Amazon FreeRTOS Qualification Program Developer Guide

Document Version V1.0.1

Amazon FreeRTOS Qualification Program: Developer Guide

Revision History

Date	Version	Change History	Compatible Amazon FreeRTOS Version
July 31, 2018	1.0.0	Initial Version.	1.3.0
August 09, 2018	1.0.1	Updates in appendixes: • Updates in Porting Order chart. • Updates in PKCS#11 “Porting” section. • File path changes in TLS “Test Setup” section and TLS Server Setup step 9. • Fix hyperlinks in MQTT prerequisite section. • Add AWS CLI config instruction in BYOC certificate creation example.	1.3.1

Contents

Revision History 3

Introduction 6

What is the Amazon FreeRTOS Qualification Program? 6

Target System Requirements 6

Document Outline 6

Qualification Process 7

(A) Get Started with AFQP 7

(A1) Contact Amazon 7

(A2) Sign Up for AWS Partner Network 7

(A3) Agree on Shared Work Plan 7

(B) Pass Qualification Test Suite 8

(B1) Download Amazon FreeRTOS Source Code 8

(B2) Set Up Your Amazon FreeRTOS Project 8

(B2.1) Preparing Amazon FreeRTOS Folders 8

(B2.2) Create the Test Project 10

(B3) Port, Build and Test Libraries and Demos 15

(B3.1) Port Libraries, Build and Test Libraries 15

(B3.2) “Hello World” Demo and Getting Started Guide 17

(B3.3) Configure your board name 17

(C) Get Verified 18

(C1) Send Reports and Ship Development Board 18

(C2) Amazon Runs Verification Tests 18

(C3) Development Board is Amazon FreeRTOS Qualified 18

FAQs 20

Contact Us 21

Appendix 22

Appendix A: configPRINT_STRING() 23

Appendix B: FreeRTOS kernel 25

Appendix C: Wi-Fi Management 27

Appendix D: FreeRTOS TCP/IP Stack 30

Appendix E: Secure Sockets 33
Appendix F: PKCS #11 36
Appendix G: TLS 39
Appendix H: MQTT 45
Appendix I: Test List 47
Appendix J: TLS Server Setup 51
Appendix K: "Hello World" Demo Project Set Up 52
Appendix L: Checklist for Qualification 55
Appendix M: Troubleshooting Porting Setup 56
Appendix N: Instructions to Create a BYOC (ECDSA) 57
Appendix O: Source for ca.config 59
Appendix P: Modify issuer in a certificate 65
Appendix Q: Shared Work Plan for Partnership 69
Appendix R: Getting Started Guide Template 72
Appendix S: Hardware Information 75
Appendix T: Glossary 76

Introduction

What is the Amazon FreeRTOS Qualification Program?

The Amazon FreeRTOS Qualification Program (AFQP) defines a process that the author of an Amazon FreeRTOS port[1] must follow, and a set of tests that the port must pass, in order for the port to be described as 'qualified by Amazon'. Amazon only distributes and supports Amazon FreeRTOS ports that have passed the qualification program. The purpose of the AFQP is to give developers confidence that qualified Amazon FreeRTOS ports behave correctly and consistently with each other.

Target System Requirements

It is recommended that Amazon FreeRTOS is only qualified on microcontrollers (MCUs) that have a minimum processing speed of 25MHz, a minimum of 65K bytes of RAM, and a minimum of 128K bytes of program memory per executable image stored on the MCU. For future qualification requirement with Over-the-air update (OTA) functionality, two executable images must be stored in program memory at the same time.

Document Outline

Figure 1 shows the Amazon FreeRTOS qualification workflow. This document guides you through each workflow step. Appendixes provide more detail on step B3, “Port Libraries, Build, and Test Project”, with one appendix per Amazon FreeRTOS library.

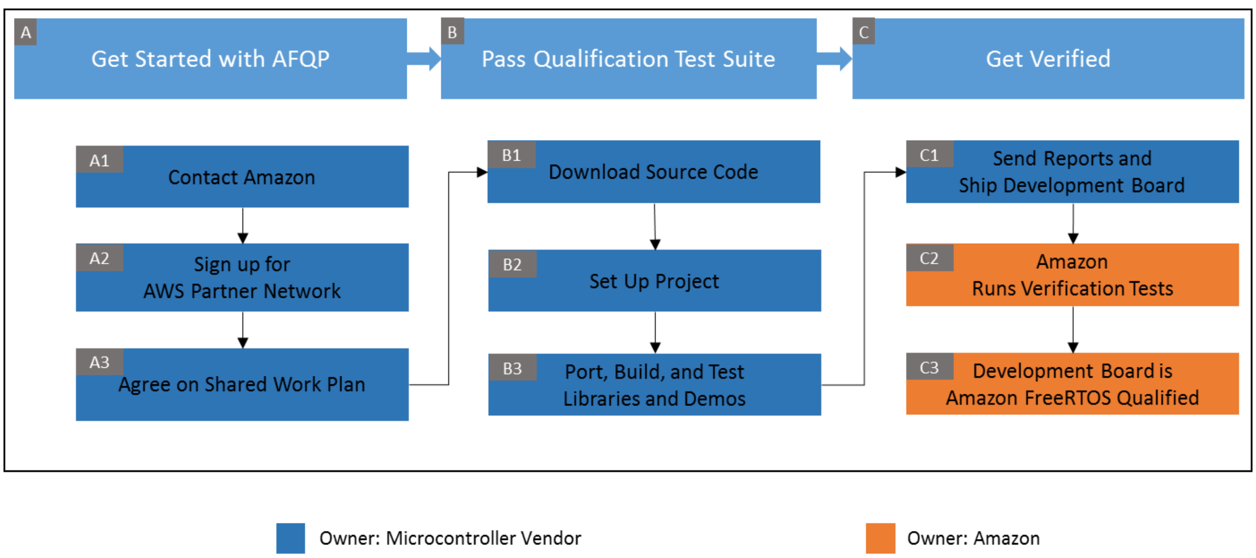


Figure 1 The Amazon FreeRTOS qualification workflow

[1] An Amazon FreeRTOS port is a board-specific implementation of APIs for certain Amazon FreeRTOS libraries. The port enables these APIs to work on the specific board, and implements the required integration with device drivers and BSPs provided by the platform vendor. It should also include any configuration adjustments (e.g. clock rate, stack size, heap size etc.) required by the board.

Qualification Process

Follow the AFQP checklist for the qualification process.

(A) Get Started with AFQP

Follow the steps below.

(A1) Contact Amazon

If you have not already done so, apply for the AFQP by sending an email to the Amazon FreeRTOS Qualification team with your name, contact, company, and board name.

(A2) Sign Up for AWS Partner Network

The AWS Partner Network (APN) is the global partner program for AWS. If you have not already done so, sign up for the APN on the APN registration page. An APN representative (referred to from here on as the "Qual-Rep") will respond and guide you through the qualification steps.

(A3) Agree on Shared Work Plan

Review and agree on a shared work plan (Appendix Q: Shared Work Plan for Partnership) with your Qual-Rep so both parties move forward with a common understanding of the joint goals, and a definition of roles, responsibilities, and commitments. You are encouraged to start working on the next step *(Pass the Qualification Test)* while you are reviewing the Shared Work Plan.

(B) Pass Qualification Test Suite

Follow the steps below.

(B1) Download Amazon FreeRTOS Source Code

You can download Amazon FreeRTOS source code and test code from GitHub:

https://github.com/aws/amazon-freertos

You should import the repository to your own private GitHub repository and configure to watch Amazon FreeRTOS public repository. You will get notifications if there are new releases on our master branch.

If you are using Windows, you must keep the file path short (for example clone to C:\AFreeRTOS rather than C:\Users\username\programs\projects\AmazonFreeRTOS\) to avoid a Windows limitation with long file paths. The chosen folder will be referred as **$AFR_HOME** from here on in the document.

(B2) Set Up Your Amazon FreeRTOS Project

At the end of this step, you will have a working project that can write to a serial console.

(B2.1) Preparing Amazon FreeRTOS Folders

All qualified Amazon FreeRTOS ports use the same directory structure. New files, including IDE project files, must be created in the correct folder locations.

The directory structure is explained below. Detailed instructions on how to create the same directory structure are listed in the grey box.

Directory Structure:

The three root level folders under $AFR_HOME are:

```
$AFR_HOME
├───demos              Contains projects that build demo applications
├───lib                Contains Amazon FreeRTOS and third-party libraries
└───tests              Contains projects that build qualification tests
```

Create your project in the `tests` folder, which is structured as follows:

```
$AFR_HOME
└───tests
    ├───common         Contains files built by all test projects
    ├───pc             Contains a reference test project for the FreeRTOS Windows port
    └───vendor         Template, to be renamed to the name of the MCU vendor
        └───board      Template, to be renamed to the name of the development board
```

Instructions:

1. Rename the **$AFR_HOME/tests/vendor** folder to the name of the company that manufactures the MCU – from here on the folder is referred to as [vendor].
2. Rename the **$AFR_HOME/tests/board** folder to the name of the development board being qualified – from here on the folder is referred to as [board].

The **$AFR_HOME/tests/[vendor]/[board]** folder is a template provided to simplify the creation of a new test project and ensures all test projects have a consistent organization. It has the following structure:

```
$AFR_HOME
└───tests
    └───[vendor]
        └───[board]
            ├───common
            │   ├───application_code    Contains main.c, which itself contains main()
            │   │   └───vendor_code     Contains vendor supplied board specific files
            │   └───config_files        Contains Amazon FreeRTOS config files
            └───ide                     Contains an IDE specific project
```

Test projects always require vendor-supplied driver libraries. Some vendor-supplied libraries, such as a header file that maps a GPIO output to an LED output, are specific to the target development board. Other vendor-supplied libraries, such as the GPIO library itself, are specific to the target MCU family.

For Vendor-supplied driver libraries that are **specific to the target development board**:

Instructions (continued):

3. Save any required vendor-supplied libraries that are specific to the board in the **$AFR_HOME/tests/[vendor]/[board]/common/application_code/vendor_code** folder.
4. Rename the **$AFR_HOME/tests/[vendor]/[board]/ide** folder to the name of the IDE that will be used to build the test project – from here on the folder is referenced as [IDE].

Vendor-supplied driver libraries that are **specific to the target MCU family** belong in the **$AFR_HOME/lib/third_party/mcu_vendor** folder, which has the following structure:

```
$AFR_HOME
└─lib
    ├───AmazonLib1      Contains an Amazon FreeRTOS library (example name only)
    ├───AmazonLib2      Contains an Amazon FreeRTOS library (example name only)
    └───third_party     Contains all non-board specific third-party libraries
        ├───Lib1        Contains a third-party library (example name only)
        ├───Lib2        Contains a third-party library (example name only)
        └───mcu_vendor  Contains vendor-supplied MCU specific libraries
            └──vendor   Template, to be renamed to the name of the MCU vendor
               └─driver_library Template, to be renamed to the library name
                  └─driver_library_version Template, to be renamed to the library version
```

Instructions (continued):

5. Rename the **$AFR_HOME/lib/third_party/mcu_vendor/vendor** folder to [vendor] (the name of the company that manufactures the MCU).
6. Likewise, rename the contained **driver_library** folder to the name of the vendor's MCU specific driver library, and the contained **driver_library_version** folder to the version number of the vendor's MCU specific driver library.
7. Copy the vendor-supplied driver library into the newly renamed **driver_library_version** folder.

NOTE: DO NOT save vendor-supplied libraries that are specific to the MCU family anywhere within either the **$AFR_HOME/tests** or **$AFR_HOME/demos** folders!

(B2.2) Create the Test Project

All qualified Amazon FreeRTOS test projects look the same when viewed from within an IDE. This section describes and demonstrates the required project structure. By the end of this section you will have a project with FreeRTOS Kernel libraries ready to run. The next section (B3) will cover porting of other Amazon FreeRTOS libraries into the project.

Instructions:

1. Read this section of the document, and then **replicate** the project it describes, but using the selected IDE, and targeting the hardware being qualified. Take care to ensure the structure of the created project matches that described below.

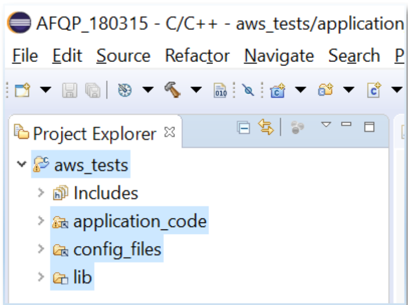
NOTE 1: All files in the project must be built in the file's original position within the folder structure. They are imported into the project by linking the files. Never directly copy files into the project's folder or use absolute file paths.

NOTE 2: If you are using an Eclipse based IDE, do not configure the project to build all the files in any given folder. Instead, add source files into the project by linking to each source file individually.

The project is called **aws_tests**. Under **aws_tests**, there are three virtual folders. In this context, a virtual folder is created in an IDE to better organize the source code. It may not correspond to a physical directory on disk. The three virtual folders are **application_code**, **config_files** and **lib**, as described below:

```
aws_tests               The project name
├──application_code       Contains application logic, in this case it is AFQP test code
├──config_files           Contains header files that configure Amazon FreeRTOS libraries
└──lib                    Contains Amazon and third-party libraries
```

Figure 2 shows how the top three virtual folders appear in an IDE. The depicted IDE is Eclipse, but the structure is the same in all IDEs.

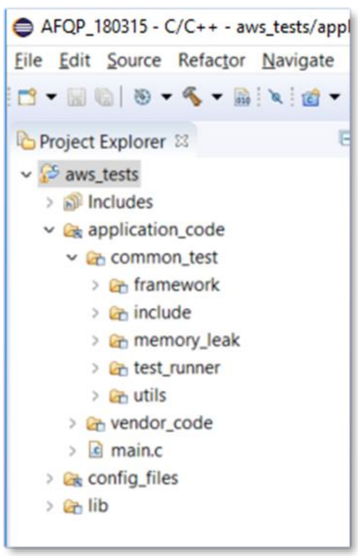


NOTE:

- The “Includes” folder is generated automatically by Eclipse. It is not part of the required structure

Figure 2 The top three virtual folders viewed in an IDE’s project explorer view

Figure 3 shows the contents of the **application_code** virtual folder.

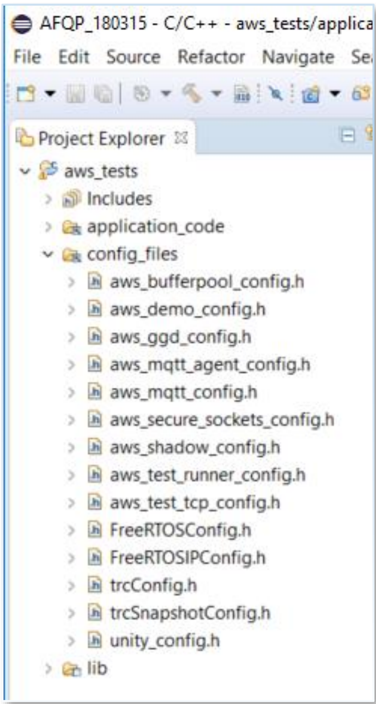


NOTE:

- The **main.c** file, and the **vendor_code** folder, are (physically on the disk) located in the **$AFR_HOME/tests/[vendor]/[board]/common/ application_code** folder.
- **common_test** is a virtual folder – it does not actually exist within the **$AFR_HOME** directory structure. The folders under **common_test** are located in the **$AFR_HOME/tests/common** folder. The project builds the source files located in those folders.

Figure 3 An IDE project with the application_code virtual folder expanded

Figure 4 shows the contents of the **config_files** virtual folder.

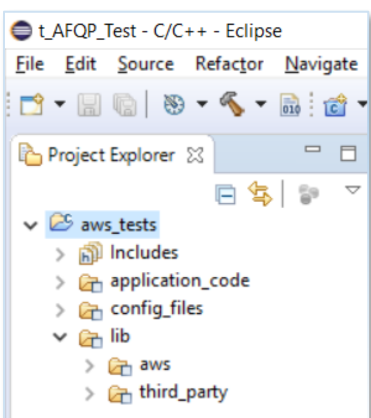


NOTE:

- The files shown under **config_files** are located in the **$AFR_HOME/tests/[vendor]/[board]/common/config_files** folder.

Figure 4 An IDE project with the config_files folder expanded

Figure 5 shows the contents of the **lib** virtual folder.



NOTE:

- **aws** and **third_party** are virtual folders.

Figure 5 IDE project with the lib group expanded

Figure 6 shows the contents of the **lib/aws** folder, which contains AWS (as opposed to third party) provided libraries. It only contains FreeRTOS kernel library at this stage. You will import more libraries in this folder during later porting effort.

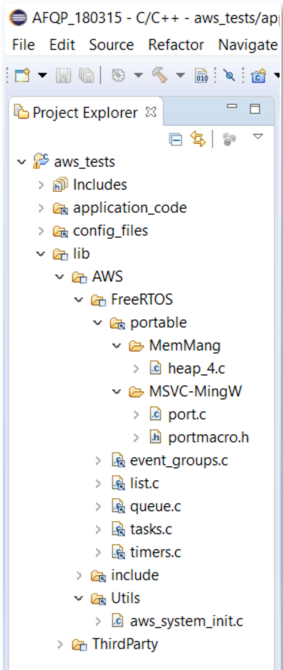


Figure 6 IDE project with the lib/AWS group expanded

NOTE:

- The files and folders shown under **lib/aws/FreeRTOS** are located in the **$AFR_HOME/lib/FreeRTOS** folder. The figure shows **lib/aws/FreeRTOS/portable/MSVC-MingW** being included in the project. That folder contains the FreeRTOS kernel Windows port and should be substituted with whichever folder contains the correct FreeRTOS port for your target IDE and MCU, see below instruction.
- The file shown under **lib/aws/FreeRTOS/portable/MemMang** is located in the **$AFR_HOME/lib/FreeRTOS/MemMang** folder. It is FreeRTOS memory management implementation.
- The files shown under **lib/aws/include** are located in the **$AFR_HOME/lib/include** folder. Although not shown in Figure 6, it includes all the header files and folders under **$AFR_HOME/lib/include**.
- **aws_system_init.c** is located in the **$AFR_HOME/lib/utils** folder.

Instructions (continued):

1. Replicate the folder structure (continued)
   a. Import the FreeRTOS Kernel port for your compiler and architecture **in place of** *lib/aws/FreeRTOS/portable/MSVC_MingW in Figure 6.* **$AFR_HOME/lib/FreeRTOS/portable** contains the FreeRTOS kernel port files organized first by compiler, and then by architecture.
   b. Import one of the FreeRTOS Kernel memory management implementation to **lib/aws/FreeRTOS/portable/MemMang**. For Amazon FreeRTOS, we use heap_4.c. For more information, please visit [FreeRTOS Memory Management]

Figure 7 shows the contents of the **lib/third_party** directory.

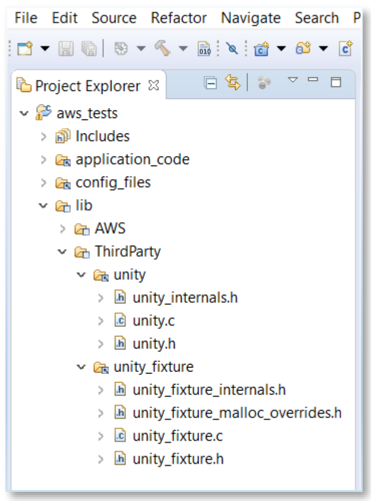


Figure 7 IDE project with the lib/third_party group expanded

NOTE:

- **unity** and **unity_fixture** are virtual folders.
- The files shown under **unity** are located in the **$AFR_HOME/tests/common/third_party/unity/src** folder.
- The files shown under **unity_fixture** are located in the **$AFR_HOME/tests/common/third_party/unity/extras/fixture** folder.
- Although not shown in Figure 7, also add the MCU specific vendor-supplied driver libraries that were saved in the **$AFR_HOME/lib/third_party/[mcu_vendor]/[vendor]/[driver_library]/[driver_library_version]** folder (see section (B2.1) Preparing Amazon FreeRTOS Folders).

Instructions (continued, to be followed after creating the project):

2. Make sure the following compiler include paths are set in the project property.
   a. **$AFR_HOME/tests/common/include**, which is aws_tests/application_code/common_tests/include when viewed in the IDE project.
   b. **$AFR_HOME/lib/include**, which is aws_tests/lib/aws/include when viewed in the IDE.
   c. **$AFR_HOME/lib/include/private**, which is aws_tests/lib/aws/include/private when viewed in the IDE.
   d. **$AFR_HOME/lib/FreeRTOS/portable/[compiler]/[architecture]**, which is aws_tests/lib/aws/FreeRTOS/portable/[compiler]/[architecture] when viewed in the IDE.
   e. **$AFR_HOME/tests/common/third_party/unity/src**, which is aws_tests/lib/third_party/unity when viewed in the IDE.
   f. **$AFR_HOME/tests/common/third_party/unity/extras/fixture/src**, which is aws_tests/lib/third_party/unity_fiture when viewed in the IDE.
   g. **$AFR_HOME/demos/vendor/board/common/config_files**, which is aws_tests/config_files when viewed in the IDE.
   h. Any paths necessitated by vendor-supplied driver libraries.

3. Define the following two project level macros in your IDE:
   - UNITY_INCLUDE_CONFIG_H
   - AMAZON_FREERTOS_ENABLE_UNIT_TESTS

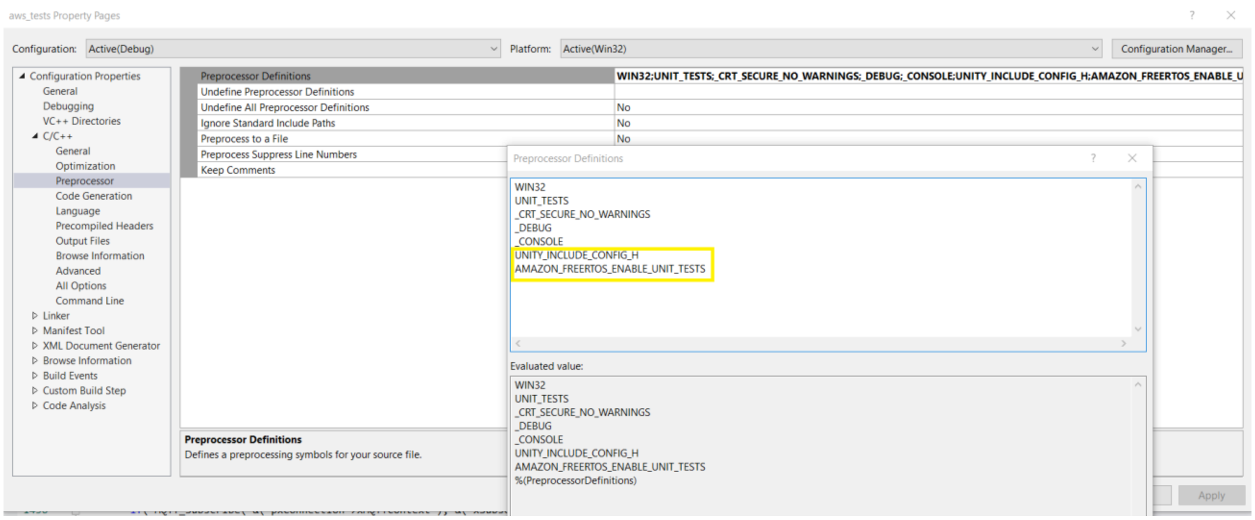


Figure 8 Preprocessor Macro Definitions Example (Visual Studio)

Visual Studio 2017 Example: Project Properties => Preprocessor => Preprocessor Definitions

4. Implement configPRINT_STRING() as described in Appendix A.
5. Make sure the new project builds successfully, that the resultant executable binary can be loaded to the target hardware. If you run the project in debug mode, the pc should stop at the first line of `main()`.

(B3) Port, Build and Test Libraries and Demos

(B3.1) Port Libraries, Build and Test Libraries

With the folder structure and test projects prepared you are ready to start porting and testing the Amazon FreeRTOS libraries. You must enable **AFQP test group** first.

$AFR_HOME/tests/[vendor]/[board]/common/config_files/aws_test_runner_config.h contains a macro defined shown below. Uncomment the following line:

```
/*#define testrunnerAFQP_ENABLED */
```

Then bring an Amazon FreeRTOS library into your test project and port the library to your hardware. The libraries are listed in below table. The detailed instruction on porting and testing procedures are listed in the appendices – one appendix per library. The order of the appendices accounts for interdependencies between libraries so should be followed in turn.

Library	Details in Location	Notes
configPRINT_STRING Macro	Appendix A	Required for console output.

FreeRTOS Kernel	Appendix B	Required for the FreeRTOS kernel.
Wi-Fi Management	Appendix C	Required only if hardware supports network connectivity over Wi-Fi.
FreeRTOS TCP/IP Stack	Appendix D	Required only if a board does not have its own TCPIP stack support.
TCP/IP Socket	Appendix E	Required for AWS cloud connectivity.
PKCS#11	Appendix F	Required for OTA* and TLS support.
TLS	Appendix G	Required for TLS support.
MQTT	Appendix H	Required for AWS cloud connectivity.

* The OTA feature is in Beta and currently not covered in this document.

There are constants defined in **$AFR_HOME/tests/[vendor]/[board]/common/config_files/*aws_test_runner_config.h*** file which can be used as a switch to trigger a test set for a library. To enable a set of tests, set the constant to 1. These constants listed in table below:

Defined constant to trigger a set of tests on a library	Default value. (set to '1' for enabling the test)
testrunnerFULL_CBOR_ENABLED	0
testrunnerFULL_OTA_AGENT_ENABLED	0
testrunnerFULL_OTA_PAL_ENABLED	0
testrunnerFULL_MQTT_ALPN_ENABLED	0
testrunnerFULL_MQTT_STRESS_TEST_ENABLED	0
testrunnerFULL_MQTT_AGENT_ENABLED	0
testrunnerFULL_TCP_ENABLED	0
testrunnerFULL_GGD_ENABLED	0
testrunnerFULL_GGD_HELPER_ENABLED	0
testrunnerFULL_SHADOW_ENABLED	0
testrunnerFULL_MQTT_ENABLED	0
testrunnerFULL_PKCS11_ENABLED	0
testrunnerFULL_CRYPTO_ENABLED	0
testrunnerFULL_TLS_ENABLED	0
testrunnerFULL_WIFI_ENABLED	0

(B3.2) "Hello World" Demo and Getting Started Guide

Prepare the "Hello World" demo project in the code packaged delivered to Amazon. The project creation process is similar to the test project creation. Please see Appendix K: "Hello World" Demo Project Set Up for detailed instructions.

Prepare a "Getting Started Guide" for your board to help users run the Hello World Demo project (and any other demos you may include). You can use the Getting Started Guide template to start and look at the guide for the Window Simulator for reference.

(B3.3) Configure your board name

Please put your board name in:
$AFR_HOME/demos/[vendor]/[board]/common/config_files/FreeRTOSConfig.h

```
#define mqttconfigMETRIC_PLATFORM    "Platform=Unknown"
```

Replace "Unknown" with your own board name.

(C) Get Verified

(C1) Send Reports and Ship Development Board

Test results are output to a UART console.

```
Starting Tests...

TEST(Full_TLS, AFQP_TLS_ConnectEC) PASS

TEST(Full_TLS, TLS_ConnectRSA) PASS

TEST(Full_TLS, TLS_ConnectMalformedCert) PASS

TEST(Full_TLS, TLS_ConnectUntrustedCert) PASS

TEST(Full_TLS, AFQP_TLS_ConnectBYOCCredentials) PASS

-----------------------
5 Tests 0 Failures 0 Ignored
OK
-------ALL TESTS FINISHED-------
```

Figure 9 Console Output of Test Results (Example)

Instructions to submit test results:

1. Once all the tests pass, copy the console output into a text file called "Test_Report.txt". If memory constraints require the tests to be run in batches, then concatenate the output generated by each batch in the same Test_Report.txt file.

   Upload the test report file to your private GitHub repository. Add the "Qual-rep" to your GitHub repository as a "contributor" so that he/she has the access to your test result and qualification-ready code. The "Qual-rep" later will be able to fetch the qualification-ready code from your private GitHub repository.

2. Ship three development boards, along with all required power adapters, cables and setup instructions, to

   Amazon ATTN: Amazon FreeRTOS AFQP
   1800 9th Ave., 15th floor
   Seattle, WA 98101

(C2) Amazon Runs Verification Tests

After receiving the test reports and the development boards, Amazon will run verification tests.

(C3) Development Board is Amazon FreeRTOS Qualified

If the verification tests are successful, your development board will be Amazon FreeRTOS qualified. On a mutually agreed timeframe, it will be included in the Amazon FreeRTOS console. The port code and driver code for the board will be merged in Amazon FreeRTOS public GitHub repository. The details of the development board, your company logo, and relevant links will be included in the Amazon FreeRTOS Getting Started webpage and Amazon FreeRTOS documents.

FAQs

1. **What is an Amazon FreeRTOS port?**
   An Amazon FreeRTOS port is a board-specific implementation of APIs for certain Amazon FreeRTOS libraries. The port enables these APIs to work on the specific board, and implements the required integration with device drivers and BSPs provided by the platform vendor. It should also include any configuration adjustments (e.g. clock rate, stack size, heap size) required by the board.

2. **Do I need to retest for minor version releases of Amazon FreeRTOS?**
   There is no need to retest for qualification with minor version releases of Amazon FreeRTOS.

3. **What network ports will need to be opened to run AFQP tests?**
   The network connections needed in the AFQP tests include

Port	Protocol
443, 8883	MQTT
8443	Greengrass Discovery

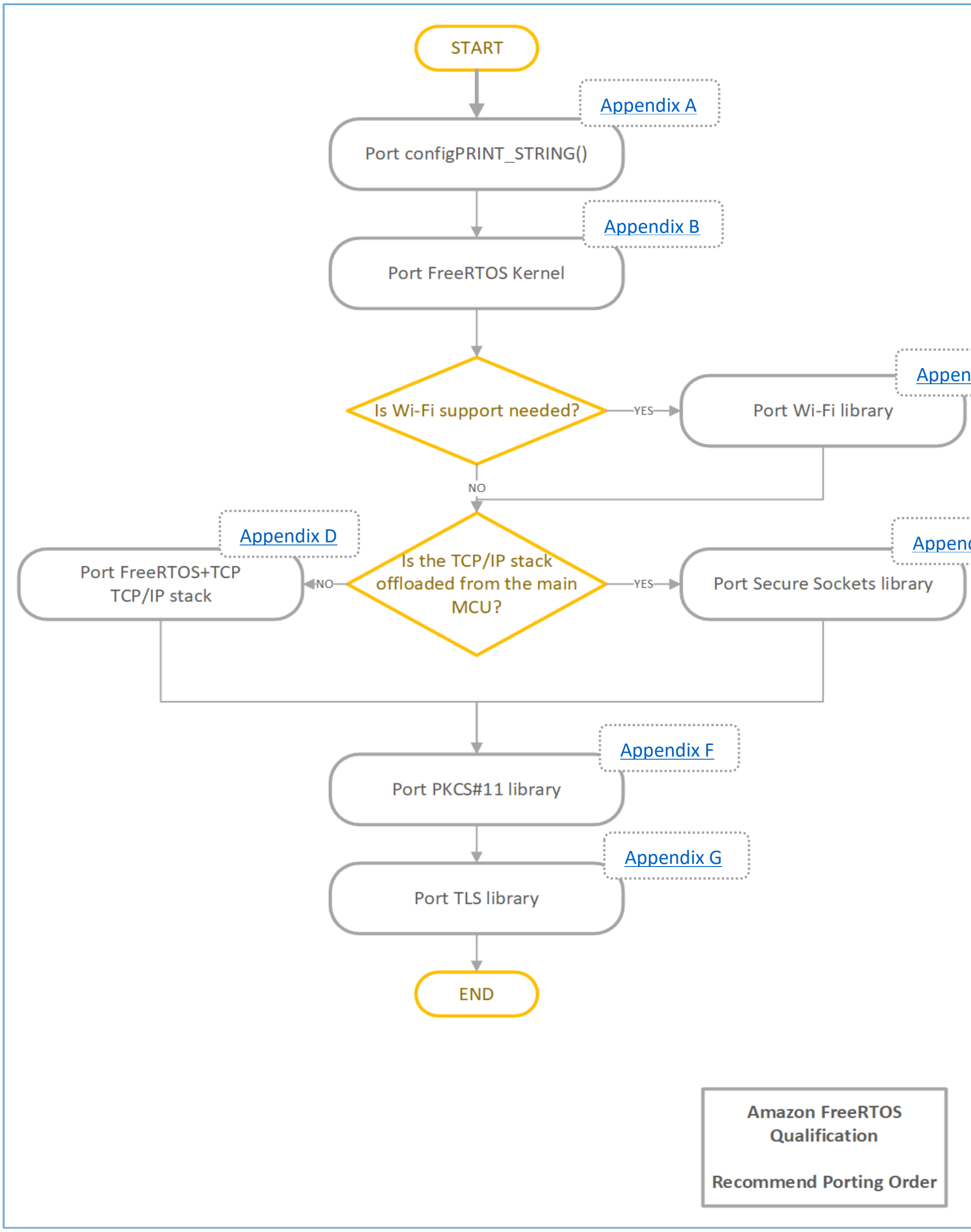
Contact Us

If you have any questions, please contact your Qual-Rep or the Amazon FreeRTOS Qualification team.

Appendix

The Appendix contains detailed descriptions of Amazon FreeRTOS libraries and macros to be ported, as well as steps to verify them.

Here is the recommended porting order:



Appendix A: configPRINT_STRING()

Description

`configPRINT_STRING()` is a macro used by the AFQP test framework to output test results as human readable ASCII strings. It must be implemented before AFQP porting and testing can begin.
These instructions assume test results are output over a UART serial port.

Pre-requisites

1. A development board that supports UART or virtual COM port output.
2. A test project that was created in accordance with the instructions provided in the body of this document, and that is building vendor-supplied UART initialization and output functions.
3. The UART initialization and output must not have any dependency on FreeRTOS.

Setup

1. Connect a terminal emulator, such as TeraTerm, to the port on the target hardware that is to be used to output test results.

Porting

1. Locate the call to `configPRINT_STRING`(“Test Message”) within the function prvMiscInitialization(), which is itself in the file `$AFR_HOME/tests/[vendor]/[board]/common/application_code/main.c.`
2. Immediately before the call to `configPRINT_STRING`(“Test Message”), add code that uses the vendor-supplied UART driver to initialize the UART to 115200 baud
3. `$AFR_HOME/tests/[vendor]/[board]/common/config_files/FreeRTOSConfig.h` contains an empty definition of `configPRINT_STRING().` The macro takes a NULL terminated ASCII C string as its only parameter. Update the empty definition of `configPRINT_STRING()` so that it calls the vendor-supplied UART output function. For example, if the UART output function has the following prototype:

```
void MyUARTOutput( char *DataToOutput, size_t LengthToOutput );
```

then you would implement `configPRINT_STRING()` as:

```
#define configPRINT_STRING( X ) MyUARTOutput( (X), strlen( (X) ) )
```

Manual Testing

Build and execute the application. If “Test Message” appears in the UART console then the console is connected and configured correctly, and `configPRINT_STRING()` is behaving as expected. If this is the case then `configPRINT_STRING()` testing is complete and the call to `configPRINT_STRING( “Test Message” )` can be removed from `prvMiscInitialization()`.

Appendix B: FreeRTOS kernel

Description

Amazon FreeRTOS uses the FreeRTOS kernel for multitasking and inter-task communications. This appendix describes how to integrate a port of the FreeRTOS kernel into the AFQP test project.

The FreeRTOS.org website contains a list of all the available kernel ports.

Porting the FreeRTOS kernel to a new architecture is out of scope of this document. Contact the Amazon FreeRTOS Qualification team if a port does not exist for your architecture.

Pre-requisites

1. An official FreeRTOS kernel port for the target MCU architecture.
2. A test project that was created in accordance with the instructions provided in the body of this document ((B2.2) Create the Test Project), and that includes the correct FreeRTOS kernel port files for the MCU and compiler in use.
3. An implementation of configPRINT_STRING() that was created and tested as described in Appendix A.

Porting

The header file `$AFR_HOME/tests/[vendor]/[board]/common/config_files/FreeRTOSConfig.h` contains application specific FreeRTOS kernel configuration settings. The FreeRTOS.org website provides a description of each configuration option. In particular, ensure the following constants are set correctly for your hardware:

Configuration definitions	Comment
`configCPU_CLOCK_HZ`	The frequency of the clock used to generate the tick interrupt.
`configMINIMAL_STACK_SIZE`	As a starting point, this can be set to whichever value is used in the official FreeRTOS demo for the FreeRTOS kernel port in use. Official FreeRTOS demos are those distributed from the FreeRTOS.org web site. Ensure stacks overflow checking is set to 2, and increase configMINIMAL_STACK_SIZE if overflows occur. To save RAM, set stack sizes to the minimum value that does not result in a stack overflow.
`configTOTAL_HEAP_SIZE`	Sets the size of the FreeRTOS heap. Like task stack sizes, the heap size can be tuned to ensure unused heap space does not consume RAM.

ARM Cortex-M3, M4 and M7 devices must also have `configPRIO_BITS and configMAX_SYSCALL_INTERRUPT_PRIORITY` set correctly.

The MQTT (Appendix H) and Secure Sockets (Appendix E) libraries have not been ported yet, so it is necessary to comment out the lines that call `BUFFERPOOL_Init()`, `MQTT_AGENT_Init()` and `SOCKETS_Init()` from within function `SYSTEM_Init()`, which is implemented in `$AFR_HOME/lib/utils/aws_system_init.c`.

Test Setup

No further setup is required for this section.

Qualification Test

1. Build and execute the project.
2. If a "." appears in the UART console every 5 seconds then the FreeRTOS kernel is operating as expected and this test is complete. Set configUSE_IDLE_HOOK to 0 in `$AFR_HOME/tests/[vendor]/[board]/common/config_files/FreeRTOSConfig.h` before moving to the next section. Setting configUSE_IDLE_HOOK to 0 stops the FreeRTOS kernel from executing vApplicationIdleHook(), and so stop the "." Being output during future test executions.
3. If a "." appears at any other frequency then check the setting of configCPU_CLOCK_HZ in `$AFR_HOME/tests/[vendor]/[board]/common/config_files/FreeRTOSConfig.h`. `configCPU_CLOCK_HZ` must be set to the correct value for your board.

Appendix C: Wi-Fi Management

Description

The Wi-Fi Management library is the Amazon FreeRTOS interface to vendor-supplied Wi-Fi drivers. Skip this section if your hardware does not support Wi-Fi.

Pre-requisites

1. A test project that was created in accordance with the instructions provided in the body of this document, and that is building vendor-supplied Wi-Fi drivers.
2. An implementation of configPRINT_STRING() that was created and tested as described in Appendix A.
3. A validated FreeRTOS kernel configuration, as described in Appendix B: FreeRTOS kernel
4. Two wireless Access Points.

Preparing the IDE Project

1. Add the source file `$AFR_HOME/lib/wifi/portable/[vendor]/[board]/aws_wifi.c` into the `[project_top_level]/lib/aws/wifi` virtual folder of the IDE project.
2. Add the source file `$AFR_HOME/tests/common/wifi/aws_test_wifi.c` into the `[project_top_level]/application_code/common_tests/wifi` virtual folder of the IDE project.
3. Enable the Wi-Fi connection code in `$AFR_HOME/tests/[vendor]/[board]/common/application_code/main.c` by deleting the `#if 0` and `#endif` compiler directives in the functions `vApplicationDaemonTaskStartupHook(void)` and `prvWifiConnect(void)`.

Porting

1. `$AFR_HOME/lib/wifi/portable/[vendor]/[board]/aws_wifi.c` contains empty definitions of a set of Wi-Fi management functions. Use the vendor-supplied Wi-Fi driver library to implement at least the subset of functions listed in the table below. `$AFR_HOME/lib/include/aws_wifi.h` provides the information necessary to complete the implementations.

Function	Description
WIFI_On	Turns on Wi-Fi module. Initializes the drivers
WIFI_ConnectAP	Connects to a Wi-Fi Access Point (AP)
WIFI_Disconnect	Disconnects from the currently connected AP
WIFI_Scan	Performs a Wi-Fi network scan
WIFI_GetIP	Retrieves the Wi-Fi interface's IP address
WIFI_GetMAC	Retrieves the Wi-Fi interface's MAC address
WIFI_GetHostIP	Retrieves the host IP address from a hostname using DNS

Since the MQTT library is not used for running Wi-Fi tests, comment out the lines that call `BUFFERPOOL_Init(), MQTT_AGENT_Init()`in function `SYSTEM_Init()`in file `$AFR_HOME/lib/utils/aws_system_init.c.` Bufferpool and mqtt_agent are used in MQTT library. If you have not ported the SOCKETS library (Appendix E), comment out the line that calls `SOCKETS_Init()`in function `SYSTEM_Init(),` in file `$AFR_HOME/lib/utils/aws_system_init.c.`

Test Setup

1. In `$AFR_HOME/tests/common/include/aws_clientcredential.h`, set the macros shown in table below to be correct for the first Wi-Fi access point.

Macro name	Value
clientcredentialWIFI_SSID	The Wi-Fi SSID as a C string (in quotes)
clientcredentialWIFI_PASSWORD	The Wi-Fi password as a C string
clientcredentialWIFI_SECURITY	Either eWiFiSecurityOpen, eWiFiSecurityWEP, eWiFiSecurityWPA, or eWiFiSecurityWPA2. eWiFiSecurityWPA2 is recommended.

2. In `$AFR_HOME/tests/common/wifi/aws_test_wifi.c`, set the macros shown in table below to be correct for the second Wi-Fi access point.

Macro name	Value
testwifiWIFI_SSID	The Wi-Fi SSID as a C string (in quotes)
testwifiWIFI_PASSWORD	The Wi-Fi password as a C string
testwifiWIFI_SECURITY	Either eWiFiSecurityOpen, eWiFiSecurityWEP, eWiFiSecurityWPA, or eWiFiSecurityWPA2. eWiFiSecurityWPA2 is recommended.

The Wi-Fi management tests listed in the bullet points below have a dependency on the Secure Sockets library, which may not have been ported yet. If the Secure Sockets library has not been ported then all the Wi-Fi management tests other than those listed in the bullets below must pass. After the Secure Sockets library has been ported it is necessary to re-run the Wi-Fi management tests to ensure all the tests (including those in the bullet points below) pass. Additionally, the tests listed in the bullet points attempt to communicate with an echo server. See the Appendix E: Secure Sockets section "Test Setup" for information on starting the echo server.

- WiFiConnectionLoop
- WiFiIsConnected
- WiFiConnectMultipleAP
- WiFiSeperateTasksConnectingAndDisconnectingAtOnce

3. Enable the Wi-Fi tests, by setting the `testrunnerFULL_WIFI_ENABLED` macro in `$AFR_HOME/tests/[vendor]/[board]/common/config_files/aws_test_runner_config.h` to 1.

Qualification Test

1. Build and execute the test project.
2. View the test results in the UART console. As noted in the 'Test Setup' section of this appendix, not all the tests will pass until porting of the Secure Sockets library is complete. If all the tests that are expected to pass are passing, then save the test results by cutting and pasting them from the UART console into a text file, and move to the next section.

Example output:

```
Starting Tests...
TEST(Full_WiFi, WiFiOnOff) PASS
TEST(Full_WiFi, WiFiMode) PASS
TEST(Full_WiFi, WiFiConnectionLoop) PASS
TEST(Full_WiFi, WiFiNetworkAddGetDelete) PASS
TEST(Full_WiFi, WiFiPowerManagementMode) PASS
TEST(Full_WiFi, WiFiGetIP) PASS
```

…

```
TEST(Full_WiFi, WIFI_NetworkGet_GetManyNetworks) PASS
TEST(Full_WiFi, WIFI_NetworkAdd_AddManyNetworks) PASS
TEST(Full_WiFi, WIFI_NetworkDelete_DeleteManyNetworks) PASS
TEST(Full_WiFi, WIFI_ConnectAP_ConnectAllChannels) PASS
-----------------------
46 Tests 0 Failures 0 Ignored
OK
----All tests finished----
```


Appendix D: FreeRTOS TCP/IP Stack

Description

FreeRTOS+TCP is the TCP/IP stack used by Amazon FreeRTOS. See https://www.freertos.org/FreeRTOS-Plus/FreeRTOS_Plus_TCP for more information. Skip this section if your target hardware offloads TCP/IP functionality to a separate network processor or module.

This appendix only describes how to create a port to the target hardware's Ethernet driver, and test as far as ensuring the Ethernet driver can connect to the network. Actually sending and receiving data is not tested until the Secure Sockets library port is complete.

Pre-requisites

1. A test project that was created in accordance with the instructions provided in the body of this document, and that is building vendor-supplied Ethernet drivers.
2. An implementation of configPRINT_STRING() that was created and tested as described in Appendix A.
3. A validated FreeRTOS kernel configuration, as described in Appendix B: FreeRTOS kernel.

Preparing the IDE Project

In all steps below, add source files to the IDE project from their existing location on the disk (by reference) – do not create duplicate copies of source files on the disk:

1. Add the source files from the `$AFR_HOME/lib/FreeRTOS-Plus-TCP/source` directory to the [project_top_level]/lib/FreeRTOS-Plus-TCP/source folder of the IDE project.
2. Add the header files from the `$AFR_HOME/lib/FreeRTOS-Plus-TCP/include` directory to the `[project_top_level]/lib/FreeRTOS-Plus-TCP/include` folder of the IDE project.
3. Add the port source files from the `$AFR_HOME/lib/FreeRTOS-Plus-TCP/source/portable/NetworkInterface/[board_family]/` directory to the `[project_top_level]/lib/FreeRTOS-Plus-TCP/portable/NetworkInterface` folder of the IDE project.
4. Add the `$AFR_HOME/lib/FreeRTOS-Plus-TCP/source/portable/BufferManagement/BufferAllocation_2.c` source file to the `[project_top_level]/lib/FreeRTOS-Plus-TCP/portable/BufferManagement` folder of the IDE project.

FreeRTOS has five example heap implementations under `$AFR_HOME /lib/FreeRTOS/portable/MemMang`. Using FreeRTOS+TCP and `BufferAllocation_2.c` requires the heap_4.c implementation.

5. Add the directory `$AFR_HOME/lib/FreeRTOS-Plus-TCP/include` to your compiler's include path.

Porting

1. Check the `$AFR_HOME/lib/FreeRTOS-Plus-TCP/source/portable/NetworkInterface/` directory to see if a port to your target hardware already exists.
2. If a port does not exist, then:
    a. Rename the `$AFR_HOME/lib/FreeRTOS-Plus-TCP/source/portable/NetworkInterface/[board_family]` directory to be appropriate for the target hardware.
    b. Follow the instruction on the FreeRTOS.org website for porting the TCP/IP stack to a different microcontroller, and if necessary, a different compiler, to create a new port that uses the vendor supplied Ethernet drivers. Implement the new port in a file called NetworkInterface.c, and save the file in the newly renamed directory.

   Note: The files in `$AFR_HOME/lib/FreeRTOS-Plus-TCP/source/portable/BufferManagement` are used by multiple ports so must not be edited.

3. Update the FreeRTOS+TCP configuration file `$AFR_HOME/tests/[vendor/[board]/common/config_files/FreeRTOSIPConfig.h` so it is correct for your target hardware. The FreeRTOS.org website describes each configuration option.

Since the MQTT library is not used for running Wi-Fi tests, comment out the lines that call `BUFFERPOOL_Init()`, `MQTT_AGENT_Init()` in function `SYSTEM_Init()` in file `$AFR_HOME/lib/utils/aws_system_init.c`. Bufferpool and mqtt_agent are used in MQTT library. If you have not ported the SOCKETS library (Appendix E), comment out the line that calls `SOCKETS_Init()` in function `SYSTEM_Init()`, in file `$AFR_HOME/lib/utils/aws_system_init.c`.

Test Setup

1. In `$AFR_HOME/tests/[vendor]/[board]/common/application_code/main.c`, uncomment the call to `FreeRTOS_IPInit()` in `main()`. By default, the IP address is acquired by DHCP. If DCHP fails or you do not want to use DHCP, you must set a static IP address in `$AFR_HOME/tests/[vendors]/[board]/common/application_code/main.c`. The following variables must be set to valid values of your actual network:

Network configuration variables	Description
uint8_t ucMACAddress[6]	
uint8_t ucIPAddress[4]	
uint8_t ucNetMask[4]	
uint8_t ucGatewayAddress[4]	
uint8_t ucDNSServerAddress[4]	
uint8_t ucMACAddress[6]	

2. In `$AFR_HOME/tests/[vendor]/[board]/common/config_files/FreeRTOSIPConfig.h` set the `ipconfigUSE_NETWORK_EVENT_HOOK` macro to 1.

3. In `$AFR_HOME/tests/[vendor]/[board]/common/application_code/main.c` add the code below at the start of `vApplicationIPNetworkEventHook()`:

```
if (eNetworkEvent == eNetworkUp)
{
    configPRINT("Network connection successful.\n\r");
}
```

Qualification Test

1. Build and execute the test project.
2. If "Network connection successful" appears in the UART console, then the Ethernet driver has successfully connected to the network and this test is complete.

Appendix E: Secure Sockets

Description

The Secure Sockets library API is based on the Berkeley sockets API. It provides the API functions necessary to create and configure a TCP socket, connect to an MQTT broker, and send and receive TCP data.

The library is called Secure Sockets as it also encapsulates TLS functionality. To create a TLS protected socket the application writer need only creates a standard TCP socket, then uses a setsockopt call make the socket use TLS.

If your target hardware does not offload TCP/IP functionality to a separate network chip then use the FreeRTOS+TCP TCP/IP stack. A Secure Sockets implementation already exists for the FreeRTOS+TCP TCP/IP stack used in conjunction with mbedTLS – so if you are using those libraries no porting is necessary, but the Secure Sockets tests must still be executed and pass.

Pre-requisites

1. If you are using Wi-Fi for network connectivity: A port of the Wi-Fi management library as described in Appendix TBD.
2. If you are using the FreeRTOS+TCP TCP/IP stack: A port of the FreeRTOS+TCP library as described in Appendix TBD.

Preparing the IDE Project

In all steps below, add source files to the IDE project from their existing location on the disk (by reference) – do not create duplicate copies of source files on the disk:

1. Add `$AFR_HOME/lib/secure_sockets/portable/[vendor]/[board]/aws_secure_sockets.c` to the `[project_top_leve]/lib/aws/secure_sockets` folder of the IDE project.
2. Add `$AFR_HOME/tests/common/secure_sockets/portable/[vendor]/[board]/aws_test_tcp_portable.h` and `$AFR_HOME/tests/common/secure_sockets/portable/[vendor]/[board]/aws_test_tcp.c` to the `[project_top_level]/application_code/common_tests/secure_sockets` folder of the IDE project.

Porting

If you use the FreeRTOS+TCP TCP/IP stack then no porting is necessary – simply copy the existing FreeRTOS+TCP port from TBD to `$AFR_HOME/lib/secure_sockets/portable/[vendor]/[board].`

If your target hardware offloads TCP/IP functionality to a separate network chip then it is necessary to implement all the functions for which stubs already exist in

`$AFR_HOME/lib/secure_sockets/portable/[vendor]/[board]/aws_secure_sockets.c`. `$AFR_HOME/lib/include/aws_secure_sockets.h` contains the information necessary to create the implementations.

Since the MQTT library is not used for running Wi-Fi tests, comment out the calls to `BUFFERPOOL_Init()` and `MQTT_AGENT_Init()` from `SYSTEM_Init()`, which is located in `$AFR_HOME/lib/utils/aws_system_init.c`. Make sure the call to `SOCKETS_init()` is uncommented.

Test Setup

1. The Secure Sockets tests require an echo server to be present on the network. For your convenience, the AFQP tests distribution contains a suitable echo server, written in Go, in the `$AFR_HOME/tests/common/utils` directory.

   To start the echo server:

   a) Ensure the latest version of Go is installed on the computer that will run the echo server (the server host). Go can be installed from https://golang.org/dl/
   b) Copy `$AFR_HOME/tests/common/utils/echo_server.go` onto the server host.
   c) Start the server by typing: "`go run echo_server.go`"
   d) In `$AFR_HOME/tests/common/tests/common/include/aws_test_tcp.h`, set the parameters shown in the table below to the IP address of the server host. The value 192.168.0.200 is an example only.

Echo Server IP address	Example value if address is 192.168.0.200
tcptestECHO_SERVER_ADDR0	192
tcptestECHO_SERVER_ADDR1	168
tcptestECHO_SERVER_ADDR2	0
tcptestECHO_SERVER_ADDR3	200

   To test that the echo server is working, open a command prompt on a computer on the same network and type "telnet 192.168.0.200 9001" from a Windows host, or "nc 192.168.0.200 9001" from a Linux host. Again, use the correct IP address for the server host – 192.168.0.200 is used as an example only.

   It may be necessary to adjust the firewall settings on the server host to enable the MCU to connect.

2. Set the tcptestTCP_ECHO_CLIENT_PORT macro in `$AFR_HOME/tests/common/tests/common/include/aws_test_tcp.h` to the port on which the echo server is listening. The provided echo server listens on port 9001. If you have problems connecting to port 9001 due to corporate network security policies, you can change the listening port in the echo server code to a port that is not restricted by your company's security policy.
3. Set the tcptestSECURE_SERVER macro to 0 in `$AFR_HOME/tests/common/secure_sockets_test_tcp.h` to run the socket tests without TLS.
4. Set the testrunnerFULL_TCP_ENABLED macro to 1 in `$AFR_HOME/tests/[vendor]/[board]/common/config_files/aws_test_runner.config.h` to enable the sockets tests.
5. Enable the testing task in `$AFR_HOME/tests/[vendor]/[board]/common/application_code/main.c` by deleting the `#if 0`

and `#endif` compiler directives, in `vApplicationIPNetworkEventHook ( void )`. This change is required for all the remaining libraries to be ported in this document.

Dependency on TLS

The test set for this library (Secure Sockets) includes some tests that require TLS which is described later in this document. The functionality verified by these tests is exercised when tcptestSECURE_SERVER macro is set to 1. You MUST come back to run this subset of the Secure Socket tests after TLS library porting is completed.

Qualification Test

1. Build and execute the test project.
2. View the test results in the UART console. If all the tests pass, then testing is complete. Save the test results by cutting and pasting them from the UART console into a text file, and move to the next section.

Example test results output:

```
Starting Tests...

TEST(Full_TCP, SOCKETS_CloseInvalidParams) PASS
```

…

```
TEST(Full_TCP, SECURE_SOCKETS_NonBlockingConnect) PASS

TEST(Full_TCP, SECURE_SOCKETS_TwoSecureConnections) PASS

TEST(Full_TCP, SECURE_SOCKETS_SetSecureOptionsAfterConnect) PASS



-----------------------

47 Tests 3 Failures 0 Ignored

FAIL

----All tests finished----
```


Appendix F: PKCS #11

Description

Amazon FreeRTOS uses the open standard PKCS #11 "CryptoKi" API as the abstraction layer for cryptographic operations, including:

- Encryption and decryption.
- Storage and enumeration of X.509 certificates.
- Storage and management of cryptographic keys.

See the open standard PKCS #11 Cryptographic Token Interface Base Specification:

http://docs.oasis-open.org/pkcs11/pkcs11-base/v2.40/os/pkcs11-base-v2.40-os.html

Storing private keys in general purpose flash memory can be convenient in evaluation and rapid prototyping scenarios. However, when it comes to production scenarios, we recommend the use of dedicated cryptographic hardware in order to reduce the threats of data theft and device duplication. Cryptographic hardware includes components with features that prevent cryptographic secret keys from being exported. In order to use such hardware with Amazon FreeRTOS, the PKCS #11 API must be ported to it.

Pre-requisites

1. A test project that was created in accordance with the instructions provided in the body of this document, and that is building vendor-supplied secure storage drivers.
2. An implementation of configPRINT_STRING() that was created and tested as described in Appendix A.
3. A validated FreeRTOS kernel configuration, as described in Appendix B: FreeRTOS kernel.

Preparing the IDE Project

In all steps below, add source files to the IDE project from their existing location on the disk (by reference) – do not create duplicate copies of source files on the disk:

1. Add `$AFR_HOME/lib/pkcs11/portable/[vendor]/[board]/aws_pkcs11_pal.c` to the `[project_top_level]/lib/aws/pkcs11` folder of the test project.
2. Add the PKCS #11 library header files from `$AFR_HOME/lib/third_party/pkcs11` to the `[project_top_level]/lib/third_party/pkcs11` folder of the test project._RB_
3. Add the PKCS #11 tests from `$AFR_HOME/tests/common/pkcs11/aws_test_pkcs11.c` to the `[project_top_level]/application_code/common_tests/pkcs11` folder folder of the test project.
4. Add the implementation of PKCS #11 for mbedTLS `$AFR_HOME/lib/pkcs11/mbedtls/aws_pkcs11_mbedtls.c` to the `[project_top_level]/lib/pkcs11` folder of the test project
5. Import the CRYPTO abstraction wrapper file for mbedTLS `$AFR_HOME/lib/crypto/aws_crypto.c` to the `[project_top_level]/lib/crypto` folder of the test project.
6. Add the mbedTLS library itself from `$AFR_HOME/lib/third_party/mbedtls/library` into the `[project_top_level]/lib/third_party/mbedtls/source` folder of the test project.

7. Add the mbedTLS library header file from `$AFR_HOME/lib/third_party/mbedtls/include` into the `[project_top_level]/lib/third_party/mbedtls/include` folder of the test project.
8. Add both `$AFR_HOME/lib/third_party/mbedtls/include` and `$AFR_HOME/lib/third_party/pkcs11` to the compiler's include path.

Porting

1. Porting the PKCS #11 API functions

   The PKCS #11 API is dependent on the implementation of cryptographic primitives such as SHA256 hashing and ECDSA signing. The Amazon FreeRTOS implementation of PKCS #11 uses the cryptographic primitives implemented in the mbedTLS library, for which a port is already provided. Modifying the existing PKCS #11 port is required if you wish to use a different software implementation of the cryptographic primitives (i.e., other than mbedTLS), or if your target hardware offloads crypto to a separate module.

2. Porting the PKCS #11 Platform Abstraction Layer (PAL) for device specific certificate and key storage

   If you decide to use the Amazon FreeRTOS implementation of PKCS #11, there is a relatively small amount of customization required in order to read and write cryptographic objects to storage (for example, onboard flash memory).

   `$AFR_HOME/lib/pkcs11/portable/[vendor][board]/aws_pkcs11_pal.c` contains stubs for the PAL functions, of which you must provide ports for at least the functions listed in the table below:

Function	Description
PKCS11_PAL_SaveFile	Write information to local storage
PKCS11_PAL_ReadFile	Read information from local storage
PKCS11_ReleaseFileDate	Cleanup after PKCS #11_PAL_ReadFile. Cleanup buffer used.

3. Implement `mbedtls_hardware_poll()`

   ⚠ You only need to port this function if you plan to use Amazon FreeRTOS' PKCS#11 implementation and the mbedTLS library for underlying cryptographic and TLS support.

   TCP/IP and TLS require cryptographic pseudo-random number generation (PRNG) for sequence number and key generation, respectively. A hardware entropy source is important for seeding the PRNG. For the mbedTLS library to work, you MUST implement mbedtls_hardware_poll() which allows the mbedTLS library to seed its PRNG using your board's entropy source. This function is located in `$AFR_HOME/lib/pkcs11/portable/[vendor][board]/aws_pkcs11_pal.c`

   For more information see, [https://docs.mbed.com/docs/mbed-os-handbook/en/5.2/advanced/tls_porting.](https://docs.mbed.com/docs/mbed-os-handbook/en/5.2/advanced/tls_porting)

Since the MQTT library is not used for running PKCS11 tests, comment out the lines that call `BUFFERPOOL_Init(), MQTT_AGENT_Init()` in function `SYSTEM_Init()` in file `$AFR_HOME/lib/utils/aws_system_init.c`. Bufferpool and mqtt_agent are used in MQTT library. If you have not ported the SOCKETS library (Appendix E), comment out the line that calls `SOCKETS_Init()` in function `SYSTEM_Init()`, in file `$AFR_HOME/lib/utils/aws_system_init.c`.

Test Setup

1. Enable the PKCS 11 test by setting the `testrunnerFULL_PKCS11_ENABLED` macro to 1 in `$AFR_HOME/tests/[vendor]/[board]/common/config_files/aws_test_runner_config.h`.

Qualification Test

Build and execute the project. The UART output indicates how many tests have run and completed successfully. Copy the results from the terminal and save it to a text file.

Example of the test results output:

```
Starting Tests...

TEST(Full_PKCS11, Digest) PASS

TEST(Full_PKCS11, Digest_ErrorConditions) PASS

TEST(Full_PKCS11, GetFunctionListInvalidParams) PASS

TEST(Full_PKCS11, InitializeFinalizeInvalidParams) PASS
```

…

```
TEST(Full_PKCS11, SignVerifyCryptoApiInteropRSA) PASS

TEST(Full_PKCS11, SignVerifyRoundTrip_MultitaskLoop)d:\treadstonetest_custom\treadstone\tests\common\pkcs11
\aws_test_pkcs11.c:2728::FAIL: This test leaks!

TEST(Full_PKCS11, KeyGenerationEcdsaHappyPath) PASS



-----------------------

33 Tests 1 Failures 0 Ignored

FAIL

----All tests finished----
```


Appendix G: TLS

Description

The AWS IoT Core MQTT broker only accepts mutually authenticated TLS connections. Amazon FreeRTOS can use either mbedTLS, in which case no porting is necessary, or an off-chip TLS implementation, such as those found on some network co-processors. To allow both options the TLS library is not accessed directly, but through a TLS abstraction layer.

In all cases, the TLS tests must be executed and pass. Preparing the tests requires IoT device configuration in the AWS cloud and certificate and key provisioning on the target hardware.

Pre-requisites

1. A port of the Secure Sockets library, as described in Appendix E: Secure Sockets.
2. A port of the PKCS #11 library, as described in Appendix F: PKCS #11.
3. An AWS account.

Preparing the IDE Project

1. Add the TLS abstraction implementation `$AFR_HOME/lib/tls/aws_tls.c` or `$AFR_HOME/lib/tls/portable/[vendor]/[board]/aws_tls.c` (if your target hardware offloads TLS to a separate processor) to the `[project_top_leve]/lib/aws/tls` folder of the test project.
2. Add the TLS tests file `$AFR_HOME/tests/common/tls/aws_test_tls.c` to the `[project_top_level]/application_code/common_tests/tls` folder of the test project.
3. Enable the tests by setting the testrunnerFULL_TLS_ENABLED macro to 1 in `$AFR_HOME/tests/[vendor]/[board]/common/config_files/aws_test_runner_config.h`.

Porting

If your target hardware offloads TLS functionality to a separate network chip then it is necessary to implement all the TLS abstraction layer functions in the table below. `$AFR_HOME/lib/include/aws_tls.h` contains the information necessary to create the implementations. Save the created file as `$AFR_HOME/lib/tls/portable/[vendor]/[board]/aws_tls.c`

Function	Description
TLS_Init	Initialize the TLS context
TLS_Connect	Negotiate TLS and connect to the server
TLS_Recv	Read the requested number of bytes from the secure connection
TLS_Send	Write the requested number of bytes to the secure connection
TLS_Cleanup	Free resources consumed by the TLS context

⚠ Since the MQTT library is not used for running Wi-Fi tests, comment out the calls to `BUFFERPOOL_Init() and MQTT_AGENT_Init()` from `SYSTEM_Init(), which is located` in `$AFR_HOME/lib/utils/aws_system_init.c.` Make sure the call to `SOCKETS_init()` is uncommented.

Test Setup

The tests in this section require use of the online AWS console, where your target hardware will be represented as a 'thing', and communicate with AWS via a custom MQTT endpoint that is tied to your AWS account.

The steps below create the certificates and keys necessary to complete qualification tests.

⚠ The tests require the created certificates and keys to be built into the target hardware's executable image. That is convenient in this test scenario, but is **not recommended for production scenarios**, where the keys should be stored securely.

Certificate Formatting Tool: It is necessary to convert the certificates and keys to C strings before building them into the executable image. The AFQP tests include a tool for that purpose. To convert certificate and key pairs into C strings:

a. Open `$AFR_HOME/tests/common/utils/CertificateConfigurationTool/PEMfileToCString.html` in a web browser.
b. Follow the instructions on the opened web page to load the certificate and private key.
c. Once loaded, follow the instruction in the opened web page to convert the opened certificate and private key to a formatted C string.

1. Set the `clientcredentialMQTT_BROKER_ENDPOINT`[] variable in `$AFR_HOME/tests/common/include/aws_clientcredential.h` to the custom end point of your AWS account – this is the URL the TLS tests connect to.

   To find your custom end point, use the URL https://aws.amazon.com/iot/ to log into your AWS account, then click the "Settings" link in the bottom left corner of the screen to open the settings window – the customer end point is displayed at the top of the settings window.

2. Noting the information below about the information you need to record during the process, follow the steps in the AWS IoT Getting Started tutorial to create the resources in AWS IoT that will represent your target hardware (Thing, Certificate and Policies).
   a. Start here: https://docs.aws.amazon.com/iot/latest/developerguide/iot-console-signin.html, and continue through each of the steps of the tutorial until you complete "Attach your Certificate to a Thing". See notes below for additional guidance about these steps.
   b. During this process,
      - Set the `clientcredentialIOT_THING_NAME` variable in `$AFR_HOME/tests/common/include/aws_clientcredential.h` to the name you assigned your 'thing' (the thing name).
      - The steps on the link above include the creation of a certificate. Download and save all three files that are generated during that process.

- The steps on the link above include creating a policy. Use the following policy to attach to the certificate:

```
{
    "Effect": "Allow",

    "Action": "iot:*",

    "Resource": "*"

}
```

This policy will allow all IoT actions on all resources. That is convenient in a test and evaluation scenario but is not recommended for production scenarios.

3. Prepare certificate/key pairs for various tests for the TLS library.

AWS IoT can use AWS IoT-generated certificates or certificates signed by a CA certificate for device authentication. In order to run the various tests, you will need to create multiple credentials as listed below:

a. Generate a certificate by AWS IoT. (RSA Certificate)
b. Generate a certificate from a CSR (Certificate Signing Request). (ECDSA cert, Malformed cert)
c. Generate a certificate from a registered CA. (Untrusted, BYOC certificate)

The credentials for the types listed above are described in detail in following table. All of the testing client certificates and private keys listed above must be stored in:

```
$AFR_HOME/tests/common/aws_clientcredential_keys.h

$AFR_HOME/tests/common/aws_test_tls.h
```

Cert/Key Variables	Description
clientcredentialCLIENT_CERTIFICATE_PEM	Device certificate used for AWS cloud communication. It is also used in TLS_ConnectRSA() test. You can set this up using AWS Console.
clientcredentialCLIENT_PRIVATE_KEY_PEM	Device private key used for AWS cloud communication. It is also used in TLS_ConnectRSA() test. You can set this up using AWS Console.
tlstestCLIENT_CERTIFICATE_PEM_EC	Certificate for P-256 elliptic curve key. It is used in TLS_ConnectEC() test.
tlstestCLIENT_PRIVATE_KEY_PEM_EC	A p-256 elliptic curve key. It is used in TLS_ConnectEC() test.

tlstestCLIENT_CERTIFICATE_PEM_MALFORMED	A RSA or ECDSA certificate that has a field modified. Used in TLS_ConnectMalformedCert() test.
tlstestCLIENT_UNTRUSTED_CERTIFICATE_PEM	A certificate is not trusted (not registered) by AWS IoT. Used in TLS_ConnectUntrustedCert().
tlstestCLIENT_UNTRUSTED_PRIVATE_KEY_PEM	The private key correspond to the untrusted certificate.
tlstestCLIENT_BYOC_CERTIFICATE_PEM	A certificate created by a CA (registered to AWS IoT). Used in TLS_ConnectBYOCCredentials().
tlstestCLIENT_BYOC_PRIVATE_KEY_PEM	The private key corresponding to the BYOC certificate.

Setup for RSA certificate/private used in TLS_ConnectRSA() :

This pair of certificate/key is generated in Test Setup Step 2. The three files you downloaded during that "Thing" creation process will be used here.

Format the certificate and the private key with the formatting tool and copy and assign them to variables

clientcredentialCLIENT_CERTIFICATE_PEM

clientcredentialCLIENT_PRIVATE_KEY_PEM

in file `$AFR_HOME/tests/common/aws_clientcredential_keys.h`.

Setup for ECDSA certificate/private key used in TLS_ConnectEC():

OpenSSL is an open source toolkit for TLS protocol. (https://www.openssl.org/). We will use openssl in examples of generating certificates below. Please use TLS V1.2. You can download it here: https://www.openssl.org/source/

1) Create a CSR with openssl:
   a) `openssl ecparam -name prime256v1 -genkey -noout -out p256_privatekey.pem`
   b) `openssl req –new –key p256_privatekey.pem –out csr.csr`
2) Create a certificate with the AWS IoT console:
   a) On the AWS IoT / Security / Certificate page, click "Create" in upper right-hand corner
   b) Click "Create with CSR" and upload the .csr file created in step 1.
   c) Download the cert .pem file, activate it and attach the same policy you used when setting up the RSA certificate.
   d) Attach the certificate to the IoT thing created when you set up the "Thing".
3) Format the certificate and the private key using the formatting tool.
4) Copy the created cert and private key to the following variables in `$AFR_HOME/tests/common/aws_test_tls.h`:
   a) `tlstestCLIENT_CERTIFICATE_PEM_EC`
   b) `tlstestCLIENT_PRIVATE_KEY_PEM_EC`

5) In `$AFR_HOME/tests/common/tls/aws_test_tls.c`, set the `tlstestMQTT_BROKER_ENDPOINT_EC` variable to the same AWS IoT message broker endpoint address in TLS Test Setup Step 1.

Setup for Malformed certificate used in TLS_ConnectMalformedCert():

The purpose of the test is to be able to use a malformed certificate to authenticate with the server. Random modification of a certificate will most likely be rejected by x509 certificate verification before the connection request is sent out. We have a suggestion to setup this malformed certificate: modifying the issuer of the certificate.

See Appendix P: Modify issuer in a certificate for process details.

Setup for BYOC (Bring You Own Certificate) certificate used in TLS_ConnectBYOCCredentials():

1) Create your own certificate with a valid rootCA/CA chain. See example in Appendix N: Instructions to Create a BYOC (ECDSA).
2) Register CAs and your own certificate in the AWS IoT console: IoT Core / Secure / Certificates / Create / Get started.
3) Format the certificate and the private key using the formatting tool.
4) Copy the certificate and private key strings to the following variables in `$AFR_HOME/tests/common/aws_test_tls.h`:
   a) `tlstestCLIENT_BYOC_CERTIFICATE_PEM`
   b) `tlstestCLIENT_BYOC_PRIVATE_KEY_PEM`

Setup for Untrusted certificate used in TLS_ConnectUntrustedCert():

1) Create your own certificate with valid rootCA/CA chain. See example in Appendix N: Instructions to Create a BYOC (ECDSA).
2) Do not register them in AWS IoT console.
3) Format the certificate and the private key using the formatting tool.
4) Copy the cert and private key strings to the following variables in `$AFR_HOME/tests/common/aws_test_tls.h`:
   a) `tlstestCLIENT_UNTRUSTED_CERTIFICATE_PEM`
   b) `tlstestCLIENT_UNTRUSTED_PRIVATE_KEY_PEM`

Qualification Test

1. Build and execute the test project.
2. View the test results in the UART console. If all the tests pass, then testing is complete. Save the test results by cutting and pasting them from the UART console into a text file, and move to the next section.

Example of the test results output:


```
Starting Tests...

TEST(Full_TLS, AFQP_TLS_ConnectEC) PASS

TEST(Full_TLS, TLS_ConnectRSA) PASS

TEST(Full_TLS, TLS_ConnectMalformedCert) PASS

TEST(Full_TLS, TLS_ConnectUntrustedCert) PASS

TEST(Full_TLS, AFQP_TLS_ConnectBYOCCredentials) PASS

-----------------------
5 Tests 0 Failures 0 Ignored
OK
-------ALL TESTS FINISHED-------
```

Once TLS porting and verification is completed, note that you must go back to run a subset of the Secure Socket tests which depend on this functionality i.e. when tcptestSECURE_SERVER macro is set to 1. See Dependency on TLS in the Secure Sockets porting section.

Appendix H: MQTT

Description

Communication between IoT devices and AWS IoT Core (the MQTT broker) uses the MQTT protocol. The MQTT library that implements the protocol does not need to be ported, but does need to pass all the MQTT tests.

The MQTT library has a dependency on the Buffer Pool library, which is used to allocate the memory necessary to hold MQTT packets.

Pre-requisites

1. A port of the Secure Sockets library, as described in Appendix E: Secure Sockets.
2. A port of the PKCS #11 library, as described in Appendix F: PKCS #11.
3. A port of the TLS library, as described in Appendix G: TLS.
4. An AWS account.
5. An IoT thing created in AWS cloud and its associated credential information. (Refer to TLS_Test_Setup_Step_1, TLS_Test_Setup_Step_2 and RSA_Certificate_Setup)

Preparing the IDE Project

In all steps below, add source files to the IDE project from their existing location on the disk (by reference) – do not create duplicate copies of source files on the disk:

1. Add the MQTT library source files from `$AFR_HOME/lib/mqtt` into the `[project_top_level]/lib/aws/mqtt` folder of the test project.
2. Add the Bufferpool source files from `$AFR_HOME/lib/bufferpool` into the `[project_top_level]/lib/bufferpool` folder of the test project.
3. Add the MQTT test source files from `$AFR_HOME/tests/common/mqtt/` to `[project_top_level]/application_code/common_tests/mqtt` folder of the test project.
4. Uncomment all the initialization functions called from `SYSTEM_Init() within $AFR_HOME/lib/utils/aws_system_init.c`.

Porting

⚠ In order to enable MQTT functionality, uncomment the calls to `BUFFERPOOL_Init()` and `MQTT_AGENT_Init()` from `SYSTEM_Init()`, which is located in `$AFR_HOME/lib/utils/aws_system_init.c`. Make sure the call to `SOCKETS_init()` is also still uncommented.

There is no additional porting required for this library.

Test Setup

These tests require the certificates and keys that were created prior to testing the TLS library.

1. Enable the MQTT tests by setting testrunnerFULL_MQTT_ENABLED to 1 in `$AFR_HOME/tests/[vendor]/[board]/common/config_files/aws_test_runner_config.h`

Qualification Test

1. Build and execute the test project.
2. View the test results in the UART console. If all the tests pass, then testing is complete. Save the test results by cutting and pasting them from the UART console into a text file.

Example of the test results output:

```
Starting Tests...
TEST(Full_MQTT, prvGetTopicFilterType_HappyCases) PASS
TEST(Full_MQTT, prvGetTopicFilterType_ErrorCases) PASS
TEST(Full_MQTT, prvDoesTopicMatchTopicFilter_MatchCases) PASS
TEST(Full_MQTT, prvDoesTopicMatchTopicFilter_NotMatchCases) PASS
TEST(Full_MQTT, MQTT_Init_HappyCase) PASS
TEST(Full_MQTT, MQTT_Init_NULLParams) PASS
TEST(Full_MQTT, MQTT_Connect_HappyCase) PASS
TEST(Full_MQTT, MQTT_Connect_BrokerRejectsConnection) PASS
TEST(Full_MQTT, MQTT_Connect_ConnACKWithoutConnect) PASS
TEST(Full_MQTT, MQTT_Connect_ReservedReturnCodeFromBroker) PASS
TEST(Full_MQTT, MQTT_Connect_ShorterConnACK) PASS
TEST(Full_MQTT, MQTT_Connect_LongerConnACK) PASS
TEST(Full_MQTT, MQTT_Connect_NULLParams) PASS
TEST(Full_MQTT, MQTT_Connect_SecondConnectWhileAlreadyConnected) PASS
TEST(Full_MQTT, MQTT_Connect_SecondConnectWhileWaitingForConnACK) PASS
TEST(Full_MQTT, MQTT_Connect_NetworkSendFailed) PASS

-----------------------
16 Tests 0 Failures 0 Ignored
OK
----All tests finished----
```


Appendix I: Test List

The tests listed here are the current tests we provide. It is subject to change.

Library	Tests	Notes
Wi-Fi	WiFiOnOff	
	WiFiMode	
	WiFiConnectionLoop	
	WiFiIsConnected	
	WiFiNetworkAddGetDelete	
	WiFiPowerManagementMode	
	WiFiGetIP	
	WiFiGetMAC	
	WiFiGetHostIP	
	WiFiScan	
	WiFiReset	
	WiFiPing	
	WiFiConnectMultipleAP	
	WiFiSeperateTasksConnectingAndDisconnectingAtOnce	
	WiFiOnOffLoop	
	WIFI_GetMode_NullParameters	
	WIFI_GetIP_NullParameters	
	WIFI_GetMAC_NullParameters	
	WIFI_GetHostIP_NullParameters	
	WIFI_Scan_NullParameters	
	WIFI_NetworkAdd_NullParameters	
	WIFI_NetworkGet_NullParameters	
	WIFI_SetPMMode_NullParameters	
	WIFI_GetPMMode_NullParameters	
	WIFI_Ping_NullParameters	
	WIFI_ConnectAP_NullParameters	
	WIFI_SetMode_InvalidMode	
	WIFI_GetHostIP_InvalidDomainName	
	WIFI_GetHostIP_DomainNameLengthExceeded	
	WIFI_NetworkDelete_DeleteNonExistingNetwork	
	WIFI_NetworkGetNonExistingNetwork	
	WIFI_SetPMMode_InvalidPMMode	
	WIFI_Ping_ZeroParameters	
	WIFI_ConnectAP_InvalidSSID	
	WIFI_ConnectAP_InvalidPassword	
	WIFI_ConnectAP_InvalidSecurityTypes	
	WIFI_ConnectAP_MaxSSIDLengthExceeded	
	WIFI_ConnectAP_MaxPasswordLengthExceeded	
	WIFI_ConnectAP_ZeroLengthSSID	
	WIFI_ConnectAP_ZeroLengthPassword	
	WIFI_ConnectAP_PasswordLengthLess	

Library	Tests	Notes
	WIFI_Scan_ZeroScanNumber	
	WIFI_NetworkGet_GetManyNetworks	
	WIFI_NetworkAdd_AddManyNetworks	
	WIFI_NetworkDelete_DeleteManyNetworks	
	WIFI_ConnectAP_ConnectAllChannels	
Secure Sockets	SOCKETS_Threadsafe_SameSocketDifferentTasks	
	SOCKETS_Threadsafe_DifferentSocketsDifferentTasks	
	SOCKETS_Connect_InvalidAddressLength	
	SOCKETS_Connect_InvalidParams	
	SOCKETS_Socket_TCP	
	SOCKETS_SetSockOpt_RCVTIMEO	
	SOCKETS_SetSockOpt_InvalidParams	
	SOCKETS_Shutdown	
	SOCKETS_ShutdownInvalidParams	
	SOCKETS_ShutdownWithoutReceiving	
	SOCKETS_Close	
	SOCKETS_CloseInvalidParams	
	SOCKETS_CloseWithoutReceiving	
	SOCKETS_Recv_ByteByByte	
	SOCKETS_Recv_On_Unconnected_socket	
	SOCKETS_SendRecv_VaryLength	
	SOCKETS_Socket_InvalidTooManySockets	
	SOCKETS_Socket_InvalidInputParams	
	SOCKETS_Send_Invalid	
	SOCKETS_Recv_Invalid	
	SOCKETS_htos_HappyCase	
	SOCKETS_inet_addr_quick_HappyCase	
	SOCKETS_NonBlocking_Test	
	SECURE_SOCKETS_Threadsafe_DifferentSocketsDifferentTasks	
	SECURE_SOCKETS_Threadsafe_SameSocketDifferentTasks	
	SECURE_SOCKETS_Connect_InvalidAddressLength	
	SECURE_SOCKETS_Connect_InvalidParams	
	SECURE_SOCKETS_NonBlockingConnect	
	SECURE_SOCKETS_NonBlocking_Test	
	SECURE_SOCKETS_SetSockOpt_SERVER_NAME_INDICATION	
	SECURE_SOCKETS_SetSockOpt_TRUSTED_SERVER_CERTIFICATE	
	SECURE_SOCKETS_SetSockOpt_RCVTIMEO	
	SECURE_SOCKETS_SetSockOpt_InvalidParams	
	SECURE_SOCKETS_Shutdown	
	SECURE_SOCKETS_ShutdownInvalidParams	
	SECURE_SOCKETS_ShutdownWithoutReceiving	
	SECURE_SOCKETS_Close	
	SECURE_SOCKETS_CloseInvalidParams	

Library	Tests	Notes
	SECURE_SOCKETS_CloseWithoutReceiving	
	SECURE_SOCKETS_Recv_ByteByByte	
	SECURE_SOCKETS_Recv_On_Unconnected_socket	
	SECURE_SOCKETS_SendRecv_VaryLength	
	SECURE_SOCKETS_SockEventHandler	
	SECURE_SOCKETS_Send_Invalid	
	SECURE_SOCKETS_SetSecureOptionsAfterConnect	
	SECURE_SOCKETS_TwoSecureConnections	
	SECURE_SOCKETS_Recv_Invalid	
TLS	TLS_ConnectEC	
	TLS_ConnectRSA	
	TLS_ConnectMalformedCert	
	TLS_ConnectUntrustedCert	
	TLS_ConnectBYOCCredentials	
PKCS #11	GetFunctionListInvalidParams	
	InitializeFinalizeInvalidParams	
	GetSlotListInvalidParams	
	OpenCloseSessionInvalidParams	
	C_PKCSHappyTestVerify	
	C_PKCSC_VerifyInvalidParams	
	C_PKCSC_VerifyInitInvalidParams	
	C_PKCSHappyTestSign	
	C_PKCSSignInvalidParams	
	C_PKCSSignInitInvalidParams	
	C_PKCSHappyTestObject	
	C_PKCSCreateObjectInvalidParameters	
	C_PKCSFindObjectsFinalInvalidParams	
	C_PKCSFindObjectsInvalidParams	
	C_PKCSFindObjectsInitInvalidParams	
	C_PKCSGetAttributeValueInvalidParams	
	C_PKCSGenerateRandomInvalidParameters	
	C_PKCSGenerateRandomTestHappyTest	
	SignVerifyCryptoApiInteropRSA	
	SignVerifyRoundTripRSANoPubKey	
	SignVerifyRoundTripWithCorrectRSAPublicKey	
	SignVerifyRoundTripWithWrongRSAPublickKey	
	SignVerifyRoundTripECNoPubKey	
	SignVerifyRoundTripWithCorrectECPublicKey	
	SignVerifyRoundTripWithWrongECPublicKey	
	TestRSAParse	
	TestECDSAParse	
	TestRSAExport	
	TestECDSAExport	
	SignVerifyRoundTrip_MultitaskLoop	

Library	Tests	Notes
	GetFunctionListInvalidParams	
	InitializeFinalizeInvalidParams	
	GetSlotListInvalidParams	
	OpenCloseSessionInvalidParams	
MQTT	MQTT_Init_HappyCase	
	MQTT_Init_NULLParams	
	MQTT_Connect_HappyCase	
	MQTT_Connect_BrokerRejectsConnection	
	MQTT_Connect_ConnACKWithoutConnect	
	MQTT_Connect_ReservedReturnCodeFromBroker	
	MQTT_Connect_ShorterConnACK	
	MQTT_Connect_LongerConnACK	
	MQTT_Connect_NULLParams	
	MQTT_Connect_SecondConnectWhileAlreadyConnected	
	MQTT_Connect_SecondConnectWhileWaitingForConnACK	
	MQTT_Connect_NetworkSendFailed	
	prvGetTopicFilterType_HappyCases	
	prvGetTopicFilterType_ErrorCases	
	prvDoesTopicMatchTopicFilter_MatchCases	
	prvDoesTopicMatchTopicFilter_NotMatchCases	
Total	148	

Appendix J: TLS Server Setup

A simple TLS echo server is provided with Amazon FreeRTOS code. It is located in `$AFR_HOME/tests/common/utils/tls_echo_server.go`.

Instructions:

1. Install the latest version of GO on your server host: https://golang.org/dl/
2. Install openssl on your server host:
    a. Linux --- https://www.openssl.org/source/
    b. Windows --- https://slproweb.com/products/Win32OpenSSL.html
3. Copy `tls_echo_server.go` to a directory you choose.
4. Generate a TLS server self-signed certificate and private key. See `$AFR_HOME/tests/common/utils/readme-gencert.txt` for the openssl commands to generate a self-signed server certificate and private key and a client certificate and private key.
5. Generate the client certificate and private key.
6. Copy the certificate and private key .pem files into a subdirectory called “certs”. The “certs” directory should be a subdirectory of the directory where the server code will run.
7. Start the TLS server by running: `go run tls_echo_server.go`
8. The server will listen on port 9000. The IP address and the port must be set in `$AFR_HOME/tests/common/tests/common/include/aws_test_tcp.h`. For example if your server’s IP address is 192.168.2.6, set the following macros:

Macro definition for TLS server	Example value if address is 192.168.0.200
`tcptestECHO_SERVER_TLS_ADDR0`	192
`tcptestECHO_SERVER_TLS_ADDR1`	168
`tcptestECHO_SERVER_TLS_ADDR2`	2
`tcptestECHO_SERVER_TLS_ADDR3`	6
`tcptestECHO_PORT_TLS`	`( 9000 )`

9. The tests will check the server certificate. In `$AFR_HOME/tests/common/tests/common/include/aws_test_tcp.h, set tcptestECHO_HOST_ROOT_CA` to your formatted server certificate.
   You can use the formatting tool to format your server certificate.

Appendix K: "Hello World" Demo Project Set Up

Amazon FreeRTOS Directory Structure

All qualified Amazon FreeRTOS ports use the same directory structure, so all new files, including IDE project files, must be created in the correct folder locations. The directory structure is explained below.

The three root level folders under `$AFR_HOME` are:

```
$AFR_HOME
├───demos    Contains projects that build demo applications
├───lib      Contains Amazon FreeRTOS and third-party libraries
└───tests    Contains projects that build qualification tests
```

Your project is to be created within the demos folder, which is structured as follows:

```
$AFR_HOME
└───demos
    ├───common     Contains files built by all demo projects
    ├───pc         Contains a demo project for the FreeRTOS Windows port
    └───vendor     Contains your vendor specific code
        └───board  Contains your board specific code
```

The `$AFR_HOME/demos/[vendor]/[board]` folder is a template provided to simplify the creation of a new test project and ensures all test projects have a consistent organization. It has the following structure:

```
$AFR_HOME
└───demos
    └───[vendor]
        └───[board]
            ├───common
            │   ├───application_code   Contains main.c
            │   │   └───vendor_code    Contains vendor supplied board specific files
            │   └───config_files       Contains Amazon FreeRTOS config files
            └───ide                    Contains an IDE specific project
```

Your demo projects always require vendor-supplied driver libraries. Some vendor-supplied libraries, such as a header file that maps a GPIO output to an LED, are specific to the target development board. Other vendor-supplied libraries, such as the GPIO library itself, are specific to the target MCU family. Do not save vendor-supplied libraries that are specific to the MCU anywhere within either the `$AFR_HOME/tests or $AFR_HOME/demos` folders.

Preparing Your Project Directories:

1. Rename the `$AFR_HOME/demos/vendor` folder to the name of the company that manufactures the MCU – from here on the folder is referred to as `[vendor]`.

2. Rename the `$AFR_HOME/demos/[vendor]/board` folder to the name of the development board being qualified – from here on the folder is referred to as `[board]`.
3. Copy your `main.c` and `main.h` in `$AFR_HOME/demos/[vendor]/[board]/common/application_code` folder. You can re-use the `main.c` in your `aws_tests` project.
4. Save any required vendor-supplied libraries that are specific to the board in the `$AFR_HOME/demos/[vendor]/[board]/common/application_code/vendor_code` folder.
5. Rename the `$AFR_HOME/demos/[vendor]/[board]/ide` folder to the name of the IDE that will be used to build the test project – from here on the folder is referenced as `[ide]`.

Create the "Hello World" Demo Project

If your IDE does not use relative paths, define a variable in the project for relative folder locations before importing Amazon FreeRTOS source files.

1. Create an IDE project `aws_demos` in the `$AFR_HOME/demos/[vendor]/[board]/[ide]` directory.
2. Create the project structure in the IDE
3. Create three top level virtual folders:
    a. `application_code`
    b. `config_files`
    c. `lib`
4. Import the `$AFR_HOME/demos/[vendor]/[board]/common/application_code` directory and its contents into the `application_code` virtual folder.
5. Import the files in `$AFR_HOME/demos/[vendor]/[board]/common/config_files` into the `config_files` virtual folder.
6. Create a virtual folder under `application_code` and call it `common_demos`.
7. Create a source folder under `common_demos`.
8. Import the files in each of the following directories into the source folder:
    a. `$AFR_HOME/demos/common/demo_runner`
    b. `$AFR_HOME/demos/common/devmode_key_provisioning` (only the .c file)
    c. `$AFR_HOME/demos/common/mqtt`
    d. `$AFR_HOME/demos/common/logging`
9. Import the following directories and its contents into `common_demos` folder.
    a. `$AFR_HOME/demos/common/include`
10. Create two virtual folders `aws` and `third_party` under virtual folder `lib`
11. Import each of the following directories and their contents into the `aws` folder:
    a. `$AFR_HOME/lib/bufferpool`
    b. `$AFR_HOME/lib/FreeRTOS`
    c. `$AFR_HOME/lib/FreeRTOS/portable/MemMang/heap_4.c`
    d. `$AFR_HOME/lib/FreeRTOS/portable/[compiler your IDE uses]`
    e. `$AFR_HOME/lib/FreeRTOS-Plus-TCP` (if you have ported this library, please refer to Appendix D on what files should be included in this project)
    f. `$AFR_HOME/lib/include`
    g. `$AFR_HOME/lib/include/private` (only .h files)
    h. `$AFR_HOME/lib/mqtt`
    i. `$AFR_HOME/lib/pkcs11/portable/[vendor]/[board]/pkcs11.c` (under pkcs11 folder)

`$AFR_HOME/lib/secure_sockets/portable/[vendor]/[board]/aws_secure_sockets.c` (under secure_sockets folder)
  j. `$AFR_HOME/lib/tls` (if you have ported TLS library)
  k. `$AFR_HOME/lib/wifi/portable/[vendor]/[board]/aws_wifi.c` (under `wifi` folder, if you have ported WI-FI library)

12. Import each of the following directories and their contents into third_party:
    a. `$AFR_HOME/lib/third_party/mcu_vendor/[vendor]/[board]/[driver_library]/[driver_library_version]` (under mcu_vendor folder)
    b. `$AFR_HOME/lib/third_party/mbedtls` (rename `../mbedtls/library` to `../mbedtls/source`)
    c. `$AFR_HOME/lib/third_party/pkcs11`
13. Make sure the following compiler include paths are set in the project property:
    a. `$AFR_HOME/demos/common/include`
    b. `$AFR_HOME/lib/include`
    c. `$AFR_HOME/lib/include/private`
    d. `$AFR_HOME/lib/FreeRTOS/portable/[compiler]/[architecture]`
    e. `$AFR_HOME/demos/vendor/board/common/config_files`
    f. `$AFR_HOME/lib/third_party/mbedtls/include`
    g. Any paths required by vendor-supplied driver libraries

Appendix L: Checklist for Qualification

- ☐ Register with Amazon Partner Network (APN).
- ☐ Agree on the Shared Work Plan with AWS.
- ☐ "Test" project passed all tests in AFQP tests version _______________.
  - ☐ Ported configPRINT_STRING() macro.
  - ☐ Configured FreeRTOS kernel according to your target MCU.
  - ☐ Ported Wi-Fi Management library (Optional if your board does not support Wi-Fi) and passed Wi-Fi Management library tests.
  - ☐ Ported FreeRTOS TCP/IP stack (Optional if you use off-chip TCP/IP stack).
  - ☐ Ported CYPTO library and passed CYPTO library tests.
  - ☐ Ported PKCS #11 library and passed tests for this library.
  - ☐ Ported TLS library (Optional if you use Amazon FreeRTOS TLS support) and passed tests for this library.
  - ☐ Ported Secure Sockets library and passed secure sockets tests.
  - ☐ Passed tests for MQTT library.
- ☐ Prepare a "Demo" project for an IDE you choose that can send "Hello World" to AWS IoT Console and receive reply through MQTT protocol.
- ☐ Put the appropriate open source license text in your code. Please refer to https://opensource.org/licenses for license text information.
- ☐ Configure your board name in
  `$AFR_HOME/demos/[vendor]/[board]/common/config_files/FreeRTOSConfig.h`
  `#define mqttconfigMETRIC_PLATFORM    "Platform=Your board name"`
- ☐ Information required for Appendix S: Hardware Information filled
- ☐ Prepare a "Getting Started Guide" for your board to help users run your "Demo" project. You can use the Getting Started Guide template to start and look at the guide for the Window Simulator for reference.

Appendix M: Troubleshooting Porting Setup

1. **Can I reach the "echo server" from two different networks (for example, from two subnets across 2 different access points)?**
   An echo server is required for successful completion of the TCP/IP and TLS tests. The echo server must be reachable from the network that the boards are connected to. Please consult your IT support to enable routing across subnets if you need devices on different subnets to communicate to a single echo server.

2. **Can I use openssl in a Windows environment?**
   Yes. Even though only a Linux distribution of openssl is provided on https://www.openssl.org/, you can find openssl distributions for Windows on the internet.

Appendix N: Instructions to Create a BYOC (ECDSA)

Prerequisite:

To follow the instructions below, you need to have **openssl** and the **AWS CLI** installed.

- **OpenSSL** is an open source toolkit for the TLS protocol. (https://www.openssl.org/). We will use openssl in examples for generating certificates below. Please use TLS V1.2. You can download it here:
  Linux --- https://www.openssl.org/source/
- **AWS CLI** installation guide: https://docs.aws.amazon.com/cli/latest/userguide/installing.html
  - MUST DO: Configure AWS CLI before use.
    Please follow the instruction here to configure AWS CLI: https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-getting-started.html

Note: during the CA certificate creation process, please consider fill in valid information. You may see errors if the organization or other fields don't align in later signing steps.

Generate a Root CA

1. Generate a root CA private key
   a. `openssl ecparam -name prime256v1 -genkey -noout -out rootCA.key`
2. Generate a root CA certificate
   a. `openssl req -x509 -new -nodes -key rootCA.key -sha256 -days 1024 -out rootCA.crt`

Generate Intermediate CA

1. Create necessary files
   a. `touch index.txt`
   b. `echo 1000 > serial`
2. Paste the ca.config file in Appendix O: Source for ca.config in the directory
3. Generate intermediate CA's private key:
   a. `openssl ecparam -name prime256v1 -genkey -noout -out intermediateCA.key`
4. Generate intermediate CA's CSR [Make sure to fill Common Name to some value]
   a. `openssl req -new -sha256 -key intermediateCA.key -out intermediateCA.csr`
5. Sign the intermediate CA's CSR with root CA
   b. `openssl ca -config ca.config -notext -cert rootCA.crt -keyfile rootCA.key -days 500 -in  intermediateCA.csr -out intermediateCA.crt`

Generate Device Certificate (ECDSA certificate as an example)

1. Generate private key
   a. `openssl ecparam -name prime256v1 -genkey -noout -out deviceCert.key`
2. Generate CSR for device certificate
   a. `openssl req -new -key deviceCert.key -out deviceCert.csr`
3. Sign the device certificate with the intermediate CA
   a. `openssl x509 -req -in deviceCert.csr -CA intermediateCA.crt -CAkey intermediateCA.key -CAcreateserial -out deviceCert.crt -days 500 -sha256`

Register both CA certificates

1. Get registration code
    a. `aws iot get-registration-code`
2. Generate private key for verification certificates
    a. `openssl ecparam -name prime256v1 -genkey -noout -out verificationCert.key`
3. Create CSR for verification certificates. **Set the Common Name field to your registration code obtained in the first step.**
    a. `openssl req -new -key verificationCert.key -out verificationCert.csr`
4. Sign a verification certificate using root CA and another one using intermediate CA
    a. `openssl x509 -req -in verificationCert.csr -CA rootCA.crt -CAkey rootCA.key -CAcreateserial -out rootCAverificationCert.crt -days 500 -sha256`
    b. `openssl x509 -req -in verificationCert.csr -CA intermediateCA.crt -CAkey intermediateCA.key -CAcreateserial -out intermediateCAverificationCert.crt -days 500 -sha256`
5. Register both CA certificates with AWS IoT
    a. `aws iot register-ca-certificate --ca-certificate file://rootCA.crt --verification-cert file://rootCAverificationCert.crt`
    b. `aws iot register-ca-certificate --ca-certificate file://intermediateCA.crt --verification-cert file://intermediateCAverificationCert.crt`
6. Activate both CA certificates
    a. `aws iot update-ca-certificate --certificate-id xxxxxxxxxxxxxxxx --new-status ACTIVE`

Register Device Certificate

1. Register the device certificate with AWS IoT
    a. `aws iot register-certificate --certificate-pem file://deviceCert.crt --ca-certificate-pem file://intermediateCA.crt`
2. Activate the device certificate
    a. `aws iot update-certificate --certificate-id xxxxxxxxxxxxxx --new-status ACTIVE`

deviceCert.crt is device certificate and deviceCert.key is device private key.

Appendix O: Source for ca.config

```
#
# OpenSSL example configuration file.
# This is mostly being used for generation of certificate requests.
#

# This definition stops the following lines choking if HOME isn't
# defined.
HOME                    = .
RANDFILE        = $ENV::HOME/.rnd

# Extra OBJECT IDENTIFIER info:
#oid_file               = $ENV::HOME/.oid
oid_section             = new_oids

# To use this configuration file with the "-extfile" option of the
# "openssl x509" utility, name here the section containing the
# X.509v3 extensions to use:
# extensions            =
# (Alternatively, use a configuration file that has only
# X.509v3 extensions in its main [= default] section.)

[ new_oids ]

# We can add new OIDs in here for use by 'ca', 'req' and 'ts'.
# Add a simple OID like this:
# testoid1=1.2.3.4
# Or use config file substitution like this:
# testoid2=${testoid1}.5.6

# Policies used by the TSA examples.
tsa_policy1 = 1.2.3.4.1
tsa_policy2 = 1.2.3.4.5.6
tsa_policy3 = 1.2.3.4.5.7

####################################################################
[ ca ]
default_ca      = CA_default            # The default ca section

####################################################################
[ CA_default ]

dir             = .                    # Where everything is kept
certs           = $dir                  # Where the issued certs are kept
crl_dir         = $dir                  # Where the issued crl are kept
database= $dir/index.txt# database index file.
#unique_subject = no                    # Set to 'no' to allow creation of
                                        # several ctificates with same subject.
new_certs_dir   = $dir                  # default place for new certs.

certificate     = $dir/cacert.pem       # The CA certificate
serial          = $dir/serial           # The current serial number
crlnumber       = $dir/crlnumber# the current crl number
                                        # must be commented out to leave a V1 CRL
crl             = $dir/crl.pem          # The current CRL
private_key     = $dir/private/cakey.pem# The private key
RANDFILE= $dir/private/.rand    # private random number file

x509_extensions = usr_cert              # The extentions to add to the cert

# Comment out the following two lines for the "traditional"
# (and highly broken) format.
name_opt        = ca_default            # Subject Name options
cert_opt        = ca_default            # Certificate field options
```



```
# Extension copying option: use with caution.
# copy_extensions = copy

# Extensions to add to a CRL. Note: Netscape communicator chokes on V2 CRLs
# so this is commented out by default to leave a V1 CRL.
# crlnumber must also be commented out to leave a V1 CRL.
# crl_extensions= crl_ext

default_days     = 365                   # how long to certify for
default_crl_days      = 30                       # how long before next CRL
default_md       = default               # use public key default MD
preserve= no                     # keep passed DN ordering

# A few difference way of specifying how similar the request should look
# For type CA, the listed attributes must be the same, and the optional
# and supplied fields are just that :-)
policy           = policy_match

# For the CA policy
[ policy_match ]
countryName                 = match
stateOrProvinceName         = match
organizationName= match
organizationalUnitName      = optional
commonName                  = supplied
emailAddress                = optional

# For the 'anything' policy
# At this point in time, you must list all acceptable 'object'
# types.
[ policy_anything ]
countryName                 = optional
stateOrProvinceName         = optional
localityName                = optional
organizationName= optional
organizationalUnitName      = optional
commonName                  = supplied
emailAddress                = optional

####################################################################
[ req ]
default_bits                = 2048
default_keyfile  = privkey.pem
distinguished_name          = req_distinguished_name
attributes                  = req_attributes
x509_extensions  = v3_ca # The extentions to add to the self signed cert

# Passwords for private keys if not present they will be prompted for
# input_password = secret
# output_password = secret

# This sets a mask for permitted string types. There are several options.
# default: PrintableString, T61String, BMPString.
# pkix   : PrintableString, BMPString (PKIX recommendation before 2004)
# utf8only: only UTF8Strings (PKIX recommendation after 2004).
# nombstr : PrintableString, T61String (no BMPStrings or UTF8Strings).
# MASK:XXXX a literal mask value.
# WARNING: ancient versions of Netscape crash on BMPStrings or UTF8Strings.
string_mask = utf8only

# req_extensions = v3_req # The extensions to add to a certificate request

[ req_distinguished_name ]
countryName                         = Country Name (2 letter code)
countryName_default                 = AU
countryName_min                     = 2
```



```
countryName_max                 = 2

stateOrProvinceName             = State or Province Name (full name)
stateOrProvinceName_default     = Some-State

localityName                    = Locality Name (eg, city)

0.organizationName              = Organization Name (eg, company)
0.organizationName_default      = Internet Widgits Pty Ltd

# we can do this but it is not needed normally :-)
#1.organizationName             = Second Organization Name (eg, company)
#1.organizationName_default     = World Wide Web Pty Ltd

organizationalUnitName          = Organizational Unit Name (eg, section)
#organizationalUnitName_default =

commonName                      = Common Name (e.g. server FQDN or YOUR name)
commonName_max                  = 64

emailAddress                    = Email Address
emailAddress_max        = 64

# SET-ex3                       = SET extension number 3

[ req_attributes ]
challengePassword               = A challenge password
challengePassword_min           = 4
challengePassword_max           = 20

unstructuredName        = An optional company name

[ usr_cert ]

# These extensions are added when 'ca' signs a request.

# This goes against PKIX guidelines but some CAs do it and some software
# requires this to avoid interpreting an end user certificate as a CA.

basicConstraints=CA:TRUE

# Here are some examples of the usage of nsCertType. If it is omitted
# the certificate can be used for anything *except* object signing.

# This is OK for an SSL server.
# nsCertType                    = server

# For an object signing certificate this would be used.
# nsCertType = objsign

# For normal client use this is typical
# nsCertType = client, email

# and for everything including object signing:
# nsCertType = client, email, objsign

# This is typical in keyUsage for a client certificate.
# keyUsage = nonRepudiation, digitalSignature, keyEncipherment

# This will be displayed in Netscape's comment listbox.
nsComment                       = "OpenSSL Generated Certificate"

# PKIX recommendations harmless if included in all certificates.
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer

# This stuff is for subjectAltName and issuerAltname.
```



```
# Import the email address.
# subjectAltName=email:copy
# An alternative to produce certificates that aren't
# deprecated according to PKIX.
# subjectAltName=email:move

# Copy subject details
# issuerAltName=issuer:copy

#nsCaRevocationUrl               = http://www.domain.dom/ca-crl.pem
#nsBaseUrl
#nsRevocationUrl
#nsRenewalUrl
#nsCaPolicyUrl
#nsSslServerName

# This is required for TSA certificates.
# extendedKeyUsage = critical,timeStamping

[ v3_req ]

# Extensions to add to a certificate request

basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment

[ v3_ca ]


# Extensions for a typical CA


# PKIX recommendation.

subjectKeyIdentifier=hash

authorityKeyIdentifier=keyid:always,issuer

# This is what PKIX recommends but some broken software chokes on critical
# extensions.
#basicConstraints = critical,CA:true
# So we do this instead.
basicConstraints = CA:true

# Key usage: this is typical for a CA certificate. However since it will
# prevent it being used as an test self-signed certificate it is best
# left out by default.
# keyUsage = cRLSign, keyCertSign

# Some might want this also
# nsCertType = sslCA, emailCA

# Include email address in subject alt name: another PKIX recommendation
# subjectAltName=email:copy
# Copy issuer details
# issuerAltName=issuer:copy

# DER hex encoding of an extension: beware experts only!
# obj=DER:02:03
# Where 'obj' is a standard or added object
# You can even override a supported extension:
# basicConstraints= critical, DER:30:03:01:01:FF

[ crl_ext ]

# CRL extensions.
# Only issuerAltName and authorityKeyIdentifier make any sense in a CRL.
```



```
# issuerAltName=issuer:copy
authorityKeyIdentifier=keyid:always

[ proxy_cert_ext ]
# These extensions should be added when creating a proxy certificate

# This goes against PKIX guidelines but some CAs do it and some software
# requires this to avoid interpreting an end user certificate as a CA.

basicConstraints=CA:FALSE

# Here are some examples of the usage of nsCertType. If it is omitted
# the certificate can be used for anything *except* object signing.

# This is OK for an SSL server.
# nsCertType                    = server

# For an object signing certificate this would be used.
# nsCertType = objsign

# For normal client use this is typical
# nsCertType = client, email

# and for everything including object signing:
# nsCertType = client, email, objsign

# This is typical in keyUsage for a client certificate.
# keyUsage = nonRepudiation, digitalSignature, keyEncipherment

# This will be displayed in Netscape's comment listbox.
nsComment                       = "OpenSSL Generated Certificate"

# PKIX recommendations harmless if included in all certificates.
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer

# This stuff is for subjectAltName and issuerAltname.
# Import the email address.
# subjectAltName=email:copy
# An alternative to produce certificates that aren't
# deprecated according to PKIX.
# subjectAltName=email:move

# Copy subject details
# issuerAltName=issuer:copy

#nsCaRevocationUrl              = http://www.domain.dom/ca-crl.pem
#nsBaseUrl
#nsRevocationUrl
#nsRenewalUrl
#nsCaPolicyUrl
#nsSslServerName

# This really needs to be in place for it to be a proxy certificate.
proxyCertInfo=critical,language:id-ppl-anyLanguage,pathlen:3,policy:foo

####################################################################
[ tsa ]

default_tsa = tsa_config1       # the default TSA section

[ tsa_config1 ]

# These are used by the TSA reply generation only.
dir             = ./demoCA              # TSA root directory
serial          = $dir/tsaserial# The current serial number (mandatory)
```

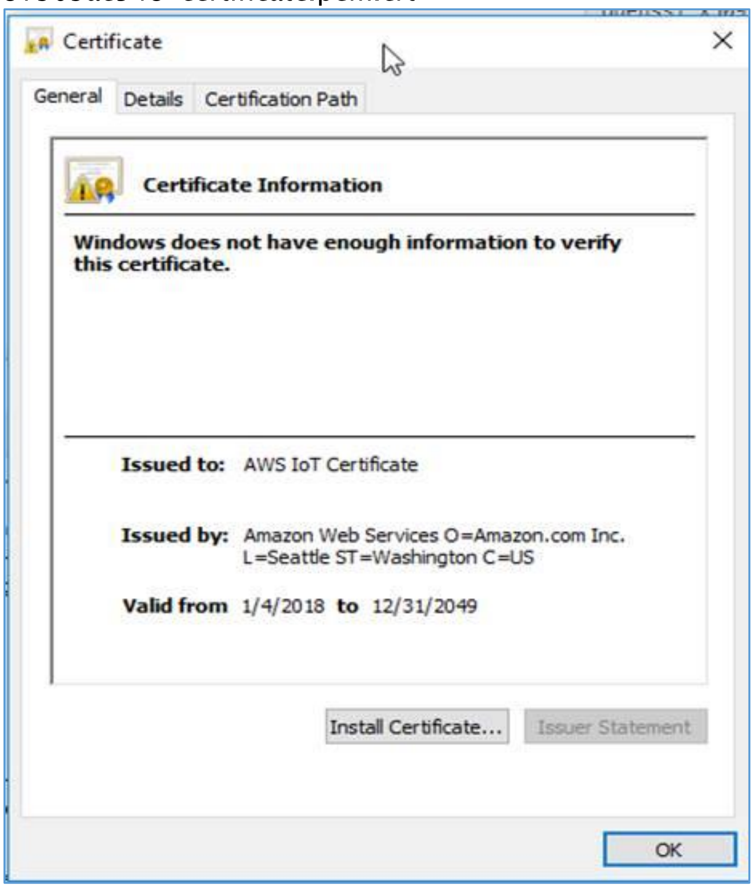


```
crypto_device    = builtin               # OpenSSL engine to use for signing
signer_cert      = $dir/tsacert.pem      # The TSA signing certificate
                                         # (optional)
certs            = $dir/cacert.pem       # Certificate chain to include in reply
                                         # (optional)
signer_key       = $dir/private/tsakey.pem # The TSA private key (optional)

default_policy   = tsa_policy1           # Policy if request did not specify it
                                         # (optional)
other_policies   = tsa_policy2, tsa_policy3      # acceptable policies (optional)
digests          = md5, sha1             # Acceptable message digests (mandatory)
accuracy= secs:1, millisecs:500, microsecs:100   # (optional)
clock_precision_digits  = 0      # number of digits after dot. (optional)
ordering         = yes   # Is ordering defined for timestamps?
                                 # (optional, default: no)
tsa_name         = yes   # Must the TSA name be included in the reply?
                                 # (optional, default: no)
ess_cert_id_chain        = no    # Must the ESS cert id chain be included?
                                 # (optional, default: no)
```


Appendix P: Modify issuer in a certificate

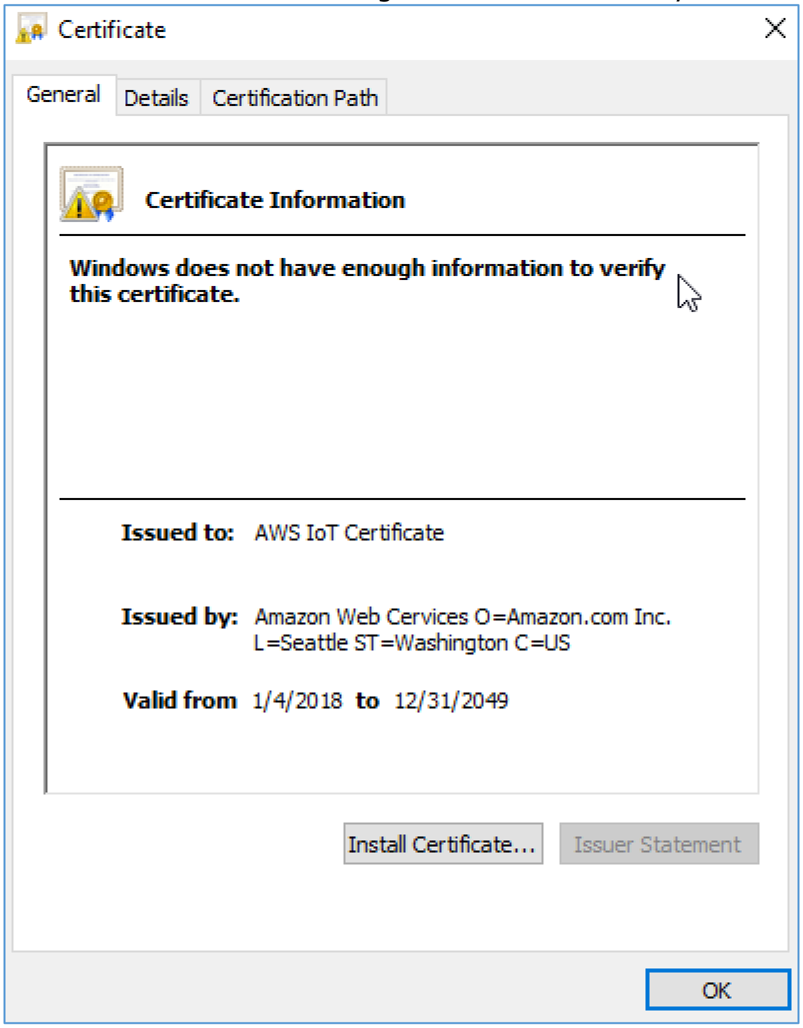
1. Take the valid client certificate that you have been using as a base. In this example is it 81909ac548-certificate.pem.crt



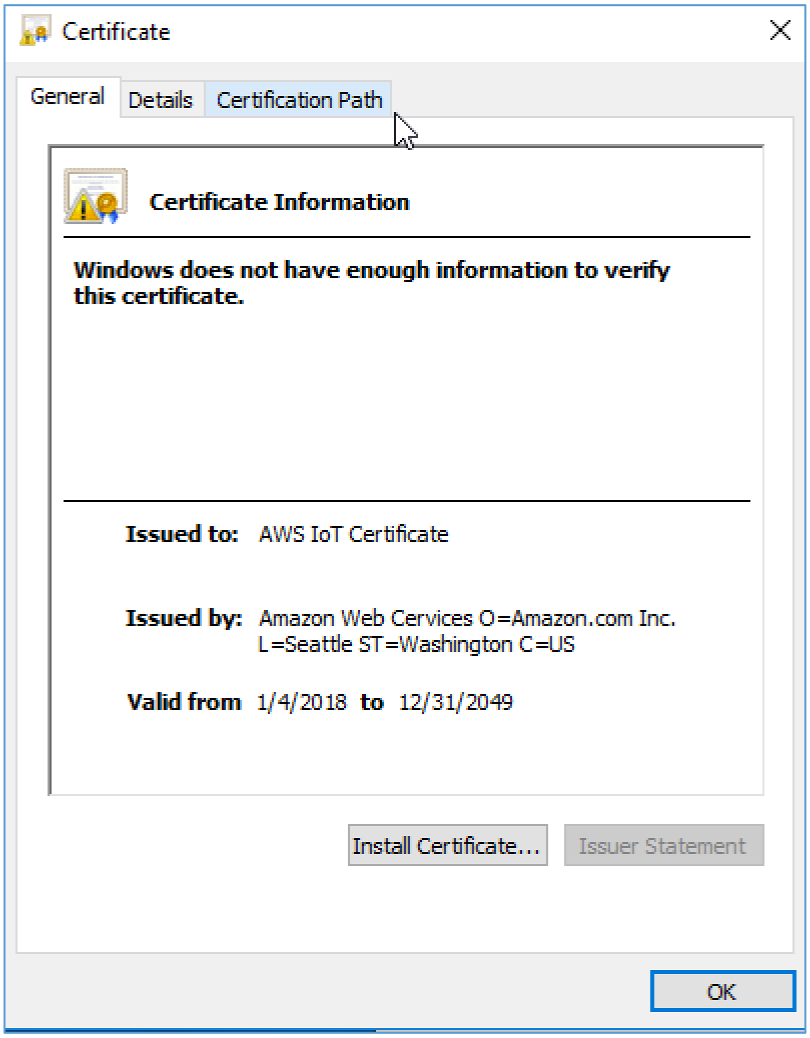
2. Convert the certificate from PEM to DER (openssl x509 -outform der -in 81909ac548-certificate.pem.crt -out 81909ac548-certificate.der.crt)
3. Open the .der certificate. “Amazon Web Services” in hex is **41 6d 61 7a 6f 6e 20 57 65 62 20 53 65 72 76 69 63 65 73**. Search for this sequence in your DER output:


```
3082 035a 3082 0242 a003 0201 0202 1500
a09a 7038 af7d f387 04cc d2bd fb9f 1cdf
c132 8bca 300d 0609 2a86 4886 f70d 0101
0b05 0030 4d31 4b30 4906 0355 040b 0c42
416d 617a 6f6e 2057 6562 2053 6572 7669
6365 7320 4f3d 416d 617a 6f6e 2e63 6f6d
2049 6e63 2e20 4c3d 5365 6174 746c 6520
5354 3d57 6173 6869 6e67 746f 6e20 433d
5553 301e 170d 3138 3031 3034 3137 3335
3036 5a17 0d34 3931 3233 3132 3335 3935
395a 301e 311c 301a 0603 5504 030c 1341
5753 2049 6f54 2043 6572 7469 6669 6361
7465 3082 0122 300d 0609 2a86 4886 f70d
0101 0105 0003 8201 0f00 3082 010a 0282
0101 00c0 d5d4 cce8 9963 fff1 dc29 781b
6a72 1ec5 7aee 280d b88a 78bb 547e d816
67fc 59d2 4b6d 1ecd 523f e256 4926 4005
7a13 e1a3 1fcf 311d 72a5 30a2 5443 3961
16e5 09f4 837c 2e44 d765 1b72 4c4f 62f8
50b9 e704 4347 c3d9 1b33 3dce 557e 67cd
cf87 9ce0 df5e 8eb6 522a d6ee 4bfb 0737
48fd e1e2 c848 31a6 dc0e 661f 1df9 d22d
3002 5ab8 71dd 6d4b 1bf7 6ecf 5c49 3170
0bf1 64d9 6437 43e4 d697 6d96 cce6 b44a
9bda e897 2abc a724 3d0a 6c62 bd58 b74e
10db 653f ee38 1b51 b24b 4a11 d708 b052
d7e1 fee7 61da bb2f 2cb2 16b0 2f0e 22dd
3b3a 7889 85c6 91a6 dd36 e64d f4c6 cacb
f76f 642e 126f 3bcd 4665 e8a9 320b ba82
ea8f f64d deac dea3 0882 62a9 2f4b be74
26ed c702 0301 0001 a360 305e 301f 0603
551d 2304 1830 1680 14d9 b618 e527 6c46
f606 16bb 1c9e 3562 4c47 0597 6330 1d06
0355 1d0e 0416 0414 eeec 1155 1654 beb6
4ecf b39c 01c9 3081 5aeb 125e 300c 0603
551d 1301 01ff 0402 3000 300e 0603 551d
0f01 01ff 0404 0302 0780 300d 0609 2a86
4886 f70d 0101 0b05 0003 8201 0100 20fc
ae39 bbdd 597c 8909 83d5 ef50 98ca 9f81
31e6 68bf 65bb 56ef 1b4d 6a63 9376 2c6b
d232 3edb 50db 0a70 1e32 df19 e317 b1c3
fa1f 1f86 db08 7022 370b e9b1 e73f 7b70
cc67 df3e 7006 e390 6eff 3755 bf8f 1530
6810 2a1b bf2a c16c 091b 32f5 6325 d3d2
a356 0ac1 9566 253c 7e79 d642 03db c460
02b8 e260 693a 2e60 5efa a856 9952 ebeb
5345 13e1 5226 b6aa 9cb2 ce7b 34b3 411f
be2c 8779 775c 2ef0 eef6 83b2 9f7d b738
cba4 dbe1 171f 236a 5730 fdc3 cd2f 3e26
4614 e27f 455a 270a 37f0 f6ef 880c 80fd
d57e 8af8 275b 76c2 1de1 d4e8 df92 5148
da39 1230 c652 303a 217c 663f 4df0 d838
a3b0 d128 16f2 6d70 49d6 9931 05e8 de60
ec3c 3e5c d6f5 9737 64e3 ecad 24a5
```


4. Modify the sequence to say 'Amazon Web Cervices', switching out the 53 to be a 43. Save the file. To verify your change, you can check out the modified cert in the windows certificate manager. See that it now says **Issued by**: Amazon Web Cervices



5. Convert your newly modified certificate back to PEM. openssl x509 -inform der -in 81909ac548-certificate.der.crt -out 81909ac548-cert-modified.pem.crt



Again, viewing this in the certificate viewer should show the modified certificate.

6. Put this certificate into the Certificate Configuration Tool (demos\common\devmode_key_provisioning\CertificateConfigurationTool) and copy the formatted output.

Appendix Q: Shared Work Plan for Partnership

The Shared Work Plan proposes certain tactical aspects of our partnership on how to qualify Amazon FreeRTOS on your IoT microcontrollers, jointly go to market and support customers.

Objective

The objective of the partnership is to provide embedded developers a platform to easily build and deploy connected microcontroller solutions securely, quickly and economically. To that end, you and AWS will work in partnership to qualify Amazon FreeRTOS on your IoT microcontrollers, drive joint marketing/PR activities, and support service requests. The near term goal is to qualify at least one of your IoT microcontrollers under the Amazon FreeRTOS Qualification Program and have the infrastructure for joint marketing, PR and customer service ready by a mutually agreed upon launch date and event (referred to from here on as the “Launch Date” and “Launch Event”.)

Details

The terms below and in the subsequent sections outline the high-level agreement between you and AWS. This is not meant as a replacement for a legal agreement but instead allows the companies to move forward with a common understanding of the joint operating agreement and a definition of roles, responsibilities, and commitments.

Under this partnership, you will be responsible for qualifying software that allows Amazon FreeRTOS to run on your IoT microcontroller (including board specific libraries and device drivers,) and AWS will be responsible for running final verification tests to confirm qualification. The public announcement of Amazon FreeRTOS support of your IoT microcontroller will be at the Launch Event, to be held on the Launch Date.

Development

Corporate Code Submissions: You can download Amazon FreeRTOS source code and test code from the Amazon FreeRTOS GitHub repository, and can then update your board specific libraries to make them compatible with the Amazon FreeRTOS software libraries. You can then share the code with AWS by uploading these updated libraries to your private GitHub repository and providing AWS access. These libraries will then be tested, code-reviewed and then distributed along with Amazon FreeRTOS source-code under the terms of MIT license. Prior to the Launch Event, AWS will work with select OEMs who will be our launch customers and share with them Amazon FreeRTOS and your libraries. AWS will keep the distribution of your libraries restricted to these OEMs under a license that prohibits them from distributing your libraries further.

Source-code Releases: AWS will continually update and release Amazon FreeRTOS source-code as major, minor or point releases. These updates could range from major feature additions to minor optimizations.

For a minor version change (e.g., 9.1.0 to 9.2.0) or point release (e.g., 9.1.1 to 9.1.2,) AWS will make best efforts to inform you 4 weeks before a new Amazon FreeRTOS source-code is released. While you will not need to requalify because these minor and point versions will be backward compatible, AWS recommends that you retest your libraries for performance with the new Amazon FreeRTOS source-code and inform AWS in case of concerns. For a major version change (e.g. 9.3.0 to 10.0.0,) AWS will make best efforts to inform you 8 weeks before a new Amazon FreeRTOS source-code is released. You should make best efforts to provide feedback, test and successfully pass the Amazon FreeRTOS Qualification Program tests within the 8 weeks. Only major version changes would require an Amazon FreeRTOS re-qualification, and you would have an opportunity to evaluate which microcontrollers you want to qualify for the major release. During the transition, AWS will continue to support the previous release, unless a bug severely affects the security or operation of the end application. At the time of release of the new version, AWS should already have your IoT microcontroller Amazon FreeRTOS-qualified with the new Amazon FreeRTOS source-code.

Amazon FreeRTOS Qualification of New Chipsets: All new chipsets that will run Amazon FreeRTOS will need to pass the Amazon FreeRTOS Qualification Program tests. For qualified chipsets, Amazon FreeRTOS source-code and your libraries will be available in the Amazon FreeRTOS console for developers to download. If you decide to distribute Amazon FreeRTOS source-code to customers before the chipset is Amazon FreeRTOS-qualified, you assumes responsibility for service support until the chipset is Amazon FreeRTOS-qualified.

Roadmap Alignment: We propose that you and AWS meet at least once a quarter and give updates to each other on new features, services, chipsets and product changes, so that AWS can enable mutual customers to benefit from the latest hardware and software technologies.

Code Distribution: AWS will distribute source-code via two channels: AWS and GitHub. Using the Amazon FreeRTOS console, developers can choose which libraries are relevant to their application and can then download those libraries along with a working sample. Using GitHub, developers can download the entire source-code.

Go-to-Market

Launch Event: You and AWS will jointly launch the Amazon FreeRTOS-qualified chipset at the Launch Event. In addition, AWS would like to share booth space, have joint demo sessions and collaborate on design contests with you at external events such as CES and Electronica.

PR Activities: AWS will encourage you to have your own press release and AWS will provide support for the same.

Marketing Activities: You and AWS will jointly work on any specific case studies or use cases that highlight the capabilities of your IoT chipset and Amazon FreeRTOS. There will be a logo specific to Amazon FreeRTOS and related products, and both you and AWS will provide a specific logo to be used in this program.

AWS will have the Amazon FreeRTOS webpage highlighting the partnership and containing details on Amazon FreeRTOS-qualified chipsets (such as Dev Kits, FAQs, and Getting Started,) with hyperlinks pointing to your chipsets. AWS will also provide marketing collateral to you so that you can incorporate Amazon FreeRTOS messaging on your websites.

Sales and FAE Enablement: AWS will participate in information sessions and supply you with any Amazon FreeRTOS-specific collateral (such as customer and workshop presentation slides) that you might need to train Sales/FAEs at its training events.

Support

OEM Support: AWS believes in a shared support model. OEMs will regularly approach both you and AWS for service requests. We will have mutually agreed upon SLAs in terms of responsiveness and participation in customer forums. AWS will be responsible for supporting service requests from OEMs and your engineers. In case the service request is board specific or pertains to the microcontroller, you will provide support to AWS customer representatives solving the case. Similarly, AWS will field technical questions from you to support your customers.

Division of Responsibilities (Technical)

Specifically, the work falls into three categories:

1. Board Specific Libraries – You, ported to be Amazon FreeRTOS compatible
2. Amazon FreeRTOS Qualification – You, with support and diagnostics from AWS. It gives developers confidence that the microcontroller they choose fully supports Amazon FreeRTOS features.
3. Amazon FreeRTOS Verification Tests – AWS, with test reports and hardware from you

Timeline

AWS will have a joint launch with you at the Launch Event, to be held on the Launch Date. You and AWS will have a chipset Amazon FreeRTOS-qualified by the Launch Event.

Source Code Versioning Scheme

Source Code Version: X.Y.Z

Version	X	Y	Z
Major Release	Change	-	-
Minor Release	No Change	Change	-
Point Release	No Change	No Change	Change

Appendix R: Getting Started Guide Template

Getting Started with the [board-name]

Provide a brief description of the board(s) that are qualified to run Amazon FreeRTOS with links to more in-depth information on your company's website

- What hardware is required?
- What host operating systems are supported?
- What IDEs are supported? (Include links to download IDEs)
- What toolchains will the developer use? (Include links to download toolchains)
- Prerequisite

Prerequisites

List any prerequisites for your board

Setting up the [board-name] Hardware

Provide instructions for setting up the hardware including:

- Jumper settings
- Driver installation (include links to supported driver versions)
- Connecting the board to a computer

Setting Up Your Environment

- Provide instructions to establish a serial connection to your board for each host operating system.
- Provide instructions and link(s) to set up the toolchain for each host operating system.
- Provide instructions for installing/configuring any board-specific software for each host operating system (anything listed here should be called out in the prerequisites section).

Download and Build Amazon FreeRTOS

- Provide instructions to download Amazon FreeRTOS from the Amazon FreeRTOS Online Connection Wizard or GitHub repository.
- Provide instructions for loading/importing the Amazon FreeRTOS sample code into your IDE.

Configure Your Project

Open `<BASE_FOLDER>\demos\common\include\aws_clientcredential.h` in your IDE or text editor.

To configure your project, you need to know your AWS IoT endpoint.

To find your AWS IoT endpoint

1. Browse to the AWS IoT console.
2. In the left navigation pane, choose **Settings**.
3. Copy your AWS IoT endpoint from the **Endpoint** text box. It should look like `<1234567890123>.iot.<us-east-1>.amazonaws.com`.
4. Open `aws_demos/application_code/common_demos/include/aws_clientcredential.h` and set `clientcredentialMQTT_BROKER_ENDPOINT` to your AWS IoT endpoint. Save your changes.

You also need to know your Wi-FI network SSID, password, and security type, and the name of the AWS IoT thing that represents your device. Valid security types are:

- `eWiFiSecurityOpen`: Open, no security.
- `eWiFiSecurityWEP`: WEP security.
- `eWiFiSecurityWPA`: WPA security.
- `eWiFiSecurityWPA2`: WPA2 security.

In the **[insert your appropriate IDE window or text editor]** window, open `aws_demos\application_code\common_demos\include\aws_clientcredential.h`.

Specify values for the following #define constants:

- `clientcredentialMQTT_BROKER_ENDPOINT`: Your AWS IoT endpoint.
- `clientcredentialIOT_THING_NAME`: The AWS IoT thing for your board.
- `clientcredentialWIFI_SSID`: The SSID for your Wi-Fi network.
- `clientcredentialWIFI_PASSWORD`: The password for your Wi-Fi network.
- `clientcredentialWIFI_SECURITY`: The security type for your Wi-Fi network.

Make sure to save your changes.

Configure Your AWS IoT Credentials

The certificate and private key must be hard-coded into the Amazon FreeRTOS demo code. This is for demo purposes only. Production level applications should store these files in a secure location. Amazon FreeRTOS is a C language project, and the certificate and private key must be specially formatted to be added to the project.

To format your certificate and private key

1. In a browser window, open `<BASE_FOLDER>\demos\common\devmode_key_provisioning\CertificateConfigurationTool\CertificateConfigurator.html`.
2. Under **Certificate PEM file**, choose the `<ID>-certificate.pem.crt` you downloaded from the AWS IoT console.

3. Under **Private Key PEM file**, choose the `<ID>-private.pem.key` you downloaded from the AWS IoT console.
4. Choose **Generate and save aws_clientcredential_keys.h**, and then save the file in `<BASE_FOLDER>\demos\common\include`. This overwrites the stub file in the directory.

Run the FreeRTOS Samples

To flash the demo application onto your board

- Provide instructions on how to flash the sample application to your board including:
  - How to connect your board to the host computer
  - How to use an IDE or other tools to flash the sample application to your board
  - How to verify the sample application is running correctly
  - Troubleshooting steps for resolving problems

Debugging the samples

- Provide instructions on how to use any on-board debugging interface or external debuggers for each supported host OS.

Appendix S: Hardware Information

General Information:

Company Name	
Company Name (short, if any) for Amazon FreeRTOS Console	
High Resolution Logo	
Link to Landing Page	
Company Description (15 words)	

Development Board Information:

Board Name	
Board Name (20 chars) for Amazon FreeRTOS Console	
High Resolution Board Image	
Board Description	
Board Description (50 chars) for Amazon FreeRTOS Console	
Microcontroller Family Name	
Board Datasheet	
Compiler Options (optimization)	
IDE with Version Number	
Link to Board Landing Page	
Getting Started Guide	
Link to Purchase Board	

Appendix T: Glossary

$AFR_HOME	The path where Amazon FreeRTOS is installed/extracted.