|  |  |  |
| --- | --- | --- |
| Cell | **Visuals** |  |
| 1 | TEXT ON SCREEN:  Download the manual and solution projects at:  [www.cypress.com/training/wicedwifi-101](http://www.cypress.com/training/wicedwifi-101) | Hi, I’m Alan Hawse, welcome back to WICED WiFi 101. The time has come for me to show you how to connect your WICED IoT device to AWS. Once connected, you will be able to Publish MQTT messages and Subscribe to MQTT Topics – that's everything you need to build your IoT devices that connect with the world. |
| 2 | TEXT ON SCREEN:  libraries/protocols/MQTT  libraries/protocols/AWS  TEXT ON SCREEN:  Remember, you can download all the projects at:  [www.cypress.com/training/wicedwifi-101](http://www.cypress.com/training/wicedwifi-101) | The WICED SDK contains libraries that make it easy to create MQTT firmware. There is a general-purpose MQTT library in libraries/protocols/MQTT, and one that we built specifically for you to use with AWS in libraries/protocols/AWS. We will focus on the AWS libraries here but the solution projects for the class show examples using both the AWS library and the general MQTT library so feel free to browse those if you want to see different libraries in action. |
| 3 |  | To include the library in your project, you first need to include it in the Make File, and include the header files in your C source file. |
| 4 | TEXT ON SCREEN:  iot/pub\_sub/publisher  iot/pub\_sub/subscriber  iot/shadow  greengrass/publisher  greengrass/subscriber | In addition to the library, there are several demo applications included in the SDK that can be used as a starting point for using MQTT with AWS. These applications are all located in the apps/demo/aws/ directory.  The projects are:   1. iot/pub\_sub/publisher which publishes messages to a topic when a button is pressed on the kit. 2. iot/pub\_sub/subscriber which subscribes to a topic and controls an LED on the kit based on the messages received, and these two can talk to each other through the Cloud. 3. iot/shadow which shows how to interact with a Thing via the shadow and also demonstrates using a temporary configuration AP and a web server in the kit to setup the WiFi configuration and the security certificates. 4. greengrass/publisher which shows how to Publish using AWS Greengrass – a software package that extends the AWS Cloud capabilities to local devices. 5. greengrass/subscriber which shows – you guessed it – how to Subscribe using AWS Greengrass. |
| 5 | 2:47  SCREEN CAPTURE:  7-5-AWS\_WICED\_capture-1.trec | First, I'll show you the IoT publisher project and afterwards I'll walk you through all of the source code. |
| 6 | SCREEN CAPTURE:  7-5-AWS\_WICED\_capture-1.trec | The security certificates and key are included in the make file – the firmware needs Amazon's Root Certificate, the certificate for your device which you created in the last chapter, and the private key for your device. Amazon already has the public key for your device since we created it along with the certificate and the private key, so we don't need to provide that in the firmware.  First let's copy in the Amazon Trust Services root CA certificate into resources/apps/iot/. We need to delete the existing rootca.cer file and rename the one that we downloaded to that exact name since that’s what we’re going to use in the firmware.  Next, let's delete the client.cer and privkey.cer files from the publisher sub-folder and copy in our files. Again, rename them to match what the firmware is going to look for.  Notice that the AWS root CA certificate is at the resources/apps/iot level while the device certificate and key are at the iot/publisher level. That’s done so that the publisher and subscriber could have different certificates – for example to have a different policy – but the AWS root CA is always the same, so we put it at the shared location.  Now I'll go back to the make file and add the custom platform that we discussed back in chapter 2 to the list of valid platforms for this project. Otherwise, it won't build for the kit/shield combination that I'm using.  Before I forget, I'm going to do a "Clean" since I just modified the files from the SDK that are not C files or header files. I want to be sure that the make process will see all of the latest versions. |
| 7 | SCREEN CAPTURE:  7-5-AWS\_WICED\_capture-1.trec | There is one change that needs to be made to the source code. If you go to line 108 you will see the name of the AWS broker is there. You should replace that with the name of your virtual broker. If you don't know the name, you can find it from the AWS Console by clicking on the Settings. One other thing to note is the Thing name on line 117. Since we aren't using the shadow, we don't need to change it. But, one important thing to know is that every Thing that connects to your Broker must have a unique name – if not, the Things will conflict with each other and things won't work properly. See, there's that complicated "things will break Things" thing!  Next, remember to update the SSID and password for your WiFi access point in the wifi\_config\_dct.h file just like you did in chapter 5. |
| 8 | SCREEN CAPTURE:  7-5-AWS\_WICED\_capture-1.trec | Finally, create a make target for the pub\_sub/publisher app and program it to the kit. |
| 9 | 6:18  SCREEN CAPTURE:  7-5-AWS\_WICED\_capture-2.trec | From the UART terminal window, I can see that the kit has connected to the WiFi network and it's made an MQTT connection to AWS. |
| 10 | SCREEN CAPTURE:  7-5-AWS\_WICED\_capture-2.trec | Next, I'll go to the AWS Console and open the Test Client. From here I can subscribe to messages from a Topic. The publisher project uses the topic WICED\_BLUB. I'll show you that in the firmware in a minute, but just trust me for now. This project doesn't send JSON, which is OK by the rules, but it's a bit unusual, so I'll display the payload as a string, and I'll Subscribe to the Topic. |
| 11 | Probably want split screen of these:  VIDEO:  Show person pressing button on kit  SCREEN CAPTURE:  7-5-AWS\_WICED\_capture-2.trec | I press the button on the kit and – look there – we got a message that says "LIGHT OFF". If I press the button again, it will send another message that says "LIGHT ON", and then the next message is back to "LIGHT OFF". We now have an IoT connected light switch. How cool is that? |
| 12 | 7:21  SCREEN CAPTURE:  7-5-AWS\_WICED\_capture-3.trec | Looking at the firmware, wiced\_aws.h and aws\_common.h are included to provide access to the library functions and resources.h is included to provide access to the resource files – the certificates and the keys.  Then we have a few defines for the topic and the messages that the firmware sends. Next is a structure for the security credentials – its empty for now –a structure to setup the AWS endpoint – this is where you put in your broker name earlier, and a structure for the Thing's info.  In the main application loop, you will see the usual wiced\_init, and wiced\_network\_up to get connected to the WiFi. Next is a call to the helper function that takes care of the security credentials. It reads the AWS Root Certificate, and the device's certificate, and the device's private key from where we stored them in the resources directory. |
| 13 | SCREEN CAPTURE:  7-5-AWS\_WICED\_capture-3.trec | Then, back in the main loop is a call to wiced\_aws\_init. It takes the structure that contains the Thing's info from earlier and the name of a callback function. The callback function is called by the library whenever an AWS event is received. You can see here that you get a callback for a connection, a disconnection, a publish event, a subscription event, and so on. In this case, the function just sets a variable during the connection and disconnection.  After AWS initialization, the firmware creates an endpoint using the broker information structure from earlier, and then it opens up the AWS MQTT connection.  Finally, the firmware goes into a loop in which it waits for a semaphore that is set when the button is pressed. When that happens, if there is a connection, the appropriate message is published using the function wiced\_aws\_publish. You give it a handle to the AWS connection, the Topic name, the message, the message length, and the quality of service that you want.  That's it! Really not that difficult, is it? I know you can make it work. |
| 14 | 9:50  SCREEN CAPTURE:  7-5-AWS\_WICED\_capture-4.trec | Now, let's do the same thing for the Subscriber app. We allowed all IoT actions when we set up the policy for our certificate, so we can just copy over the device certificate and private key from the publisher resources to the subscriber resources. The AWS root CA certificate is always the same and it is in a shared location already so that's all set. Now let's run "Clean" to make sure the build will see all of the latest certificates.  Then, update the broker name in the source code – it's on line 119 this time. We will again just leave the Thing Name on line 128 as it is since it's different from the name that we used in the Publisher.  In the make file I'll add the custom platform to the list of valid platforms for this project.  Next, don't forget to put your WiFi SSID and password in the wifi\_config\_dct.h file for this project and finally create a make target and program it to the kit. |
| 15 | 11:01  SCREEN CAPTURE:  7-5-AWS\_WICED\_capture-5.trec | In the UART terminal I can see that it connects to WiFi, opens an MQTT connection to AWS, and then subscribes to the Topic WICED\_BULB. |
| 16 | Probably want split screen of these:  VIDEO:  Show LED turning on/off  SCREEN CAPTURE:  7-5-AWS\_WICED\_capture-5.trec | On the Test Client, I will choose "Publish to a Topic" and I'll enter the Topic name. Again, this project isn't expecting JSON for some reason, so I'll delete the JSON stuff and just enter "LIGHT ON". Now I click the "Publish to topic" button and look – the LED turns ON. I can then send "LIGHT OFF" to turn the light OFF. Now we have an IoT connected light bulb – how awesome is that? |
| 17 | 11:44  SCREEN CAPTURE:  7-5-AWS\_WICED\_capture-6.trec | The firmware is very similar to the publisher app, so I won't go through it here in detail, but you can review it on your own. One thing I will point out is the AWS callback function. In the case of the subscriber, it handles the payload received event which is called when a message is received for a Topic that you've subscribed to. |
| 18 | VIDEO:  Show 2 kits – 1 button and the other LED. | Now let's put it all together. I'll plug in one kit with the publisher firmware and another with the subscriber firmware. When I press the button on the publisher kit, it will send a message all the way to Amazon.com and the AWS MQTT broker. The broker then pushes the message on the subscriber and the light turns ON. Now we now have an IoT connected light switch controlling an IoT connected light bulb through the Cloud all using the wonderful magic that is the WICED SDK. |
| 19 |  | This is the end of HTTP and MQTT. I'm going to talk more about AMQP and CoAP in later chapters. If you've watched all of the videos up until this point – or read the manual on your own - you now have all of the tools you need to create a complete IoT device using WICED – you can setup platform files for your own custom hardware, read sensors from the pins using GPIOs or I2C, provide user information using LEDs or the OLED display, use the RTOS, use libraries, connect to WiFi and AWS using secure TLS sockets, and send and receive MQTT and HTTP messages.  In the next chapter, I'll demonstrate a complete IoT device and challenge you to replicate it on your own as the final test of your newly developed mad, amazing, wonderful, fatastic IoT skills. |
| 20 | TEXT ON SCREEN:  Cypress Developer Community  community.cypress.com  Show video of email and twitter windows. | As always, you can post your comments and questions in our WiFi developer community or as always, you are welcome to email me at alan\_hawse@cypress.com or tweet me @askioexpert with your comments, suggestions, criticisms and questions, but I would encourage you – you've got the tools to win – go forth and build amazing devices. Thank you. |