FreeRTOS

资格认证指南

FreeRTOS: 资格认证指南

Amazon Web Services 文档中描述的 Amazon Web Services 服务或功能可能因区域而异。要查看适用于中国区域的差异，请参阅中国的 Amazon Web Services 服务入门。

Table of Contents

AmazonFreeRTOS 资格认证计划

什么是 FreeRTOS

FreeRTOS 与世界领先的芯片公司合作开发了 15 年，现在每 175 秒下载一次，是面向微控制器和小型微处理器的市场领先的实时操作系统 (RTOS)。FreeRTOS 根据 MIT 开源许可免费分发，其中包含一个内核和一组持续增加的库，可广泛应用于各个行业领域。FreeRTOS 的设计重点是可靠性和易用性。

FreeRTOS 包含用于连接、安全性和无线 (OTA) 更新的库。FreeRTOS 还包括演示应用程序，展示资格认证主板。

FreeRTOS 是一个开源项目。您可以在 GitHub 网站 (https://github.com/aws/amazon-freertos) 上下载源代码，提供更改或增强功能或报告问题。我们根据 MIT 开源许可发布 FreeRTOS 代码，以便您可以在商业和个人项目中使用它。

我们也欢迎为 FreeRTOS 文档（FreeRTOS 用户指南、FreeRTOS 移植指南, 和FreeRTOS 资格认证指南)。有关文档的 markdown 来源，请访问 https://github.com/awsdocs/aws-freertos-docs。这是根据知识共享 (CC BY-ND) 许可证发布的。

FreeRTOS 内核和组件是单独发布的，并使用语义版本控制。定期发布集成的 FreeRTOS 版本。所有版本使用基于日期的版本控制，格式为 YYYYMM.NN，其中：

- Y 表示年份。
- M 表示月份。
- N 表示指定月份的版本顺序（00 表示第一个版本）。

例如，2021 年 6 月的第二个版本为 2021 年 6 月的 202106.01。

以前，FreeRTOS 版本在主要版本中使用语义版本控制。虽然它已改用基于日期的版本控制（FreeRTOS 1.4.8 更新为 FreeRTOSAmazon参考集成 201906.00），FreeRTOS 内核和每个单独的 FreeRTOS 库仍保留语义版本控制。在语义版本控制中，版本号本身 (X.Y.Z) 表示版本是主要版本、次要版本还是修订版本。您可以使用库的语义版本评估新版本对应用程序的影响以及适用范围。

LTS 版本的维护方式与其他版本类型不同。除了解决缺陷以外，还经常使用新功能更新主要版本和次要版本。LTS 版本仅使用解决严重缺陷和安全漏洞的更改进行更新。在发布后，不会在给定 LTS 版本中引入新功能。它们在发布后至少保留三个日历年，并为设备制造商提供使用稳定基准的选项，而不是使用主要和次要版本表示的更动态的基准。

什么是Amazon资格认证计划？

这些区域有：AmazonFreeRTOS 的设备资格认证计划验证移植到基于微控制器的主板的预集成 FreeRTOS 项目，使开发人员确信 FreeRTOS 移植的行为正确且与Amazon IoT。

亚马逊合作伙伴网络中的用户可以使用Amazon设备资格认证计划，使微控制器 (MCU) 开发主板正式取得资 FreeRTOS 认证。

合格主板可在 Amazon Partner Device Catalog 上列出。

要使设备取得资格认证，必须将 FreeRTOS 移植到设备，然后按照Amazon设备认证计划步骤。有关更多信息，请参阅Amazon设备资格认证计划页和 Amazon设备资格认证计划指南。

有关设备如何取得资格认证的信息，请参阅设备资格认证 (p. 4)。

资格认证常见问题

Q: 我能否让没有 Wi-Fi 或以太网的 MCU 取得资格认证？

A: 是。有一些合格的 MCU 使用外部 Wi-Fi 模块，并将各种功能转移到 Wi-Fi 模块，包括 TCP/IP 和 TLS。其中一个例子是使用 Inventek Wi-Fi 模块的 STM32L4 探索套件。按照设备资格认证计划提交流程，让我们知道我们如何帮助您完成工作。

Q: 如果 FreeRTOS 版本在我开始移植之前的版本之后发布，我是否需要使用最新版本重新来过？

A: 应始终移植 FreeRTOS 的最新版本。如果在您进行移植的过程中，我们发布 FreeRTOS 的新版本，那么您仍然可以使用之前的版本。

Q: 我的主板使用的内核架构我曾经修改过，并且不属于 FreeRTOS 官方版本。我是否仍符合资格？

A: 很遗憾，我们只接受官方内核端口。这些端口可以从GitHub。如果您有不受支持的架构或其他功能要添加到现有内核端口，请联系您当地的 APN 代表。

Q: 如果我想要将设备目录中列出的端口更新为 FreeRTOS 的新版本，是否需要重新取得资格？

A: 更新端口后，运行Amazon IoT再次检查设备测试器并检查FreeRTOS 资格审查清单 (p. 21)以查看是否有任何项目受到影响（尤其是《入门指南》）。提交一个新的设备资格认证计划服务单，并附上您的传递日志副本，以便更新 Device Catalog 列表以指向您的新端口。

Q: 我的设备不支持 Wi-Fi。FreeRTOS Wi-Fi 库是否需要移植 FreeRTOS？

A: 主要的要求是设备可以连接到 Amazon 云。如果设备可以连接到Amazon云通过安全的以太网连接，Wi-Fi 库不是必需的。

Q: 我的设备不支持低功耗蓝牙或无线 (OTA) 更新。资格认证是否需要移植这些 FreeRTOS 库？

A: 对于资格认证，低功耗蓝牙和 OTA 移植是可选的。

Q: 我的主板上没有片上 TCP/IP 功能。FreeRTOS 资格认证是否需要特定的 TCP/IP 堆栈？

A: 如果主板不具备片上 TCP/IP 功能，那么可以使用 FreeRTOS+TCP TCP/IP 堆栈或 LWIP TCP/IP 堆栈的最新版本来通过 TCP/IP 资格认证要求。有关 FreeRTOS 支持的最新版本的 lwIP，请参阅 GitHub 网站上的 changelog.md 文件。有关更多信息，请参阅 。移植 TCP/IP 堆栈中的FreeRTOS 移植指南。

Q: 资格认证是否需要特定的 TLS 堆栈？

A: FreeRTOS 支持 mbedTLS 和片外 TLS 实施，如在某些网络处理器上找到的实施。无论设备的 FreeRTOS 端口使用哪个 TLS 实施，该移植都必须通过 TLS 的 Device Tester 验证测试。有关更多信息，请参阅 。移植 TLS 库中的FreeRTOS 移植指南。

Q: 我的设备是否需要将所有Amazon IoT资格认证测试是否需要移植设备测试？ 是否有未通过所有测试而符合条件的方法？

A: 设备必须通过所有必需的验证测试才能取得 FreeRTOS 资格认证。仅有的例外是对于 Wi-Fi、低功耗蓝牙和 OTA。

Q: 我的设备仅使用其中一个协议（HTTP、MQTT），并且仅使用其中一个可用通信通道（Wi-Fi、以太网、BLE）。如果仅使用一个协议通信通道组合便通过所有与 OTA 相关的 IDT 测试，那么我的设备是否将在设备目录中被列为符合 OTA 要求的设备？

A: 是。如果可能，我们鼓励您在设备上获得其他合格的组合。这样一来，您便能为更多的客户使用案例提供支持。

Q: 根据资格要求，我们将在我们自己的回购中托管我们的 FreerTOS 端口。我们应该在回购中包括什么文件夹和演示来支持？

A: 托管所有必要的文件和文件夹，使端口成为从存储库下载端口的客户的开箱即用体验。你应该包含你的整个`freertos_kernel`、`libraries`, 和`tools`文件夹以及`docs`文件夹中，`projects`文件夹，以及`vendors`文件夹为您的供应商特定文件。还包括您的整个演示文件夹。

Note

必须支持 CoremQtt 相互身份验证演示。其他演示由您自行决定。另外，`tools`文件夹不是必需的。但是，我们建议您托管此文件夹，以帮助客户进行测试。

如果您对资格认证的疑问在本页面或其余部分中都找不到答案FreeRTOS 资格认证指南，请联系您的Amazon代表或FreeRTOS 工程团队。

文档历史记录

请参阅FreeRTOS 移植和资格认证文档的修订历史记录at将 FreeRTOS 移植到您的 IoT 设备中的FreeRTOS移植指南。

设备资格认证

使设备取得资格认证

1. 将 FreeRTOS 库移植到设备。

   Note

   目前，FFreeRTOS OTA 和低功耗蓝牙库的移植并不是资格认证所必需的。
   如果设备不支持 Wi-Fi，则可改用以太网连接来连接到 Amazon 云。移植 FreeRTOS Wi-Fi 库并不是必需的。

   有关将 FreeRTOS 移植到设备的说明，请参阅FreeRTOS 移植指南。
2. 使用验证移植Amazon IoTFreeRTOS 的 Device Tester。

   在使用 Device Tester 验证移植资格时，必须在 `device.json` 配置文件的 `features` 属性中指定以下移植的有关信息：

   - 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"
     }
     ```

   设备测试程序使用此信息确定根据移植的 FreeRTOS 代码要运行哪些测试。默认情况下，Device Tester 运行其他所有必需的库移植测试。

   有关的信息Amazon IoTFreeRTOS 的 Device Tester，请参阅。使用Amazon IoTFreeRTOS 的 Device Tester（在 FreeRTOS 用户指南中）。
3. 创建以下内容以便提交资格认证：

   - Hello World 演示应用程序，该应用程序通过 MQTT 将消息从设备发布到 Amazon 云。

有关信息，请参阅 设置 Hello World 演示 (p. 5)。

- 适用于的 Device FreeRTOS 入门指南。

  有关信息，请参阅 为设备创建入 FreeRTOS 指南 (p. 7)。
- A`CMakeLists.txt`文件，为设备构建 FreeRTOS 应用程序。

  Note

  通过 Amazon 设备资格计划对主板进行资格认证不需要 CMake 列表文件。仅在 FreeRTOS 控制台上列出设备时需要该文件。使用 CMake 为您的平台构建项目文件同样需要该文件。

  有关信息，请参阅 为平台创建 CMakeLists.txt 文件 (p. 9)。
- 硬件平台的详细信息列表。

  有关信息，请参阅 FreeRTOS 资格审查清单 (p. 21)。
- 适用于设备 FreeRTOS 端口的开源许可文件。

  有关信息，请参阅 为代码提供开源许可 (p. 20)。
- （针对符合 OTA 更新要求的主板）有关代码签名的说明。

  有关示例，请参阅。创建代码签名证书（在 FreeRTOS 用户指南中）。
- （针对符合 OTA 更新要求且使用自定义引导加载程序的主板）有关自定义引导加载程序应用程序的信息和说明。

  有关要求列表，请参阅。移植启动加载程序演示（在 FreeRTOS 移植指南中）。

要在 FreerTOS 控制台上列出设备，要使设备的代码位于 GitHub 上，要使设备接收入门文档支持，这些项都是必需的。

4. 提交您的合格图板，以便在Amazon合作伙伴设备目录通过设备列表门户在 APN 合作伙伴平台上。所有提交中均必须包含 Amazon IoT Device Tester 测试结果文件，指示您通过了所有必需的测试案例。您必须首先注册成为 APN 合作伙伴才能提交主板以列出。

可以使用FreeRTOS 资格审查清单 (p. 21)来跟踪资格认证所需的步骤列表。

设置 Hello World 演示

要取得资 FreeRTOS 认证，可设置 Hello World 演示应用程序以运行于取得认证的设备上。该演示通过 MQTT 将消息从设备发布到 Amazon 云。

设置 Hello World 演示

1. 按照中的说明进行操作配置 FreeRTOS 下载用于演示 (p. 5)配置 FreeRTOS 下载的目录结构以适合设备。
2. 按照创建演示项目 (p. 6)中的说明，在 IDE 中创建演示项目。

设置完演示之后，为设备创建入门指南。该指南引导用户设置设备以运行 Hello World 演示。

配置 FreeRTOS 下载用于演示

在下载的根目录 (*`freertos`*) 中，`vendors` 文件夹的结构如下所示：

```
vendors
```

```
+ - vendor    (Template, to be renamed to the name of the MCU vendor)
    + - boards
    |   + - board    (Template, to be renamed to the name of the development board)
    |       + - aws_demos
    |       + - aws_tests
    |       + - CMakeLists.txt
    |       + - ports
    + - driver_library    (Template, to be renamed to the library name)
        + - driver_library_version    (Template, to be renamed to the library version)
```

这些区域有：*vendor*和*board*文件夹是模板文件夹，我们提供这些文件夹是为了简化创建演示和测试项目的过程。其目录结构可确保所有演示和测试项目具有一致的组织结构。

aws_demos 文件夹具有以下结构：

```
vendors/vendor/boards/board/aws_demos
+ - application_code    (Contains main.c, which contains main())
|   + - vendor_code    (Contains vendor-supplied, board-specific files)
|   + - main.c    (Contains main())
+ - config_files    (Contains FreeRTOS config files)
```

配置演示项目文件

将演示应用程序的 `main.c` 和 `main.h` 文件复制到 `application_code` 文件夹。您可以重复使用`main.c`from的`aws_tests`project你用来测试你的端口。

1. 将任何必需的供应商提供的特定于主板的库保存到 *vendor*_code 文件夹。

   Important

   不要将供应商提供的目标主板 MCU 系列通用库保存到 `aws_tests` 或 `aws_demos` 的任何子目录中。
2. 将 *vendor*_code 文件夹中的 *vendor* 替换为供应商的名称。
3. 将 *board* 文件夹重命名为开发主板的名称。

配置完演示项目文件之后，可以在 IDE 中创建项目。有关说明，请参阅创建演示项目 (p. 6)。

如果您在创建 CMake 列表文件 (p. 9)，请确保为演示项目提供了 CMakelist 条目。

创建演示项目

配置 FreeRTOS 下载之后，可以创建一个具有 Hello World 演示所需项目结构的 IDE 项目。

按照以下说明创建一个具有演示应用程序所需的 IDE 项目结构的 IDE 项目。

Important

如果使用的是基于 Eclipse 的 IDE，请勿将项目配置为在任何文件夹中构建所有文件。相反，通过分别链接到每个源文件来将源文件添加到项目。

1. 创建一个名为 `aws_demos` 的项目，并将该项目保存到 `projects/`*vendor*`/`*board*`/`*ide* 目录中。
2. 在 IDE 中，在 `aws_demos` 下创建两个虚拟文件夹：

   - `application_code`
   - `config_files`

   在 `aws_demos` 下，IDE 项目中现在应该有两个虚拟子目录：`application_code` 和 `config_files`。

Note

Eclipse 会生成一个额外的 `includes` 文件夹。此文件夹不属于所需的结构。

3. 将 `aws_demos/application_code` 及其子目录下的所有文件夹和文件导入到 IDE 中的 `application_code` 虚拟文件夹。
4. 将 `aws_demos/config_files` 中的所有文件导入到 IDE 中的 `config_files` 虚拟文件夹。
5. 将以下目录中的所有文件夹和文件导入到 IDE 中的 `application_code` 虚拟文件夹。

   - *`freertos`*`/demos/demo_runner`
   - *`freertos`*`/demos/coreMQTT`
   - *`freertos`*`/libraries/.../provisioning/src`
6. 在 IDE 中，将 *`freertos`*`/demos/include` 目录及其内容导入到 `application_code` 虚拟文件夹中。
7. 将以下目录及其内容导入 `aws_demos` IDE 项目中：

   Note

   仅导入适用于您的平台和移植的文件及目录。

   - *`freertos`*`/libraries`
   - *`freertos`*`/freertos_kernel`
   - *`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/`*`driver_library`*`/`*`driver_library_version`*
8. 打开项目的 IDE 属性，并将以下路径添加到编译器的包含路径：

   - *`freertos`*`/demos/include`
   - *`freertos`*`/freertos_kernel/portable/`*`compiler`*`/`*`architecture`*
   - *`freertos`*`/libraries/3rdparty/mbedtls/include`
   - *`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files`
   - 供应商提供的驱动程序库所需的任何路径。

为设备创建入 FreeRTOS 指南

要取得资格认证，必须为设备创建 FreerTOS 入门指南。本指南引导用户设置硬件和开发环境，以便为 FFreeRTOS 设备开发应用程序，并在设备上构建、运行和闪存 Freertos Hello World 演示。

本指南必须在面向公众的网站提供给客户。在 Amazon 合作伙伴设备目录中列出的合格主板必须提供本指南的 URL。

该指南必须包含以下说明：

- 设置设备硬件。
- 设置开发环境。
- 构建并运行演示项目。
- 调试。
- 故障排除。

我们还建议该指南中包括：

- 指向 MCU 数据表的链接。
- 印刷电路板 (PCB) 原理图。
- 默认映像启动控制台日志。

Important

如果说明因操作系统而异，则必须针对 Windows、Linux 和 macOS 操作系统提供说明。

在为主板编写指南时，可遵照入门指南模板 (p. 8)。可以在FreeRTOS 用户指南。

入门指南模板

编写概述，提供对主板的简要说明。本部分应回答以下问题：

- 运行 FreeRTOS Hello World 演示需要什么硬件？

  提供指向公司网站页面的链接，以了解更多详细信息。
- 为主板开发应用程序时，支持哪些 IDE？

  提供指向 IDE 用户指南和下载页面的链接。
- 开发需要用到哪些工具链和其他软件实用程序？

  提供指向用户指南和下载页面的链接。
- 在主板上开始使用 FreeRTOS 是否要具备其他任何先决条件？

  提供指向购买页面、用户指南和下载页面的链接。

设置硬件

本部分阐述如何设置平台的硬件。请确保提供指向用于设置硬件的任何用户指南或其他文档的链接。

这些说明包括以下内容：

- 配置跳线设置。
- 下载并安装驱动程序。

  提供指向支持的驱动程序版本的下载页面和其他文档的链接。
- 将主板连接到计算机。
- 设置硬件所需的其他任何步骤。

设置开发环境

本部分阐述如何设置平台支持的开发环境。请确保提供指向每一项的任何下载页面、用户指南或其他文档的链接。

这些说明包括以下内容：

- 建立串行连接。
- 下载并安装工具链。
- 下载并安装支持的 IDE。
- 为设备开发和调试应用程序所需的其他任何软件。

构建并运行 FreeRTOS 演示项目

构建 FreeRTOS 演示

本部分阐述如何在支持的 IDE 中构建 FreeRTOS 演示代码，或使用支持的命令行工具进行构建。

Note

必须提供有关在主板上使用 CMake 构建演示应用程序的说明。

运行 FreeRTOS 演示项目

本部分阐述如何在主板上闪存并运行 FFreeRTOS 演示代码。

Debugging

本部分阐述如何使用板载调试程序或外部调试程序。

Troubleshooting

本部分阐述用于解决常见问题或潜在问题的故障排除技巧。

为平台创建 CMakeLists.txt 文件

A`CMakeLists.txt`文件用于在 FreeRTOS 控制台上列出设备，并使开发人员可以在没有 IDE 的情况下为设备构建 FreerTOS 代码。

Note

通过 Amazon 设备资格计划对主板进行资格认证不需要 CMake 列表文件。仅在 FreeRTOS 控制台上列出设备时需要该文件。

有关 CMake 构建系统的更多信息，请参阅 CMake.org。

按照中的说明进行操作根据 CMakeLists.txt 模板为平台创建列表文件 (p. 10)，根据 FreeRTOS 提供的模板创建 CMake 列表文件。

Important

提交 CMake 列表文件之前，必须验证该文件可用于使用 CMake 构建 FreeRTOS 测试项目和 Hello World 演示项目。
有关说明，请参阅使用 CMake 构建 FreeRTOS (p. 16)。

Prerequisites

请先确保主机符合以下先决条件，再继续下一步：

- 设备的编译工具链必须支持计算机的操作系统。CMake 支持所有版本的 Windows、macOS 和 Linux。

  不支持 Windows Subsystem for Linux (WSL)。在 Windows 计算机上使用本机 CMake。
- 必须安装了 CMake 3.13 版或更高版本。

  可以从 CMake.org 下载 CMake 的二进制发行版。

  Note

  如果下载 CMake 的二进制发行版，请确保先将 CMake 可执行文件添加到 PATH 环境变量，然后再从命令行使用 CMake。

  也可以使用程序包管理器下载并安装 CMake，例如，macOS 上的 homebrew、Windows 上的 scoop 或 chocolatey。

  Note

  许多 Linux 发行版的程序包管理器中的 CMake 程序包版本已过时。如果发布版本的程序包管理器不包括最新版本的 CMake，那么可以尝试`linuxbrew`或者`nix`。

- 必须具有兼容的本机构建系统。

  CMake 可以针对许多本机构建系统，包括GNU Make或者忍者。Make 和 Ninja 都可以使用程序包管理器安装在 Linux、macOS 和 Windows 上。如果在 Windows 上使用 Make，则可从 Equation 安装独立版本，或安装捆绑了 Make 的 MinGW。

  Note

  MinGW 中的 Make 可执行文件名为 `mingw32-make.exe`，而不是 `make.exe`。

  我们建议使用 Ninja，因为它不仅速度快于 Make，还可提供对所有桌面操作系统的本机支持。

根据 CMakeLists.txt 模板为平台创建列表文件

A`CMakeLists.txt`模板文件是随 FreeRTOS 提供的，它位于*`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/CMakeLists.txt`。

`CMakeLists.txt` 模板文件由以下四个部分组成：

- FreeRTOS 控制台元数据 (p. 10)
- 编译器设置 (p. 11)
- FreeRTOS 可移植层 (p. 12)
- FreeRTOS 演示和测试 (p. 15)

可以按照说明编辑列表文件的这四个部分，以与平台相匹配。您可以参考 *`freertos`*`/vendors` 中其他符合条件的供应商主板的 `CMakeLists.txt` 文件以作为示例。

整个文件中会调用以下两个主要函数：

`afr_set_board_metadata(`*`name value`*`)`

此函数为 FreeRTOS 控制台定义元数据。该函数是在 *`freertos`*`/tools/cmake/afr_metadata.cmake` 中定义的。

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

此函数定义与 FreeRTOS 模块（即库）关联的可移植层目标。此函数采用以下名称形式创建 CMake `GLOBAL INTERFACE IMPORTED` 目标：`AFR:`*`module_name`*`::mcu_port`。如果使用 `DEPENDS`，则其他目标通过 `target_link_libraries` 链接在一起。该函数是在 *`freertos`*`/tools/cmake/afr_module.cmake` 中定义的。

FreeRTOS 控制台元数据

模板文件的第一部分定义用于在 FreeRTOS 控制台中显示主板信息的元数据。使用函数 `afr_set_board_metadata(`*`name value`*`)` 定义模板中列出的每个字段。下表提供了每个字段的说明。

字段名称	值说明
`ID`	主板的唯一 ID。
`DISPLAY_NAME`	要显示在 FreeRTOS 控制台上的主板的名称。
`DESCRIPTION`	FreeRTOS 控制台的主板的简短说明。
`VENDOR_NAME`	主板供应商的名称。

字段名称	值说明
FAMILY_NAME	主板 MCU 系列的名称。
DATA_RAM_MEMORY	主板 RAM 的大小，后跟单位缩写。例如，使用 KB 表示千字节。
PROGRAM_MEMORY	主板程序内存的大小，后跟单位缩写。例如，使用 MB 表示兆字节。
CODE_SIGNER	用于 OTA 更新的代码签名平台。使用 AmazonFreeRTOS-Default 作为 SHA256 散列算法和 ECDSA 加密算法。如果要使用不同的代码签名平台，请联系我们。
SUPPORTED_IDE	主板支持的 IDE 的 ID 列表，用分号分隔。
IDE_*ID*_NAME	支持的 IDE 的名称。Replace*ID*中列出了 IDE 的 IDSUPPORTED_IDE字段。
IDE_*ID*_COMPILER	对于所支持的 IDE，支持的编译器的名称列表，用分号分隔。Replace*ID*中列出了 IDE 的 IDSUPPORTED_IDE字段。
KEY_IMPORT_PROVISIONING	如果电路板演示项目从预配置的aws_clientcredential_keys.h头文件；在这种情况下，快速 Connect将在 FreeRTOS 控制台中启用。 如果预期的主板配置机制是 JITR/JITP 或多账户注册，则设置为 FALSE；在这种情况下，快速 Connect将在 FreeRTOS 控制台中禁用。

编译器设置

模板文件的第二部分为主板定义编译器设置。要创建具有编译器设置的目标，可调用 afr_mcu_port 函数，用 compiler 代替 *module_name*，以创建名为 AFR::compiler::mcu_port 的 INTERFACE 目标。内核会公开链接到此 INTERFACE 目标，以便编译器设置能以传递的方式填充到所有模块。

在列表文件的此部分中，使用标准的内置 CMake 函数定义编译器设置。在定义编译器设置时，请遵循以下最佳实践：

- 使用 target_compile_definitions 提供编译定义和宏。
- 使用 target_compile_options 提供编译器标记。
- 使用 target_include_directories 提供包含目录。
- 使用 target_link_options 提供链接器标记。
- 使用 target_link_directories 提供链接器搜索目录。
- 使用 target_link_libraries 提供链接的库。

> **Note**
>
> 如果在其他位置定义编译器设置，则不需要重复文件此部分中的信息。相反，使用 DEPENDS 调用 afr_mcu_port，可从其他位置引入目标定义。
> 例如：

```
# your_target is defined somewhere else. It does not have to be in the same file.
```

```
afr_mcu_port(compiler DEPENDS your_target)
```

使用 DEPENDS 调用 afr_mcu_port 时，它会调用 target_link_libraries(AFR::*module_name*::mcu_port INTERFACE *your_targets*)，这会填充所需 AFR::compiler::mcu_port 目标的编译器设置。

使用多个编译器

如果主板支持多个编译器，则可使用 AFR_TOOLCHAIN 变量动态选择编译器设置。该变量设置为使用的编译器的名称，该名称应该与位于 *freertos*/tools/cmake/toolchains 中的工具链文件的名称相同。

例如：

```
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()
```

高级编译器设置

如果要设置更高级的编译器设置，如设置基于编程语言的编译器标记，或更改不同版本的设置及调试配置，则可使用 CMake 生成器表达式。

例如：

```
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 在读取列表文件时，不会在配置阶段评估 CMake 生成器表达式。它们会在生成阶段进行评估，当 CMake 完成列表文件的读取，并为目标构建系统生成构建文件时。

FreeRTOS 可移植层

模板文件的第三部分为 FreeRTOS（即库）定义所有可移植层目标。

您必须使用afr_mcu_port(*module_name*)函数为计划实施的每个 FreeRTOS 模块定义可移植层目标。

只要使用任何所需的 CMake 函数，即可使用任何所需的 CMake 函数，只要afr_mcu_port调用所创建目标的名称提供了构建相应 FreeRTOS 模块所需的信息。

afr_mcu_port 函数采用以下名称形式创建 GLOBAL INTERFACE 库目标：AFR::*module_name*::mcu_port。作为 GLOBAL 目标时，可以在 CMake 列表文件中进行引用。作为INTERFACE目标时，不能构建为独立的目标或库，但可以编译到相应的 FreeRTOS 模块。作为 IMPORTED 目标时，其名称在目标名称中包含命名空间 (::)，例如，AFR::kernel::mcu_port。

默认情况下，无相应可移植层目标的模块处于禁用状态。如果运行 CMake 配置 FreeRTOS，而未定义任何可移植层目标，那么应该看到以下输出：

```
FreeRTOS modules:
  Modules to build:
```

```
Disabled by user:
Disabled by dependency:  kernel, posix, pkcs11, secure_sockets, mqtt, ...

Available demos:
Available tests:
```

在更新CMakeLists.txt文件，将启用相应的 FreeRTOS 模块。您还应该能构建其依赖性要求随后得以满足的任何 FreeRTOS 模块。例如，如果已启用 CoremQtt 库，则 Device Shadow 库也会启用，因为它唯一的依赖项是 CoremQtt 库。

Note

FreeRTOS 内核依赖性是最低要求。如果 FreeRTOS 内核依赖性未满足，CMake 配置将失败。

设置内核移植目标

要创建内核移植目标 AFR::kernel::mcu_port，可使用模块名称 kernel 调用 afr_mcu_port。调用 afr_mcu_port 时，为 FreeRTOS 可移植层和驱动程序代码指定目标。创建目标后，可以提供依赖性信息以及 FreeRTOS 可移植层和驱动程序代码信息，供目标使用。

按照说明设置内核移植目标。

设置内核移植目标

1. 为驱动程序代码创建目标。

   例如，可以为驱动程序代码创建 STATIC 库目标：

   ```
   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"
   )
   ```

   或者，可以为驱动程序代码创建 INTERFACE 库目标：

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

   Note

   INTERFACE 库目标无构建输出。如果使用的是 INTERFACE 库目标，则驱动程序代码将编译到内核库。

2. 配置 FreeRTOS 可移植层：

   ```
   add_library(freertos_port INTERFACE)
   target_sources(
       freertos_port
       INTERFACE
           "${AFR_MODULES_DIR}/freertos_kernel/portable/GCC/ARM_CM4F/port.c"
   ```

```
        "${AFR_MODULES_DIR}/freertos_kernel/portable/GCC/ARM_CM4F/portmacro.h"
        "${AFR_MODULES_DIR}/freertos_kernel/portable/MemMang/heap_4.c"
)
target_include_directories(
    freertos_port
    INTERFACE
        "${AFR_MODULES_DIR}/freertos_kernel/portable/GCC/ARM_CM4F"
        "${include_path_to_FreeRTOSConfig_h}
)
```

Note

要配置 FreeRTOS 可移植层，也可以在 `AFR::kernel::mcu_port` 目标中直接指定这些源文件及其包含目录。

3. 创建内核可移植层目标：

```
# 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. 要测试列表文件和配置，可以编写一个简单的应用程序以使用 FreeRTOS 内核移植。有关使用 CMake 开发和构建 FreeRTOS 应用程序的更多信息，请参阅使用 CMake 构建 FreeRTOS (p. 16)。
5. 创建演示之后，将 `add_executable` 和 `target_link_libraries` 调用添加到列表文件，并将内核编译为静态库，以检验内核可移植层的配置是否正确。

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

设置 FreeRTOS 模块的移植目标

为内核添加可移植层目标之后，可以为其他 FreeRTOS 模块添加可移植层目标。

例如，为 Wi-Fi 模块添加可移植层：

```
afr_mcu_port(wifi)
target_sources(
    AFR::wifi::mcu_port
    INTERFACE "${AFR_MODULES_DIR}/vendors/vendor/boards/board/ports/wifi/iot_wifi.c"
)
```

此 Wi-Fi 模块可移植层示例只有一个基于驱动程序代码的实施文件。

如果要为 FreeRTOS 安全套接字模块添加可移植层，则该模块依赖于 TLS。这会使其可移植层目标较 Wi-Fi 模块的可移植层目标略微复杂。FreeRTOS 提供了基于 mbedTLS 的默认 TLS 实施，您可以与之链接：

```
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
)
```

在此示例代码中，标准 CMake 函数 `target_link_libraries` 声明安全套接字可移植层依赖于 `AFR::tls`。

可以通过使用目标名称来引用所有 FreeRTOS 模块`AFR::`*`module_name`*。例如，可以使用相同的语法声明对 FreeRTOS-Plus-TCP 的依赖性：

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

Note

如果平台自己处理 TLS，则可直接使用驱动程序代码。如果将驱动程序代码直接用于 TLS，则无需调用`target_link_libraries`，因为所有 FreeRTOS 模块隐式依赖于包含驱动程序代码的内核。
由于所有非内核 FreeRTOS 模块隐式依赖于内核，因此其移植层不需要将内核指定为依赖项。不过，POSIX 模块定义为可选的内核模块。如果要使用 POSIX，则必须将其显式包括在内核可移植层中。例如：

```
# By default, AFR::posix target does not expose standard POSIX headers in its
 public
# interface, i.e., You need to use "freertos_plus_posix/source/
FreeRTOS_POSIX_pthread.c"  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
)
```

FreeRTOS 演示和测试

模板文件的最后一个部分定义 FreeRTOS 的演示和测试目标。对于满足依赖性要求的每个演示和测试，会自动创建 CMake 目标。

在本节中，使用 `add_executable` 函数定义可执行目标。如果要编译测试，则使用 `aws_tests` 作为目标名称。如果要编译演示，则使用 `aws_demos`。可能还需要提供其他项目设置，如链接器脚本和构建后命令。例如：

```
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")
```

随后调用 `target_link_libraries`，将可用的 CMake 演示或测试目标链接到可执行目标。

Note

仍需要修改 `aws_demos/config_files/aws_demo_config.h` 和 `aws_tests/config_files/aws_test_runner_config.h`，以启用演示和测试。

运行构建后命令

有关运行构建后命令的信息，请参阅 add_custom_command。使用第二个签名。例如：

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

Note

CMake 支持许多常见的、独立于平台的操作，如创建目录、复制文件等。有关 CMake 命令行操作的更多信息，请参阅 CMake 命令行工具参考。可以使用内置变量 `${CMAKE_COMMAND}`，从 CMake 列表文件引用 CMake 命令行工具。

使用 CMake 构建 FreeRTOS

默认情况下，CMake 将主机操作系统作为目标系统。要将 CMake 用于交叉编译，请提供工具链文件以指定要使用的编译器。您可以在 *freertos*`/tools/cmake/toolchains` 中找到一些示例。

如果使用的编译器不同于随 FreeRTOS 提供的编译器，则先编写此工具链文件，然后再使用 CMake 构建 FreerTOS。在 CMake 读取顶级 `CMakeLists.txt` 文件之前，还必须设置 `CMAKE_TOOLCHAIN_FILE` 变量。`CMAKE_TOOLCHAIN_FILE` 变量指定要使用的编译器，并设置某些 CMake 变量，如系统名称和默认搜索路径。有关使用 CMake 交叉编译的更多信息，请参阅正式 CMake Wiki 中的交叉编译。

`CMakeLists.txt` 和工具链文件必须位于正确的位置。使用 CMake 构建 FreeRTOS 之前，请确保已在本地计算机上设置 FreerTOS 目录结构，以便与GitHub。有关说明，请参阅 README.md 文件。

构建基于 CMake 的项目

1. 运行 CMake 为本机构建系统生成构建文件，如 Make 或 Ninja。

   可以使用 CMake 命令行工具或 CMake GUI 为本机构建系统生成构建文件。

   有关生成 FreeRTOS 构建文件的信息，请参阅。生成构建文件（CMake 命令行工具）(p. 16)和生成构建文件 (CMake GUI) (p. 18)。
2. 调用本机构建系统，将项目制作为可执行文件。

   有关如何制作 FreeRTOS 构建文件的信息，请参阅。根据生成的构建文件构建 FreeRTOS (p. 19)。

生成构建文件（CMake 命令行工具）

可以使用 CMake 命令行工具 (`cmake`)，从命令行界面或终端生成 FreeRTOS 构建文件。

要生成编译文件，可运行 `cmake`。对于 `DVENDOR` 选项，请指定供应商。对于 `DBOARD` 选项，请指定主板。对于 `DCOMPILER` 选项，请指定编译器。使用 `S` 选项指定源代码的位置。使用 `B` 选项指定写入所生成文件的位置。

Note

编译器必须包含在系统的 PATH 变量中，或者必须指定编译器的位置。

例如，如果供应商是 Texas Instruments，主板是 CC3220 Launchpad，编译器是 GCC for ARM，那么可以发出以下命令，将源文件从当前目录构建到名为 *build-directory* 的目录：

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

Note

如果使用的是 Windows，则必须指定本机构建系统，因为 CMake 默认使用 Visual Studio。例如：

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

或者:

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

正则表达式 ${VENDOR}.* 和 ${BOARD}.* 用于搜索匹配的主板，因此对于 VENDOR 和 BOARD 选项，不必使用完整的供应商和主板名称。在只有单个名称匹配的情况下，部分名称也是可行的。例如，以下命令可从同一源文件生成相同的构建文件：

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

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

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

如果要使用不在默认目录 cmake/toolchains 中的工具链文件，则可使用 CMAKE_TOOLCHAIN_FILE 选项。例如：

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

如果工具链文件未使用编译器的绝对路径，并且编译器未添加到 PATH 环境变量中，那么 CMake 可能无法找到编译器。要确保 CMake 找到工具链文件，可以使用 AFR_TOOLCHAIN_PATH 选项。此选项将搜索指定的工具链目录路径以及 bin 下的工具链子文件夹。例如：

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

要启用调试，可将 CMAKE_BUILD_TYPE 设置为 debug。启用此选项后，CMake 将调试标志添加到编译选项，并构建带调试符号的 FreeRTOS。

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

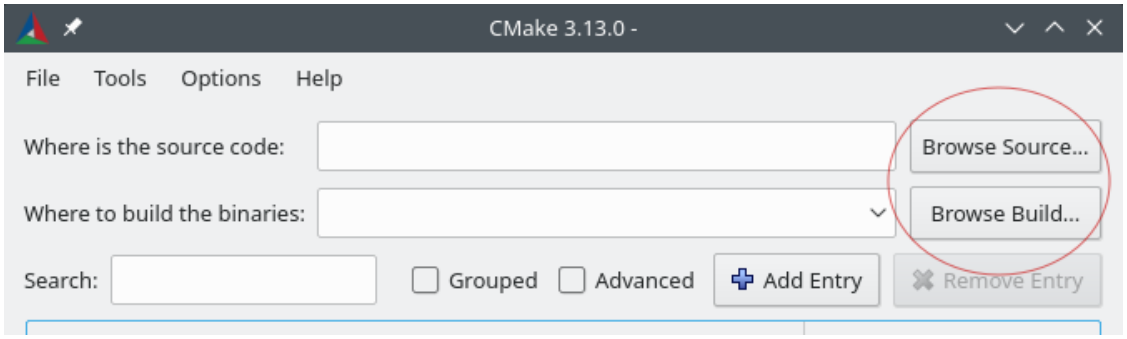
也可以将 `CMAKE_BUILD_TYPE` 设置为 `release`，将优化标志添加到编译选项。

生成构建文件 (CMake GUI)

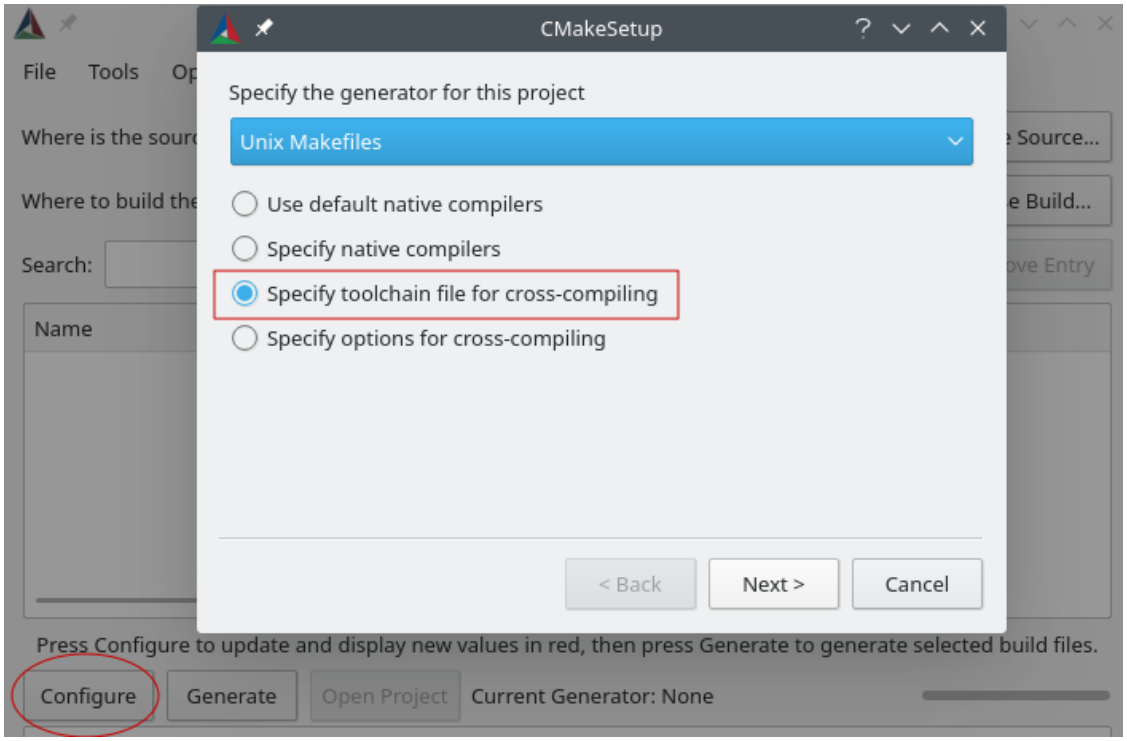
可以使用 CMake GUI 生成 FreeRTOS 构建文件。

使用 CMake GUI 生成构建文件

1. 从命令行中发出 `cmake-gui` 以启动 GUI。
2. 选择 Browse Source (浏览源) 并指定源输入，然后选择 Browse Build (浏览构建) 并指定构建输出。



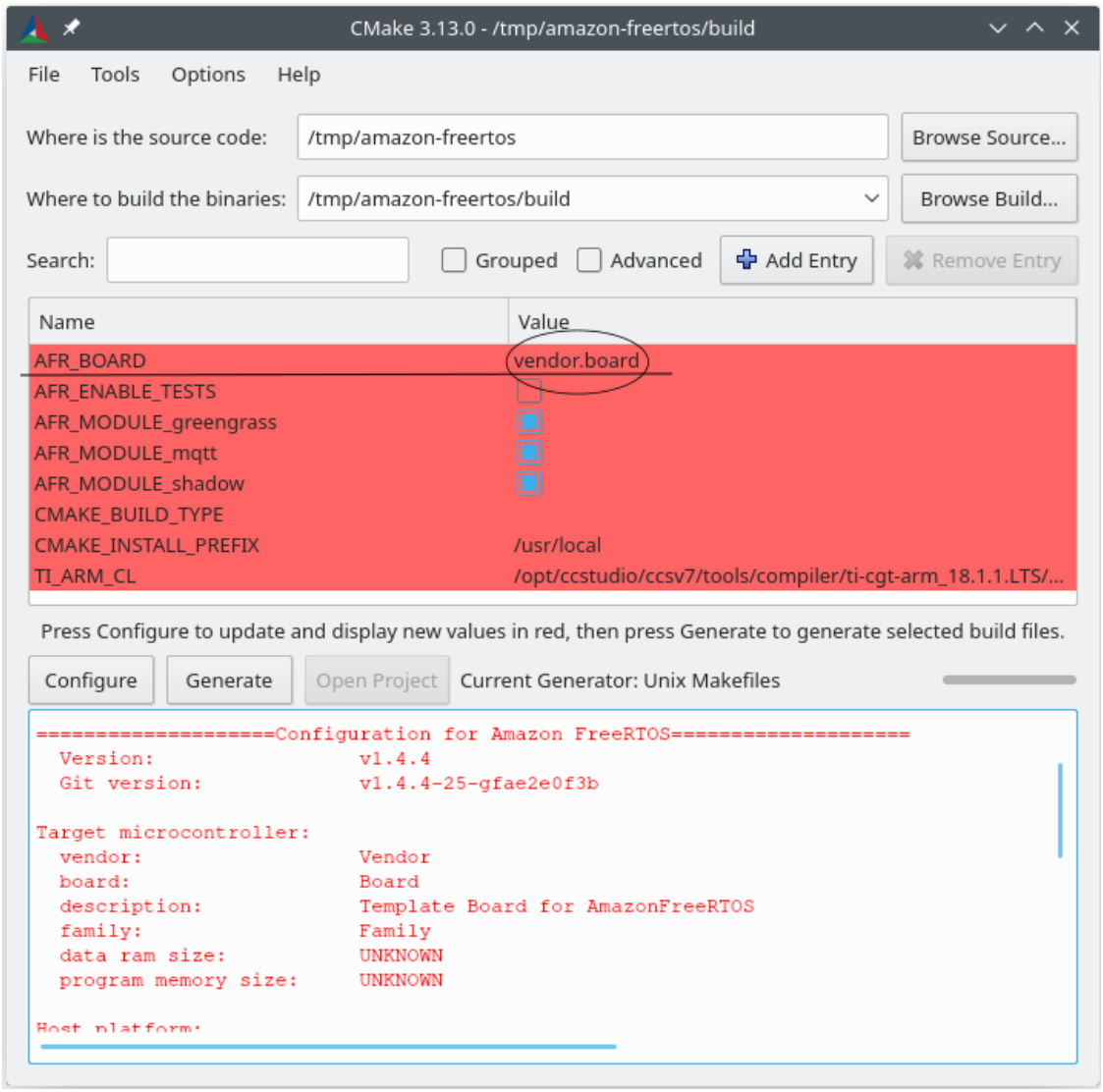
3. 选择 Configure (配置)，然后在 Specify the build generator for this project (指定此项目的构建生成器) 下，查找并选择构建系统，以用于构建所生成的构建文件。



4. 选择 Specify toolchain file for cross-compiling (指定用于交叉编译的工具链文件)，然后选择 Next (下一步)。
5. 选择工具链文件（例如，*`freertos`*`/tools/cmake/toolchains/arm-ti.cmake`），然后选择 Finish (完成)。

   FreeRTOS 的默认配置为模板主板，该主板不提供任何可移植层目标。结果，将显示一个包含消息 Error in configuration process 的窗口。

6. GUI 现在应如下所示：



选择 AFR_BOARD，选择主板，然后选择 Configure (配置)。

7. 选择 Generate (生成)。CMake 生成构建系统文件（例如，makefile 或 ninja 文件），这些文件位于第一步指定的构建目录中。按照下一节中的说明生成二进制映像。

根据生成的构建文件构建 FreeRTOS

可以使用本机构建系统构建，方法是从输出二进制目录调用构建系统命令。例如，如果构建文件输出目录为 `build`，并且您使用 Make 作为本机构建系统，则可运行以下命令：

```
cd build-directory
make -j4
```

也可以使用 CMake 命令行工具构建 FreeRTOS。CMake 提供了抽象层用于调用本机构建系统。例如：

```
cmake --build build-directory
```

以下是 CMake 命令行工具构建模式的其他一些常见用例：

```
# Take advantage of CPU cores.
cmake --build build-directory --parallel 8
```

```
# Build specific targets.
cmake --build build-directory --target afr_kernel
```

```
# Clean first, then build.
cmake --build build-directory --clean-first
```

有关 CMake 构建模式的更多信息，请参阅 CMake 文档。

为代码提供开源许可

为进行资格认证，请提供移植的 FreeRTOS 代码的开源许可。要确定要使用的许可证，请参阅开源计划网站上的许可证和标准。

FreeRTOS 资格审查清单

可使用以下清单来跟踪资格认证项。

您必须通过每一项认证才能列入Amazon合作伙伴设备目录。

- 查看将 FreeRTOS 移植到设备时必须遵循的步骤。这些步骤总结在FreeRTOS 移植流程图。有关更多信息，请参阅 。FreeRTOS 移植指南。
  - 您必须移植一个 FreeRTOS 资格认证的内核架构，并且不能自行对其进行修改。有关更多信息，请参阅 。配置 FreeRTOS 内核移植中的FreeRTOS 移植指南。
- 验证您的 FreeRTOS 端口Amazon IoTDevice Tester。
  - 您的设备资格认证门户 (DQP) 提交信息中需要成功的 IDT 日志（所有测试组都通过一个日志传递）。
  - 所有资格认证提交都必须通过 APN 合作伙伴中心的设备列表门户完成。
- 创建 “Hello World” 演示。
  - 请参阅 设置 Hello World 演示 (p. 5)。
- 创建入门指南(GSG) 适用于设备
  - 请参阅 为设备创建入 FreeRTOS 指南 (p. 7)。
- 使用代码创建并放置适当的开源许可证文本文件。
  - FreeRTOS 是根据MIT 许可证。
- 提供一个可访问的位置来下载您的代码。
  - 我们建议您使用 GitHub 存储库，但不要使用个人 GitHub 存储库。请使用官方的公司 GitHub 存储库。
- 减轻有关随机数生成器 (RNG) 的以下威胁：
  - 为了减轻网络欺骗和中间人攻击可能导致未经授权的数据泄露所带来的风险，FreeRTOS 资格认证需要使用真正的硬件随机数生成器 (TRNG)。实施 DHCP、DNS、TCP/IP 和 TLS 等协议的 FreeRTOS 库推荐使用 TRNG。与 NIST 发布的指南一致，FreeRTOS 使用主板上的 TRNG 作为标准实施 CTR_DRBG 的熵源。有关更多信息，请参阅精灵特性。

    根据 NIST SP 800-90B 的描述，TRNG 是一种“物理噪声源”（2.2.1 部分），可以生成“独立同分布”(IID) 示例（第 5 部分），例如环形振荡器。
  - 为了控制 BOM 成本和一些客户使用案例，某些主板将不具有专用的 TRNG。如果您有获得这些主板的资格，请在文件标头中添加以下通告`core_pkcs11.h`您已移植的核心API。查看我们的更改日志，了解类似的主板示例。

    > **Note**
    >
    > 为了获得最佳的安全实践，我们建议您使用真正随机化且符合FreeRTOS 资格审查清单。硅供应商在此文件中提出的随机数生成器方法本质上并不是真正随机的。请与硅供应商联系，了解有关所实施方法的详细信息。
- 如果您符合 OTA 资格，请验证您是否降低了移植 OTA 库中的FreeRTOS 移植指南。

将被指定为支持长期支持 (LTS) 版本Amazon合作伙伴设备目录，则必须提供清单文件。标准资格认证不需要执行此操作。您必须使用 FreeRTOS 的 LTS 版本，并包含`manifest.yml`文件。清单文件的要求在FreeRTOS 清单文件指令 (p. 22)，并且模板作为清单 .yml 示例 (p. 24)。默认情况下，FreeRTOS 存储库包含兼容的清单文件。

要列入联机配置向导，请与您的 APN 代表联系并提供以下项目：

- 创建 CMake 列表文件，并使用该文件构建测试和演示应用程序。
  - 有关说明，请参阅为平台创建 CMakeLists.txt 文件 (p. 9)。

Note

CMake 列表文件不需要通过Amazon设备认证计划。仅在 FreeRTOS 控制台上列出设备时需要该文件。

- 提供设备的以下硬件信息：
  - 用于优化的编译器选项。
  - 支持的 IDE，及支持的最新版本号。
  - 用于构建目标可执行文件的 CLI 命令。
  - 用于闪存目标的 CLI 命令。

FreeRTOS 清单文件指令

所有基于 LTS 的 FreeRTOS 项目都需要清单文件，以帮助客户界定正在使用的依赖关系版本、哪些库是 LTS，以及帮助客户更快地评估和使用软件的其他元数据。

该文件应满足以下要求：

- 命名`manifest.yml`
- 放置在*`freertos`*/vendors/*`vendor`*/boards/*`board`*/directory
- 采用 YAML 格式并按照YAML 1.2 规格

参数可以按任何顺序排列，但我们建议您将它们按下面列出的顺序排列，以获得最佳的可读性。在文件中添加注释，以帮助买家使用或了解您的包裹。

文件路径

位于软件包或库的根目录。每个程序包只能有一个清单。引入的依赖关系可能有自己的清单文件。

参数

name

程序包的名称。所有空格都应替换为下划线 (_)。例如，`My project name - 2020`应该更改为`My_project_name_-_2020`。

- 类型：字符串
- 需要：true
- MinLength: 1
- MaxLength: 40

版本

程序包的版本。版本可以是发行版本或版本标签。

- 类型：字符串
- 需要：true
- MinLength: 1
- MaxLength: 30

description

程序包的人类可读格式的描述。描述应清楚地描述包裹是什么以及它提供的内容。

- 类型：字符串
- 需要：true
- MinLength: 30
- MaxLength: 255

dependencies

用户成功构建此软件包所需的所有第一级依赖项列表，这些依赖项可由 Git、Subversion 或 Mercurial 源代码主机检索。不要包含通过 Git，SVG 或 hg 不可用的依赖关系。请勿包含用于测试、文档生成或开发的依赖关系。为了确保为一般公众提供良好的体验，我们建议您避免列出门控或私有的依赖关系。

- 类型：数组
- 需要：false
- MinLength: 0

依赖关系 [] .name

依赖项的程序包名称。这必须与依赖关系的`name`参数。

- 类型：字符串
- 需要：true
- MinLength: 1
- MaxLength: 40

依赖关系 [] .版本

依赖关系的版本。版本可以是发行版本或版本标签。如果包本身中包含任何依赖关系，则版本必须与依赖关系中的清单文件匹配。

- 类型：字符串
- 需要：true
- MinLength: 1
- MaxLength: 30

依赖关系 []。存储库

描述依赖关系源代码的位置。

- 类型：词典
- 需要：true

依赖关系 [] .存储库.类型

存储库的类型。

- 类型：字符串
- 需要：true
- 枚举：[git，svn，hg]

依赖关系 [] .存储库.url

存储库位置的 URL。这必须是一个带有协议前缀的完整 URL（例如 https://github.com/*ACCOUNT_NAME*/*####*）。

- 类型：字符串
- 需要：true

依赖关系 [] .存储库.分支

所使用的依赖关系的分支。如果软件包使用默认的主分支，则不要包含此参数以保持清单的最小长度。

- 类型：字符串
- 需要：false

许可

库的 SPDX 许可证标识符。有关完整列表，请参阅https://spdx.org/licenses/。它应该与`LICENSE`文件包含在存储库的根目录中（如果存在）。

- 类型：字符串
- 需要：true

清单 .yml 示例

```
---
# This is an example of the manifest file that is included at the root of all FreeRTOS
 repos.

name : "Project_Name"
version: "202012.00-LTS"
description: "Clear concise description of this project."

dependencies:
  - name: "dependency_1"
    version: "v1.0.0"
    repository:
      type: "git"
      url: "https://github.com/account/dependency_1"
      branch: "1.x"
  - name: "dependency_2"
    version: "v1.0.1_LTS"
    repository:
      type: "git"
      url: "https://github.com/account/dependency_1"

license: "MIT"
```

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。