Amazon FreeRTOS

Qualification Guide

Amazon FreeRTOS: Qualification Guide

Table of Contents

What Is Amazon FreeRTOS Qualification?

Amazon Partner Network Partners can use the AWS Device Qualification Program to officially qualify a microcontroller (MCU) development board for Amazon FreeRTOS.

Amazon FreeRTOS qualification gives developers a reliable and consistent experience across a range of qualified, MCU-based development boards. By abstracting some of the complexity of embedded development, Amazon FreeRTOS qualification makes it possible for developers to focus on designing the application code for their product, so they can rapidly evaluate, prototype, and productize IoT solutions. This qualification process helps bridge the gap between cloud developers and embedded engineers.

Qualified boards are eligible for listing on the AWS Partner Device Catalog. Amazon FreeRTOS-ported code for qualified boards is eligible for listing on the Amazon FreeRTOS console and on GitHub. Amazon also provides getting started documentation for each qualified platform.

To qualify a device for Amazon FreeRTOS, you must port Amazon FreeRTOS to your device, and then follow the Device Qualification Program steps. For information, see the AWS Partner Network website.

For information about qualifying your device for Amazon FreeRTOS, see Qualifying Your Device (p. 5).

Qualification FAQs

If a version of Amazon FreeRTOS is released after I have started porting a previous version, do I need to start over using the latest version?

> Always start porting the latest version of Amazon FreeRTOS. If your porting process is underway when a new version is released, you might be able to qualify for the Amazon FreeRTOS version that you started porting. Consult your AWS representative if you have questions.

My device does not support Wi-Fi. Is a port of the Amazon FreeRTOS Wi-Fi library required to qualify for Amazon FreeRTOS?

> The primary requirement is that your device can connect to the AWS Cloud. If your device can connect to the AWS Cloud across a secure ethernet connection, the Wi-Fi library is not a requirement.

My device does not support Bluetooth Low Energy (BLE) or over-the-air (OTA) updates. Are ports for these Amazon FreeRTOS libraries required to qualify for Amazon FreeRTOS?

> BLE and OTA ports are optional for qualification.

My board does not have on-chip TCP/IP functionality. Is a particular TCP/IP stack required for Amazon FreeRTOS qualification?

> If your board does not have on-chip TCP/IP functionality, you can use either the FreeRTOS+TCP TCP/IP stack or version 2.0.3 of the lwIP TCP/IP stack to pass TCP/IP qualification requirements. For more information, see Porting a TCP/IP Stack in the *Amazon FreeRTOS Porting Guide*.

Is a particular TLS stack required for qualification?

> Amazon FreeRTOS supports mbedTLS and off-chip TLS implementations, such as those found on some network processors. No matter which TLS implementation is used by your device's port

of Amazon FreeRTOS, the port must pass the Device Tester validation tests for TLS. For more information, see Porting the TLS Library in the *Amazon FreeRTOS Porting Guide*.

Does my device need to pass all of the AWS IoT Device Tester validation tests to qualify? Is there a way to qualify without passing all of the tests?

Your device must pass all of the required validation tests to qualify for Amazon FreeRTOS. The only exceptions are for Wi-Fi, BLE, and OTA.

If you have questions about qualification that are not answered on this page or in the rest of the Amazon FreeRTOS Qualification Guide, contact your AWS representative or the Amazon FreeRTOS engineering team.

Documentation History

Revision History of Amazon FreeRTOS Porting and Qualification Documentation

Date	Porting and Qualification Documentation Version	Change History	Amazon FreeRTOS Version
May 21, 2019	1.4.8 (Porting Guide) 1.4.8 (Qualification Guide)	• Porting documentation moved to the Amazon FreeRTOS Porting Guide • Qualification documentation moved to the Amazon FreeRTOS Qualification Guide	1.4.8
February 25, 2019	1.1.6	• Removed download and configuration instructions from Getting Started Guide Template Appendix (page 84)	1.4.5 1.4.6 1.4.7
December 27, 2018	1.1.5	• Updated Checklist for Qualification appendix with CMake requirement (page 70)	1.4.5 1.4.6
December 12, 2018	1.1.4	• Added lwIP porting instructions to TCP/IP porting appendix (page 31)	1.4.5
November 26, 2018	1.1.3	• Added BLE porting appendix (page 52) • Added AWS IoT Device Tester for Amazon FreeRTOS	1.4.4

Date	Porting and Qualification Documentation Version	Change History	Amazon FreeRTOS Version
		testing information throughout document • Added CMake link to Information for listing on the Amazon FreeRTOS Console appendix (page 85)	
November 7, 2018	1.1.2	• Updated PKCS #11 PAL interface porting instructions in PKCS #11 porting appendix (page 38) • Updated path to `CertificateConfigurator.html` (page 76) • Updated Getting Started Guide Template appendix (page 80)	1.4.3
October 8, 2018	1.1.1	• Added new "Required for AFQP" column to `aws_test_runner_config.h` test configuration table (page 16) • Updated Unity module directory path in Create the Test Project section (page 14) • Updated "Recommended Porting Order" chart (page 22) • Updated client certificate and key variable names in TLS appendix, Test Setup (page 40) • File paths changed in Secure Sockets porting appendix, Test Setup (page 34); TLS porting appendix, Test Setup (page 40); and TLS Server Setup appendix (page 57)	1.4.2

Date	Porting and Qualification Documentation Version	Change History	Amazon FreeRTOS Version
August 27, 2018	1.1.0	• Added OTA Updates porting appendix (page 47) • Added Bootloader porting appendix (page 51)	1.4.0 1.4.1
August 9, 2018	1.0.1	• Updated "Recommended Porting Order" chart (page 22) • Updated PKCS #11 porting appendix (page 36) • File paths changed in TLS porting appendix, Test Setup (page 40), and TLS Server Setup appendix, step 9 (page 51) • Fixed hyperlinks in MQTT porting appendix, Prerequisites (page 45) • Added AWS CLI config instructions to examples in Instructions to Create a BYOC appendix (page 57)	1.3.1 1.3.2
July 31, 2018	1.0.0	Initial version of the Amazon FreeRTOS Qualification Program Guide	1.3.0

Qualifying Your Device

To qualify your device for Amazon FreeRTOS

1. Port the Amazon FreeRTOS libraries to your device.

   Note
   Currently, ports of the Amazon FreeRTOS OTA and BLE libraries are not required for qualification.
   If your device does not support Wi-Fi, you can use an ethernet connection to connect to the AWS Cloud instead. A port of the Amazon FreeRTOS Wi-Fi library is not necessarily required.

   For instructions on porting Amazon FreeRTOS to your device, see the Amazon FreeRTOS Porting Guide.
2. Validate your ports with AWS IoT Device Tester for Amazon FreeRTOS.

   When using Device Tester to validate your ports for qualification, you must specify information in the `features` attribute of the `device.json` configuration file about the following ports:

   - TCP/IP

   ```
   {
       "name": "TCP/IP",
       "value": "On-chip | Offloaded | No"
   }
   ```

   - TLS

   ```
   {
       "name": "TLS",
       "value": "On-chip | Offloaded | No"
   }
   ```

   - Wi-Fi

   ```
   {
       "name": "WIFI",
       "value": "Yes | No"
   }
   ```

   - OTA

   ```
   {
       "name": "OTA",
       "value": "Yes | No"
   }
   ```

   Device Tester uses this information to determine which tests to run against your ported Amazon FreeRTOS code. Device Tester runs all other required library port tests by default.

   For information about AWS IoT Device Tester for Amazon FreeRTOS, see Using AWS IoT Device Tester for Amazon FreeRTOS in the Amazon FreeRTOS User Guide.
3. Work with your AWS representative to submit your ported Amazon FreeRTOS code.

Note
You must pass the Device Tester validation tests before you can submit the code.

4. Create the following for qualification submission:

   - A "Hello World" demo application that publishes messages from your device to the AWS Cloud over MQTT.

     For information, see Setting Up a Hello World Demo (p. 6).
   - A "Getting Started with Amazon FreeRTOS" guide for your device.

     For information, see Creating a Getting Started with Amazon FreeRTOS Guide for Your Device (p. 10).
   - A `CMakeLists.txt` file for building Amazon FreeRTOS applications for your device.

     For information, see Creating a CMakeLists.txt File for Your Platform (p. 11).
   - A list of detailed information for your hardware platform.

     For information, see Hardware Information for Amazon FreeRTOS Qualification (p. 24).
   - An appropriate open source license file for your device's Amazon FreeRTOS port.

     For information, see Providing an Open Source License for Your Code (p. 25).
   - (For boards qualifying for OTA updates) Instructions for code-signing.

     For examples, see Create a Code-Signing Certificate in the Amazon FreeRTOS User Guide.
   - (For boards qualifying for OTA updates that use custom bootloader) Information and instructions on the custom bootloader application.

     For a list of requirements, see Porting the Bootloader Demo in the Amazon FreeRTOS Porting Guide.

   These items are required for your device to be listed on the Amazon FreeRTOS console, for your device's code to be on GitHub, and for your device to receive Getting Started documentation support.
5. Verify that you have all that you need to submit your board for qualification with the Amazon FreeRTOS qualification check script.

   For more information about running the qualification check script, see Amazon FreeRTOS Qualification Check Script (p. 25).
6. Submit your qualification items using the Device Listing Portal.

You can use the Amazon FreeRTOS Qualification Checklist (p. 26) to keep track of the list of required steps for qualification.

Setting Up a Hello World Demo

To qualify for Amazon FreeRTOS, set up a Hello World demo application that runs on your qualified device. This demo publishes messages from your device to the AWS Cloud over MQTT.

To set up the Hello World demo

1. Follow the instructions in Configuring the Amazon FreeRTOS Download for Demos (p. 7) to configure the directory structure of your Amazon FreeRTOS download to fit your device.

2. Follow the instructions in Creating the Demo Project (p. 8) to create a demo project in your IDE.

After you set up the demo, create a "Getting Started with Amazon FreeRTOS" guide for your device. This guide walks users through setting up your device to run the Hello World demo.

Configuring the Amazon FreeRTOS Download for Demos

Under the download's root directory (*<amazon-freertos>*), there are five folders: `cmake`, `demos`, `lib`, `tests`, and `tools`.

```
<amazon-freertos>
+ - cmake    (Contains files for CMake support)
+ - demos    (Contains projects that build demo applications)
+ - lib    (Contains Amazon FreeRTOS and third-party libraries)
+ - tests    (Contains projects that build porting tests)
+ - tools    (Contains tools for configuration, development, and testing)
```

The `demos` folder is structured as follows:

```
<amazon-freertos>
+ - demos
    + - common    (Contains files built by all demo projects)
    + - pc    (Contains a demo 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)
```

The *<board>* folder is a template folder we provide to make it easier to create a demo project. Its directory structure ensures that all demo projects have a consistent organization. The *<board>* folder has the following structure:

```
<amazon-freertos>
+ - demos
    + - <vendor>
        + - <board>
            + - common
            |   + - application_code    (Contains main.c, which contains main())
            |   |   + - <vendor>_code    (Contains vendor-supplied, board-specific files)
            |   + - config_files    (Contains Amazon FreeRTOS config files)
            + - <ide>    (Contains an IDE-specific project)
```

To configure the demo project files

Copy `main.c` and `main.h` files for the demo application to the `application_code` folder. You can reuse the `main.c` from the `aws_tests` project that you used to test your ports.

1. Save any required vendor-supplied, board-specific libraries to the *<vendor>*`_code` folder.

   Important
   Do not save vendor-supplied libraries that are common across a target board's MCU family to any subdirectories of *<amazon-freertos>*`/tests` or *<amazon-freertos>*`/demos`.
2. Replace *<vendor>* in the *<vendor>*`_code` folder with the name of the vendor.
3. Rename the *<board>* folder to the name of the development board.

After you configure the demo project files, you can create the project in the IDE. For instructions, see Creating the Demo Project (p. 8).

Creating the Demo Project

After you configure your Amazon FreeRTOS download, you can create an IDE project with the required project structure for the Hello World demo.

Follow the instructions below to create an IDE project with the required IDE project structure for demo applications. These instructions were written using an Eclipse-based IDE, but Amazon FreeRTOS demo projects look the same in most IDEs.

Important
You must import all files in their original position in the directory structure. Never directly copy files into the project's folder, and never use absolute file paths.
If you are using an Eclipse-based IDE, do not configure the project to build all the files in any folder. Instead, add source files to a project by linking to each source file individually.

1. Create a project named `aws_demos` in the *<amazon-freertos>*/`demos/`*<vendor>*`/`*<board>*`/`*<ide>* directory.
2. In your IDE, create three virtual folders under `aws_demos`:

   - `application_code`
   - `config_files`
   - `lib`

   Under `aws_demos`, there should now be three virtual subdirectories in the IDE project:

   ```
   aws_demos    (The project name)
   + - application_code    (Contains application logic, in this case, demo code)
   + - config_files    (Contains header files that configure Amazon FreeRTOS libraries)
   + - lib    (Contains Amazon and third-party libraries)
   ```

   Note
   Eclipse generates an additional `includes` folder. This folder is not a part of the required structure.
3. Import all of the folders and files in *<amazon-freertos>*`/demos/`*<vendor>*`/`*<board>*`/common/application_code` into the `application_code` virtual folder.
4. Import all of the files in *<amazon-freertos>*`/demos/`*<vendor>*`/`*<board>*`/common/config_files` into the `config_files` virtual folder.
5. Under `application_code`, create a folder named `common_demos`.
6. Under `common_demos`, create a folder named `source`, and import all of the folders and files in all of the following directories into that folder:

   - *<amazon-freertos>*`/demos/common/demo_runner`
   - *<amazon-freertos>*`/demos/common/devmode_key_provisioning` (only the `.c` file)
   - *<amazon-freertos>*`/demos/common/mqtt`
   - *<amazon-freertos>*`/demos/common/logging`
7. Import the *<amazon-freertos>*`/demos/common/include` directory and its contents into the `common_demos` folder.
8. Under the `lib` virtual folder, create two virtual folders named `aws` and `third_party`.
9. Import the following files or directories and their contents into the `aws` folder:

   Note
   If you are importing a file or folder whose directory path extends beyond the `lib` directory, you must create virtual folders in the IDE to match the directory path after *<amazon-freertos>*`/lib`.

- *<amazon-freertos>*/lib/bufferpool
- *<amazon-freertos>*/lib/FreeRTOS
- *<amazon-freertos>*/lib/FreeRTOS/portable/MemMang/heap_4.c
- *<amazon-freertos>*/lib/FreeRTOS/portable/*<your-compiler>*
- *<amazon-freertos>*/lib/FreeRTOS-Plus-TCP

  Note
  If you have the FreeRTOS+TCP library, see Porting a TCP/IP Stack in the Amazon FreeRTOS Porting Guide to determine if other files must be included in this project.
- *<amazon-freertos>*/lib/include
- *<amazon-freertos>*/lib/include/private (only the .h files)
- *<amazon-freertos>*/lib/mqtt
- *<amazon-freertos>*/lib/pkcs11/portable/*<vendor>*/*<board>*/pkcs11.c
- *<amazon-freertos>*/lib/secure_sockets/portable/*<vendor>*/*<board>*/aws_secure_sockets.c
- *<amazon-freertos>*/lib/tls

  Note
  Import the tls directory only if you have ported the TLS library.
- *<amazon-freertos>*/lib/wifi/portable/*<vendor>*/*<board>*/aws_wifi.c

  Note
  Import the aws_wifi.c file only if you have ported the Wi-Fi library.

10. Import the following directories and their contents into third_party:

    Note
    If you are importing a file or folder whose directory path extends beyond the third_party directory, you must create virtual folders in the IDE to match the full directory structure, after *<amazon-freertos>*/lib/third_party.

    - *<amazon-freertos>*/lib/third_party/mcu_vendor/*<vendor>*/*<board>*/*<driver_library>*/*<driver_library_version>*
    - *<amazon-freertos>*/lib/third_party/mbedtls

      Note
      Rename mbedtls/library to mbedtls/source.
    - *<amazon-freertos>*/lib/third_party/pkcs11
11. Open your project's IDE properties, and add the following paths to your compiler's include path:

    - *<amazon-freertos>*/demos/common/include
    - *<amazon-freertos>*/lib/include
    - *<amazon-freertos>*/lib/include/private
    - *<amazon-freertos>*/lib/FreeRTOS/portable/*<compiler>*/*<architecture>*
    - *<amazon-freertos>*/lib/third_party/mbedtls/include
    - *<amazon-freertos>*/demos/*<vendor>*/*<board>*/common/config_files
    - Any paths required for vendor-supplied driver libraries.

Creating a Getting Started with Amazon FreeRTOS Guide for Your Device

To qualify for Amazon FreeRTOS, you need to create a Getting Started with Amazon FreeRTOS guide for your device. This guide walks users through setting up the hardware and development environment for developing applications for Amazon FreeRTOS devices, and building, running, and flashing the Amazon FreeRTOS Hello World demo on a device.

Your guide must include the following instructions:

- Setting up the device hardware.
- Setting up the development environment.
- Building and running the demo project.
- Debugging.
- Troubleshooting.

We also recommend that your guide includes:

- A link to the MCU datasheet.
- A Printed Circuit Board (PCB) schematic.
- A default image boot up console log.

> **Important**
> Where instructions differ by operating system, you must provide instructions for Windows, Linux, and macOS operating systems.

Follow the Getting Started Guide Template (p. 10) when you write the guide for your board. You can find examples of published guides for other qualified boards in the Amazon FreeRTOS User Guide.

Getting Started Guide Template

Write an overview that provides a brief description of the board. This section should answer the following questions:

- Which hardware is required to run the Amazon FreeRTOS Hello World demo?

  Provide links to pages on your company website for more detail.
- Which IDEs are supported for developing applications for the board?

  Provide links to IDE user guides and download pages.
- Which toolchains and other software utilities are required for development?

  Provide links to user guides and download pages.
- Are there any other prerequisites for getting started with Amazon FreeRTOS on the board?

  Provide links to purchasing pages, user guides, and download pages.

Setting Up Your Hardware

In this section, provide instructions for setting up the platform's hardware. Make sure that you provide links to any user guides or other documentation for setting up hardware.

These instructions include the following:

- Configuring jumper settings.
- Downloading and installing drivers.

  Provide links to download pages and other documentation for supported driver versions.
- Connecting the board to a computer.
- Any other steps required to set up the hardware.

Setting Up the Development Environment

In this section, provide instructions for setting up the platform's supported development environment. Make sure that you provide links to any download pages, user guides, or other documentation for each item.

These instructions include the following:

- Establishing a serial connection.
- Downloading and installing the toolchain.
- Downloading and installing a supported IDE.
- Any other software that is required to develop and debug applications for the device.

Build and Run the Amazon FreeRTOS Demo Project

Build the Amazon FreeRTOS Demo

In this section, provide instructions for building the Amazon FreeRTOS demo code in a supported IDE, or with supported command line tools.

> **Note**
> You must provide instructions for building the demo application on your board with CMake.

Run the Amazon FreeRTOS Demo Project

In this section, provide instructions for flashing and running the Amazon FreeRTOS demo code on your board.

Debugging

In this section, provide instructions for using on-board or external debuggers.

Troubleshooting

In this section, provide troubleshooting tips for resolving common or potential problems.

Creating a CMakeLists.txt File for Your Platform

To qualify a device for Amazon FreeRTOS, you submit a `CMakeLists.txt` file for the device. This file is used to list your device on the Amazon FreeRTOS console, and it makes it possible for developers to build Amazon FreeRTOS code for the device without an IDE.

For more information about the CMake build system, see CMake.org.

Follow the instructions in Creating a List File for Your Platform from the CMakeLists.txt Template (p. 12) to create a CMake list file from the template provided with Amazon FreeRTOS.

Important
Before submitting your CMake list file, you must verify that you can use the file to build the Amazon FreeRTOS test project and the Hello World demo project with CMake.
For instructions, see Building Amazon FreeRTOS with CMake (p. 19).

Prerequisites

Make sure that your host machine meets the following prerequisites before you continue:

- Your device's compilation toolchain must support the machine's operating system. CMake supports all versions of Windows, macOS, and Linux.

  Windows subsystem for Linux (WSL) is not supported. Use native CMake on Windows machines.
- You must have CMake version 3.13 or later installed.

  You can download the binary distribution of CMake from CMake.org.

  Note
  If you download the binary distribution of CMake, make sure that you add the CMake executable to the PATH environment variable before you use CMake from command line.

  You can also download and install CMake using a package manager, like homebrew on macOS, and scoop or chocolatey on Windows.

  Note
  The CMake package versions in the package managers for many Linux distributions are out-of-date. If your distribution's package manager does not include the the latest version of CMake, you can try `linuxbrew` or `nix`.
- You must have a compatible native build system.

  CMake can target many native build systems, including GNU Make or Ninja. Both Make and Ninja can be installed with package managers on Linux, macOS, and Windows. If you are using Make on Windows, you can install a standalone version from Equation, or you can install MinGW, which bundles Make.

  Note
  The Make executable in MinGW is called `mingw32-make.exe`, instead of `make.exe`.

  We recommend that you use Ninja, because it is faster than Make and also provides native support to all desktop operating systems.

Creating a List File for Your Platform from the CMakeLists.txt Template

A `CMakeLists.txt` template file is provided with Amazon FreeRTOS, under *<amazon-freertos>*/`cmake/vendors/`*<vendor>*`/`*<board>*`/CMakeLists.txt`.

The `CMakeLists.txt` template file consists of four sections:

- Amazon FreeRTOS Console Metadata (p. 13)
- Compiler Settings (p. 14)
- Amazon FreeRTOS Portable Layers (p. 15)

- Amazon FreeRTOS Demos and Tests (p. 18)

Follow the instructions to edit these four sections of the list file to match your platform. You can refer to the `CMakeLists.txt` files for other qualified boards under *<amazon-freertos>*`/cmake/vendors` as examples.

Two primary functions are called throughout the file:

`afr_set_board_metadata(`*<name> <value>*`)`

This function defines metadata for the Amazon FreeRTOS console. The function is defined in *<amazon-freertos>*`/cmake/afr_metadata.cmake`.

`afr_mcu_port(`*<module_name>* `[<DEPENDS> [`*targets*`...]])`

This function defines the portable-layer target associated with an Amazon FreeRTOS module (that is, library). It creates a CMake `GLOBAL INTERFACE IMPORTED` target with a name of the form `AFR:`*<module_name>*`::mcu_port`. If `DEPENDS` is used, additional targets are linked with `target_link_libraries`. The function is defined in *<amazon-freertos>*`/cmake/afr_module.cmake`.

Amazon FreeRTOS Console Metadata

The first section of the template file defines the metadata that is used to display a board's information in the Amazon FreeRTOS console. Use the function `afr_set_board_metadata(`*<name> <value>*`)` to define each field listed in the template. This table provides descriptions of each field.

Field Name	Value Description
`ID`	A unique ID for the board.
`DISPLAY_NAME`	The name of the board as you want it displayed on the Amazon FreeRTOS console.
`DESCRIPTION`	A short description of the board for the Amazon FreeRTOS console.
`VENDOR_NAME`	The name of the vendor of the board.
`FAMILY_NAME`	The name of the board's MCU family.
`DATA_RAM_MEMORY`	The size of the board's RAM, followed by abbreviated units. For example, use KB for kilobytes.
`PROGRAM_MEMORY`	The size of the board's program memory, followed by abbreviated units. For example, use "MB" for megabytes.
`CODE_SIGNER`	The code-signing platform used for OTA updates. Use AmazonFreeRTOS-Default for SHA256 hash algorithm and ECDSA encryption algorithm. If you want to use a different code-signing platform, contact us.
`SUPPORTED_IDE`	A semicolon-delimited list of IDs for the IDEs that the board supports.

Field Name	Value Description
IDE_*<ID>*_NAME	The name of the supported IDE. Replace *<ID>* with the ID listed for the IDE in the SUPPORTED_IDE field.
IDE_*<ID>*_COMPILER	A semicolon-delimited list of names of supported compilers for the supported IDE. Replace *<ID>* with the ID listed for the IDE in the SUPPORTED_IDE field.

Compiler Settings

The second section of the template file defines the compiler settings for your board. To create a target that holds the compiler settings, call the `afr_mcu_port` function with `compiler` in place of the *<module_name>* to create an `INTERFACE` target with the name `AFR::compiler::mcu_port`. The kernel publicly links to this `INTERFACE` target so that the compiler settings are transitively populated to all modules.

Use the standard, built-in CMake functions to define the compiler settings in this section of the list file. As you define the compiler settings, follow these best practices:

- Use `target_compile_definitions` to provide compile definitions and macros.
- Use `target_compile_options` to provide compiler flags.
- Use `target_include_directories` to provide include directories.
- Use `target_link_options` to provide linker flags.
- Use `target_link_directories` to provide linker-search directories.
- Use `target_link_libraries` to provide libraries to link against.

Note
If you define the compiler settings somewhere else, you don't need to duplicate the information in this section of the file. Instead, call `afr_mcu_port` with `DEPENDS` to bring in the target definition from another location.
For example:

```
# <your_target> is defined somewhere else. It does not have to be in the same file.
afr_mcu_port(compiler DEPENDS <your_target>)
```

When you call `afr_mcu_port` with `DEPENDS`, it calls `target_link_libraries(AFR::<module_name>::mcu_port INTERFACE <your_targets>)`, which populates the compiler settings for the required `AFR::compiler::mcu_port` target.

Using Multiple Compilers

If your board supports multiple compilers, you can use the `AFR_TOOLCHAIN` variable to dynamically select the compiler settings. This variable is set to the name of the compiler you are using, which should be same as the name of the toolchain file found under *<amazon-freertos>*`/cmake/toolchains`.

For example:

```
if("${AFR_TOOLCHAIN}" STREQUAL "arm-gcc")
```

```
    afr_mcu_port(compiler DEPENDS my_gcc_settings).
elseif("${AFR_TOOLCHAIN}" STREQUAL "arm-iar")
    afr_mcu_port(compiler DEPENDS my_iar_settings).
else()
    message(FATAL_ERROR "Compiler ${AFR_TOOLCHAIN} not supported.")
endif()
```

Advanced Compiler Settings

If you want to set more advanced compiler settings, such as setting compiler flags based on programming language, or changing settings for different release and debug configurations, you can use CMake generator expressions.

For example:

```
set(common_flags "-foo")
set(c_flags "-foo-c")
set(asm_flags "-foo-asm")
target_compile_options(
    my_compiler_settings INTERFACE
    $<$<COMPILE_LANGUAGE:C>:${common_flags} ${c_flags}> # This only have effect on C files.
    $<$<COMPILE_LANGUAGE:ASM>:${common_flags} ${asm_flags}> # This only have effect on ASM
 files.
)
```

CMake generator expressions are not evaluated at the configuration stage, when CMake reads list files. They are evaluated at the generation stage, when CMake finishes reading list files and generates build files for the target build system.

Amazon FreeRTOS Portable Layers

The third section of the template file defines all of the portable layer targets for Amazon FreeRTOS (that is, libraries).

You must use the `afr_mcu_port(`*`<module_name>`*`)` function to define a portable layer target for each Amazon FreeRTOS module that you plan to implement.

You can use any CMake functions you want, as long as the `afr_mcu_port` call creates a target with a name that provides the information required to build the corresponding Amazon FreeRTOS module.

The `afr_mcu_port` function creates a GLOBAL INTERFACE IMPORTED library target with a name of the form `AFR::`*`<module_name>`*`::mcu_port`. As a `GLOBAL` target, it can be referenced in CMake list files. As an `INTERFACE` target, it is not built as a standalone target or library, but compiled into the corresponding Amazon FreeRTOS module. As an `IMPORTED` target, its name includes a namespace (`::`) in the target name (for example, `AFR::kernel::mcu_port`).

Modules without corresponding portable layer targets are disabled by default. If you run CMake to configure Amazon FreeRTOS, without defining any portable layer targets, you should see the following output:

```
Amazon FreeRTOS modules:
  Modules to build:
  Disabled by user:
  Disabled by dependency:  kernel, posix, pkcs11, secure_sockets, mqtt, ...

  Available demos:
  Available tests:
```

As you update the `CMakeLists.txt` file with porting layer targets, the corresponding Amazon FreeRTOS modules are enabled. You should also be able to build any Amazon FreeRTOS module whose dependency requirements are subsequently satisfied. For example, if the MQTT library is enabled, the Device Shadow library is also enabled, because its only dependency is the MQTT library.

Note
The FreeRTOS kernel dependency is a minimum requirement. The CMake configuration fails if the FreeRTOS kernel dependency is not satisfied.

Setting Up the Kernel Porting Target

To create the kernel porting target (`AFR::kernel::mcu_port`), call `afr_mcu_port` with the module name `kernel`. When you call `afr_mcu_port`, specify the targets for the FreeRTOS portable layer and driver code. After you create the target, you can provide the dependency information and the FreeRTOS portable layer and driver code information for the target to use.

Follow the these instructions to set up the kernel porting target.

To set up the kernel porting target

1. Create a target for the driver code.

   For example, you can create a `STATIC` library target for the driver code:

   ```
   add_library(my_board_driver STATIC ${driver_sources})

   # Use your compiler settings
   target_link_libraries(
       my_board_driver
       PRIVATE AFR::compiler::mcu_port
   # Or use your own target if you already have it.
   #   PRIVATE ${compiler_settings_target}
   )

   target_include_directories(
       my_board_driver
       PRIVATE "<include_dirs_for_private_usage>"
       PUBLIC "<include_dirs_for_public_interface>"
   )
   ```

   Or you can create an `INTERFACE` library target for the driver code:

   ```
   # No need to specify compiler settings since kernel target has them.
   add_library(my_board_driver INTERFACE ${driver_sources})
   ```

   Note
   An `INTERFACE` library target does not have build output. If you use an `INTERFACE` library target, the driver code is compiled into the kernel library.

2. Configure the FreeRTOS portable layer:

   ```
   add_library(freertos_port INTERFACE)
   target_sources(
       freertos_port
       INTERFACE
           "${AFR_MODULES_DIR}/FreeRTOS/portable/GCC/ARM_CM4F/port.c"
           "${AFR_MODULES_DIR}/FreeRTOS/portable/GCC/ARM_CM4F/portmacro.h"
           "${AFR_MODULES_DIR}/FreeRTOS/portable/MemMang/heap_4.c"
   )
   target_include_directories(
   ```

```
    freertos_port
    INTERFACE
        "${AFR_MODULES_DIR}/FreeRTOS/portable/GCC/ARM_CM4F"
        "${include_path_to_FreeRTOSConfig_h}
)
```

Note
You can also configure the FreeRTOS portable layer by specifying these source files and their include directories directly in the `AFR::kernel::mcu_port` target.

3. Create the kernel portable layer target:

```
# Bring in driver code and freertos portable layer dependency.
afr_mcu_port(kernel DEPENDS my_board_driver freertos_port)

# If you need to specify additional configurations, use standard CMake functions with
# AFR::kernel::mcu_port as the target name.
target_include_directories(
    AFR::kernel::mcu_port
    INTERFACE
        "${additional_includes}" # e.g. board configuration files
)
target_link_libraries(
    AFR::kernel::mcu_port
    INTERFACE
        "${additional_dependencies}"
)
```

4. To test your list file and configuration, you can write a simple application that uses the FreeRTOS kernel port. For more information about developing and building Amazon FreeRTOS applications with CMake, see Building Amazon FreeRTOS with CMake (p. 19).
5. After you create the demo, add `add_executable` and `target_link_libraries` calls to the list file, and compile the kernel as a static library to verify that the kernel portable layer is correctly configured.

```
add_executable(
    <my_demo>
    main.c
)
target_link_libraries(
    <my_demo>
    PRIVATE AFR::kernel
)
```

Setting Up the Porting Targets for Amazon FreeRTOS Modules

After you add the portable layer target for the kernel, you can add portable layer targets for other Amazon FreeRTOS modules.

For example, to add the portable layer for the Wi-Fi module:

```
afr_mcu_port(wifi)
target_sources(
    AFR::wifi::mcu_port
    INTERFACE "${AFR_MODULES_DIR}/wifi/portable/<vendor>/<board>/aws_wifi.c"
)
```

This example Wi-Fi module portable layer has only one implementation file, which is based on the driver code.

If you want to add the portable layer for the Amazon FreeRTOS Secure Sockets module, the module depends on TLS. This makes its portable layer target slightly more complicated than that of the Wi-Fi module. Amazon FreeRTOS provides a default TLS implementation based on mbedTLS that you can link to:

```
afr_mcu_port(secure_sockets)
target_sources(
    AFR::secure_sockets::mcu_port
    INTERFACE ${portable_layer_sources}
)
target_link_libraries(
    AFR::secure_sockets::mcu_port
    AFR::tls
)
```

In this example code, the standard CMake function `target_link_libraries` states that the Secure Sockets portable layer depends on `AFR::tls`.

You can reference all Amazon FreeRTOS modules by using their target name `AFR::`*`<module_name>`*. For example, you can use the same syntax to also state a dependency on FreeRTOS-Plus-TCP:

```
target_link_libraries(
    AFR::secure_sockets::mcu_port
    AFR::freertos_plus_tcp
    AFR::tls
)
```

Note
If your platform handles TLS by itself, you can use your driver code directly. If you use your driver code directly for TLS, you don't need to call `target_link_libraries`, because all Amazon FreeRTOS modules implicitly depend on the kernel that includes your driver code. Because all non-kernel Amazon FreeRTOS modules implicitly depend on the kernel, their porting layers don't require you to specify the kernel as a dependency. The POSIX module, however, is defined as an optional kernel module. If you want to use POSIX, you must explicitly include it in your kernel portable layer. For example:

```
# By default, AFR::posix target does not expose standard POSIX headers in its
 public
# interface, i.e., You need to use "FreeRTOS_POSIX/pthread.h"  instead of
 "pthread.h".
# Link to AFR::use_posix instead if you need to use those headers directly.
target_link_libraries(
    AFR::kernel::mcu_port
    INTERFACE AFR::use_posix
)
```

Amazon FreeRTOS Demos and Tests

The final section of the template file defines the demo and test targets for Amazon FreeRTOS. CMake targets are created automatically for each demo and test that satisfies the dependency requirements.

In this section, define an executable target with the `add_executable` function. Use `aws_tests` as the target name if you're compiling tests, or `aws_demos` if you're compiling demos. You might need to provide other project settings, such as linker scripts and post-build commands. For example:

```
if(AFR_IS_TESTING)
    set(exe_target aws_tests)
```

```
else()
    set(exe_target aws_demos)
endif()

set(CMAKE_EXECUTABLE_SUFFIX ".elf")
add_executable(${exe_target} "${board_dir}/application_code/main.c")
```

`target_link_libraries` is then called to link available CMake demo or test targets to your executable target.

Note
You still need to modify *<amazon-freertos>*`/demos/common/demo_runner/aws_demo_runner.c` and *<amazon-freertos>*`/tests/common/test_runner/aws_test_runner.c` to enable demos and tests.

Running Post-build Commands

For information about running post-build commands, see add_custom_command. Use the second signature. For example:

```
# This should run an external command "command --arg1 --arg2".
add_custom_command(
    TARGET ${exe_target} POST_BUILD COMMAND "command" "--arg1" "--arg2"
)
```

Note
CMake supports many common, platform-independent operations for creating directories, copying files, and so on. For more information about CMake command-line operations, see the CMake command-line tool reference. You can reference the CMake command-line tool from a CMake list file with the built-in variable `${CMAKE_COMMAND}`.

Building Amazon FreeRTOS with CMake

CMake targets your host operating system as the target system by default. To use CMake for cross compiling, provide a toolchain file that specifies the compiler that you want to use. You can find some examples in *<amazon-freertos>*`/cmake/toolchains`.

If you're using a compiler different from the one provided with Amazon FreeRTOS, write this toolchain file before you build Amazon FreeRTOS with CMake. You must also set the `CMAKE_TOOLCHAIN_FILE` variable before CMake reads your top-level `CMakeLists.txt` file. The `CMAKE_TOOLCHAIN_FILE` variable specifies which compiler to use and sets some CMake variables, like the system name and the default search path. For more information about cross compiling with CMake, see Cross Compiling on the official CMake wiki.

The `CMakeLists.txt` and toolchain files must be in the correct locations. Before you build Amazon FreeRTOS with CMake, make sure that you have correctly set up the Amazon FreeRTOS directory structure. For example:

```
<amazon-freertos>
+ - cmake
|   + - vendors
|   |   + - <vendor>
|   |       + - <board>
|   |           + - CMakeLists.txt
|   + - toolchains
|       + - <toolchain_name>.cmake
+ - demos
+ - lib
```

```
+ - tests
+ - tools
+ - CMakeLists.txt
```

To build a CMake-based project

1. Run CMake to generate the build files for a native build system, like Make or Ninja.

   You can use either the CMake command-line tool or the CMake GUI to generate the build files for your native build system.

   For information about generating Amazon FreeRTOS build files, see Generating Build Files (CMake Command-Line Tool) (p. 20) and Generating Build Files (CMake GUI) (p. 21).
2. Invoke the native build system to make the project into an executable.

   For information about making Amazon FreeRTOS build files, see Building Amazon FreeRTOS from Generated Build Files (p. 23).

Generating Build Files (CMake Command-Line Tool)

You can use the CMake command-line tool (`cmake`) to generate Amazon FreeRTOS build files from the command line or terminal.

To generate the build files, run `cmake`. For the `DVENDOR` option, specify the vendor. For the `DBOARD` option, specify the board. For the `DCOMPILER` option, specify the compiler. Use the `S` option to specify where your source code is. Use the `B` option to specify where to write the generated files.

Note
The compiler must be in the system's `PATH` variable, or you must specify the location of the compiler.

For example, if the vendor is Texas Instruments, and the board is the CC3220 Launchpad, and the compiler is GCC for ARM, you can issue the following command to build the source files from the current directory to a directory named `build`:

```
cmake -DVENDOR=ti -DBOARD=cc3220_launchpad -DCOMPILER=arm-ti -S . -B build
```

Note
If you are using Windows, you must specify the native build system because CMake uses Visual Studio by default. For example:

```
cmake -DVENDOR=ti -DBOARD=cc3220_launchpad -DCOMPILER=arm-ti -S . -B build -G Ninja
```

Or:

```
cmake -DVENDOR=ti -DBOARD=cc3220_launchpad -DCOMPILER=arm-ti -S . -B build -G "Unix
 Makefiles"
```

The regular expressions `${VENDOR}.*` and `${BOARD}.*` are used to search for a matching board, so you don't have to use the full names of the vendor and board for the `VENDOR` and `BOARD` options. Partial names work, provided there is a single match. For example, the following commands generate the same build files from the same source:

```
cmake -DVENDOR=ti -DCOMPILER=arm-ti -S . -B build
```

```
cmake -DBOARD=cc3220 -DCOMPILER=arm-ti -S . -B build
```

```
cmake -DVENDOR=t -DBOARD=cc -DCOMPILER=arm-ti -S . -B build
```

You can use the `CMAKE_TOOLCHAIN_FILE` option if you want to use a toolchain file that is not located in the default directory `cmake/toolchains`. For example:

```
cmake -DBOARD=cc3220 -DCMAKE_TOOLCHAIN_FILE='/path/to/toolchain_file.cmake' -S . -B build
```

If the toolchain file does not use absolute paths for your compiler, and you didn't add your compiler to the `PATH` environment variable, CMake might not be able to find it. To make sure that CMake finds your toolchain file, you can use the `AFR_TOOLCHAIN_PATH` option. This option searches the specified toolchain directory path and the toolchain's subfolder under `bin`.

To enable debugging, set the `CMAKE_BUILD_TYPE` to `debug`. With this option enabled, CMake adds debug flags to the compile options, and builds Amazon FreeRTOS with debug symbols.

```
# Build with debug symbols
cmake -DBOARD=cc3220 -DCOMPILER=arm-ti -DCMAKE_BUILD_TYPE=debug -S . -B build
```

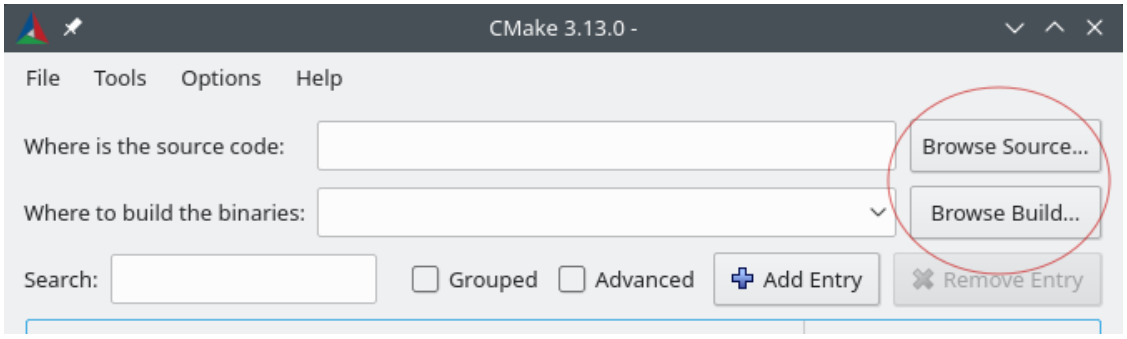
You can also set the `CMAKE_BUILD_TYPE` to `release` to add optimization flags to the compile options.

Generating Build Files (CMake GUI)

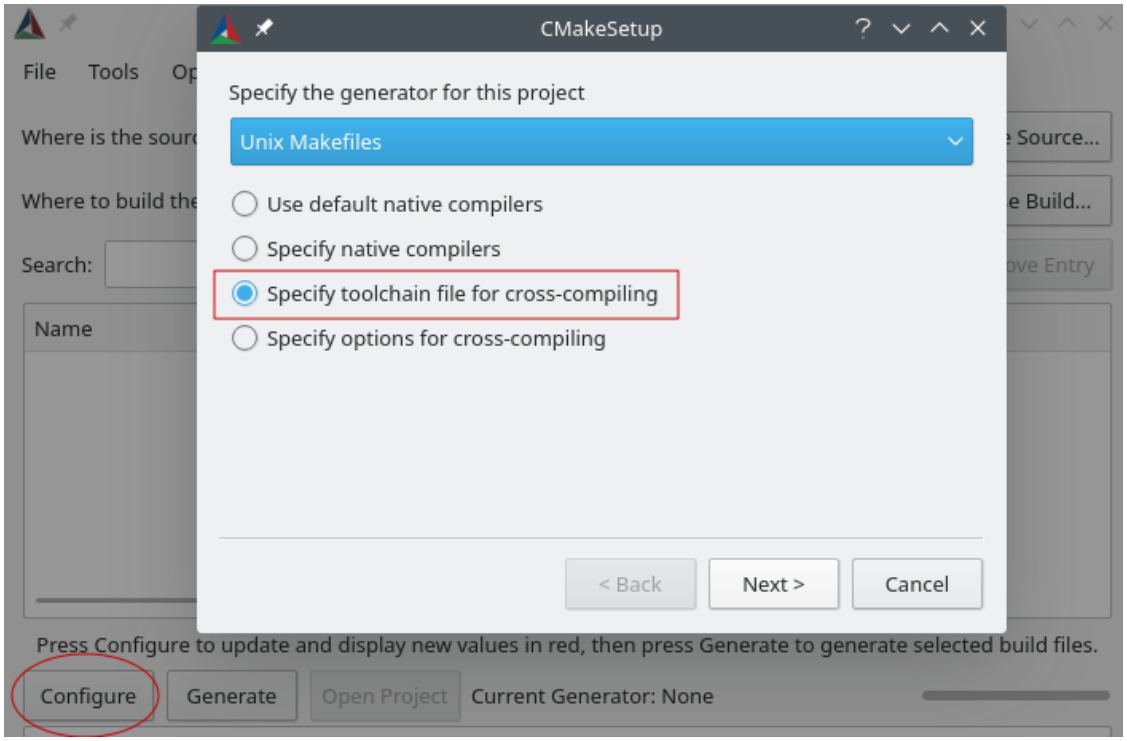
You can use the CMake GUI to generate Amazon FreeRTOS build files.

To generate build files with the CMake GUI

1. From the command line, issue `cmake-gui` to start the GUI.
2. Choose **Browse Source** and specify the source input, and then choose **Browse Build** and specify the build output.

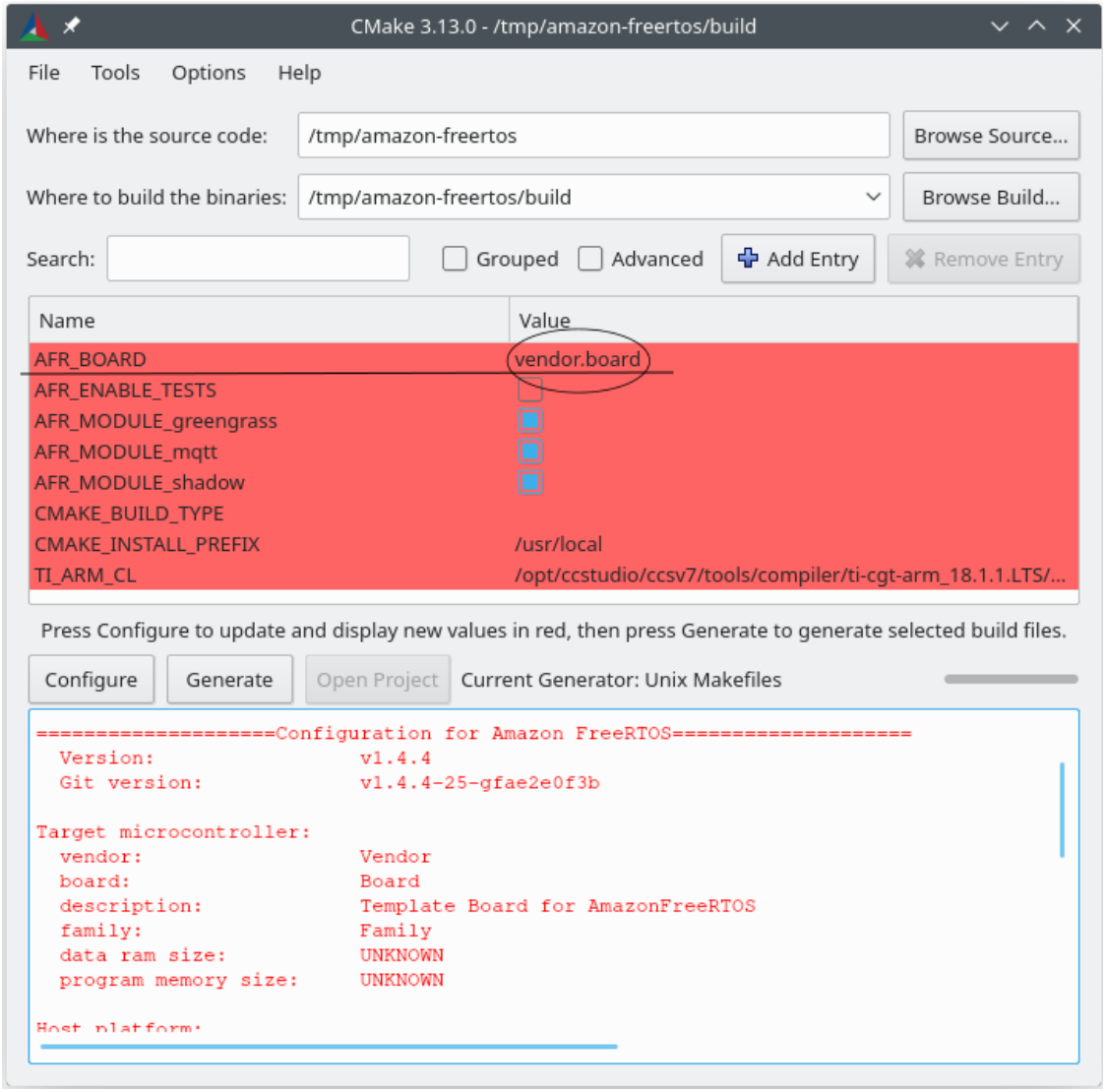


3. Choose **Configure**, and under **Specify the build generator for this project**, find and choose the build system that you want to use to build the generated build files.



4. Choose **Specify toolchain file for cross-compiling**, and then choose **Next**.
5. Choose the toolchain file (for example, `amazon-freertos/cmake/toolchains/arm-ti.cmake`), and then choose **Finish**.

   The default configuration for Amazon FreeRTOS is the template board, which does not provide any portable layer targets. As a result, a window appears with the message Error in configuration process.
6. The GUI should now look like this:



Choose **AFR_BOARD**, choose your board, and then choose **Configure** again.

7. Choose **Generate**. After CMake generates the native build system files, the files should appear in the output binaries directory that you specified in the first step.

Building Amazon FreeRTOS from Generated Build Files

You can build Amazon FreeRTOS with a native build system by calling the build system command from the output binaries directory. For example, if your build file output directory is `build`, and you are using Make as your native build system, run the following commands:

```
cd build
make -j4
```

You can also use the CMake command-line tool to build Amazon FreeRTOS. CMake provides an abstraction layer for calling native build systems. For example:

```
cmake --build <build_dir>
```

Here are some other common uses of the CMake command-line tool's build mode:

```
# Take advantage of CPU cores.
cmake --build <build_dir> --parallel 8
```

```
# Build specific targets.
cmake --build <build_dir> --target afr_kernel
```

```
# Clean first, then build.
cmake --build <build_dir> --clean-first
```

For more information about the CMake build mode, see the CMake documentation.

Hardware Information for Amazon FreeRTOS Qualification

Submit the following information, along with the other items required for qualification:

Company Information

- Full company name.
- Abbreviated company name (if applicable), for the Amazon FreeRTOS console.
- High resolution company logo.
- Brief description of your company.
- URL of your company's landing page.

Board Information

- Full board name.
- Abbreviated board name, for the Amazon FreeRTOS console (under 20 characters).
- High resolution board image.
- Brief description of board.
- URL of board's landing page.
- Brief description of board, for the Amazon FreeRTOS console (under 50 characters).
- Microcontroller family name.
- Board datasheet.
- Compiler options, for optimization.
- Supported IDE, with latest supported version number.
- CLI command to build target executables.
- CLI command to flash target.
- URL to the purchase page.

Providing an Open Source License for Your Code

For qualification, provide an open source license for your ported Amazon FreeRTOS code. To determine which license you want to use, see the License and Standards on the Open Source Initiative website.

Add the open source license to each file that is part of your submission, including the portable files.

Amazon FreeRTOS Qualification Check Script

After you have ported Amazon FreeRTOS, tested your ports, and gathered the other qualification items, run a qualification check script before you submit.

For instructions on running the script, see the script's README file on GitHub.

Note
The script might return warnings for libraries that you did not port. You can ignore these warnings.

Amazon FreeRTOS Qualification Checklist

Use following checklist to help you keep track of qualification items.

- ☐ Port Amazon FreeRTOS.

  Make sure that you have ported and tested the following all of the libraries, according to the instructions in the Amazon FreeRTOS Porting Guide.

  - ☐ Implement `configPRINT_STRING()` macro.

    For instructions, see Implementing the configPRINT_STRING() macro in the *Amazon FreeRTOS Porting Guide*.

  - ☐ Configure FreeRTOS kernel for the target board.

    For instructions, see Configuring a FreeRTOS Kernel Port in the *Amazon FreeRTOS Porting Guide*.

  - ☐ Port Wi-Fi library.

    Note
    This port is not required for qualification if your board does not support Wi-Fi.

    For instructions, see Porting the Wi-Fi Library in the *Amazon FreeRTOS Porting Guide*.

  - ☐ Port TCP/IP stack.

    For instructions, see Porting a TCP/IP Stack in the *Amazon FreeRTOS Porting Guide*.

  - ☐ Port Secure Sockets library.

    For instructions, see Porting the Secure Sockets Library in the *Amazon FreeRTOS Porting Guide*.

  - ☐ Port PKCS #11 library.

    For instructions, see Porting the PKCS #11 Library in the *Amazon FreeRTOS Porting Guide*.

  - ☐ Port TLS library.

    For instructions, see Porting the TLS Library in the *Amazon FreeRTOS Porting Guide*.

  - ☐ Test MQTT library.

    For instructions, see Setting Up the MQTT Library for Testing in the *Amazon FreeRTOS Porting Guide*.

  - ☐ Port OTA library.

    Note
    This port is currently not required for qualification.

    For instructions, see Porting the OTA Library in the *Amazon FreeRTOS Porting Guide*.

  - ☐ Port BLE library.

    Note
    This port is currently not required for qualification.

    For instructions, see Porting the BLE Library in the *Amazon FreeRTOS Porting Guide*.

- ☐ Validate your Amazon FreeRTOS ports with AWS IoT Device Tester.

  For more information, see Using AWS IoT Device Tester for Amazon FreeRTOS in the *Amazon FreeRTOS User Guide*.

- ☐ Set up Hello World demo.

  For instructions, see Setting Up a Hello World Demo (p. 6).
- ☐ Create a Getting Started Guide for your device.

  For instructions, see Creating a Getting Started with Amazon FreeRTOS Guide for Your Device (p. 10).
- ☐ Create a CMake list file, and build the test and demo applications with the file.

  For instructions, see Creating a CMakeLists.txt File for Your Platform (p. 11).
- ☐ Provide hardware information for your device.

  For instructions, see Hardware Information for Amazon FreeRTOS Qualification (p. 24).
- ☐ Create and place an appropriate open source license text file with your code.

  For instructions, see Providing an Open Source License for Your Code (p. 25).
- ☐ Run the Amazon FreeRTOS qualification check script.

  For instructions, see Amazon FreeRTOS Qualification Check Script (p. 25).