aws

用户指南

FreeRTOS



Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

FreeRTOS: 用户指南

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon 的商标和商业外观不得用于任何非 Amazon 的商品或服务，也不得以任何可能引起客户混淆、贬低或诋毁 Amazon 的方式使用。所有非 Amazon 拥有的其他商标均为各自所有者的财产，这些所有者可能附属于 Amazon、与 Amazon 有关联或由 Amazon 赞助，也可能不是如此。

Table of Contents

什么是 FreeRTOS？ 1
下载 FreeRTOS 源代码 1
符合 FreeRTOS 条件的硬件平台 1
其他资源 2
FreeRTOS 版本 2
FreeRTOS 长期支持 3
FreeRTOS 扩展维护计划 3
FreeRTOS 架构 3
开发工作流程 4
FreeRTOS 内核基础知识 5
FreeRTOS 内核调度器 5
内核内存分配 5
管理应用程序内存 6
任务间协调 7
队列 7
信号灯和互斥对象 7
Direct-to-task 通知 8
流缓冲区 8
消息缓冲区 9
对称多处理 (SMP) 支持 10
软件计时器 11
低功耗支持 11
FreeRTOSConfig.h 11
AWS IoT 适用于嵌入式 C 的设备 SDK 13
了解 FreeRTOS 的常用 IO APIs 14
库 14
通用 IO - 基本 14
通用 IO-BLE 15
适用于 Amazon 通用软件的通用 IO 16
什么是 ACS？ 16
资格认证计划 16
开始使用 FreeRTOS 17
开始使用 Quick Connect 17
探索 FreeRTOS 库 17

打造安全耐用的 AWS IoT 产品 17
开发您的 AWS IoT 应用程序 18
AWS IoT Device Tester 适用于 FreeRTOS 19
FreeRTOS 资格认证套件 19
了解自定义测试套件 20
支持的适用于 FreeRTOS 的 IDT 版本 20
适用于 FreeRTOS 的 IDT 的最新版本 21
早期 IDT 版本 23
不受支持的 IDT 版本 26
下载适用于 FreeRTOS 的 IDT 50
手动下载 IDT 50
以编程方式下载 IDT 51
带有 FreeRTOS 资格套件 2.0 的 IDT (FRQ 2.0) 56
设置 LTS 资格先决条件 57
首次测试您的微控制器板 66
使用 IDT UI 运行 FreeRTOS 资格认证套件 80
运行 FreeRTOS 资格赛 2.0 套件 95
免费查看 IDT RTOSresults 98
解读 FreeRTOS 结果的 IDT 99
免费查看 IDT RTOSlogs 101
带有 FreeRTOS 资格套件 1.0 的 IDT (FRQ 1.0) 102
设置 1.0 资格先决条件 103
首次测试您的微控制器板 107
使用 IDT UI 运行 FreeRTOS 资格认证套件 125
运行蓝牙低功耗测试 134
运行 FreeRTOS 资格套件 139
查看 IDT for FreeRTOS 结果 144
解读 FreeRTOS 结果的 IDT 145
查看 FreeRTOS 日志的 IDT 148
开发和运行自己的 IDT 测试套件 148
下载适用于 FreeRTOS 的最新版本的 IDT 149
测试套件工作流程 149
教程：构建和运行示例 IDT 测试套件 150
教程：开发一个简单的 IDT 测试套件 155
测试套件版本 232
排查 错误 233

对设备配置进行故障排除 234
超时错误疑难解答 245
蜂窝功能和 AWS 费用 246
资格报告生成策略 246
AWS 的托管策略 AWS IoT Device Tester 246
托管式策略 246
策略更新 253
支持策略 256
安全性 AWS 258
身份和访问管理 258
受众 259
使用身份进行身份验证 259
使用策略管理访问 262
如何将 FreeRTOS 与 IAM 配合使用 264
基于身份的策略示例 269
故障排除 271
合规性验证 273
恢复能力 274
基础结构安全性 274
Amazon-FreeRTOS Github 存储库迁移指南 275
附录 275
档案 280
FreeRTOS 用户指南存档 280
FreeRTOS 用户指南之前的内容 280
开始使用 FreeRTOS 280
Over-the-Air 更新 459
FreeRTOS 库 535
FreeRTOS 演示 593
..... dccx

什么是 FreeRTOS？

与世界领先的芯片公司合作开发了 15 年，现在每 170 秒有一次下载，FreeRTOS 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)。根据 MIT 开源许可免费分发，FreeRTOS 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 的设计非常注重可靠性和易用性。

FreeRTOS 包括用于连接、安全 over-the-air和 (OTA) 更新的库。FreeRTOS 还包括演示应用程序，可在合格的主板上显示 FreeRTOS 功能。

FreeRTOS 是一个开源项目。您可以在 https://github.com/FreeRTOS/FreeRTOS GitHub 网站上下载源代码、贡献更改或增强，或者报告问题。

我们根据 MIT 开源许可证发布 FreeRTOS 代码，以便您可以在商业和个人项目中使用这些代码。

我们也欢迎您对 FreeRTOS 文档 (《FreeRTOS 用户指南》、《FreeRTOS 移植指南》和《FreeRTOS 资格认证指南》) 提供意见或建议。要查看文档的 markdown 源代码，请参阅https://github.com/awsdocs/aws-freertos-docs。这是根据知识共享 (CC BY-ND) 许可证发布的。

下载 FreeRTOS 源代码

从 freertos.org 的下载页面下载最新的 FreeRTOS 和长期支持 (LTS) 程序包。

符合 FreeRTOS 条件的硬件平台

以下硬件平台符合 FreeRTOS 的条件：

- ATECC608A 零接触配置套件适用于 AWS IoT
- Cypress CYW9439 07 AEVAL1 F 开发套件
- Cypress CYW9549 07 AEVAL1 F 开发套件
- Cypress CY8 CKIT-064S0S2-4343W 套件
- Espressif-C ESP32 DevKit
- Espressif ESP-WROVER-KIT
- Espressif ESP-WROOM-32SE
- Espressif-S2-Saola-1 ESP32
- 英飞凌 00 XMC48 物联网连接套件
- Marvell MW32 0 AWS IoT 入门套件

- Marvell MW322 AWS IoT 入门套件
- MediaTek MT7697Hx 开发套件
- 微芯片好奇号 PIC32 MZEF 捆绑包
- 北欧 n RF5284 0-DK
- NuMaker-iot-m487
- 恩智浦 LPC54 018物联网模块
- OPTIGA Trust X 安全解决方案
- 瑞萨 RX65 N RSK 物联网模块
- STMicroelectronicsSTM32L4 探索套件物联网节点
- 德州仪器 CC322 0SF-LAUNCHXL
- Microsoft Windows 7 或更高版本，至少双核处理器以及有线以太网连接
- Xilinx Avnet MicroZed 工业物联网套件

AWS 合作伙伴设备目录中也列出了取得认证的设备。

有关新设备资格鉴定的信息，请参阅 FreeRTOS 资格认证指南。

其他资源

这些资源可能对您很有用。

- 有关其他 FreeRTOS 文档，请参阅 freertos.org。
- 如果 FreeRTOS 工程团队对 FreeRTOS 有疑问，可以在 FreeRTOS 页面上打开一个问题。 GitHub
- 有关 FreeRTOS 的技术问题，请访问 FreeRTOS 社区论坛。
- 有关将设备连接至的更多信息 AWS IoT，请参阅《AWS IoT Core 开发者指南》中的设备配置。
- 有关技术支持 AWS，请参阅 Supp AWS ort。
- 有关 AWS 账单、账户服务、活动、滥用行为或其他问题的问题 AWS，请参阅 “联系我们” 页面。

FreeRTOS 版本

各个库使用 x.y.z 风格的版本号，类似于语义版本控制。x 是主版本号，y 是次要版本号，从 2022 年开始，z 是补丁号。在 2022 年之前，z 是一个小版本号，它要求第一个 LTS 库的补丁号必须为“x.y.z LTS 补丁 2”。

库包使用 yyyymm.x 风格的日期戳版本号。yyyy 是年份，mm 是月份，x 是显示当月发行顺序的可选序列号。对于 LTS 程序包，x 是 LTS 版本的连续补丁号。程序包中包含的各个库是该库当天的最新版本。对于 LTS 程序包，它是当天最初作为 LTS 版本发布的 LTS 库的最新补丁版本。

FreeRTOS 长期支持

FreeRTOS 长期支持 (LTS) 版本在发布后至少两年内会收到安全和关键错误修复（如有必要）。通过这种持续的维护，您可以在整个开发和部署周期中加入错误修复，而不会因为更新到 FreeRTOS 库的新主要版本而造成代价高昂的中断。

凭借 FreeRTOS LTS，您可以获得构建安全互联的 IoT 和嵌入式产品所需的全套库。对于已投入生产的设备上库的更新，LTS 有助于降低相关的维护和测试成本。

FreeRTOS LTS 包括 FreeRTOS 内核和物联网库：freertos+TCP、CoreMQTT、CoreHttp PKCS11、coreJson、OTA、Jobs 和 Device Shadow。 AWS IoT AWS IoT AWS IoT Device Defender AWS IoT 有关更多信息，请参阅 FreeRTOS LTS 库。

FreeRTOS 扩展维护计划

AWS 还提供 FreeRTOS 延长维护计划 (EMP)，该计划为您选择的 FreeRTOS 长期支持 (LTS) 版本提供安全补丁和关键错误修复，最长可延长十年。凭借 FreeRTOS EMP，基于 FreeRTOS 的长期性设备可依赖功能稳定的版本且可接收多年的安全更新。您可以及时收到有关 FreeRTOS 库即将发布的补丁的通知，因此您可以计划在物联网 (IoT) 设备上部署安全补丁。

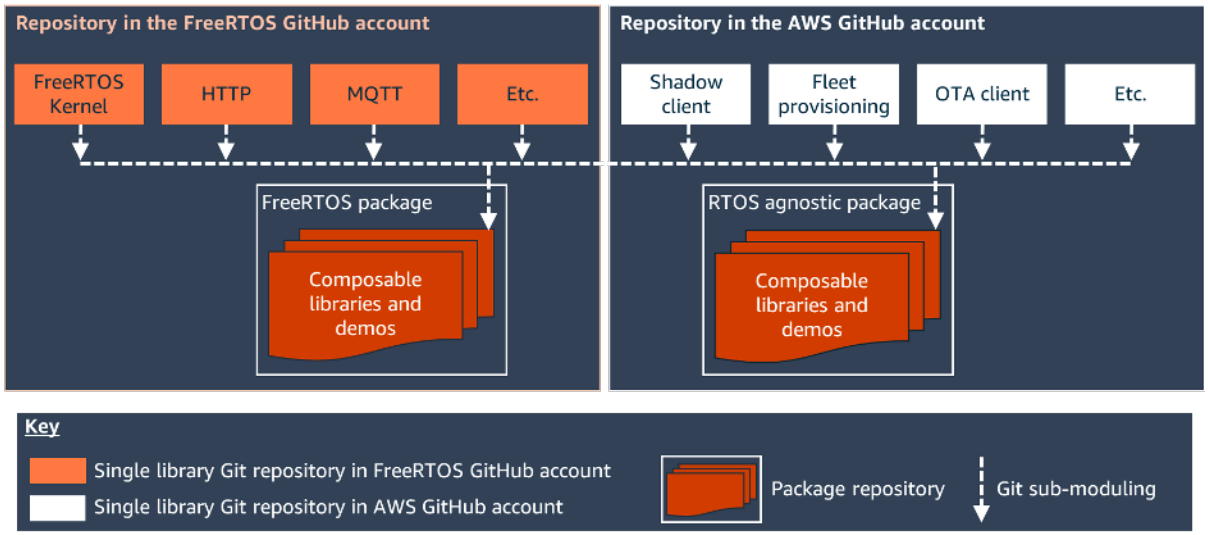
要了解有关 FreeRTOS EMP 的更多信息，请参阅功能页面。

FreeRTOS 架构

FreeRTOS 包含两种类型的存储库，即单库存储库和程序包存储库。每个单库存储库都包含一个库的源代码，而不包含任何构建项目或示例。程序包存储库包含多个库，并且可以包含演示库用法的预配置项目。

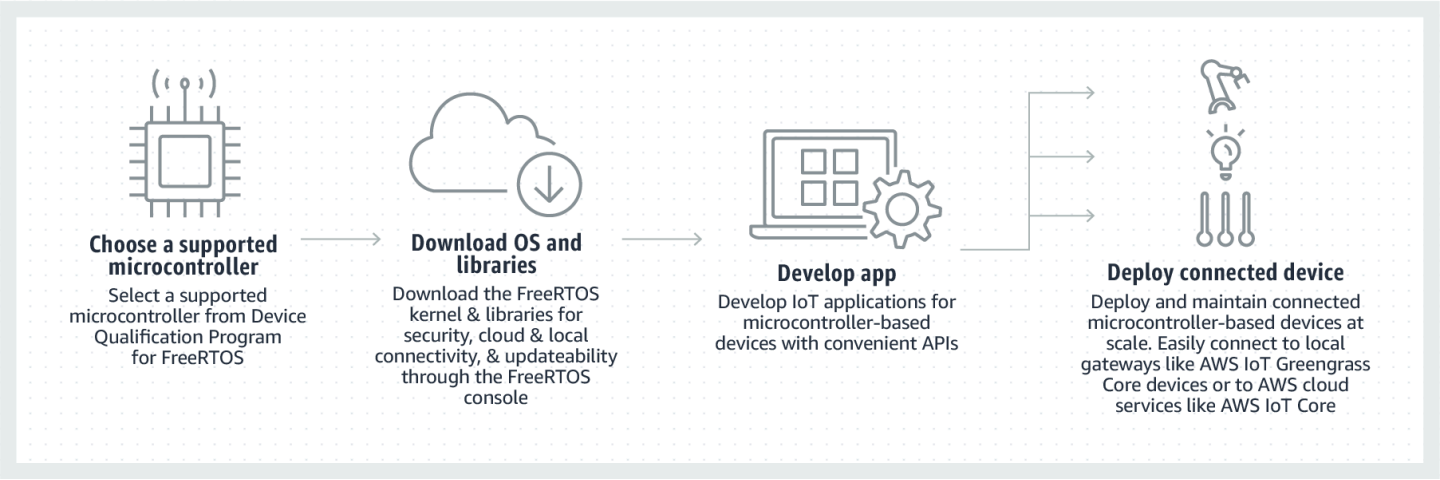
虽然程序包存储库包含多个库，但不包含这些库的副本。程序包存储库将其包含的库作为为 git 子模块引用。使用子模块可确保每个单独的库都有单一的真实来源。

单个库 git 存储库分为两个 GitHub 组织。包含 FreeRTOS 特定库（例如 FreerTOS+TCP）或通用库（例如 CoreMQTT，它适用于任何 MQTT 代理，因此与云无关）的存储库位于 FreeRTOS 组织中。GitHub 包含 AWS IoT 特定库（例如 AWS IoT over-the-air 更新客户端）的存储库位于 AWS GitHub组织中。下图介绍了结构。



开发工作流程

您可通过下载 FreeRTOS 来开始开发。解压缩程序包并将其导入您的 IDE。然后，您可以在所选硬件平台上开发应用程序，使用适合您设备的开发流程制造和部署这些设备。已部署的设备可以连接到 AWS IoT 服务或 AWS IoT Greengrass 作为完整物联网解决方案的一部分。



FreeRTOS 内核基础知识

FreeRTOS 内核是一个实时操作系统，支持各种架构。它的基础非常适合构建嵌入式微控制器应用程序。它提供了以下功能：

- 多任务计划程序。
- 多个内存分配选项（包括创建完全静态分配的系统的功能）。
- 任务间协调基元，包括任务通知、消息队列、多种信号灯类型以及流和消息缓冲区。
- 支持多核微控制器上的对称多处理 (SMP)。

FreeRTOS 内核在关键部分或中断内部从不执行非确定性操作，例如，遍历链接列表。FreeRTOS 内核包含一个高效的软件计时器实施，不使用任何 CPU 时间（除非计时器需要维护）。被封锁的任务不需要耗时的定期维护。 Direct-to-task通知允许快速发送任务信号，几乎没有 RAM 开销。它们可用于大多数任务间和 interrupt-to-task信令场景。

FreeRTOS 内核设计为小型、简单且易于使用。典型的 RTOS 内核二进制映像大小为 4000 到 9000 字节。

有关 FreeRTOS 内核的大部分 up-to-date文档，请参阅 FreerTOS.org。FreerTOS.org 提供了许多关于使用 FreeRTOS 内核的详细教程和指南，包括 FreeRTOS FreeRTOS 内核快速入门指南和 FreeRTOS 文档中更深入的 RTOS 实现。

FreeRTOS 内核调度器

采用 RTOS 的嵌入式应用程序可以结构化为一组独立的任务。每个任务都在自己的上下文中执行，独立于其他任务。在任何时间点，应用程序中都只有一个任务在运行。每个任务应当在何时运行由实时 RTOS 计划程序决定。每个任务都提供有自己的堆栈。当某个任务被换出以便运行另一个任务时，该任务的执行上下文将保存到任务堆栈，以便稍后在换回该任务恢复其运行时，可以还原执行上下文。

为提供确定性的实时行为，FreeRTOS 任务计划程序允许为任务分配严格的优先级。RTOS 可确保为能够执行的最高优先级任务分配处理时间。如果优先级相同的多个任务同时准备好运行，则这些任务需要共享处理时间。FreeRTOS 还会创建空闲任务，仅在没有其他任务准备好运行时执行它。

内核内存分配

每次在创建任务、队列或其他 RTOS 对象时，RTOS 内核都需要 RAM。RAM 可采用以下分配方式：

- 在编译时静态分配。
- 由 RTOS API 对象创建函数从 RTOS 堆动态分配。

在动态创建 RTOS 对象时，使用标准 C 库 `malloc()` 和 `free()` 函数并不始终恰当，原因如下：

- 它们在嵌入式系统中可能不可用。
- 它们占用了宝贵的代码空间。
- 它们通常不是线程安全的。
- 它们不是确定性的。

出于这些原因，FreeRTOS 会在其可移动层保留内存分配 API。可移动层位于实施核心 RTOS 功能的源文件外部，因此您可以针对正在开发的实时系统，提供特定于应用程序的适当实施。当 RTOS 内核需要 RAM 时，它会调用 `pvPortMalloc()`，而不是 `malloc()`()。在释放 RAM 时，RTOS 内核调用 `vPortFree()`，而不是 `free()`。

管理应用程序内存

当应用程序需要内存时，可以从 FreeRTOS 堆进行分配。FreeRTOS 提供了多种堆管理方案，复杂性和功能各不相同。您也可以提供自己的堆实施。

FreeRTOS 内核包含以下五个堆实施：

heap_1

是最简单的实施。不允许释放内存。

heap_2

允许释放内存，但不合并相邻的空闲数据块。

heap_3

对标准的 `malloc()` 和 `free()` 进行包装以确保线程安全。

heap_4

合并相邻的空闲数据块以避免碎片。包括绝对地址放置选项。

heap_5

类似于 heap_4。可以跨越多个非相邻内存区域中的堆。

任务间协调

本部分包含了有关 FreeRTOS 基元的信息。

主题

- 队列
- 信号灯和互斥对象
- Direct-to-task 通知
- 流缓冲区
- 消息缓冲区

队列

队列是任务间通信的主要方式。可用于在任务之间以及中断与任务之间发送消息。大多数情况下，队列用作线程安全先进先出 (FIFO) 缓冲区，新数据将发送到队列的后面。（数据也可以发送到队列的前面。）消息通过队列以复制方式发送，这意味着是将数据（可能是指向更大缓冲区的指针）本身复制到队列中，而不只是存储对数据的引用。

队列 APIs 允许指定阻塞时间。如果任务尝试读取空队列，则该任务将置为“被阻止”状态，直到队列中有可用数据，或者阻止时间结束。处于“被阻止”状态的任务不会占用任何 CPU 时间，以便其他任务运行。同样，如果任务尝试写入已满队列，则该任务将置为“被阻止”状态，直到队列中有可用空间，或者阻止时间结束。如果对同一队列阻止了多个任务，则具有最高优先级的任务将首先取消阻止。

其他 FreeRTOS 原语， direct-to-task例如通知以及流和消息缓冲区，为许多常见的设计场景中的队列提供了轻量级替代方案。

信号灯和互斥对象

FreeRTOS 内核提供了二元信号灯、计数信号灯和互斥对象，以用于相互排斥和同步的情况。

二元信号灯只能有两个值。如果要在任务之间或任务与中断之间实施同步，二元信号灯是不错的选择。计数信号灯可以有两个以上的值。该信号灯允许多个任务共享资源或执行更复杂的同步操作。

互斥对象是包括优先级继承机制的二元信号灯。这意味着，如果高优先级任务在尝试获取当前由较低优先级任务所持有的互斥对象时遭到阻止，则持有令牌的任务的优先级将临时提升至被阻止任务的优先级。此机制旨在确保较高优先级任务尽可能在最短时间内处于“被阻止”状态，从而最大程度地减少优先级反转的发生。

Direct-to-task 通知

任务通知允许任务与其他任务交互并与中断服务例程 (ISRs) 同步，而无需像信号量这样的单独通信对象。每个 RTOS 任务都有一个 32 位通知值，用于存储通知的内容（如果有）。RTOS 任务通知即直接发送给任务的事件，可以取消阻止接收的任务，以及有选择地更新所接收任务的通知值。

RTOS 任务通知可用作二元信号灯、计数信号灯和队列（在某些情况下）的更快速的轻型替代方案。相比于可用于执行同等功能的其他 FreeRTOS 基元，任务通知在速度和 RAM 开销两方面均具有优势。但是，任务通知只能用在事件接收方只能是一个任务的情况下。

流缓冲区

通过流缓冲区，可以将字节流从中断服务例程传递到任务，或者从一个任务传递到另一个。字节流可以是任意长度，且并不一定具有开头或结尾。可以一次写入任意数量的字节，也可以一次读取任意数量的字节。可通过将 `stream_buffer.c` 源文件包含在项目中来启用流缓冲区功能。

流缓冲区假定只有一个任务或中断写入缓冲区（即写入器），并且只从缓冲区读取一个任务或中断（即读取器）。写入器和读取器是不同的任务或中断服务例程才安全，具有多个写入器或读取器是不安全的。

流缓冲区实现使用 direct-to-task通知。因此，流缓冲区 API 将调用的任务置于“被阻止”状态，调用该 API 可以改变该调用任务的通知状态和通知值。

发送数据

`xStreamBufferSend()` 用于将数据发送到任务的流缓冲区。`xStreamBufferSendFromISR()` 用于将数据发送到中断服务例程 (ISR) 的流缓冲区。

`xStreamBufferSend()` 允许指定阻止时间。如果在调用 `xStreamBufferSend()` 时，将非零阻止时间写入流缓冲区且缓冲区已满，则任务将置为“被阻止”状态，直到有可用空间或者阻止时间结束。

`sbSEND_COMPLETED()` 和 `sbSEND_COMPLETED_FROM_ISR()` 是将数据写入流缓冲区时调用的宏（由 FreeRTOS API 在内部调用）。它采用更新的流缓冲区的句柄。这两个宏会查看流缓冲区中是否有被阻止的任务等待数据，如果有，会从“被阻止”状态中删除该任务。

您可以在 `FreeRTOSConfig.h` 中提供自己的 `sbSEND_COMPLETED()` 实施，以更改此默认行为。如果利用流缓冲区在多核处理器的核心之间传递数据，该功能很有用。在此情况下，可以执行 `sbSEND_COMPLETED()` 在另一个 CPU 核心中生成中断，然后该中断的服务例程可以使用 `xStreamBufferSendCompletedFromISR()` API 进行检查，如有必要则取消阻止等待数据的任务。

接收数据

xStreamBufferReceive() 用于从任务的流缓冲区读取数据。xStreamBufferReceiveFromISR() 用于从中断服务例程 (ISR) 的流缓冲区读取数据。

xStreamBufferReceive() 允许指定阻止时间。如果在调用 xStreamBufferReceive() 时，从流缓冲区读取非零阻止时间但缓冲区为空，则任务将置为“被阻止”状态，直到流缓冲区中有可用的指定数据量，或者阻止时间结束。

在取消阻止任务之前流缓冲区中必须具备的数据量，称为流缓冲区的触发级别。如果被阻止的任务触发级别为 10，则当至少 10 字节写入缓冲区或者任务的阻止时间结束时，将取消阻止该任务。如果读取任务尚未达到触发级别但其阻止时间已经到期，则该任务将接收写入缓冲区的任何数据。任务的触发级别必须设置为介于 1 与流缓冲区大小之间的值。流缓冲区的触发级别在调用 xStreamBufferCreate() 时设置。可以通过调用 xStreamBufferSetTriggerLevel() 进行更改。

sbRECEIVE_COMPLETED() 和 sbRECEIVE_COMPLETED_FROM_ISR() 是从流缓冲区读取数据时调用的宏（由 FreeRTOS API 内部调用）。宏会查看流缓冲区中是否有被阻止的任务等待缓冲区中有可用空间，如果有，它们会从“被阻止”状态中删除该任务。您可以在 FreeRTOSConfig.h 中提供其他实施来更改 sbRECEIVE_COMPLETED() 的默认行为。

消息缓冲区

通过消息缓冲区，可以将可变长度的离散消息从中断服务例程传递到任务，或者从一个任务传递到另一个。例如，可以将长度为 10、20 和 123 字节的消息写入消息缓冲区，或者从同一消息缓冲区中读取这些消息。10 字节的消息只能以 10 字节消息而不是单独字节的形式读取。消息缓冲区构建在流缓冲区实施之上。您可以通过将 stream_buffer.c 源文件包含在项目中来启用消息缓冲区功能。

消息缓冲区假定只有一个任务或中断写入缓冲区（即写入器），并且只从缓冲区读取一个任务或中断（即读取器）。写入器和读取器是不同的任务或中断服务例程才安全，具有多个写入器或读取器是不安全的。

消息缓冲区实现使用 direct-to-task通知。因此，流缓冲区 API 将调用的任务置于“被阻止”状态，调用该 API 可以改变该调用任务的通知状态和通知值。

要启用消息缓冲区来处理可变大小的消息，应先将每条消息的长度写入消息缓冲区，然后再写入消息本身。长度存储在类型为 size_t 的变量中，在 32 字节架构上通常为 4 字节。因此，将一条 10 字节消息写入消息缓冲区时，实际占用的缓冲区空间为 14 字节。同样，将一条 100 字节消息写入消息缓冲区时，实际使用的缓冲区空间为 104 字节。

发送数据

xMessageBufferSend() 用于将数据从任务发送到消息缓冲区。xMessageBufferSendFromISR() 用于将数据从中断服务例程 (ISR) 发送到消息缓冲区。

xMessageBufferSend() 允许指定阻止时间。如果在调用 xMessageBufferSend() 时，将非零阻止时间写入消息缓冲区且缓冲区已满，则任务将置为“被阻止”状态，直到消息缓冲区中有可用空间，或者阻止时间结束。

sbSEND_COMPLETED() 和 sbSEND_COMPLETED_FROM_ISR() 是将数据写入流缓冲区时调用的宏（由 FreeRTOS API 在内部调用）。宏采用单个参数，即更新的流缓冲区的句柄。这两个宏会查看流缓冲区中是否有被阻止的任务等待数据，如果有，会从“被阻止”状态中删除该任务。

您可以在 FreeRTOSConfig.h 中提供自己的 sbSEND_COMPLETED() 实施，以更改此默认行为。如果利用流缓冲区在多核处理器的核心之间传递数据，该功能很有用。在此情况下，可以执行 sbSEND_COMPLETED() 在另一个 CPU 核心中生成中断，然后该中断的服务例程可以使用 xStreamBufferSendCompletedFromISR() API 进行检查，如有必要则取消阻止等待数据的任务。

接收数据

xMessageBufferReceive() 用于将数据从消息缓冲区读取到任务中。xMessageBufferReceiveFromISR() 用于将数据从消息缓冲区读取到中断服务例程 (ISR) 中。xMessageBufferReceive() 允许指定阻止时间。如果在调用 xMessageBufferReceive() 时，从消息缓冲区读取非零阻止时间但缓冲区为空，则任务将置为“被阻止”状态，直到有可用数据或者阻止时间结束。

sbRECEIVE_COMPLETED() 和 sbRECEIVE_COMPLETED_FROM_ISR() 是从流缓冲区读取数据时调用的宏（由 FreeRTOS API 内部调用）。宏会查看流缓冲区中是否有被阻止的任务等待缓冲区中有可用空间，如果有，它们会从“被阻止”状态中删除该任务。您可以在 FreeRTOSConfig.h 中提供其他实施来更改 sbRECEIVE_COMPLETED() 的默认行为。

对称多处理 (SMP) 支持

FreeRTOS 内核支持 SMP，使 FreeRTOS 的一个内核实例能够在多个相同的处理器内核上调度任务。核心架构必须相同且共享相同的内存。

除了这些额外版本外，FreeRTOS API 在单核和 SMP 版本之间基本保持不变。 APIs因此，为 FreeRTOS 单核版本编写的应用程序应使用 SMP 版本进行编译，这样工作量就会很少。但是，可能会存在一些功能问题，因为一些适用于单核应用程序的假设可能不适用于多核应用程序。

一个常见的假设是，当优先级较高的任务正在运行时，优先级较低的任务无法运行。虽然在单核系统上确实如此，但多核系统则不然，因为多个任务可以同时运行。如果应用程序依靠相对任务优先级来提供互斥性，则它可能会在多核环境中观察到意外结果。

另一个常见的假设是，它 ISRs 不能与其他任务同时运行，也不能与其他任务同时运行。在多核环境中，情况不再是这样。应用程序编写者在访问任务和之间共享的数据时需要确保适当的相互排斥 ISRs。

软件计时器

采用软件计时器，可以在未来的设定时间执行函数。由计时器执行的函数称为计时器的回调函数。从启动计时器到执行计时器回调函数之间的时间称为计时器的周期。FreeRTOS 内核提供了高效的软件计时器实施，原因如下：

- 它不会从中断上下文内执行计时器回调函数。
- 它不会占用任何处理时间，除非计时器实际上已过期。
- 它不会给滴答中断增加任何处理开销。
- 中断处于禁用状态时，它不会搜索任何链接列表结构。

低功耗支持

与大多数嵌入式操作系统一样，FreeRTOS 内核使用硬件计时器来生成周期性的滴答中断，以用于测量时间。常规硬件计时器实施的节能受限于必须定期退出然后重新进入低功耗状态来处理滴答中断。如果滴答中断的频率太高，则为每次滴答中断进入和退出低功耗状态所消耗的能量和时间，会超过除最轻节能模式之外其他任何可能的节能收益。

为解决此限制问题，FreeRTOS 为低功耗应用程序提供了非滴答计时器模式。FreeRTOS 非滴答空闲模式在空闲时间段（即不存在可执行的应用程序任务的时间段）内将停止周期性的滴答中断，然后在重新启动滴答中断时对 RTOS 滴答计数值进行校正调整。通过停止滴答中断，微控制器可以维持在深度节能状态，直到中断发生，或者到了 RTOS 内核将任务转换为就绪状态的时间。

配置内核（Fre RTOSConfig e .h）

您可以使用 `FreeRTOSConfig.h` 标头文件为特定主板和应用程序配置 FreeRTOS 内核。每个基于内核构建的应用程序都必须在其预处理程序包含路径中有一个 `FreeRTOSConfig.h` 标头文件。`FreeRTOSConfig.h` 是特定于应用程序的，并且应置于应用程序目录下，而不是置于 FreeRTOS 内核源代码目录中。

FreeRTOS 演示和测试应用程序的 FreeRTOSConfig.h 文件位于 *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/FreeRTOSConfig.h 和 *freertos*/vendors/*vendor*/boards/*board*/aws_tests/config_files/FreeRTOSConfig.h 中。

有关要在 FreeRTOSConfig.h 中指定的可用配置参数的列表，请参阅 FreeRTOS.org。

AWS IoT 适用于嵌入式 C 的设备 SDK

> **Note**
>
> 该 SDK 供经验丰富的嵌入式软件开发人员使用。

AWS IoT Device SDK for Embedded C (C-SDK) 是 MIT 开源许可下的 C 源文件集合，可用于嵌入式应用程序，将物联网设备安全地连接到。 AWS IoT Core它包括 MQTT 客户端、HTTP 客户端、JSON 解析 AWS IoT 器和 Device Shadow、 AWS IoT 作业、 AWS IoT 队列配置和 AWS IoT Device Defender 库。该开发工具包以源代码形式分发，可构建到客户固件和应用程序代码、其他库以及您选择的操作系统 (OS) 中。

AWS IoT Device SDK for Embedded C 通常针对需要优化 C 语言运行时的资源受限的设备。您可以在任何操作系统上使用 SDK，也可以将其托管在任何类型的处理器上（例如 MCUs 和 MPUs）。但是，如果您的设备有足够的内存和处理资源，我们建议您使用更高等级的AWS IoT 设备 SDKs。

有关更多信息，请参阅下列内容：

- AWS IoT 适用于嵌入式 C 的设备 SDK
- AWS IoT 适用于嵌入式 C 的设备 SDK 已开启 GitHub
- AWS IoT 适用于嵌入式 C 的设备 SDK 自述文件
- AWS IoT 嵌入式 C 示例的设备 SDK

了解 FreeRTOS 的常用 IO APIs

通用 IO APIs 充当硬件抽象层 (HAL)，在驱动程序和更高级别的应用程序代码之间提供通用接口。FreeRTOS Common IO 为在支持的参考板上访问常用串行设备提供了一套 APIs 标准；不包括 APIs 这些标准的实现。它们与这些外围设备进行 APIs 通信和交互，使您的代码能够跨平台运行。如果没有通用 I/O，则编写代码以使用低级设备这一操作是特定于芯片供应商的。

> **Note**
>
> FreeRTOS不需要实现通用IO即可运行，但它将尝试使用通用 APIs IO作为与基于微控制器的主板上的特定外围设备而不是供应商特定的主板上的特定外围设备进行接口的一种方式。 APIs APIs

通常，设备驱动程序独立于基础操作系统，并且特定于给定的硬件配置。HAL 提取出特定驱动程序的工作原理的详细信息，并提供一个统一的 API 来控制此类设备。您可以使用相同的方法 APIs 访问基于多个微控制器 (MCU-) 的参考板上的各种设备驱动程序。

库

目前，FreeRTOS 提供了两个通用 IO 库：“通用 IO - 基本”和“通用 IO - BLE”。

通用 IO - 基本

概览

C@@ ommon IO-基本 APIs 版提供基本 I/O 外设和基于 MCU 的主板上可能找到的功能。Common IO-基本存储库可在上找到GitHub。

支持的外围设备

- ADC
- GPIO
- I2C
- PWM
- SPI
- UART

- 看门狗
- 闪存
- RTC
- EFUSE
- 重置
- I2S
- 性能计数器
- 硬件平台信息

支持的特征

- 同步读/写

  此函数直到传输了请求的数据量后才返回。

- 异步读/写

  此函数立即返回，并且数据以异步方式传输。在此操作完成后，将调用已注册用户回调。

外围设备特定的

- I2C

  将多个操作合并到一个事务中。用于在一个事务中依次执行写入操作和读取操作。

- SPI

  在主设备和辅助设备之间传输数据，这意味着写入操作和读取操作会同时进行。

API 参考

如需完整的 API 参考，请参阅通用 IO - 基本 API 参考。

通用 IO-BLE

概览

通用 IO - BLE 提供了制造商低功耗蓝牙堆栈的抽象。提供了以下接口，可用于控制设备以及执行 GAP 和 GATT 操作。Common IO-BLE 存储库可在上使用 GitHub。

蓝牙设备管理器：

提供了一个用于控制蓝牙设备、执行设备发现操作和其他与连接相关的任务的接口。

BLE 适配器管理器：

为特定于 BLE 的 GAP API 函数提供了一个接口。

经典蓝牙适配器管理器：

提供了一个用于控制设备的 BT 经典功能的接口。

GATT 服务器：

提供了一个使用蓝牙 GATT 服务器功能的接口。

GATT 客户端：

提供了一个使用蓝牙 GATT 客户端功能的接口。

A2DP 连接接口：

为本地设备的 A2DP 源配置文件提供了一个接口。

API 参考

如需完整的 API 参考，请参阅通用 IO - BLE API 参考。

适用于 Amazon 通用软件的通用 IO

通用 IO APIs 是 A mazon 设备通用软件所需的实现的一部分，特别是在供应商设备移植套件 (DPK) 中实现。

什么是 ACS？

亚马逊设备通用软件 (ACS) 是一种可让您更快地将亚马逊设备集成到您的设备 SDKs 上的软件。ACS 为连接、设备移植工具包 (DPK) 和多层测试套件等常见功能提供了统一的 API 集成层、经过预先验证且节省内存的组件。

资格认证计划

Amazon Common Software for Devices 资格认证计划可验证在基于微控制器的特定开发主板上运行的 ACS DPK（设备移植工具包）版本是否符合该计划已发布的最佳实践，以及是否足以通过资格认证计划规定的 ACS 强制测试。

ACS 芯片组供应商页面上会列出符合该计划资格的供应商。

有关资格认证的信息，请联系适用于设备的 ACS 团队。

开始使用 FreeRTOS

主题

- 开始使用 Quick Connect
- 探索 FreeRTOS 库
- 打造安全耐用的 AWS IoT 产品
- 开发您的 AWS IoT 应用程序

开始使用 Quick Connect

要快速探索 AWS IoT，请从 AWS Quick Connect 演示开始。Quick Connect 演示易于设置，可将合作伙伴提供的符合 FreeRTOS 标准的主板连接到 AWS IoT。

要更好地了解 AWS IoT 控制台，请按照AWS IoT 入门教程 AWS IoT 进行操作。您可以使用所选开发板的构建系统和工具来修改 Quick Connect 演示版中提供的演示源代码以连接到您的 AWS 账户。您的账户中来自 AWS IoT 控制台的数据流现已可见。

探索 FreeRTOS 库

一旦你了解了物联网设备如何协同 AWS IoT 工作，就可以开始探索 FreeRTOS 库和长期支持 (LTS) 库了。

基 AWS IoT 于 FreeRTOS 的设备的一些常用库有：

- FreeRTOS 内核
- coreMQTT
- AWS IoT Over-the-Air （ OTA ）

访问 freertos.org，获取特定于库的技术文档和演示。

打造安全耐用的 AWS IoT 产品

请参阅精选 FreeRTO AWS IoT S 集成，了解提高物联网设备软件安全性和稳定性的最佳实践。这些 FreeRTOS IoT 集成旨在结合使用 FreeRTOS 软件和合作伙伴提供的具有硬件安全功能的主板来提高安全性。按原样在生产中使用这些集成，或者将其用作设计的模型。

开发您的 AWS IoT 应用程序

请按照以下步骤为您的 AWS IoT 产品创建应用程序项目：

1. 从 freertos.org 下载最新的 FreeRTOS 或长期支持 (LTS) 版本，或者从 Freertos-LTS 存储库中克隆。 GitHub 如果可用，您还可以从 MCU 供应商的工具链中将所需的 FreeRTOS 库集成到项目中。
2. 按照《FreeRTOS 移植指南》创建项目、设置开发环境，并将 FreeRTOS 库集成到项目中。使用 FreerTOS-Libraries-Integration-Tests 存储库 GitHub 来验证移植。

AWS IoT Device Tester 适用于 FreeRTOS

适用于 FreeRTOS 的 IDT 是一种用于验证 FreeRTOS 操作系统的数据吞吐量的工具。Device Tester (IDT) 首先打开与设备的 USB 或 UART 连接。然后，它会闪烁配置为在各种条件下测试设备功能的 FreeRTOS 的图像。 AWS IoT Device Tester 套件是可扩展的，IDT 用于客户 AWS IoT 测试编排。

适用于 FreeRTOS 的 IDT 在与待测试设备连接的主机（Windows、Mac 或 Linux）上运行。IDT 配置和编排测试用例，并汇总结果。它还提供命令行界面来管理测试执行。

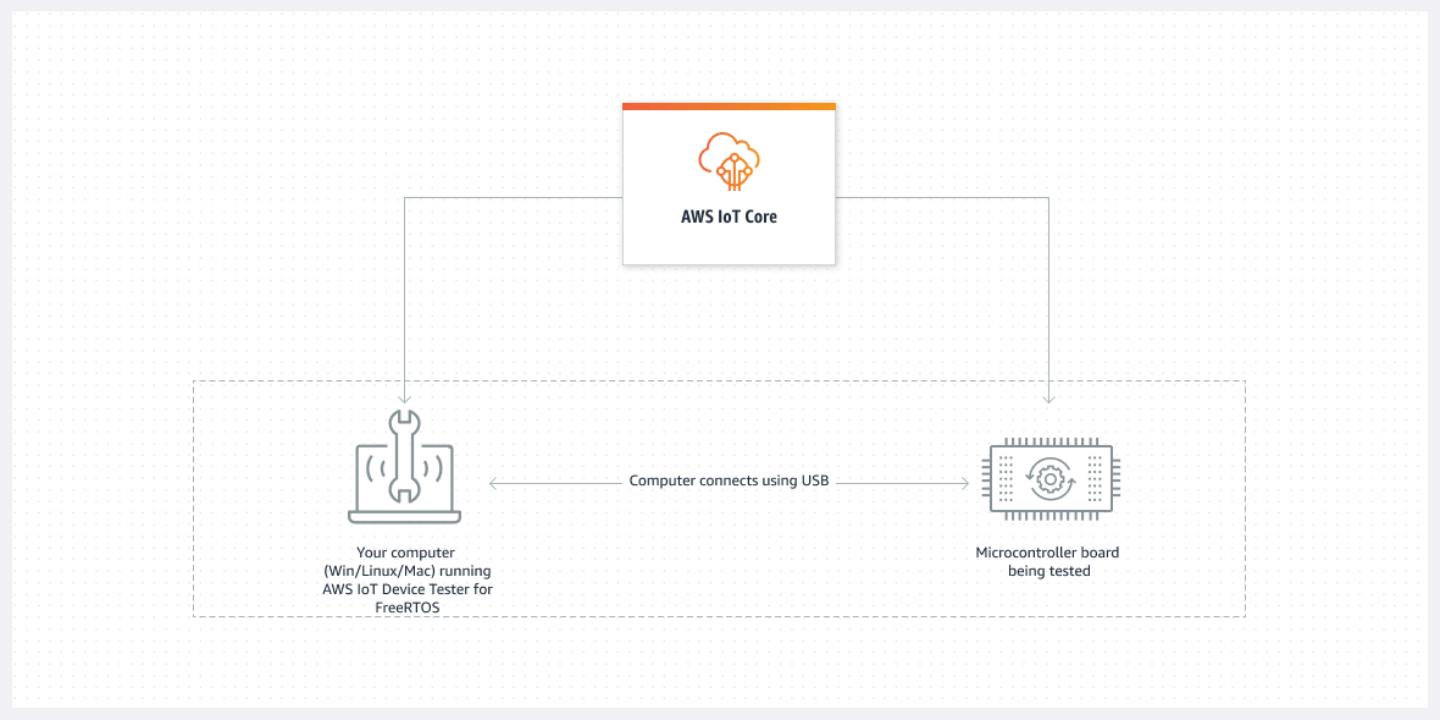
FreeRTOS 资格认证套件

适用于 FreeRTOS 的 IDT 会验证您的微控制器上的 FreeRTOS 端口，以及它是否能够以可靠和安全的方式与其进行有效通信。 AWS IoT 具体而言，它会验证是否正确实施了 FreeRTOS 库的移植层接口。它还使用执行 end-to-end测试 AWS IoT Core。例如，它验证您的主板是否能够发送和接收 MQTT 消息并正确处理它们。

FreeRTOS 资格认证 (FRQ) 2.0 套件使用来 FreeRTOS-Libraries-Integration-Tests自 FreeRTOS 资格认证指南和《设备顾问》中定义的测试用例。

适用于 FreeRTOS 的 IDT 会生成测试报告，您可以将其提交 AWS 给合作伙伴网络 (APN)，以便将您的 FreeRTOS 设备纳入合作伙伴设备目录。 AWS 有关更多信息，请参阅 AWS 设备资格认证计划。

下图显示了适用于 FreeRTOS 资格认证的测试基础设施设置。



适用于 FreeRTOS 的 IDT 将测试资源组织到测试套件和测试组中：

- 测试套件是一组测试组，用于验证设备运行的是否为特定版本的 FreeRTOS。
- 测试组是与特定功能（如 BLE 和 MQTT 消息收发）相关的一组单独测试用例。

有关更多信息，请参阅 测试套件版本

了解自定义测试套件

适用于 FreeRTOS 的 IDT 将标准化配置设置和结果格式与测试套件环境相结合。该环境允许您为设备和设备软件开发自定义测试套件。您可以添加自定义测试来用于自己的内部验证，也可以将其提供给客户进行设备验证。

自定义测试套件的配置方式决定运行自定义测试套件时必须向用户提供的设置配置。有关更多信息，请参阅 开发和运行自己的 IDT 测试套件。

支持的版本 AWS IoT Device Tester

本主题列出了支持的适用于 FreeRTOS 的 AWS IoT Device Tester 版本。作为最佳实践，我们建议您使用支持目标 FreeRTOS 版本的适用于 FreeRTOS 的 IDT 最新版本。适用于 FreeRTOS 的 IDT 的每

个版本有一个或多个支持的对应 FreeRTOS 版本。在发布新版本的 FreeRTOS 时，我们建议您下载适用于 FreeRTOS 的新版本 IDT。

下载软件即表示您同意下载档案中包含的 AWS IoT Device Tester 许可协议。

> **Note**
>
> 当你使用 AWS IoT Device Tester FreeRTOS 时，我们建议你更新到最新 FreerTOS-LTS 版本的最新补丁版本。

> **Important**
>
> 截至 2022 年 10 月， AWS IoT Device Tester AWS IoT FreeRTOS 资格认证 (FRQ) 1.0 不会生成签名的资格报告。使用 IDT AWS IoT FRQ 1.0 版本，您无法通过AWS 设备资格认证计划使新 FreeRTOS 设备在AWS 合作伙伴设备目录中列出。虽然您无法使用 IDT FRQ 1.0 确定 FreeRTOS 设备的资格，但您可以继续使用 FRQ 1.0 测试 FreeRTOS 设备。我们建议您使用 IDT FRQ 2.0 来进行资格认证，并在 AWS 合作伙伴设备目录中列出 FreeRTOS 设备。

AWS IoT Device Tester 适用于 FreeRTOS 的最新版本

使用以下链接下载适用于 FreeRTOS 的 IDT 的最新版本。

AWS IoT Device Tester 适用于 FreeRTOS 的最新版本

AWS IoT Device Tester 版本	测试套件版本	支持的 FreeRTOS 版本	下载链接	发行日期	发布说明
IDT v4.9.0	FRQ_2.5.0	• 202112.00 • 202212.00 • 202212.01 • 所有使用 FreeRTOS LTS 库的 FreeRTOS	• Linux • macOS • Windows	2023.04.04	• 支持针对使用 FreeRTOS 库的 FreeRTOS 202112、202212、2 和 FreeRTOS 202210-LT

AWS IoT Device Tester 版本	测试套件版本	支持的 FreeRTOS 版本	下载链接	发行日期	发布说明
		202210-LTS 的补丁。			S 的所有补丁进行测试。有关更多信息，请参阅 README.md。您必须在 `manifest.yml` 中包含 FreeRTOS-LTS 的补丁版本。 • 改善了 OTA E2E 测试的运行时间。 • 将 `device.json` 中列出的设备数量限制为 1。 • 少量错误修复和改进。

> **Note**
>
> 我们不建议多个用户从共享位置（如 NFS 目录或 Windows 网络共享文件夹）运行 IDT。该实践可能会导致崩溃或数据损坏。建议您将 IDT 包解压缩到本地驱动器，并在本地工作站上运行 IDT 二进制文件。

适用于 FreeRTOS 的 IDT 的早期版本

适用于 FreeRTOS 的 IDT 的以下早期版本也受支持。

for FreeR AWS IoT Device Tester TOS 的早期版本

AWS IoT Device Tester 版本	测试套件版本	支持的 FreeRTOS 版本	下载链接	发行日期	发布说明
IDT v4.8.1	FRQ_2.4.0	• 202112.00 • 202212.00 • 202212.01 • 所有使用 FreeRTOS LTS 库的 FreeRTOS 202210-LTS 的补丁。	• Linux • macOS • Windows	2023.01.23	• 有关更多信息，请参阅 README.MD 。您必须在 `manifest.yml` 中包含 FreeRTOS-LTS 的补丁版本。 • 少量错误修复和改进。
IDT v4.6.0	FRQ_2.3.0	• 202112.00 • 202212.00 • 202212.01 • 202210-LTS 使用 FreeRTOS LTS 库。	• Linux • macOS • Windows	2022.11.16	• 有关更多信息，请参阅 README.MD 。您必须在 `manifest.yml` 中包含 FreeRTOS-LTS 的补丁版本。 • 有关 FreeRTOS 202210-LTS 版本中包含的内容

AWS IoT Device Tester 版本	测试套件版本	支持的 FreeRTOS 版本	下载链接	发行日期	发布说明
					的更多信息，请参阅上的 changelog.md 文件。GitHub • 添加了通过基于 Web 的用户界面 AWS IoT Device Tester 为 FreeRTOS 配置和运行的功能。要开始，请参阅适用于 FreeRTOS 资格套件 2.0 (FRQ 2.0) 的 IDT 用户界面。 • 添加了一个选项，用于保留在运行时创建并用于测试后调试的源代码的修改副本。请参阅配置构建、刷写和

AWS IoT Device Tester 版本	测试套件版本	支持的 FreeRTOS 版本	下载链接	发行日期	发布说明
					测试设置了解更多信息。 • 添加了适用于 Java 的 IDT 客户端开发工具包支持。有关 IDT 客户端开发工具包的更多信息，请参阅开发和运行自己的 IDT 测试套件。

AWS IoT Device Tester 版本	测试套件版本	支持的 FreeRTOS 版本	下载链接	发行日期	发布说明
IDT v4.5.11	FRQ_2.2.0	• 202112.00 • 202212.00 • 202212.01 • 202210-LTS 使用 FreeRTOS LTS 库。	• Linux • macOS • Windows	2022.10.14	• 有关更多信息，请参阅 README.MD 。您必须在 `manifest.yml` 中包含 FreeRTOS-LTS 的补丁版本。 • 有关 FreeRTOS 202210-LTS 版本中包含的内容的更多信息，请参阅 GitHub 上的 changelog.md 文件。 • 少量错误修复和改进。

有关更多信息，请参阅 了解以下方面的支持政策 AWS IoT Device Tester。

不受支持的适用于 FreeRTOS 的 IDT 版本

本部分列出了不受支持的适用于 FreeRTOS 的 IDT 版本。不受支持的版本不会收到错误修复或更新。有关更多信息，请参阅 了解以下方面的支持政策 AWS IoT Device Tester。

不再支持以下版本的 IDT-FreeRTOS。

for FreeRT AWS IoT Device Tester OS 不支持的版本

AWS IoT Device Tester 版本	测试套件版本	支持的 FreeRTOS 版本	发行日期	发布说明
IDT v4.5.10	FRQ_2.1.4	• 202112.00 • 使用 FreeRTOS LTS 库的 202012-LTS。	2022.09.02	• 有关 FreeRTOS 202012-LTS 版本中包含的内容的更多信息，请参阅上的 changelog.md 文件。 GitHub • 解决了影响 OTA End to End 测试组的问题。 • 从资格认证运行中移除了 FullTransportInterfacePlainText 。通过使用 -\-group-id 标记，仍然可以将纯文本作为开发测试组运行。 • 改进了控制台以及文件输出的日志记录和可读性。 • 少量错误修复和改进。

AWS IoT Device Tester 版本	测试套件版本	支持的 FreeRTOS 版本	发行日期	发布说明
IDT v4.5.9	FRQ_2.1.3	• 202112.00 • 使用 FreeRTOS LTS 库的 202012.04-LTS。	2022.08.17	• 有关 FreeRTOS 202012.04-LTS 版本中包含的内容的更多信息，请参阅上的 changelog.md 文件。 GitHub • 解决了影响 `FreeRTOSIntegrity` 测试组的问题。 • 通过移除“MQTT Connect 指数回退重试”测试用例更新了 `FullCloudIoT` 测试组。 • 少量错误修复和改进。

AWS IoT Device Tester 版本	测试套件版本	支持的 FreeRTOS 版本	发行日期	发布说明
IDT v4.5.6	FRQ_2.1.2	• 202112.00 • 使用 FreeRTOS LTS 库的 202012.04-LTS。	2022.06.29	• 有关 FreeRTOS 202012.04-LTS 版本中包含的内容的更多信息，请参阅上的 changelog.md 文件。 GitHub • 添加了针对 AWS IoT Core Device Advisor测试主板的新测试组 `FullCloudIoT` 。 • 解决了影响 OTA E2E 测试组的问题。 • 少量错误修复和改进。

AWS IoT Device Tester 版本	测试套件版本	支持的 FreeRTOS 版本	发行日期	发布说明
IDT v4.5.5	FRQ_2.1.1	• 202112.00 • 使用 FreeRTOS LTS 库的 202012.04-LTS。	2022.06.06	• 有关 FreeRTOS 202012.04-LTS 版本中包含的内容的更多信息，请参阅上的 changelog.md 文件。 GitHub • 添加了针对 AWS IoT Core Device Advisor测试主板的新测试组 `FullCloudIoT` 。 • 解决了影响免费RTOSVersion 和免费RTOSIntegrity 测试用例的问题。 • 少量错误修复和改进。

AWS IoT Device Tester 版本	测试套件版本	支持的 FreeRTOS 版本	发行日期	发布说明
IDT v4.5.5	FRQ_2.1.0	• 202107.00 • 202112.00 • 使用 FreeRTOS LTS 库的 202012.04-LTS。	2022.05.31	• 有关 FreeRTOS 202012.04-LTS 版本中包含的内容的更多信息，请参阅上的 changelog.md 文件。 GitHub • 添加了针对 AWS IoT Core Device Advisor测试主板的新测试组 `FullCloudIoT` 。 • 少量错误修复和改进。

AWS IoT Device Tester 版本	测试套件版本	支持的 FreeRTOS 版本	发行日期	发布说明
IDT v4.5.4	FRQ_2.0.0	• 202112.00 • 使用 FreeRTOS LTS 库的 202012.04-LTS。	2022.05.09	• 有关 FreeRTOS 202012.04-LTS 版本中包含的内容的更多信息，请参阅上的 changelog.md 文件。 GitHub • 取消了仅使用存储库中的 Amazon FreeRTOS 版本对看板进行资格认证的要求。aws/amazon-freertos GitHub • 少量错误修复和改进。

AWS IoT Device Tester 版本	测试套件版本	支持的 FreeRTOS 版本	发行日期	发布说明
IDT v4.5.2	FRQ_1.6.2	202107.00	2022.01.25	• 有关 FreeRTOS 202107.00 版本中包含的内容的更多信息，请参阅上的 changelog .md 文件。GitHub • 实现用于配置自定义测试套件的新 IDT 测试编排工具。有关更多信息，请参阅配置 IDT 测试编排工具。 • 少量错误修复和改进。
IDT v4.0.3	FRQ_1.5.1	202012.00	2021.07.30	• 支持在硬件安全模块上对具有锁定凭证的设备进行资格认证。 • 少量错误修复和改进。

AWS IoT Device Tester 版本	测试套件版本	支持的 FreeRTOS 版本	发行日期	发布说明
IDT v4.3.0	FRQ_1.6.1	202107.00	2021.07.26	• 有关 FreeRTOS 202107.00 版本中包含的内容的更多信息，请参阅上的 changelog .md 文件。 GitHub • 添加了通过基于 Web 的用户界面 AWS IoT Device Tester 为 FreeRTOS 配置和运行的功能。要开始，请参阅使用 IDT 用户界面运行 FreeRTOS 资格套件。

AWS IoT Device Tester 版本	测试套件版本	支持的 FreeRTOS 版本	发行日期	发布说明

IDT v4.3.0	FRQ_1.6.0	202107.00	2021.07.21	• 有关 FreeRTOS 202107.00 版本中包含的内容的更多信息，请参阅上的 changelog.md 文件。 GitHub
				• 从 OTA 资格认证中删除以下测试用例：
				  • OTA 代理
				  • OTA 缺少文件名
				  • OTA 配置的最大数据块数量
				• 从 OTA 资格认证中移除 OTA 数据平面 Both 测试组。在 device.json 文件中，OTADataPlaneProtocol 配置现在仅接受 HTTP 或 MQTT 作为支持的值。
				• 对 userdata.json 文件中

AWS IoT Device Tester 版本	测试套件版本	支持的 FreeRTOS 版本	发行日期	发布说明
				的 freertosFileConfiguration 配置执行以下更改，以便将更改实施到 FreeRTOS 源代码： • 将为 otaAgentTestsConfig 和 otaAgentDemosConfig 指定的文件名从 aws_ota_agent_config.h 更改为 ota_config.h 。 • 添加新的 otaDemosConfig 可选配置，以便指定新的 ota_demo_config.h 文件的文件路径。 • 将一个新字段 testStart

AWS IoT Device Tester 版本	测试套件版本	支持的 FreeRTOS 版本	发行日期	发布说明
				Delayms 添加到 userdata.json ，以便指定从刷写设备到运行 FreeRTOS 测试组与启动运行测试之间的延迟。该值应以毫秒为单位。该延迟可用于让 IDT 有机会进行连接，这样就不会错过任何测试输出。

AWS IoT Device Tester 版本	测试套件版本	支持的 FreeRTOS 版本	发行日期	发布说明
IDT v4.0.1	FRQ_1.4.1	202012.00	2021.01.19	• 有关 FreeRTOS 202012.00 版本中包含的内容的更多信息，请参阅中的 changelog .md 文件。 GitHub • 引入了其他 OTA (Over-the-air) E2E (end-to-end) 测试用例。 • 支持对运行 FreeRTOS 202012.00 （使用 FreeRTOS LTS 库）的开发主板进行资格认证。 • 增加了对使用蜂窝网络连接的 FreeRTOS 开发主板进行资格认证的支持。 • 修复了 Echo 服务器配置中的一个错误。

AWS IoT Device Tester 版本	测试套件版本	支持的 FreeRTOS 版本	发行日期	发布说明
				• 使您能够使用适用于 FreeRTOS 的 AWS IoT Device Tester 开发和运行自己的自定义测试套件。有关更多信息，请参阅 开发和运行自己的 IDT 测试套件。 • 提供代码签名 IDT 应用程序，因此在 Windows 或 macOS 下运行该应用程序时无需授予权限。 • 完善了 BLE 测试结果解析逻辑。

AWS IoT Device Tester 版本	测试套件版本	支持的 FreeRTOS 版本	发行日期	发布说明
IDT v3.4.0	FRQ_1.3.0	202011.01	2020.11.05	• 有关更多详细信息，请参阅中的 Changelog.md 文件。 GitHub • 修复了 “RSA” 不是有效的 PKCS11 配置选项的错误。 • 修复了 OTA 测试后无法正确清理 Amazon S3 存储桶的错误。 • 更新以支持 FullMQTT 测试组中的新测试用例。

AWS IoT Device Tester 版本	测试套件版本	支持的 FreeRTOS 版本	发行日期	发布说明
IDT v3.3.0	FRQ_1.2.0	202007.00	2020.09.17	• 有关更多详细信息，请参阅中的 Changelog.md 文件。 GitHub • 用于验证 Over The Air (OTA) 更新暂停和恢复功能的新 end-to-end测试。 • 修复了导致 eu-central-1 区域中的用户无法通过 OTA 测试的配置验证的错误。 • 向 `run-suite` 命令添加了 `--update-idt` 参数。您可以使用此选项设置对 IDT 更新提示的响应。 • 向 `run-suite` 命令添加了 `--update-managed-policy` 参数。您可以使用此选项设置对托管策略

AWS IoT Device Tester 版本	测试套件版本	支持的 FreeRTOS 版本	发行日期	发布说明
				更新提示的响应。 • 内部改进和错误修复，包括： • 对于自动更新测试套件，改进了配置文件升级。

AWS IoT Device Tester 版本	测试套件版本	支持的 FreeRTOS 版本	发行日期	发布说明

IDT v3.0.2	FRQ_1.0.1	202002.00		• 有关更多信息，请参阅中的 Changelog.md 文件。 GitHub
				• 在 IDT 内添加测试套件的自动更新。IDT 现在可以下载适用于您的 FreeRTOS 版本的最新测试套件。使用此功能，您可以：
				  • 使用 `upgrade-test-suite` 命令下载最新的测试套件。
				  • 启动 IDT 时，通过设置标志下载最新的测试套件。
				  使用`-u` *`flag`*选项，其中*`flag`*可以是 “y” 表示始终下

AWS IoT Device Tester 版本	测试套件版本	支持的 FreeRTOS 版本	发行日期	发布说明
				载，或者 n “” 使用现有版本。 当有多个测试套件版本可用时，除非您在启动 IDT 时指定测试套件 ID，否则将使用最新版本。 • 使用新的 `list-supported-versions` 选项列出已安装的 IDT 版本支持的 FreeRTOS 和测试套件版本。 • 列出组中的测试用例并运行单个测试。 测试套件使用 `major.minor.patch` 格式进行版本化，从 1.0.0 开始。

AWS IoT Device Tester 版本	测试套件版本	支持的 FreeRTOS 版本	发行日期	发布说明
				• 添加了 `list-supported-products` 命令 – 列出已安装的 IDT 版本支持的 FreeRTOS 和测试套件版本。 • 添加 `list-test-cases` 命令 – 列出测试组中可用的测试用例。 • 添加 `test-id` 命令的 `run-suite` 选项 – 使用此选项可在测试组中运行单个测试用例。

AWS IoT Device Tester 版本	测试套件版本	支持的 FreeRTOS 版本	发行日期	发布说明
IDT v1.7.1	FRQ_1.0.0	202002.00		• 有关更多详细信息，请参阅中的 Changelog.md 文件。 GitHub • 支持 over-the-air (OTA) end-to-end 测试用例的自定义代码签名方法，因此您可以使用自己的代码签名命令和脚本对 OTA 负载进行签名。 • 在测试开始之前添加串行端口的预检查。如果串行端口在 `device.json` 文件中配置不当，测试将迅速失败，并显示改进的错误消息。 • 添加了运行 AWS IoT Device Tester 所需权限的 AWS 托管策略 `AWSIoTDeviceTesterForFreeRT`

AWS IoT Device Tester 版本	测试套件版本	支持的 FreeRTOS 版本	发行日期	发布说明
				`OSFullAccess` 。如果新版本需要额外的权限，我们会将这些权限添加到此托管策略中，以便您不必手动更新 IAM 权限。 • 结果目录中命为 `AFQ_Report.xml` 的文件现在是 `FRQ_Report.xml` 。
IDT v1.6.2	FRQ_1.0.0	202002.00		• 支持通过 HTTPS 进行的 OTA 的可选测试，从而使您的 Amazon FreeRTOS 开发主板具有资格。 • 在测试中支持 AWS IoT ATS 端点。 • 支持在测试套件开始之前告知用户最新的 IDT 版本的功能。

AWS IoT Device Tester 版本	测试套件版本	支持的 FreeRTOS 版本	发行日期	发布说明
IDT v1.5.2	FRQ_1.0.0	201910.00		• 支持具有安全元件（板载密钥）的 FreeRTOS 设备的资格认证。 • 支持适用于 SecureSockets 和 Wi-Fi 测试组的可配置 Echo 服务器端口。 • 支持使用超时乘数标志来增大超时值，这在对超时相关错误进行故障排除时非常有用。 • 增加了日志解析的错误修复。 • 在测试中支持 iot ats 终端节点。

AWS IoT Device Tester 版本	测试套件版本	支持的 FreeRTOS 版本	发行日期	发布说明
IDT v1.4.1	FRQ_1.0.0	201908.00		• 增加了对新 PKCS11 库和测试用例更新的支持。 • 引入了可操作的错误代码。有关更多信息，请参阅 IDT 错误代码 • 更新了用于运行 IDT 的 IAM 策略。
IDT v1.3.2	FRQ_1.0.0	201906.00		• 增加了对测试低功耗蓝牙 (BLE) 功能的支持。 • 改善了 IDT 命令行界面 (CLI) 命令的用户体验。 • 更新了用于运行 IDT 的 IAM 策略。
IDT-FreeRTOS v1.2	FRQ_1.0.0	• FreeRTOS v1.4.8 • FreeRTOS v1.4.9		添加了通过 CMAKE 构建系统测试 FreeRTOS 设备的支持。
IDT-FreeRTOS v1.1	FRQ_1.0.0			

AWS IoT Device Tester 版本	测试套件版本	支持的 FreeRTOS 版本	发行日期	发布说明
IDT-FreeRTOS v1.0	FRQ_1.0.0			

下载适用于 FreeRTOS 的 IDT

本主题介绍下载适用于 FreeRTOS 的 IDT 的选项。您可以使用以下软件下载链接之一，也可以按照说明以编程方式下载 IDT。

> **Important**
>
> 截至 2022 年 10 月， AWS IoT Device Tester AWS IoT FreeRTOS 资格认证 (FRQ) 1.0 不会生成已签署的资格报告。使用 IDT AWS IoT FRQ 1.0 版本，您无法通过AWS 设备资格认证计划将新 FreeRTOS 设备列入合作伙伴设备目录。AWS 虽然您无法使用 IDT FRQ 1.0 确定 FreeRTOS 设备的资格，但您可以继续使用 FRQ 1.0 测试 FreeRTOS 设备。我们建议您使用 IDT FRQ 2.0 来进行资格认证，并在 AWS 合作伙伴设备目录中列出 FreeRTOS 设备。

主题

- 手动下载 IDT
- 以编程方式下载 IDT

下载软件即表示您同意下载档案中包含的 AWS IoT Device Tester 许可协议。

> **Note**
>
> IDT 不支持由多个用户从共享位置（如 NFS 目录或 Windows 网络共享文件夹）运行。建议您将 IDT 包解压缩到本地驱动器，并在本地工作站上运行 IDT 二进制文件。

手动下载 IDT

本主题列出了支持的适用于 FreeRTOS 的 IDT 版本。作为最佳实践，我们建议您使用支持目标版本的 FreeRTOS 的最新版本。 AWS IoT Device Tester 新版本的 FreeRTOS 可能要求您下载 AWS IoT

Device Tester的新版本。如果与您 AWS IoT Device Tester 正在使用的FreeRTOS版本不兼容，则在开始测试运行时会收到通知。

请参阅 支持的版本 AWS IoT Device Tester。

以编程方式下载 IDT

IDT 提供了一个 API 操作，您可以使用该操作来检索 URL，也可以在其中以编程方式下载 IDT。您还可以使用此 API 操作来检查是否具有最新版本的 IDT。此 API 操作具有以下端点。

```
https://download.devicetester.iotdevicesecosystem.amazonaws.com/latestidt
```

要调用此 API 操作，您必须具有执行 **iot-device-tester:LatestIdt** 操作的权限。包括您的 AWS 签名，并以此`iot-device-tester`作为服务名称

API 请求

HostOs — 主机的操作系统。从以下选项中进行选择：

- `mac`
- `linux`
- `windows`

TestSuiteType — 测试套件的类型。选择以下选项：

FR – 适用于 FreeRTOS 的 IDT

ProductVersion

（可选）FreeRTOS 的版本。该服务返回适用于此 FreeRTOS 版本的 IDT 的最新兼容版本。如果不指定此选项，则该服务将返回 IDT 的最新版本。

API 响应

API 响应采用以下格式。`DownloadURL` 包括一个 zip 文件。

```
{
    "Success": True or False,
    "Message": Message,
    "LatestBk": {
        "Version": The version of the IDT binary,
        "TestSuiteVersion": The version of the test suite,
```

```
        "DownloadURL": The URL to download the IDT Bundle, valid for one hour
    }
}
```

示例

您可以参考以下示例以编程方式下载 IDT。这些示例使用您在 AWS_ACCESS_KEY_ID 和 AWS_SECRET_ACCESS_KEY 环境变量中存储的凭证。要遵循最佳安全实践，请勿在代码中存储您的凭证。

Example

示例：使用 cURL 版本 7.75.0 或更高版本下载（Mac 和 Linux）

如果您的 cURL 版本为 7.75.0 或更高版本，则可以使用 aws-sigv4 标记对 API 请求进行签名。该示例使用 jq 来解析响应中的下载 URL。

> **Warning**
>
> 该aws-sigv4标志要求 curl GET 请求的查询参数按HostOs/ProductVersion/TestSuiteType或HostOs/TestSuiteType的顺序排列。如果参数顺序不符合要求，则会导致从 API Gateway 获得不匹配的规范字符串签名错误。
> 如果包含可选参数 ProductVersion，则必须使用支持的产品版本，如 for FreeRTOS AWS IoT Device Tester 的支持版本中所述。

- 替换*us-west-2*为你的 AWS 区域。有关区域代码的列表，请参阅区域端点。
- *linux*替换为主机的操作系统。
- *202107.00*替换为你的 FreeRTOS 版本。

```
url=$(curl --request GET "https://
download.devicetester.iotdevicesecosystem.amazonaws.com/latestidt?
HostOs=linux&ProductVersion=202107.00&TestSuiteType=FR" \
--user $AWS_ACCESS_KEY_ID:$AWS_SECRET_ACCESS_KEY \
--aws-sigv4 "aws:amz:us-west-2:iot-device-tester" \
| jq -r '.LatestBk["DownloadURL"]')

curl $url --output devicetester.zip
```

Example

示例：使用早期版本的 cURL 下载（Mac 和 Linux）

您可以将以下 curl 命令与您签名并计算的 AWS 签名一起使用。有关如何签名和计算签名的更多信息，请参阅 AWS 签名 AWS API 请求。

- *linux*替换为主机的操作系统。
- *Timestamp*替换为日期和时间，例如**20220210T004606Z**。
- *Date*替换为日期，例如**20220210**。
- 替换*AWSRegion*为你的 AWS 区域。有关区域代码的列表，请参阅区域端点。
- *AWSSignature*替换为您生成的AWS 签名。

```
curl --location --request GET 'https://
download.devicetester.iotdevicesecosystem.amazonaws.com/latestidt?
HostOs=linux&TestSuiteType=FR' \
--header 'X-Amz-Date: Timestamp \
--header 'Authorization: AWS4-HMAC-SHA256 Credential=$AWS_ACCESS_KEY_ID/Date/AWSRegion/
iot-device-tester/aws4_request, SignedHeaders=host;x-amz-date, Signature=AWSSignature'
```

Example

示例：使用 Python 脚本下载

此示例使用 Python 请求库。此示例改编自AWS 通用参考中的 Python 示例，用于签署 AWS API 请求。

- 请将 *us-west-2* 替换为您的区域。有关区域代码的列表，请参阅区域端点。
- *linux*替换为主机的操作系统。

```
# Copyright 2010-2022 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#
# This file is licensed under the Apache License, Version 2.0 (the "License").
# You may not use this file except in compliance with the License. A copy of the
#License is located at
#
# http://aws.amazon.com/apache2.0/
#
# This file is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS
```

```
# OF ANY KIND, either express or implied. See the License for the specific
# language governing permissions and limitations under the License.

# See: http://docs.aws.amazon.com/general/latest/gr/sigv4_signing.html
# This version makes a GET request and passes the signature
# in the Authorization header.
import sys, os, base64, datetime, hashlib, hmac
import requests # pip install requests
# ************* REQUEST VALUES *************
method = 'GET'
service = 'iot-device-tester'
host = 'download.devicetester.iotdevicesecosystem.amazonaws.com'
region = 'us-west-2'
endpoint = 'https://download.devicetester.iotdevicesecosystem.amazonaws.com/latestidt'
request_parameters = 'HostOs=linux&TestSuiteType=FR'

# Key derivation functions. See:
# http://docs.aws.amazon.com/general/latest/gr/signature-v4-examples.html#signature-v4-
examples-python
def sign(key, msg):
    return hmac.new(key, msg.encode('utf-8'), hashlib.sha256).digest()

def getSignatureKey(key, dateStamp, regionName, serviceName):
    kDate = sign(('AWS4' + key).encode('utf-8'), dateStamp)
    kRegion = sign(kDate, regionName)
    kService = sign(kRegion, serviceName)
    kSigning = sign(kService, 'aws4_request')
    return kSigning

# Read AWS access key from env. variables or configuration file. Best practice is NOT
# to embed credentials in code.
access_key = os.environ.get('AWS_ACCESS_KEY_ID')
secret_key = os.environ.get('AWS_SECRET_ACCESS_KEY')
if access_key is None or secret_key is None:
    print('No access key is available.')
    sys.exit()

# Create a date for headers and the credential string
t = datetime.datetime.utcnow()
amzdate = t.strftime('%Y%m%dT%H%M%SZ')
datestamp = t.strftime('%Y%m%d') # Date w/o time, used in credential scope

# ************* TASK 1: CREATE A CANONICAL REQUEST *************
# http://docs.aws.amazon.com/general/latest/gr/sigv4-create-canonical-request.html
```

```
# Step 1 is to define the verb (GET, POST, etc.)--already done.
# Step 2: Create canonical URI--the part of the URI from domain to query
# string (use '/' if no path)
canonical_uri = '/latestidt'
# Step 3: Create the canonical query string. In this example (a GET request),
# request parameters are in the query string. Query string values must
# be URL-encoded (space=%20). The parameters must be sorted by name.
# For this example, the query string is pre-formatted in the request_parameters
 variable.
canonical_querystring = request_parameters
# Step 4: Create the canonical headers and signed headers. Header names
# must be trimmed and lowercase, and sorted in code point order from
# low to high. Note that there is a trailing \n.
canonical_headers = 'host:' + host + '\n' + 'x-amz-date:' + amzdate + '\n'
# Step 5: Create the list of signed headers. This lists the headers
# in the canonical_headers list, delimited with ";" and in alpha order.
# Note: The request can include any headers; canonical_headers and
# signed_headers lists those that you want to be included in the
# hash of the request. "Host" and "x-amz-date" are always required.
signed_headers = 'host;x-amz-date'
# Step 6: Create payload hash (hash of the request body content). For GET
# requests, the payload is an empty string ("").
payload_hash = hashlib.sha256(('').encode('utf-8')).hexdigest()
# Step 7: Combine elements to create canonical request
canonical_request = method + '\n' + canonical_uri + '\n' + canonical_querystring + '\n'
 + canonical_headers + '\n' + signed_headers + '\n' + payload_hash

# ************* TASK 2: CREATE THE STRING TO SIGN*************
# Match the algorithm to the hashing algorithm you use, either SHA-1 or
# SHA-256 (recommended)
algorithm = 'AWS4-HMAC-SHA256'
credential_scope = datestamp + '/' + region + '/' + service + '/' + 'aws4_request'
string_to_sign = algorithm + '\n' +  amzdate + '\n' +  credential_scope + '\n' +
 hashlib.sha256(canonical_request.encode('utf-8')).hexdigest()
# ************* TASK 3: CALCULATE THE SIGNATURE *************
# Create the signing key using the function defined above.
signing_key = getSignatureKey(secret_key, datestamp, region, service)
# Sign the string_to_sign using the signing_key
signature = hmac.new(signing_key, (string_to_sign).encode('utf-8'),
 hashlib.sha256).hexdigest()

# ************* TASK 4: ADD SIGNING INFORMATION TO THE REQUEST *************
# The signing information can be either in a query string value or in
# a header named Authorization. This code shows how to use a header.
```

```
# Create authorization header and add to request headers
authorization_header = algorithm + ' ' + 'Credential=' + access_key + '/' +
 credential_scope + ', ' +  'SignedHeaders=' + signed_headers + ', ' + 'Signature=' +
 signature
# The request can include any headers, but MUST include "host", "x-amz-date",
# and (for this scenario) "Authorization". "host" and "x-amz-date" must
# be included in the canonical_headers and signed_headers, as noted
# earlier. Order here is not significant.
# Python note: The 'host' header is added automatically by the Python 'requests'
 library.
headers = {'x-amz-date':amzdate, 'Authorization':authorization_header}

# ************* SEND THE REQUEST *************
request_url = endpoint + '?' + canonical_querystring
print('\nBEGIN REQUEST++++++++++++++++++++++++++++++++++++')
print('Request URL = ' + request_url)
response = requests.get(request_url, headers=headers)
print('\nRESPONSE++++++++++++++++++++++++++++++++++++')
print('Response code: %d\n' % response.status_code)
print(response.text)

download_url = response.json()["LatestBk"]["DownloadURL"]
r = requests.get(download_url)
open('devicetester.zip', 'wb').write(r.content)
```

带有 FreeRTOS 资格套件 2.0 的 IDT (FRQ 2.0)

FreeRTOS 资格认证套件 2.0 是 FreeRTOS 资格认证套件的更新版本。我们建议开发者使用 FRQ 2.0，因为它包含相关的测试用例，可以验证运行 FreeRTOS 长期支持 (LTS) 库的设备。

FreeRTOS 的 IDT 会验证你的微控制器上的 FreeRTOS 端口，以及它是否与之有效通信。 AWS IoT 具体而言，它会验证移植层与 FreeRTOS 库的接口，以及 FreeRTOS 测试存储库的实现是否正确。它还使用执行 end-to-end测试 AWS IoT Core。IDT 为 FreeRTOS 运行的测试在 FreeRTOS 存储库中定义。 GitHub

适用于 FreeRTOS 的 IDT 作为嵌入式应用程序运行测试，它会在待测微控制器设备上执行刷写。应用程序二进制映像包括 FreeRTOS、移植的 FreeRTOS 接口以及主板设备驱动程序。测试的目的是验证移植的 FreeRTOS 接口在设备驱动程序上正常工作。

IDT for FreeRTOS 会生成测试报告，您可以提交这些报告 AWS IoT 以将您的硬件列入 AWS 合作伙伴设备目录。有关更多信息，请参阅 AWS 设备资格认证计划。

适用于 FreeRTOS 的 IDT 在与待测试设备连接的主机（Windows、Mac 或 Linux）上运行。IDT 配置和编排测试用例并汇总结果。它还提供命令行界面来管理测试运行过程。

为了测试您的设备，适用于 FreeRTOS 的 IDT 会创建 AWS IoT 诸如东西、FreeRTOS 群组、Lambda 函数之类的资源。要创建这些资源，适用于 FreeRTOS 的 IDT 使用中`config.json`配置 AWS 的凭据代表您进行 API 调用。这些资源将在测试过程的不同时间进行预置。

当您在主机上运行适用于 FreeRTOS 的 IDT 时，它将执行以下步骤：

1. 加载和验证您的设备和凭证配置。
2. 使用所需的本地资源和云资源执行选定测试。
3. 清除本地资源和云资源。
4. 生成测试报告，指明您的主板是否已通过资格认证所需的测试。

设置 LTS 资格先决条件

本节介绍使用测试微控制器的 AWS IoT Device Tester先决条件。

FreeRTOS 资格认证准备

> **Note**
>
> AWS IoT Device Tester for FreeRTOS 强烈建议使用最新版本的 FreerTOS-LTS 版本的最新补丁版本。

IDT for FRQ 2.0 是 FreeRTOS 的资格认证计划。在运行 IDT FRQ 2.0 进行资格认证之前，您必须完成《FreeRTOS 资格认证指南》中的主板资格认证。要移植库、测试和设置 `manifest.yml`，请参阅《FreeRTOS 移植指南》中的 FreeRTOS 库移植。FRQ 2.0 包含不同的资格认证流程。有关详细信息，请参阅《FreeRTOS 资格认证指南》中的资格认证最新更改。

必须存在 FreeRTOS-Libraries-Integration-Tests 存储库才能运行 IDT。有关如何克隆此存储库并将其移植到源项目的信息，请参阅 README.md。 FreeRTOS-Libraries-Integration-Tests必须包含`manifest.yml`位于项目根目录中的，IDT 才能运行。

> **Note**
>
> IDT 取决于 `UNITY_OUTPUT_CHAR` 的测试存储库的实现。测试输出日志和设备日志不得相互交错。有关更多详细信息，请参阅《FreeRTOS 移植指南》中的实现库日志记录宏部分。

下载适用于 FreeRTOS 的 IDT

为了执行资格认证测试，FreeRTOS 的每个版本都有对应的适用于 FreeRTOS 的 IDT 版本。从 FreeRTOS 支持的版本中下载适用于 FreeRTOS 的 IDT 的相应版本。 AWS IoT Device Tester

将适用于 FreeRTOS 的 IDT 提取到文件系统上您具有读写权限的位置。由于 Microsoft Windows 对路径长度具有字符数限制，因此将适用于 FreeRTOS 的 IDT 提取到根目录，如 `C:\` 或 `D:\`。

> **Note**
>
> 多个用户不得从共享位置（如 NFS 目录或 Windows 网络共享文件夹）运行 IDT。这会会导致崩溃或数据损坏。我们建议您将 IDT 包解压缩到本地驱动器。

下载 Git

作为先决条件，IDT 必须安装 Git 才能确保源代码的完整性。

按照GitHub指南中的说明安装 Git。要验证当前安装的 Git 版本，请在终端输入命令 `git --version`。

> **Warning**
>
> IDT 使用 Git 来匹配目录的干净或占用状态。如果未安装 Git，`FreeRTOSIntegrity` 测试组要么会失败，要么会无法按预期运行。如果 IDT 返回 `git executable not found` 或 `git command not found` 之类的错误，请安装或重新安装 Git，然后重试。

主题

- 创建一个 AWS 账户
- AWS IoT Device Tester 托管策略
- （可选）安装 AWS Command Line Interface

创建一个 AWS 账户

> **Note**
>
> 只有以下内容支持完整的 IDT 资格套件 AWS 区域
>
> - 美国东部（弗吉尼亚州北部）
> - 美国西部（俄勒冈）
> - 亚太地区（东京）
> - 欧洲地区（爱尔兰）

为了测试您的设备，适用于 FreeRTOS 的 IDT 会创建诸如 AWS IoT 东西、FreeRTOS 组和 Lambda 函数之类的资源。要创建这些资源，IDT for FreeRTOS 需要您创建和配置 AWS 一个账户，以及一项授予 IDT for FreeRTOS 在运行测试时代表您访问资源的权限的 IAM 策略。

以下步骤用于创建和配置您的 AWS 账户。

1. 如果您已经有一个 AWS 帐户，请跳到下一步。创建一个 AWS 账户。
2. 按照创建 IAM 角色中的步骤操作。此时请勿添加权限或策略。
3. 要运行 OTA 资格认证测试，请转至步骤 4。否则，请转到步骤 5。
4. 将 OTA IAM 权限内联策略附加到您的 IAM 角色。

   a.

   > **Important**
   >
   > 以下策略模板授予创建角色、创建策略和将策略附加到角色的 IDT 权限。适用于 FreeRTOS 的 IDT 将这些权限用于创建角色的测试。尽管策略模板不向用户提供管理员权限，但这些权限可用于获得对您 AWS 账户的管理员访问权限。

   b. 执行以下步骤，将必要权限附加到您的 IAM 角色：

      i. 在权限页面上，请选择添加权限。

      ii. 选择创建内联策略。

      iii. 选择 JSON 选项卡，然后将以下权限复制到 JSON 文本框中。如果您不在中国区域，请使用大多数区域下的模板。如果您在中国区域，请使用北京和宁夏区域下的模板。

Most Regions

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iotdeviceadvisor:*",
            "Resource": [
                "arn:aws:iotdeviceadvisor:*:*:suiterun/*/*",
                "arn:aws:iotdeviceadvisor:*:*:suitedefinition/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::*:role/idt*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService":
 "iotdeviceadvisor.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "execute-api:Invoke*",
                "iam:ListRoles",
                "iot:Connect",
                "iot:CreateJob",
                "iot:DeleteJob",
                "iot:DescribeCertificate",
                "iot:DescribeEndpoint",
                "iot:DescribeJobExecution",
                "iot:DescribeJob",
                "iot:DescribeThing",
                "iot:GetPolicy",
                "iot:ListAttachedPolicies",
                "iot:ListCertificates",
                "iot:ListPrincipalPolicies",
                "iot:ListThingPrincipals",
                "iot:ListThings",
```

```
                "iot:Publish",
                "iot:UpdateThingShadow",
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:DescribeLogGroups",
                "logs:DescribeLogStreams",
                "logs:PutLogEvents",
                "logs:PutRetentionPolicy"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iotdeviceadvisor:*",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "logs:DeleteLogGroup",
            "Resource": "arn:aws:logs:*:*:log-group:/aws/iot/
deviceadvisor/*"
        },
        {
            "Effect": "Allow",
            "Action": "logs:GetLogEvents",
            "Resource": "arn:aws:logs:*:*:log-group:/aws/iot/
deviceadvisor/*:log-stream:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreatePolicy",
                "iam:DetachRolePolicy",
                "iam:DeleteRolePolicy",
                "iam:DeletePolicy",
                "iam:CreateRole",
                "iam:DeleteRole",
                "iam:AttachRolePolicy"
            ],
            "Resource": [
                "arn:aws:iam::*:policy/idt*",
                "arn:aws:iam::*:role/idt*"
            ]
        },
```

```
        {
            "Effect": "Allow",
            "Action": [
                "ssm:GetParameters"
            ],
            "Resource": [
                "arn:aws:ssm:*::parameter/aws/service/ami-amazon-linux-
latest/amzn2-ami-hvm-x86_64-gp2"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeInstances",
                "ec2:RunInstances",
                "ec2:CreateSecurityGroup",
                "ec2:CreateTags",
                "ec2:DeleteTags"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateKeyPair",
                "ec2:DeleteKeyPair"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:key-pair/idt-ec2-ssh-key-*"
            ]
        },
        {
            "Effect": "Allow",
            "Condition": {
                "StringEqualsIgnoreCase": {
                    "aws:ResourceTag/Owner": "IoTDeviceTester"
                }
            },
            "Action": [
                "ec2:TerminateInstances",
                "ec2:DeleteSecurityGroup",
                "ec2:AuthorizeSecurityGroupIngress",
```

```
                "ec2:RevokeSecurityGroupIngress"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Beijing and Ningxia Regions

以下策略模板可在北京和宁夏区域中使用。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreatePolicy",
                "iam:DetachRolePolicy",
                "iam:DeleteRolePolicy",
                "iam:DeletePolicy",
                "iam:CreateRole",
                "iam:DeleteRole",
                "iam:AttachRolePolicy"
            ],
            "Resource": [
                "arn:aws-cn:iam::*:policy/idt*",
                "arn:aws-cn:iam::*:role/idt*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ssm:GetParameters"
            ],
            "Resource": [
                "arn:aws-cn:ssm:*::parameter/aws/service/ami-amazon-
linux-latest/amzn2-ami-hvm-x86_64-gp2"
            ]
        },
        {
```

```
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeInstances",
                "ec2:RunInstances",
                "ec2:CreateSecurityGroup",
                "ec2:CreateTags",
                "ec2:DeleteTags"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateKeyPair",
                "ec2:DeleteKeyPair"
            ],
            "Resource": [
                "arn:aws-cn:ec2:*:*:key-pair/idt-ec2-ssh-key-*"
            ]
        },
        {
            "Effect": "Allow",
            "Condition": {
                "StringEqualsIgnoreCase": {
                    "aws-cn:ResourceTag/Owner": "IoTDeviceTester"
                }
            },
            "Action": [
                "ec2:TerminateInstances",
                "ec2:DeleteSecurityGroup",
                "ec2:AuthorizeSecurityGroupIngress",
                "ec2:RevokeSecurityGroupIngress"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

iv. 完成后，选择查看策略。

v. 输入IDTFreeRTOSIAMPermissions作为策略名称。

vi. 选择创建策略。

5. 将AWSIoTDeviceTesterForFreeRTOSFull访问权限授予您的 IAM 角色。

a. 将必要的权限附加到您的 IAM 角色：

i. 在权限页面上，请选择添加权限。

ii. 选择附加策略。

iii. 搜索AWSIoTDeviceTesterForFreeRTOSFull访问策略。选中该复选框。

b. 选择添加权限。

6. IDT 的导出凭证。有关详细信息，请参阅获取用于 CLI 访问的 IAM 角色凭证。

AWS IoT Device Tester 托管策略

`AWSIoTDeviceTesterForFreeRTOSFullAccess`托管策略包含版本检查、auto update 功能和指标收集的以下 AWS IoT Device Tester 权限。

- `iot-device-tester:SupportedVersion`

  授 AWS IoT Device Tester 予获取受支持产品、测试套件和 IDT 版本列表的权限。
- `iot-device-tester:LatestIdt`

  授 AWS IoT Device Tester 予获取可供下载的最新 IDT 版本的权限。
- `iot-device-tester:CheckVersion`

  授 AWS IoT Device Tester 予检查 IDT、测试套件和产品的版本兼容性的权限。
- `iot-device-tester:DownloadTestSuite`

  授 AWS IoT Device Tester 予下载测试套件更新的权限。
- `iot-device-tester:SendMetrics`

  授 AWS 予收集 AWS IoT Device Tester 内部使用情况指标的权限。

（可选）安装 AWS Command Line Interface

您可能更喜欢使用 AWS CLI 来执行某些操作。如果您没有安装 AWS CLI ，请按照安装 AWS CLI 中的说明执行操作。

通过 AWS CLI aws configure从命令行运行来配置要使用的 AWS 区域。有关支持 IDT for FreeRTOS 的 AWS 区域的信息，AWS 请参阅区域和终端节点。有关 aws configure 的更多信息，请参阅使用 aws configure 进行快速配置。

首次测试您的微控制器板

您可以使用适用于 FreeRTOS 的 IDT 来测试 FreeRTOS 库的实现。移植完主板设备驱动程序的 FreeRTOS 库后， AWS IoT Device Tester 使用在微控制器主板上运行资格测试。

添加库移植层

要为您的设备移植 FreeRTOS，请参阅《FreeRTOS 移植指南》。在实现 FreeRTOS 测试存储库和移植 FreeRTOS 层时，您必须向 `manifest.yml` 提供每个库（包括测试存储库）的路径。此文件将位于源代码的根目录中。有关详细信息，请参阅清单文件说明。

配置您的 AWS 凭据 AWS IoT Device Tester 以与 AWS 云端通信

您需要配置您的 AWS 凭据才能与 AWS 云端通信。 AWS IoT Device Tester 有关更多信息，请参阅设置用于开发的 AWS 凭证和区域。在*devicetester_extract_location*/devicetester_freertos_*[win|mac|linux]*/configs/config.json配置文件中指定了有效的 AWS 凭据。

```
"auth": {
   "method": "environment"
}

"auth": {
    "method": "file",
    "credentials": {
        "profile": "<your-aws-profile>"
    }
}
```

该config.json文件的auth属性具有控制 AWS 身份验证的方法字段，可以声明为文件或环境。将该字段设置为 environment 会从主机的环境变量中提取您的 AWS 凭据。将该字段设置为文件会从 `.aws/credentials` 配置文件中导入指定的配置文件。

主题

- 在适用于 FreeRTOS 的 IDT 中创建设备池

- 配置构建、刷写和测试设置

在适用于 FreeRTOS 的 IDT 中创建设备池

要测试的设备排列在设备池中。每个设备池包含一个或多个相同的设备。您可以配置适用于 FreeRTOS 的 IDT 来测试某个池中的单个设备或多个设备。为了加快资格认证测试过程，适用于 FreeRTOS 的 IDT 可以并行测试具有相同规格的设备。它使用轮询方法，在设备池中的各个设备上执行不同的测试组。

device.json 文件的顶层有一个数组。每个数组属性都是一个新的设备池。每个设备池都有一个设备数组属性，该属性声明了多个设备。在模板中有一个设备池，该设备池中只有一个设备。您可以通过编辑 configs 文件夹中 device.json 模板的 devices 部分，将一个或多个设备添加到设备池中。

> **Note**
>
> 同一个池中的所有设备必须具有相同的技术规格和 SKU。要允许针对不同测试组并行构建源代码，适用于 FreeRTOS 的 IDT 需要将源代码复制到适用于 FreeRTOS 的 IDT 提取文件夹的结果文件夹中。您在构建或刷写命令中使用 testdata.sourcePath 变量来引用源代码路径。适用于 FreeRTOS 的 IDT 使用所复制源代码的临时路径来替换此变量。有关更多信息，请参阅 适用于 FreeRTOS 变量的 IDT。

以下 device.json 示例文件用于创建具有多个设备的设备池。

```
[
    {
        "id": "pool-id",
        "sku": "sku",
        "features": [
           {
              "name": "Wifi",
              "value": "Yes | No"
           },
           {
              "name": "Cellular",
              "value": "Yes | No"
           },
           {
              "name": "BLE",
              "value": "Yes | No"
```

```
          },
          {
             "name": "PKCS11",
             "value": "RSA | ECC | Both"
          },
          {
              "name": "OTA",
              "value": "Yes | No",
              "configs": [
              {
                  "name": "OTADataPlaneProtocol",
                  "value": "MQTT | HTTP | None"
              }
            ]
          },
          {
             "name": "KeyProvisioning",
             "value": "Onboard | Import | Both | No"
          }
        ],
        "devices": [
          {
            "id": "device-id",
            "connectivity": {
              "protocol": "uart",
              "serialPort": "/dev/tty*"
            },
            "secureElementConfig" : {
              "publicKeyAsciiHexFilePath": "absolute-path-to/public-key-txt-file:
 contains-the-hex-bytes-public-key-extracted-from-onboard-private-key",
              "publiDeviceCertificateArn": "arn:partition:iot:region:account-
id:resourcetype:resource:qualifier",
              "secureElementSerialNumber": "secure-element-serialNo-value",
              "preProvisioned"           : "Yes | No",
              "pkcs11JITPCodeVerifyRootCertSupport": "Yes | No"
            },
            "identifiers": [
              {
                "name": "serialNo",
                "value": "serialNo-value"
              }
            ]
          }
        ]
```

```
    }
]
```

在 device.json 文件中使用以下属性：

id

用户定义的字母数字 ID，用于唯一地标识设备池。属于同一个池的设备必须具有相同的类型。运行一组测试时，池中的设备将用于对工作负载进行并行化处理。

sku

唯一标识您正在测试的主板的字母数字值。该 SKU 用于跟踪符合条件的主板。

> **Note**
>
> 如果您想在 AWS 合作伙伴设备目录中发布您的主板，则在此处指定的 SKU 必须与您在发布过程中使用的 SKU 相匹配。

features

包含设备支持的功能的阵列。 AWS IoT Device Tester 使用此信息来选择要运行的资格测试。

支持的值为：

Wifi

指示您的主板是否具有 Wi-Fi 功能。

Cellular

指示您的主板是否具有蜂窝功能。

PKCS11

表示主板支持的公钥加密算法。 PKCS11 是资格认证所必需的。支持的值包括 ECC、RSA 和 Both。Both 表示主板同时支持 ECC 和 RSA 算法。

KeyProvisioning

指示将受信任的 X.509 客户端证书写入主板的方法。

有效值为 Import、Onboard、Both 和 No。需要 Onboard、Both 或 No 密钥预配才能进行资格认证。只有 Import 不是有效的资格认证选项。

- 如果您的主板允许导入私有密钥，才能使用 Import。选择Import不是有效的资格鉴定配置，应仅用于测试目的，特别是 PKCS11 测试用例。 Onboard，Both或者No是资格认证所必需的。
- 如果您的主板支持板载私有密钥（例如，如果您的设备具有安全元件，或者如果您希望生成自己的设备密钥对和证书），请使用 Onboard。确保您在每个设备部分中添加一个 secureElementConfig 元素，并将公有密钥文件的绝对路径放在 publicKeyAsciiHexFilePath 字段中。
- 如果您的主板支持导入私有密钥和生成板载密钥以进行密钥预配，则使用 Both。
- 如果您的主板不支持密钥预配，请使用 No。只有当您的设备也已预先配置时，No 才是有效的选项。

OTA

表示您的主板是否支持 over-the-air (OTA) 更新功能。OtaDataPlaneProtocol 属性指示设备支持哪个 OTA 数据平面协议。OTA 需要使用 HTTP 或 MQTT 数据面板协议才能获得资格。要在测试时跳过运行 OTA 测试，请将 OTA 功能设置为 No，并将 OtaDataPlaneProtocol 属性设置为 None。这样不会运行资格认证。

BLE

指示您的主板是否支持低功耗蓝牙 (BLE) 功能。

devices.id

用户定义的测试的设备的唯一标识符。

devices.connectivity.serialPort

主机连接到所测试设备时使用的串行端口。

devices.secureElementConfig.PublicKeyAsciiHexFilePath

如果您的主板不是 pre-provisioned，或者未提供 PublicDeviceCertificateArn，则需要。由于密钥配置Onboard是必填类型，因此 FullTransportInterface TLS 测试组当前必须填写此字段。如果您的设备是 pre-provisioned，则 PublicKeyAsciiHexFilePath 为可选设备，无需将其包含在内。

以下数据块是该文件的绝对路径，包含从 Onboard 私有密钥中提取的十六进制字节公有密钥。

```
3059 3013 0607 2a86 48ce 3d02 0106 082a
8648 ce3d 0301 0703 4200 04cd 6569 ceb8
1bb9 1e72 339f e8cf 60ef 0f9f b473 33ac
6f19 1813 6999 3fa0 c293 5fae 08f1 1ad0
```

```
41b7 345c e746 1046 228e 5a5f d787 d571
dcb2 4e8d 75b3 2586 e2cc 0c
```

如果您的公有密钥是 .der 格式，则可以直接对公有密钥进行十六进制编码以生成十六进制文件。

要从 .der 公有密钥生成十六进制文件，请输入以下 xxd 命令：

```
xxd -p pubkey.der > outFile
```

如果您的公有密钥是 .pem 格式，则可以提取 base64 编码的标头和脚注并将其解码为二进制格式。然后，对二进制字符串进行十六进制编码以生成十六进制文件。

要为 .pem 公有密钥生成十六进制文件，请执行以下操作：

1. 运行以下 base64 命令，以便从公有密钥中删除 base64 标头和脚注。然后将名为 `base64key` 的解码密钥输出到文件 `pubkey.der` 中：

   ```
   base64 –decode base64key > pubkey.der
   ```

2. 运行 xxd 命令以将 `pubkey.der` 转换为十六进制格式。将生成的密钥另存为 *`outFile`*

   ```
   xxd -p pubkey.der > outFile
   ```

`devices.secureElementConfig.PublicDeviceCertificateArn`

上传到您的安全元素的证书的 ARN。 AWS IoT Core有关将证书上传到的信息 AWS IoT Core，请参阅《开发人员指南》AWS IoT 中的 X.509 客户端证书。

`devices.secureElementConfig.SecureElementSerialNumber`

（可选）安全元件的序列号。可选择将此序列号用于为 JITR 密钥预配创建设备证书。

`devices.secureElementConfig.preProvisioned`

（可选）如果设备预配置了带有锁定凭证的安全元件，无法导入、创建或销毁对象，则设置为“是”。如果将此属性设置为是，则必须提供相应的 pkcs11 标签。

`devices.secureElementConfig.pkcs11JITPCodeVerifyRootCertSupport`

（可选）如果设备的核心PKCS11 实现支持 JITP 存储，则设置为 “是”。这将在测试核心 PKCS 11 时启用 JITP `codeverify` 测试，并且需要提供代码验证密钥、JITP 证书和根证书 PKCS 11 标签。

identifiers

（可选）任意名称/值对的数组。您可以在下一部分所述的构建和刷写命令中使用这些值。

配置构建、刷写和测试设置

适用于 FreeRTOS 的 IDT 会自动构建测试并将其刷写到您的主板上。要启用此功能，您必须配置 IDT，以便为您的设备运行构建和刷写命令。构建和刷写设置在位于 `config` 文件夹的 `userdata.json` 模板文件中配置。

为测试设备配置设置

构建、刷写和测试设置在 `configs/userdata.json` 文件中进行。以下 JSON 示例显示如何配置适用于 FreeRTOS 的 IDT 来测试多个设备：

```
{
    "sourcePath": "</path/to/freertos>",
    "retainModifiedSourceDirectories": true | false,
    "freeRTOSVersion": "<freertos-version>",
    "freeRTOSTestParamConfigPath": "{{testData.sourcePath}}/path/from/source/path/to/
test_param_config.h",
    "freeRTOSTestExecutionConfigPath": "{{testData.sourcePath}}/path/from/source/path/
to/test_execution_config.h",
    "buildTool": {
        "name": "your-build-tool-name",
        "version": "your-build-tool-version",
        "command": [
            "<build command> -any-additional-flags {{testData.sourcePath}}"
        ]
    },
    "flashTool": {
        "name": "your-flash-tool-name",
        "version": "your-flash-tool-version",
        "command": [
            "<flash command> -any-additional-flags {{testData.sourcePath}} -any-
additional-flags"
        ]
    },
    "testStartDelayms": 0,
    "echoServerConfiguration": {
      "keyGenerationMethod": "EC | RSA",
      "serverPort": 9000
    },
```

```
    "otaConfiguration": {
        "otaE2EFirmwarePath": "{{testData.sourcePath}}/relative-path-to/ota-image-
generated-in-build-process",
        "otaPALCertificatePath": "/path/to/ota/pal/certificate/on/device",
        "deviceFirmwarePath" : "/path/to/firmware/image/name/on/device",
        "codeSigningConfiguration": {
            "signingMethod": "AWS | Custom",
            "signerHashingAlgorithm": "SHA1 | SHA256",
            "signerSigningAlgorithm": "RSA | ECDSA",
            "signerCertificate": "arn:partition:service:region:account-
id:resource:qualifier | /absolute-path-to/signer-certificate-file",
            "untrustedSignerCertificate": "arn:partition:service:region:account-
id:resourcetype:resource:qualifier",
            "signerCertificateFileName": "signerCertificate-file-name",
            "compileSignerCertificate": true | false,
            // ***********Use signerPlatform if you choose AWS for
 signingMethod***************
            "signerPlatform": "AmazonFreeRTOS-Default | AmazonFreeRTOS-TI-CC3220SF"

            ]
         }
    },
    **********
    This section is used for PKCS #11 labels of private key, public key, device
 certificate, code verification key, JITP certificate, and root certificate.
    When configuring PKCS11, you set up labels and you must provide the labels of the
 device certificate, public key,
    and private key for the key generation type (EC or RSA) it was created with. If
 your device supports PKCS11 storage of JITP certificate,
    code verification key, and root certificate, set
 'pkcs11JITPCodeVerifyRootCertSupport' to 'Yes' in device.json and provide the
 corresponding labels.
    **********
    "pkcs11LabelConfiguration":{
        "pkcs11LabelDevicePrivateKeyForTLS": "<device-private-key-label>",
        "pkcs11LabelDevicePublicKeyForTLS": "<device-public-key-label>",
        "pkcs11LabelDeviceCertificateForTLS": "<device-certificate-label>",
        "pkcs11LabelPreProvisionedECDevicePrivateKeyForTLS": "<preprovisioned-ec-
device-private-key-label>",
        "pkcs11LabelPreProvisionedECDevicePublicKeyForTLS": "<preprovisioned-ec-device-
public-key-label>",
        "pkcs11LabelPreProvisionedECDeviceCertificateForTLS": "<preprovisioned-ec-
device-certificate-label>",
```

```
        "pkcs11LabelPreProvisionedRSADevicePrivateKeyForTLS": "<preprovisioned-rsa-
device-private-key-label>",
        "pkcs11LabelPreProvisionedRSADevicePublicKeyForTLS": "<preprovisioned-rsa-
device-public-key-label>",
        "pkcs11LabelPreProvisionedRSADeviceCertificateForTLS": "<preprovisioned-rsa-
device-certificate-label>",
        "pkcs11LabelCodeVerifyKey": "<code-verification-key-label>",
        "pkcs11LabelJITPCertificate": "<JITP-certificate-label>",
        "pkcs11LabelRootCertificate": "<root-certificate-label>"
    }
  }
```

下面列出了在 `userdata.json` 中使用的属性：

`sourcePath`

移植的 FreeRTOS 源代码的根目录的路径。

`retainModifiedSourceDirectories`

（可选）检查是否保留在构建和刷写期间用于调试目的的已修改源目录。如果设置为 `true`，则修改的源目录名为 retainedSrc，位于每个测试组运行的结果日志文件夹中。如果未包含，则该字段默认为 `false`。

`freeRTOSTestParamConfigPath`

FreeRTOS-Libraries-Integration-Tests集成的`test_param_config.h`文件路径。此文件必须使用`{{testData.sourcePath}}`占位符变量使其相对于源代码根目录。 AWS IoT Device Tester 使用此文件中的参数来配置测试。

`freeRTOSTestExecutionConfigPath`

FreeRTOS-Libraries-Integration-Tests集成的`test_execution_config.h`文件路径。此文件必须使用`{{testData.sourcePath}}`占位符变量使其相对于存储库根目录。 AWS IoT Device Tester 使用此文件来控制必须运行哪些测试。

`freeRTOSVersion`

FreeRTOS 的版本，包括您的实现中使用的补丁版本。有关与 FreeRTOS AWS IoT Device Tester 兼容的 FreeRTOS 版本，请参阅 FreeRTOS 的支持版本。 AWS IoT Device Tester

buildTool

用于构建源代码的命令。生成命令中对源代码路径的所有引用都必须替换为 AWS IoT Device Tester 变量{{testData.sourcePath}}。使用{{config.idtRootPath}}占位符引用相对于 AWS IoT Device Tester 根路径的生成脚本。

flashTool

将映像刷写到设备的命令。flash 命令中对源代码路径的所有引用都必须替换为 AWS IoT Device Tester 变量{{testData.sourcePath}}。使用{{config.idtRootPath}}占位符引用相对于 AWS IoT Device Tester 根路径的 Flash 脚本。

> **Note**
>
> FRQ 2.0 的新集成测试结构不需要路径变量，例如，{{enableTests}} 和 {{buildImageName}}。OTA 端到端测试使用 FreeRTOS-Libraries-Integration-Tests GitHub 存储库中提供的配置模板运行。如果 GitHub 存储库中的文件存在于您的父源项目中，则源代码在两次测试之间不会更改。如果需要为 OTA 端到端测试创建不同的构建映像，则必须在构建脚本中构建此映像，并在 otaConfiguration 下指定的 userdata.json 文件中指定该映像。

testStartDelayms

指定 FreeRTOS 测试运行器在开始运行测试之前将等待多少毫秒。如果由于网络或其他延迟问题，待测试设备在 IDT 有机会连接并开始日志记录之前就开始输出重要的测试信息，这可能很有用。此值仅适用于 FreeRTOS 测试组，不适用于不使用 FreeRTOS 测试运行器的其他测试组，例如 OTA 测试。如果您收到预期为 10，但收到为 5相关的错误，则应将此字段设置为 5000。

echoServerConfiguration

为 TLS 测试设置 Echo 服务器的配置。该字段为必填。

keyGenerationMethod

使用此选项对 Echo 服务器进行配置。选项是 EC 或 RSA。

serverPort

Echo 服务器运行时使用的端口号。

otaConfiguration

OTA PAL 和 OTA E2E 测试的配置。该字段为必填。

otaE2EFirmwarePath

IDT 用于 OTA 端到端测试的 OTA bin 映像的路径。

otaPALCertificatePath

设备上 OTA PAL 测试证书的路径。这用于验证签名。例如，ecdsa-sha256-signer.crt.pem。

deviceFirmwarePath

要引导的固件映像的硬编码名称的路径。如果您的设备不使用文件系统引导固件，请将此字段指定为 'NA'。如果您的设备使用文件系统引导固件，请指定固件引导映像的路径或名称。

codeSigningConfiguration

signingMethod

代码签名方法。可能的值为 AWS 或自定义。

> **Note**
>
> 对于北京和宁夏区域，请使用自定义。 AWS 该区域不支持代码签名。

signerHashingAlgorithm

设备所支持的哈希算法。可能的值为 SHA1 或 SHA256。

signerSigningAlgorithm

设备所支持的签名算法。可能的值为 RSA 或 ECDSA。

signerCertificate

用于 OTA 的可信证书。对于 AWS 代码签名方法，请使用上传到 Certificate Manager 的可信证书的 Amazon 资源名称 (ARN)。 AWS 对于自定义代码签名方法，请使用签署人证书文件的绝对路径。有关创建信任证书的信息，请参阅创建代码签名证书。

untrustedSignerCertificate

在某些 OTA 测试中用作不可信证书的第二个证书的 ARN 或文件路径。有关创建证书的信息，请参阅创建代码签名证书。

signerCertificateFileName

设备上代码签署证书的文件名。此值必须与您在运行 `aws acm import-certificate` 命令时提供的文件名相匹配。

compileSignerCertificate

确定签名验证证书状态的布尔值。有效值为 `true` 和 `false`。

如果未配置或刷写代码签署人签名验证证书，则将此值设置为 true。它必须编译到项目中。 AWS IoT Device Tester 获取可信证书并将其编译为。`aws_codesigner_certificate.h`

signerPlatform

C AWS ode Signer 在创建 OTA 更新任务时使用的签名和哈希算法。目前，此字段的可能值为 `AmazonFreeRTOS-TI-CC3220SF` 和 `AmazonFreeRTOS-Default`。

- 如果为 SHA1 和 RSA，则选择 `AmazonFreeRTOS-TI-CC3220SF`。
- 如果为 SHA256 和 ECDSA，则选择 `AmazonFreeRTOS-Default`。
- 如果您的配置需要 SHA256 | RSA 或 SHA1 | ECDSA，请与我们联系以获取进一步的支持。
- 如果您为 `signingMethod` 选择 `Custom`，则配置 `signCommand`。

signCommand

命令中需要两个占位符 `{{inputImageFilePath}}` 和 `{{outputSignatureFilePath}}`。`{{inputImageFilePath}}` 是要签名的由 IDT 构建的映像的文件路径。`{{outputSignatureFilePath}}` 是将由脚本生成的签名的文件路径。

pkcs11LabelConfiguration

PKCS11 标签配置需要至少一组设备证书标签、公钥标签和私钥标签标签的标签才能运行 PKCS11 测试组。所需的 PKCS11 标签取决于`device.json`文件中的设备配置。如果在中将预配置设置为“是”`device.json`，则所需的标签必须是以下标签之一，具体取决于为该 PKCS11 功能选择的内容。

- `PreProvisionedEC`
- `PreProvisionedRSA`

如果在 `device.json` 中将预配置设置为否，则所需的标签为：

- `pkcs11LabelDevicePrivateKeyForTLS`
- `pkcs11LabelDevicePublicKeyForTLS`
- `pkcs11LabelDeviceCertificateForTLS`

只有在 `device.json` 文件中为 `pkcs11JITPCodeVerifyRootCertSupport` 选择是时，才需要以下三个标签。

- pkcs11LabelCodeVerifyKey
- pkcs11LabelRootCertificate
- pkcs11LabelJITPCertificate

这些字段的值应与《FreeRTOS 移植指南》中定义的值相匹配。

pkcs11LabelDevicePrivateKeyForTLS

（可选）此标签用于私有密钥的 PKCS #11 标签。对于支持板载和导入密钥预配的设备，此标签用于测试。此标签可能与为预先配置的用例定义的标签不同。如果在 device.json 中将密钥预配置设置为否，将已预先配置设置为是，则不会定义。

pkcs11LabelDevicePublicKeyForTLS

（可选）此标签用于公有密钥的 PKCS #11 标签。对于支持板载和导入密钥预配的设备，此标签用于测试。此标签可能与为预先配置的用例定义的标签不同。如果在 device.json 中将密钥预配置设置为否，将已预先配置设置为是，则不会定义。

pkcs11LabelDeviceCertificateForTLS

（可选）此标签用于设备证书的 PKCS #11 标签。对于支持板载和导入密钥预配的设备，此标签将用于测试。此标签可能与为预先配置的用例定义的标签不同。如果在 device.json 中将密钥预配置设置为否，将已预先配置设置为是，则不会定义。

pkcs11LabelPreProvisionedECDevicePrivateKeyForTLS

（可选）此标签用于私有密钥的 PKCS #11 标签。对于具有安全元素或硬件限制的设备，这将使用不同的标签来保留 AWS IoT 凭据。如果您的设备支持使用 EC 密钥进行预配置，请提供此标签。当 device.json 中的预置备设置为是 时，必须提供标签 pkcs11LabelPreProvisionedRSADevicePrivateKeyForTLS 或同时提供两者。此标签可能与为注册和导入的用例定义的标签不同。

pkcs11LabelPreProvisionedECDevicePublicKeyForTLS

（可选）此标签用于公有密钥的 PKCS #11 标签。对于具有安全元素或硬件限制的设备，这将使用不同的标签来保留 AWS IoT 凭据。如果您的设备支持使用 EC 密钥进行预配置，请提供此标签。当 device.json 中的预置备设置为是 时，必须提供标签 pkcs11LabelPreProvisionedRSADevicePublicKeyForTLS 或同时提供两者。此标签可能与为注册和导入的用例定义的标签不同。

pkcs11LabelPreProvisionedECDeviceCertificateForTLS

（可选）此标签用于设备证书的 PKCS #11 标签。对于具有安全元素或硬件限制的设备，这将使用不同的标签来保留 AWS IoT 凭据。如果您的设备支持使用 EC 密钥进

行预配置，请提供此标签。当 device.json 中的预置备设置为是 时，必须提供标签 pkcs11LabelPreProvisionedRSADeviceCertificateForTLS 或同时提供两者。此标签可能与为注册和导入的用例定义的标签不同。

pkcs11LabelPreProvisionedRSADevicePrivateKeyForTLS

（可选）此标签用于私有密钥的 PKCS #11 标签。对于具有安全元素或硬件限制的设备，这将使用不同的标签来保留 AWS IoT 凭据。如果您的设备支持使用 RSA 密钥进行预配置，请提供此标签。当 device.json 中的预置备设置为是 时，必须提供标签 pkcs11LabelPreProvisionedECDevicePrivateKeyForTLS 或同时提供两者。

pkcs11LabelPreProvisionedRSADevicePublicKeyForTLS

（可选）此标签用于公有密钥的 PKCS #11 标签。对于具有安全元素或硬件限制的设备，这将使用不同的标签来保留 AWS IoT 凭据。如果您的设备支持使用 RSA 密钥进行预配置，请提供此标签。当 device.json 中的预置备设置为是 时，必须提供标签 pkcs11LabelPreProvisionedECDevicePublicKeyForTLS 或同时提供两者。

pkcs11LabelPreProvisionedRSADeviceCertificateForTLS

（可选）此标签用于设备证书的 PKCS #11 标签。对于具有安全元素或硬件限制的设备，这将使用不同的标签来保留 AWS IoT 凭据。如果您的设备支持使用 RSA 密钥进行预配置，请提供此标签。当 device.json 中的预置备设置为是 时，必须提供标签 pkcs11LabelPreProvisionedECDeviceCertificateForTLS 或同时提供两者。

pkcs11LabelCodeVerifyKey

（可选）此标签用于代码验证密钥的 PKCS #11 标签。如果您的设备支持 PKCS #11 存储 JITP 证书、代码验证密钥和根证书，请提供此标签。当 device.json 中的 pkcs11JITPCodeVerifyRootCertSupport 设置为是时，必须提供此标签。

pkcs11LabelJITPCertificate

（可选）此标签用于设备 JITP 证书的 PKCS #11 标签。如果您的设备支持 PKCS #11 存储 JITP 证书、代码验证密钥和根证书，请提供此标签。当 device.json 中的 pkcs11JITPCodeVerifyRootCertSupport 设置为是时，必须提供此标签。

适用于 FreeRTOS 变量的 IDT

生成代码和刷新设备的命令可能需要连接或有关设备的其他信息才能成功运行。 AWS IoT Device Tester 允许您在 Flash 中引用设备信息并使用生成命令JsonPath。通过使用简单的 JsonPath表达式，您可以获取device.json文件中指定的所需信息。

路径变量

适用于 FreeRTOS 的 IDT 定义了可在命令行和配置文件中使用的以下路径变量：

{{testData.sourcePath}}

扩展到到源代码路径。如果使用该变量，则必须在刷写和构建命令中使用该变量。

{{device.connectivity.serialPort}}

扩展到串行端口。

{{device.identifiers[?(@.name == 'serialNo')].value[0]}}

扩展到您设备的序列号。

{{config.idtRootPath}}

扩展到 AWS IoT Device Tester 根路径。

适用于 FreeRTOS 资格套件 2.0 (FRQ 2.0) 的 IDT 用户界面

AWS IoT Device Tester for F reeRTOS（FreeRTOS 的 IDT）包括一个基于 Web 的用户界面 (UI)，您可以在其中与 IDT 命令行应用程序和相关的配置文件进行交互。您可以使用适用于 FreeRTOS 的 IDT 用户界面为您的设备创建新配置或修改现有配置。您还可以使用 UI 调用 IDT 应用程序并对您的设备运行 FreeRTOS 测试。

有关如何使用命令行运行资格认证测试的信息，请参阅首次测试您的微控制器板。

本节介绍了适用于 FreeRTOS UI 的 IDT 的先决条件以及如何从 UI 运行资格认证测试。

主题

- 设置 IDT 先决条件
- 配置 AWS 凭据以使用 IDT 用户界面
- 打开适用于 FreeRTOS UI 的 IDT
- 创建新的配置
- 修改现有配置
- 运行资格认证测试

设置 IDT 先决条件

要通过 AWS IoT Device Tester (IDT) for FreeRTOS 用户界面运行测试，必须完成 IDT FreeRTOS 资格认证 (FRQ) 2.x 设置 LTS 资格先决条件 页面上的先决条件。

配置 AWS 凭据以使用 IDT 用户界面

您必须为在中创建的用户配置 IAM AWS 用户证书创建一个 AWS 账户。您可以采用以下两种方法之一来指定凭证：

- 在凭证文件中
- 作为环境变量

使用 AWS 凭证文件配置凭证

IDT 使用与 AWS CLI相同的凭证文件。有关更多信息，请参阅配置和凭证文件。

凭证文件的位置因您使用的操作系统而异：

- macOS 和 Linux – `~/.aws/credentials`
- Windows – `C:\Users\UserName\.aws\credentials`

按以下格式将您的 AWS 凭证添加到`credentials`文件中：

```
[default]
aws_access_key_id = your_access_key_id
aws_secret_access_key = your_secret_access_key
```

> **Note**
>
> 如果您不使用`default` AWS 配置文件，则必须在 IDT for FreeRTOS 用户界面中指定配置文件名称。有关配置文件的更多信息，请参阅配置和凭据文件设置。

使用环境变量配置 AWS 凭证

环境变量是由操作系统维护且由系统命令使用的变量。如果您关闭 SSH 会话，则不会保存。适用于 FreeRTOS 的 IDT 用户界面使用`AWS_ACCESS_KEY_ID`、`AWS_SECRET_ACCESS_KEY`和环境变量来存储您的证书。 AWS

要在 Linux、macOS 或 Unix 上设置这些变量，请使用 export：

```
export AWS_ACCESS_KEY_ID=your_access_key_id
export AWS_SECRET_ACCESS_KEY=your_secret_access_key
```

要在 Windows 上设置这些变量，请使用 set：

```
set AWS_ACCESS_KEY_ID=your_access_key_id
set AWS_SECRET_ACCESS_KEY=your_secret_access_key
```

打开适用于 FreeRTOS UI 的 IDT

本主题介绍如何打开 IDT for FreeRTOS 用户界面以使用 FreeRTOS 资格套件。

打开适用于 FreeRTOS UI 的 IDT

1. 下载支持的适用于 FreeRTOS 版本的 IDT。然后将下载的存档解压缩到您拥有读写权限的目录中。
2. 转到适用于 FreeRTOS 的 IDT 安装的目录：

   ```
   cd devicetester-extract-location/bin
   ```

3. 运行以下命令以打开适用于 FreeRTOS UI 的 IDT：

   Linux

   ```
   .devicetester_ui_linux_x86-64
   ```

   Windows

   ```
   ./devicetester_ui_win_x64-64
   ```

   macOS

   ```
   ./devicetester_ui_mac_x86-64
   ```

> **Note**
>
> 在 macOS 中，要允许您的系统运行此 UI，请转到系统首选项 -> 安全和隐私。当您运行测试时，可能需要再执行此命令三次。

适用于 FreeRTOS UI 的 IDT 将在您的默认浏览器中打开。以下浏览器的最新三个主要版本支持此 UI：

- Google Chrome
- Mozilla Firefox
- Microsoft Edge
- Apple Safari for macOS

> **Note**
>
> 为了获得更好的体验，我们建议使用 Google Chrome 或 Mozilla Firefox 来访问适用于 FreeRTOS UI 的 IDT。此 UI 不支持 Microsoft Internet Explorer。

> **Important**
>
> 在打开 UI 之前，您必须配置您的 AWS 凭据。如果您尚未配置凭证，请关闭适用于 FreeRTOS UI 的 IDT 浏览器窗口，按照配置 AWS 凭据以使用 IDT 用户界面中的步骤操作，然后重新打开适用于 FreeRTOS UI 的 IDT。

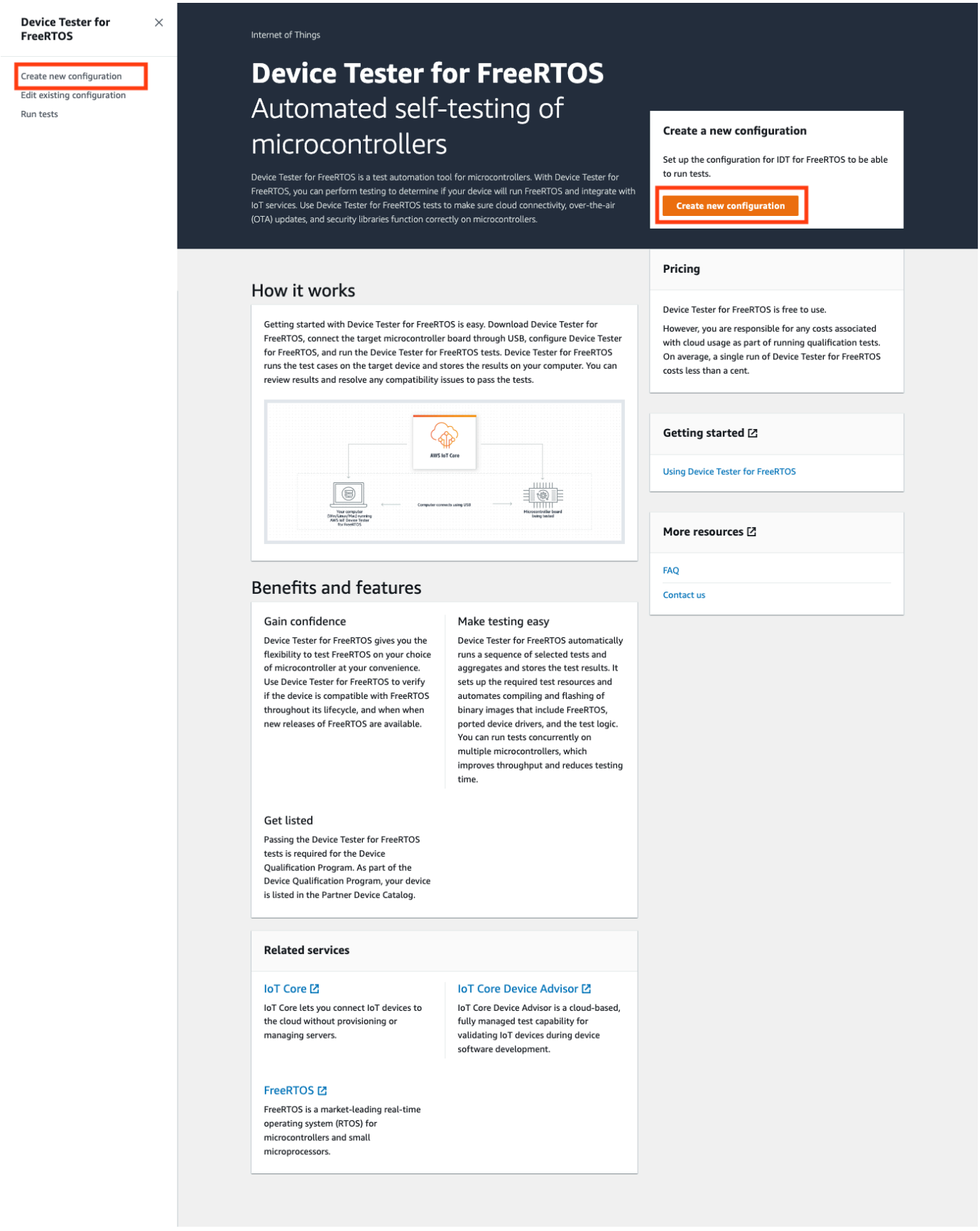
创建新的配置

如果您是首次使用的用户，则必须创建一个新配置来设置适用于 FreeRTOS 的 IDT 运行测试所需的 JSON 配置文件。然后，您可以运行测试或修改已创建的配置。

有关 `config.json`、`device.json` 和 `userdata.json` 文件的示例，请参阅首次测试您的微控制器板。

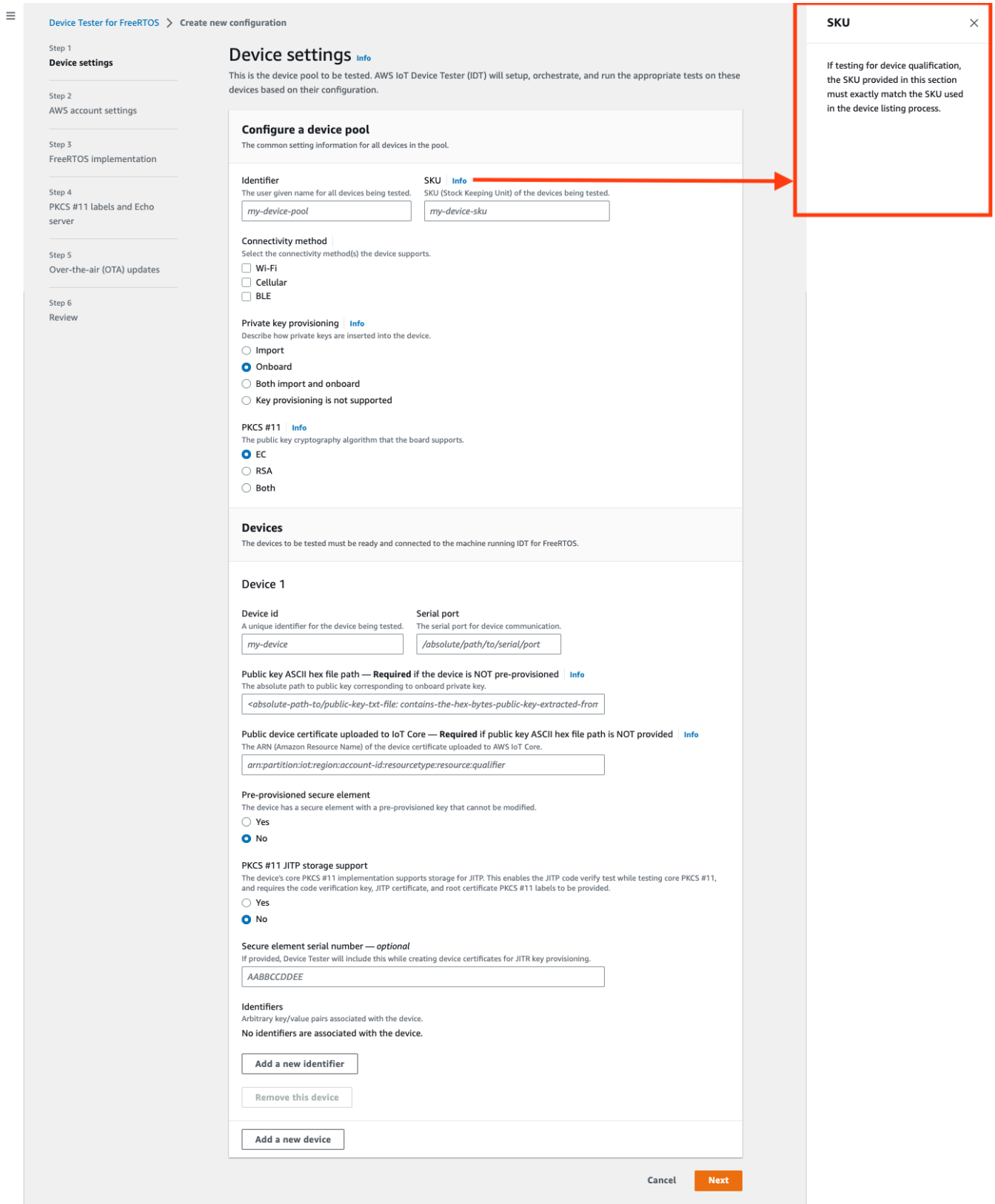
创建新的配置

1. 在适用于 FreeRTOS UI 的 IDT 中，打开导航菜单，然后选择创建新配置。

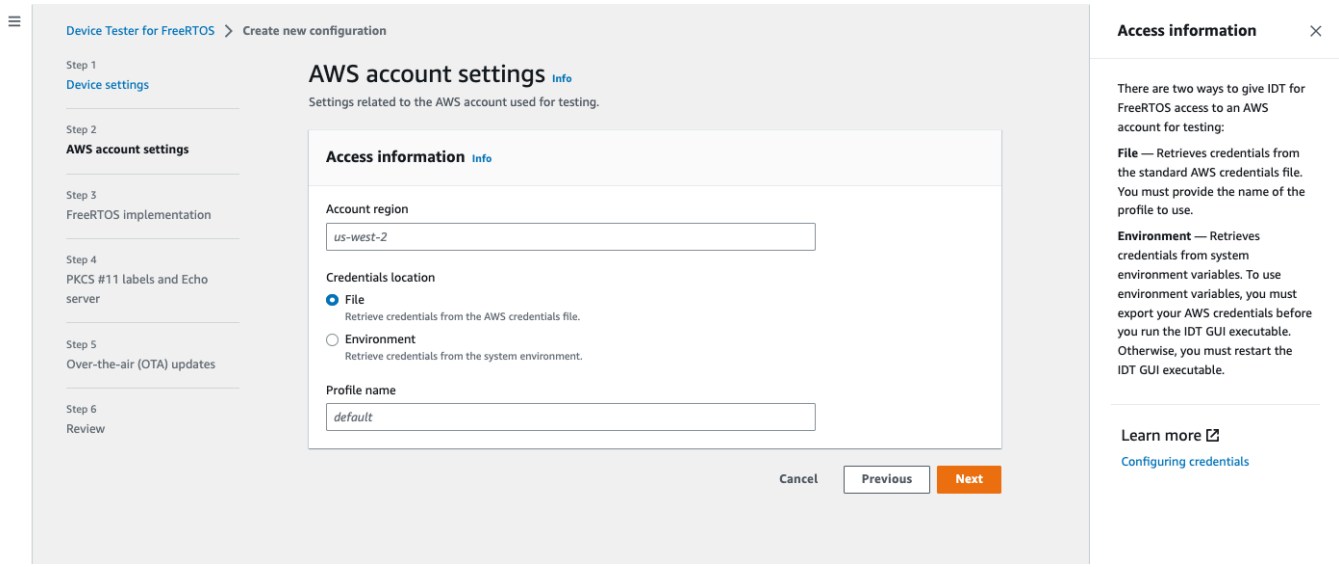

Device Tester for FreeRTOS
Create new configuration
Edit existing configuration
Run tests
Internet of Things
Device Tester for FreeRTOS
Automated self-testing of microcontrollers
Device Tester for FreeRTOS is a test automation tool for microcontrollers. With Device Tester for FreeRTOS, you can perform testing to determine if your device will run FreeRTOS and integrate with IoT services. Use Device Tester for FreeRTOS tests to make sure cloud connectivity, over-the-air (OTA) updates, and security libraries function correctly on microcontrollers.
Create a new configuration
Set up the configuration for IDT for FreeRTOS to be able to run tests.
Create new configuration
How it works
Getting started with Device Tester for FreeRTOS is easy. Download Device Tester for FreeRTOS, connect the target microcontroller board through USB, configure Device Tester for FreeRTOS, and run the Device Tester for FreeRTOS tests. Device Tester for FreeRTOS runs the test cases on the target device and stores the results on your computer. You can review results and resolve any compatibility issues to pass the tests.
AWS IoT Core
Computer connects using USB
Benefits and features
Gain confidence
Device Tester for FreeRTOS gives you the flexibility to test FreeRTOS on your choice of microcontroller at your convenience. Use Device Tester for FreeRTOS to verify if the device is compatible with FreeRTOS throughout its lifecycle, and when when new releases of FreeRTOS are available.
Make testing easy
Device Tester for FreeRTOS automatically runs a sequence of selected tests and aggregates and stores the test results. It sets up the required test resources and automates compiling and flashing of binary images that include FreeRTOS, ported device drivers, and the test logic. You can run tests concurrently on multiple microcontrollers, which improves throughput and reduces testing time.
Get listed
Passing the Device Tester for FreeRTOS tests is required for the Device Qualification Program. As part of the Device Qualification Program, your device is listed in the Partner Device Catalog.
Related services
IoT Core
IoT Core lets you connect IoT devices to the cloud without provisioning or managing servers.
IoT Core Device Advisor
IoT Core Device Advisor is a cloud-based, fully managed test capability for validating IoT devices during device software development.
FreeRTOS
FreeRTOS is a market-leading real-time operating system (RTOS) for microcontrollers and small microprocessors.
Pricing
Device Tester for FreeRTOS is free to use.
However, you are responsible for any costs associated with cloud usage as part of running qualification tests. On average, a single run of Device Tester for FreeRTOS costs less than a cent.
Getting started
Using Device Tester for FreeRTOS
More resources
FAQ
Contact us

2. 按照配置向导输入用于运行资格认证测试的 IDT 配置设置。该向导在 *devicetester-extract-location*/config 目录中的 JSON 配置文件中配置以下设置。

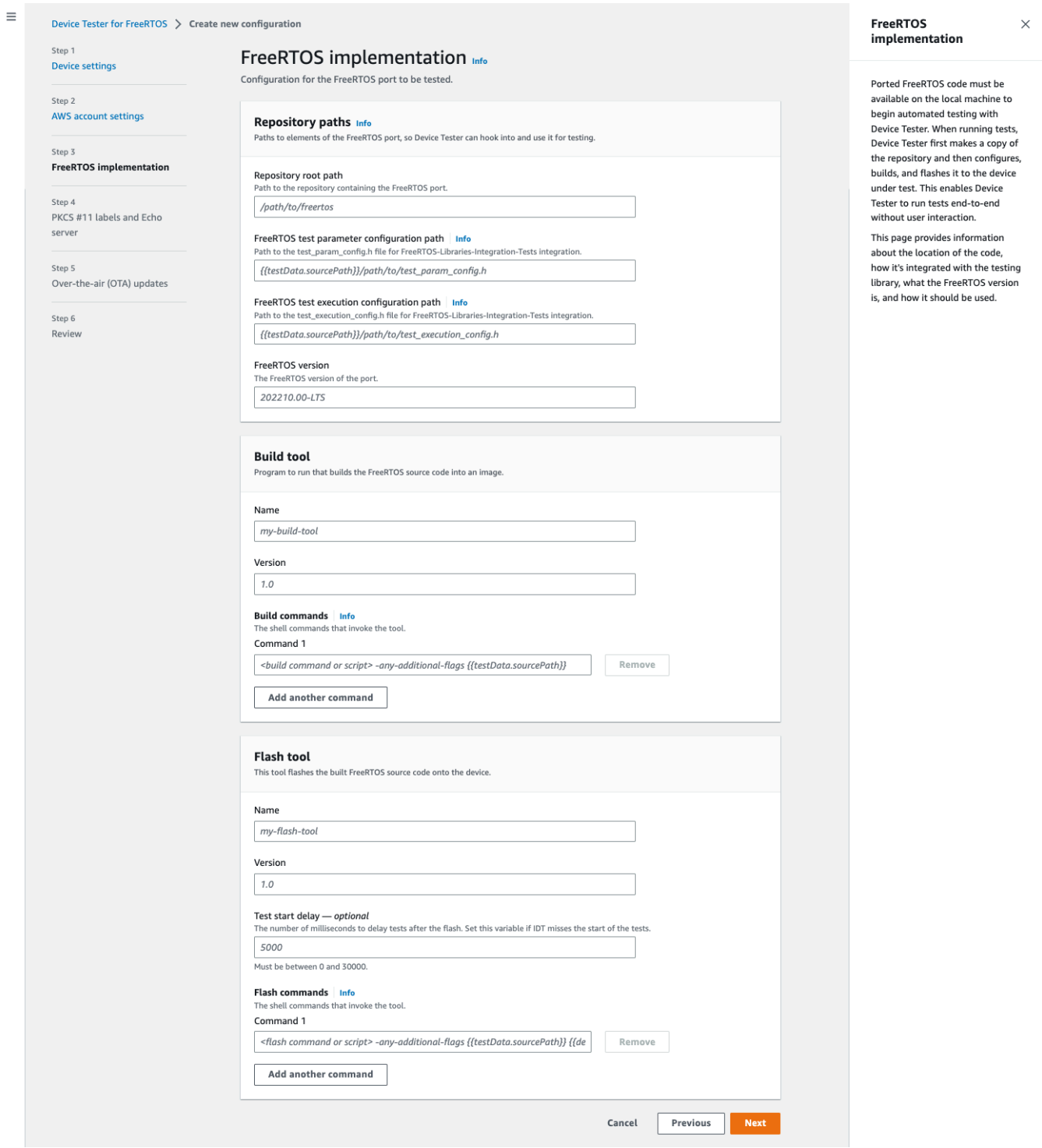
   - 设备设置 – 要测试的设备的设备池设置。这些设置在 id 和 sku 字段中配置，设备池的设备数据块位于 config.json 文件中。



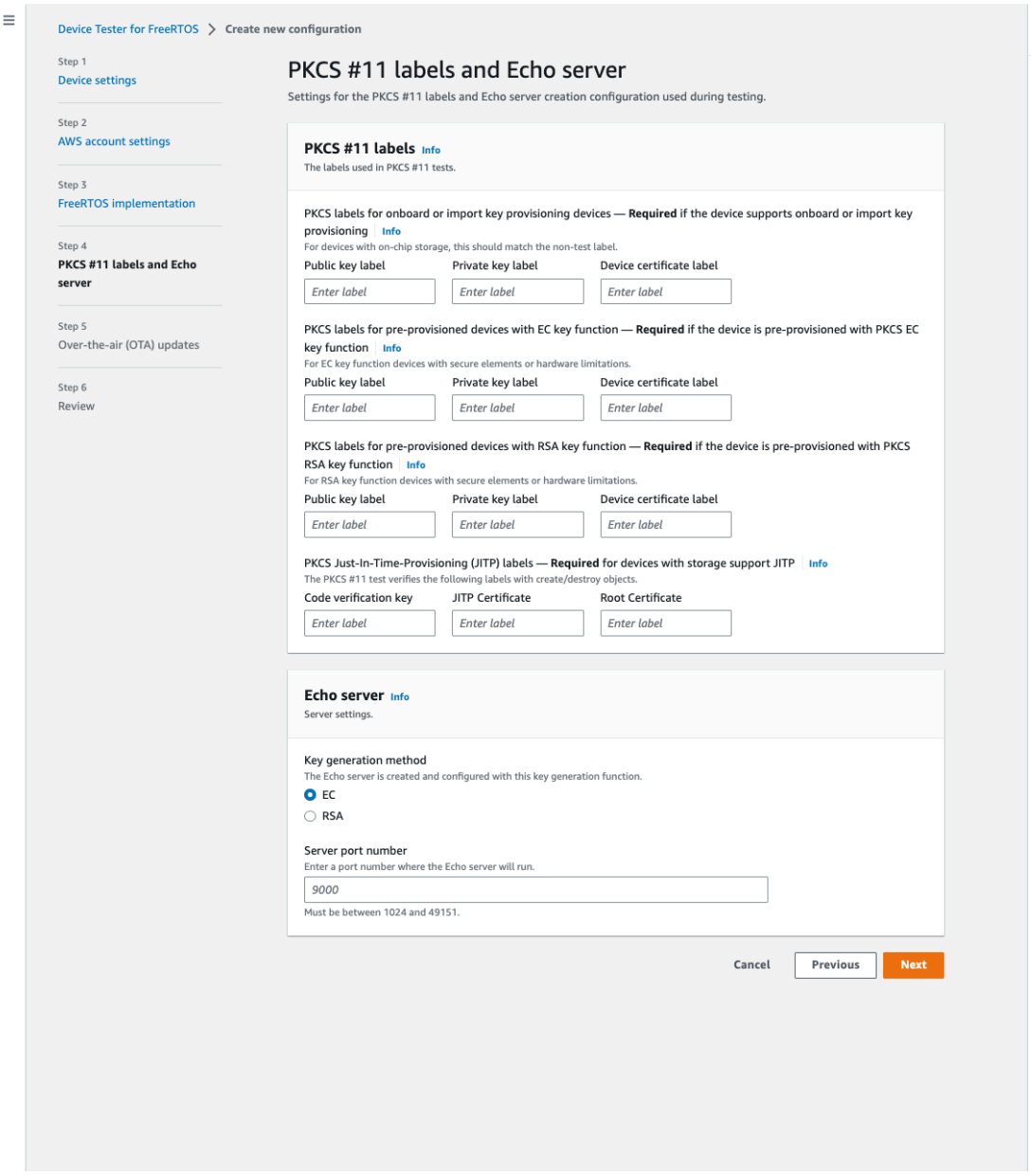
- AWS 帐户设置 — 适用于 FreeRTOS 的 IDT 在测试运行期间用于 AWS 创建资源 AWS 账户 的信息。这些设置在 `config.json` 文件中进行配置。



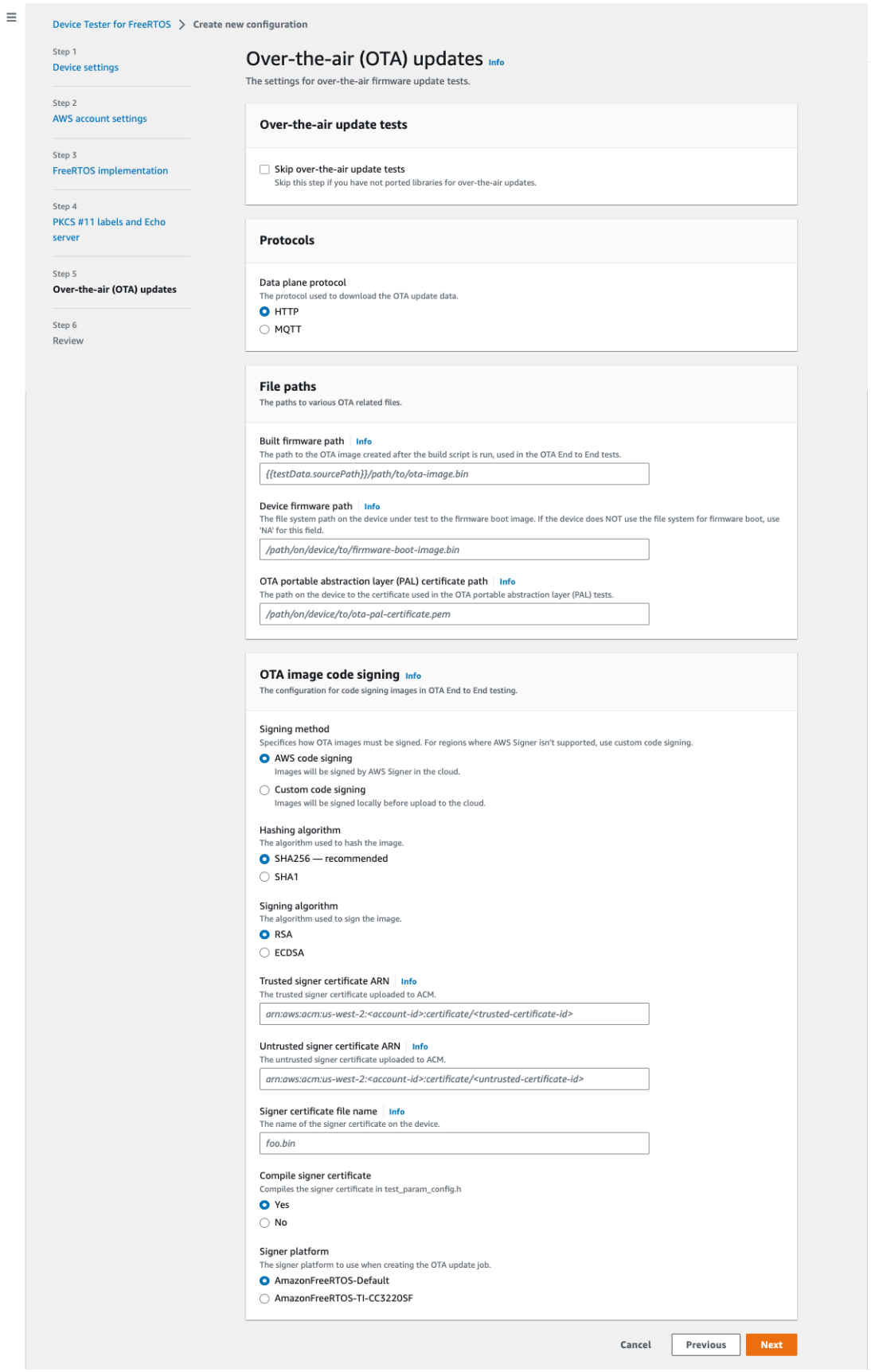
- FreeRTOS 实现 — FreeRTOS 存储库和移植代码的绝对路径，以及您想要运行 IDT FRQ 的 FreeRTOS 版本。`FreeRTOS-Libraries-Integration-Tests` GitHub 存储库中执行和参数配置头文件的路径。适用于您的硬件的构建和刷写命令，允许 IDT 在您的主板上自动构建和刷写测试。这些设置在 `userdata.json` 文件中进行配置。



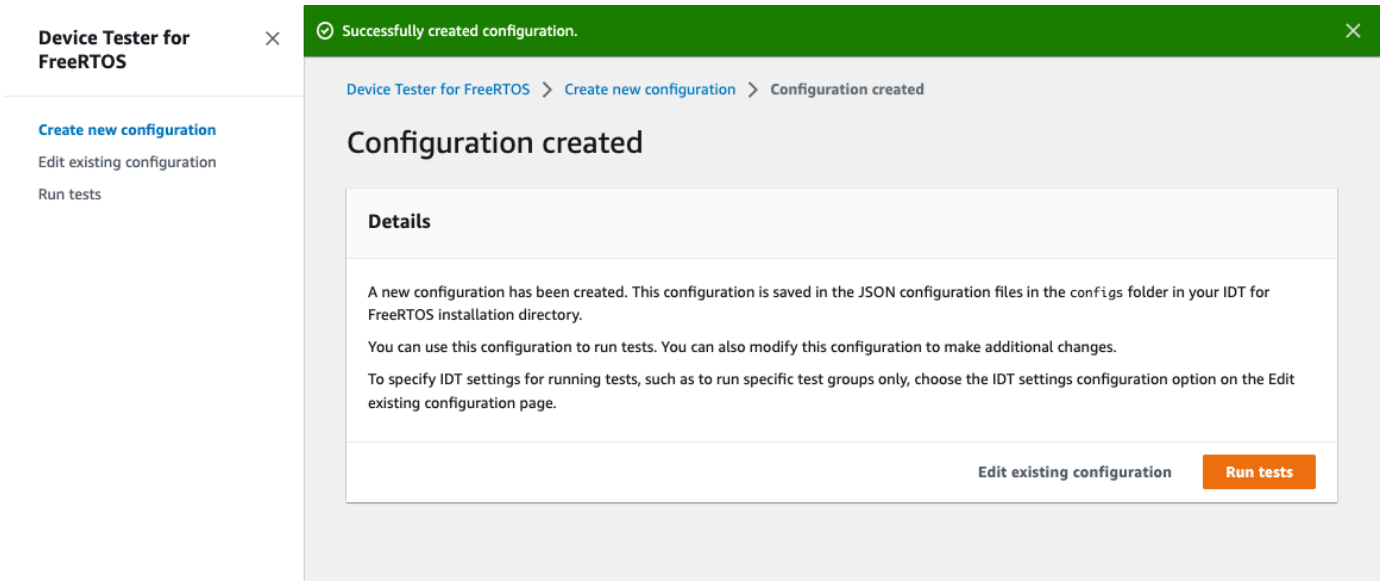
- PKCS #11 标签和 Echo 服务器 – PKCS #11 标签对应于根据关键功能和密钥预配方法在硬件中配置的密钥。传输接口测试的 Echo 服务器配置设置。这些设置在 `userdata.json` 和 `device.json` 文件中进行配置。



- Over-the-air (OTA) 更新-控制 OTA 功能测试的设置。这些设置在 `device.json` 和 `userdata.json` 文件的 `features` 数据块中配置。


Device Tester for FreeRTOS > Create new configuration
Step 1
Device settings
Step 2
AWS account settings
Step 3
FreeRTOS implementation
Step 4
PKCS #11 labels and Echo server
Step 5
Over-the-air (OTA) updates
Step 6
Review
Over-the-air (OTA) updates Info
The settings for over-the-air firmware update tests.
Over-the-air update tests
Skip over-the-air update tests
Skip this step if you have not ported libraries for over-the-air updates.
Protocols
Data plane protocol
The protocol used to download the OTA update data.
HTTP
MQTT
File paths
The paths to various OTA related files.
Built firmware path Info
The path to the OTA image created after the build script is run, used in the OTA End to End tests.
{{testData.sourcePath}}/path/to/ota-image.bin
Device firmware path Info
The file system path on the device under test to the firmware boot image. If the device does NOT use the file system for firmware boot, use 'NA' for this field.
/path/on/device/to/firmware-boot-image.bin
OTA portable abstraction layer (PAL) certificate path Info
The path on the device to the certificate used in the OTA portable abstraction layer (PAL) tests.
/path/on/device/to/ota-pal-certificate.pem
OTA image code signing Info
The configuration for code signing images in OTA End to End testing.
Signing method
Specifices how OTA images must be signed. For regions where AWS Signer isn't supported, use custom code signing.
AWS code signing
Images will be signed by AWS Signer in the cloud.
Custom code signing
Images will be signed locally before upload to the cloud.
Hashing algorithm
The algorithm used to hash the image.
SHA256 — recommended
SHA1
Signing algorithm
The algorithm used to sign the image.
RSA
ECDSA
Trusted signer certificate ARN Info
The trusted signer certificate uploaded to ACM.
arn:aws:acm:us-west-2:<account-id>:certificate/<trusted-certificate-id>
Untrusted signer certificate ARN Info
The untrusted signer certificate uploaded to ACM.
arn:aws:acm:us-west-2:<account-id>:certificate/<untrusted-certificate-id>
Signer certificate file name Info
The name of the signer certificate on the device.
foo.bin
Compile signer certificate
Compiles the signer certificate in test_param_config.h
Yes
No
Signer platform
The signer platform to use when creating the OTA update job.
AmazonFreeRTOS-Default
AmazonFreeRTOS-TI-CC3220SF
Cancel
Previous
Next
Over-the-air (OTA) updates
IDT for FreeRTOS runs tests to verify OTA update behavior, including end-to-end (E2E) and portable abstraction layer (PAL) tests.
These tests are required to qualify a device.
Learn more
FreeRTOS OTA Update tests

3. 在审核页面上，验证您的配置信息。



审核配置完成后，要运行资格认证测试，请选择运行测试。

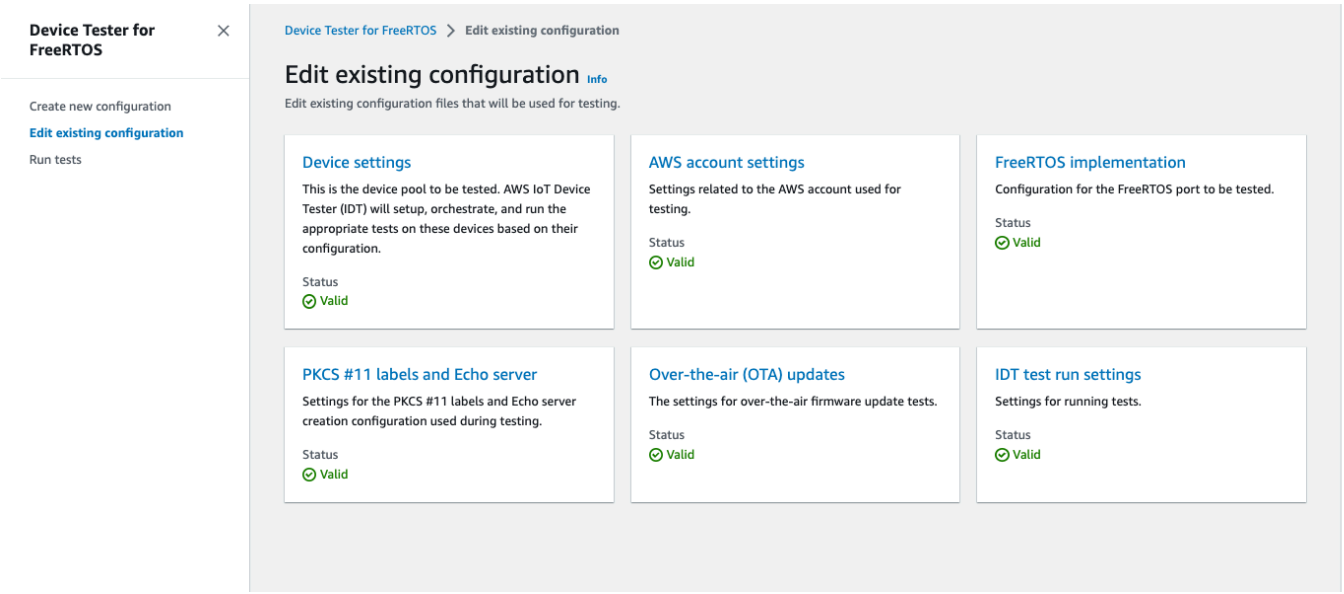
修改现有配置

如果您已经为适用于 FreeRTOS 的 IDT 设置了配置文件，则可以使用适用于 FreeRTOS UI 的 IDT 修改现有配置。现有的配置文件必须位于 *devicetester-extract-location*/config 目录中。

修改配置

1. 在适用于 FreeRTOS UI 的 IDT 中，打开导航菜单，然后选择编辑现有配置。

   配置控制面板会显示有关现有配置设置的信息。如果配置不正确或不可用，则配置的状态为 `Error validating configuration`。

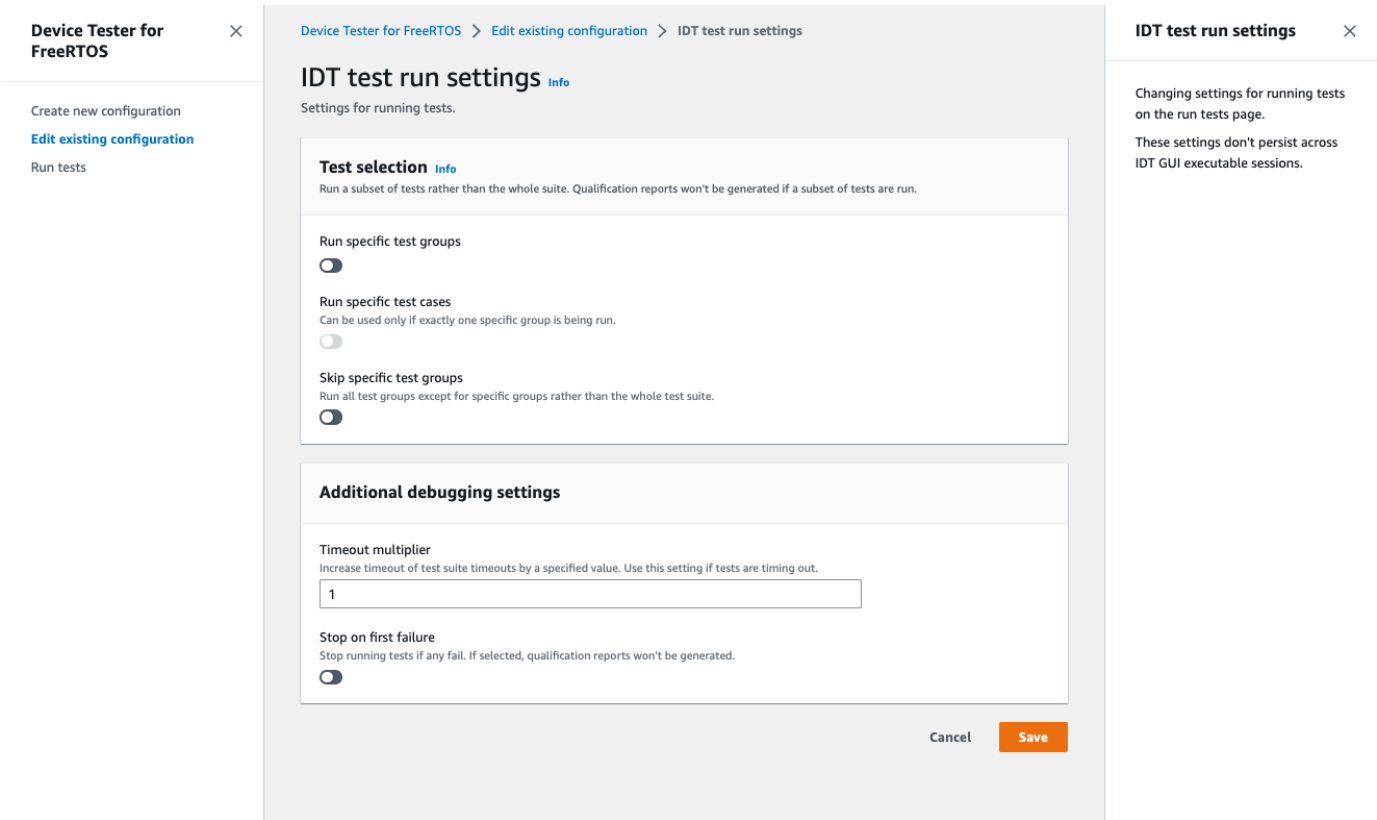


2. 要修改现有配置设置，请完成以下步骤：

   a. 选择配置设置的名称以打开其设置页面。

   b. 修改设置，然后选择保存，以便重新生成相应的配置文件。

3. 要修改适用于 FreeRTOS 的 IDT 测试运行设置，请在编辑视图中选择 IDT 测试运行设置：



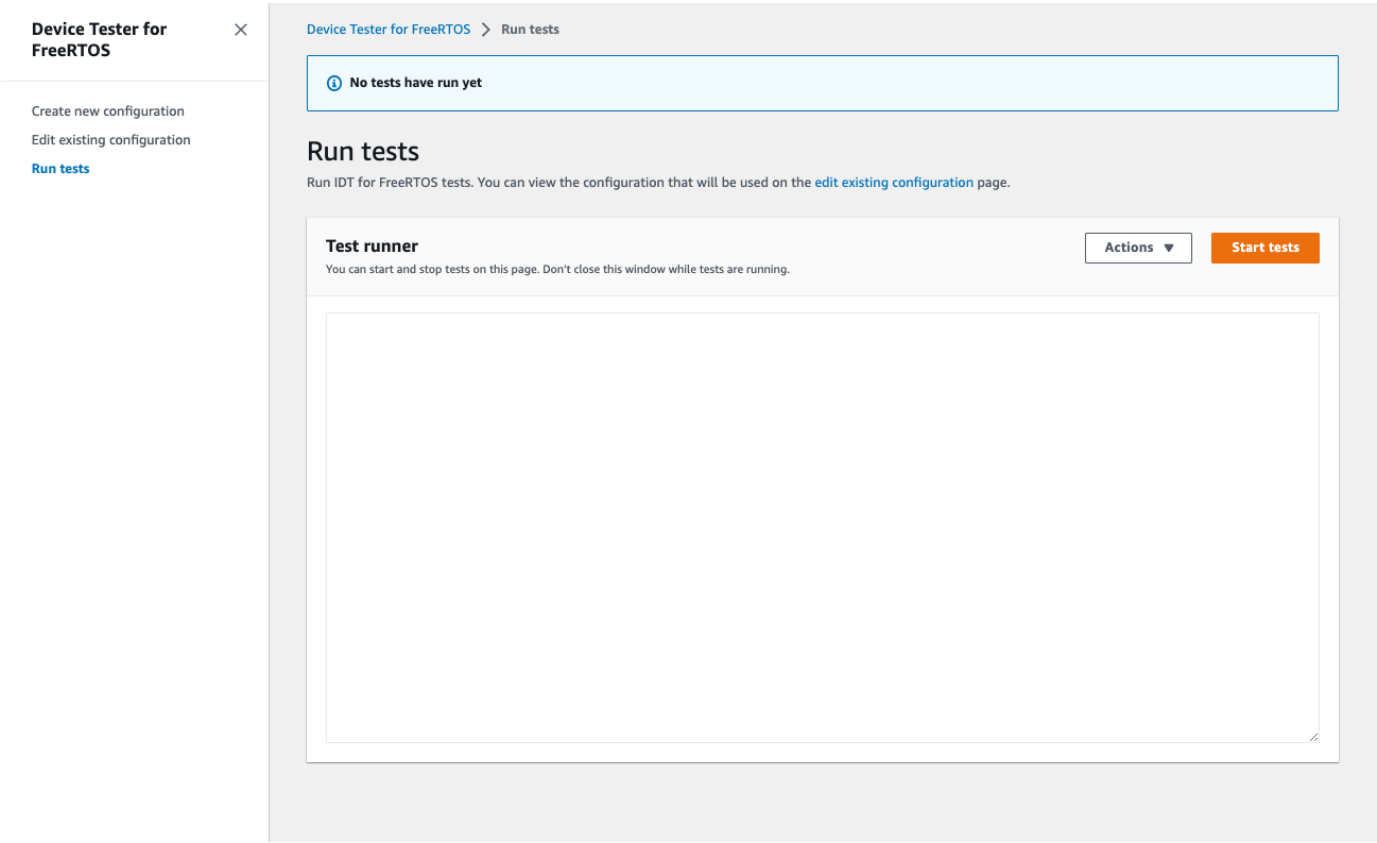
修改配置完成后，请确认所有配置设置均可通过验证。如果每个配置设置的状态为 `Valid`，则可以使用此配置运行资格认证测试。

运行资格认证测试

为适用于 FreeRTOS UI 的 IDT 创建配置后，就可以运行资格认证测试了。

运行资格认证测试

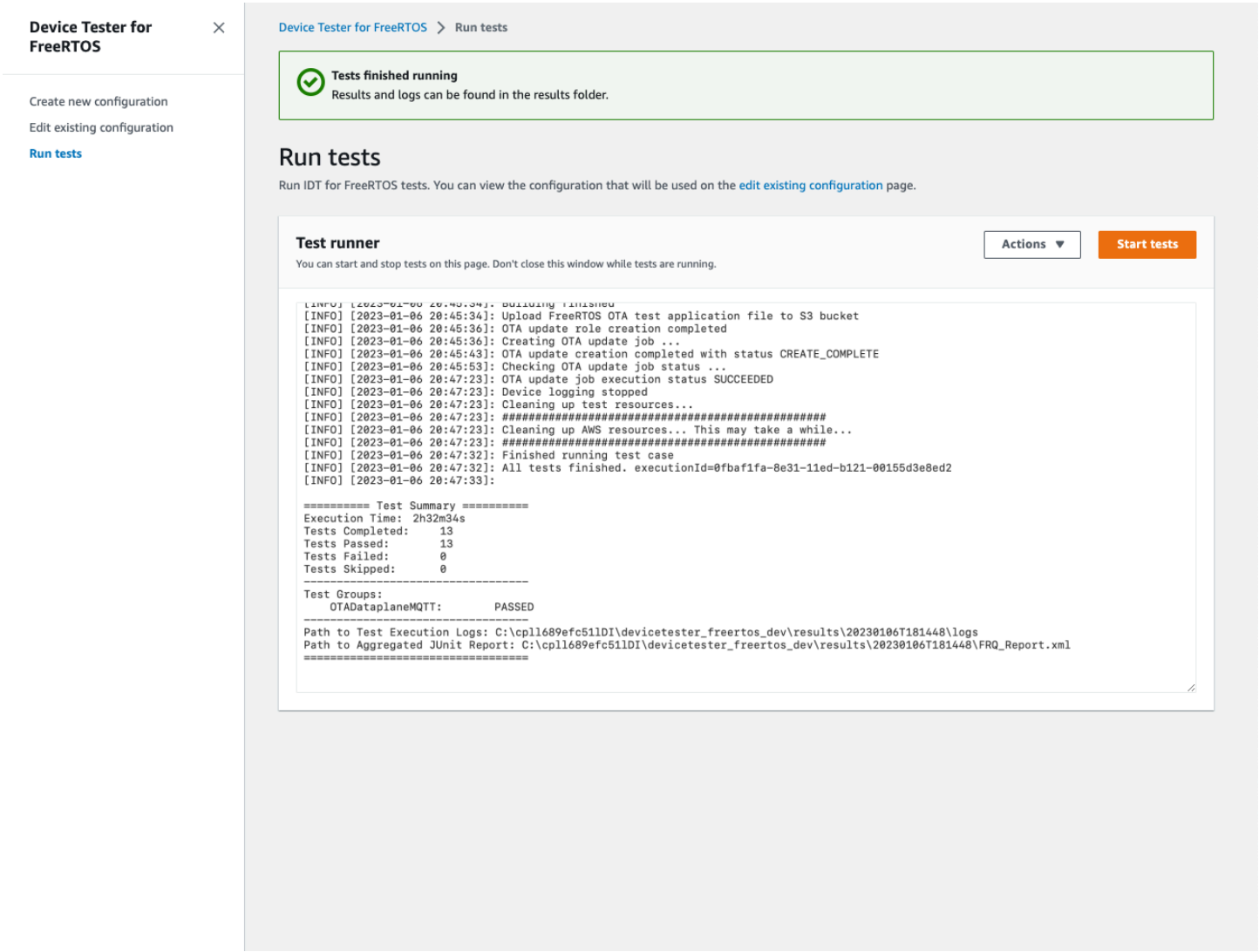
1. 在导航菜单中，选择运行测试。
2. 要开始运行测试，请选择开始测试。默认情况下，会针对您的设备配置运行所有适用的测试。适用于 FreeRTOS 的 IDT 会在所有测试完成后生成一份资格认证报告。



适用于 FreeRTOS 的 IDT 运行资格认证测试。然后，它会在测试运行器控制台中显示测试运行摘要和所有错误。测试运行完成后，您可以从以下位置查看测试结果和日志：

- 测试结果位于 *devicetester-extract-location*/results/*execution-id* 目录中。
- 测试日志位于 *devicetester-extract-location*/results/*execution-id*/logs 目录中。

有关测试结果和日志的更多信息，请参阅免费查看 IDT RTOSresults和免费查看 IDT RTOSlogs。



运行 FreeRTOS 资格赛 2.0 套件

使用 for AWS IoT Device Tester FreeRTOS 的可执行文件与 FreeRTOS 的 IDT 进行交互。以下命令行示例向您显示如何针对某个设备池（一组相同的设备）运行资格测试。

IDT v4.5.2 and later

```
devicetester_[linux | mac | win] run-suite  \
    --suite-id suite-id  \
    --group-id group-id  \
    --pool-id your-device-pool \
    --test-id test-id  \
    --userdata userdata.json
```

对某个设备池运行一组测试。userdata.json 文件必须位于 *devicetester_extract_location*/devicetester_freertos_*[win|mac|linux]*/configs/ 目录中。

> **Note**
>
> 如果您在 Windows 上运行适用于 FreeRTOS 的 IDT，请使用正斜杠 (/) 指定 userdata.json 文件的路径。

使用以下命令运行特定测试组：

```
devicetester_[linux | mac | win] run-suite  \
    --suite-id FRQ_1.99.0  \
    --group-id group-id  \
    --pool-id pool-id  \
    --userdata userdata.json
```

如果您在单个设备池上（即仅在 device.json 文件中定义了一个设备池）运行单个测试套件，则 suite-id 和 pool-id 参数为可选。

使用以下命令运行测试组中的特定测试用例：

```
devicetester_[linux | mac | win_x86-64] run-suite  \
    --group-id group-id  \
    --test-id test-id
```

您可以使用 list-test-cases 命令列出测试组中的测试用例。

适用于 FreeRTOS 的 IDT 命令行选项

group-id

（可选）要以逗号分隔的列表形式运行的测试组。如果未指定，IDT 将运行测试套件中的所有测试组。

pool-id

（可选）要测试的设备池。如果您在 device.json 中定义多个设备池，则需要执行此操作。如果您只有一个设备池，则可以省略此选项。

suite-id

（可选）要运行的测试套件版本。如果未指定，IDT 将在系统上使用测试目录中的最新版本。

test-id

（可选）要以逗号分隔的列表形式运行的测试。如果指定，则 `group-id` 必须指定单个组。

Example

```
devicetester_[linux | mac | win_x86-64] run-suite --group-id FreeRTOSVersion --
test-id FreeRTOSVersion
```

h

使用帮助选项了解有关 `run-suite` 选项的更多信息。

Example

示例

```
devicetester_[linux | mac | win_x86-64] run-suite -h
```

适用于 FreeRTOS 命令的 IDT

适用于 FreeRTOS 的 IDT 命令支持以下操作：

IDT v4.5.2 and later

`help`

列出有关指定命令的信息。

`list-groups`

列出给定套件中的组。

`list-suites`

列出可用套件。

`list-supported-products`

列出支持的产品和测试套件版本。

list-supported-versions

列出当前 IDT 版本支持的 FreeRTOS 和测试套件版本。

list-test-cases

列出指定组中的测试用例。

run-suite

对某个设备池运行一组测试。

使用 `--suite-id` 选项可以指定测试套件版本，省略它可以使用系统上的最新版本。

使用 `--test-id` 运行单个测试用例。

Example

```
devicetester_[linux | mac | win_x86-64] run-suite --group-id FreeRTOSVersion --
test-id FreeRTOSVersion
```

> **Note**
>
> 从 IDT v3.0.0 开始，IDT 在线检查较新的测试套件。有关更多信息，请参阅 测试套件版本。

免费查看 IDT RTOSresults

在运行时，IDT 会将错误写入控制台、日志文件和测试报告中。IDT 在完成资格测试套件后，会将测试运行摘要写入控制台并生成两个测试报告。可在 *devicetester-extract-location*/results/*execution-id*/ 中找到这些报告。两个报告都捕获资格测试套件执行的结果。

`awsiotdevicetester_report.xml`这是您提交给的资格测试报告， AWS 用于在 AWS 合作伙伴设备目录中列出您的设备。该报告包含以下元素：

- 适用于 FreeRTOS 版本的 IDT。
- 所测试的 FreeRTOS 版本。
- 设备所支持的 FreeRTOS 功能基于所通过的测试。
- 在 `device.json` 文件中指定的 SKU 和设备名称。
- 在 `device.json` 文件中指定的设备的功能。

- 测试用例结果的摘要汇总。
- 基于设备功能，按照所测试的库细分的测试用例结果。

FRQ_Report.xml是标准 JUnit XML 格式的报告。您可以将它集成到 CI/CD 平台中，例如 Jenkins、Bamboo 等。该报告包含以下元素：

- 测试用例结果的摘要汇总。
- 基于设备功能，按照所测试的库细分的测试用例结果。

解读 FreeRTOS 结果的 IDT

awsiotdevicetester_report.xml 或 FRQ_Report.xml 中的报告部分列出了所执行的测试的结果。

第一个 XML 标签 <testsuites> 包含测试执行的整体摘要。例如：

```
<testsuites name="FRQ results" time="5633" tests="184" failures="0"
errors="0" disabled="0">
```

<testsuites> 标签中使用的属性

name

测试套件的名称。

time

运行资格套件所用的时间（以秒为单位）。

tests

执行的测试用例数量。

failures

已运行但未通过的测试用例数。

errors

适用于 FreeRTOS 的 IDT 无法执行的测试用例数。

disabled

此属性未使用，可以忽略。

如果没有测试用例失败或错误，则您的设备符合运行 FreeRTOS 的技术要求，并且可以与服务互操作。 AWS IoT 如果您选择在 AWS 合作伙伴设备目录中列出您的设备，则可以使用此报告作为资格证据。

如果出现测试用例失败或错误，可以通过检查 <testsuites> XML 标签来确定失败的测试用例。<testsuites> 标签内部的 <testsuite> XML 标签显示测试组的测试用例结果摘要。

```
<testsuite name="FreeRTOSVersion" package="" tests="1" failures="0"
time="2" disabled="0" errors="0" skipped="0">
```

其格式类似于 <testsuites> 标签，但具有名为 skipped 的属性，此属性未使用，可以忽略。在每个 <testsuite> XML 标签内部，对于一个测试组，所执行的每个测试用例都有 <testcase> 标签。例如：

```
<testcase classname="FRQ FreeRTOSVersion" name="FreeRTOSVersion"
attempts="1"></testcase>
```

<awsproduct> 标签中使用的属性

name

所测试的产品的名称。

version

所测试的产品的版本。

features

验证的功能。标记为 required 的功能需要提交您的主板信息以供资格审核。以下代码段演示了它在 awsiotdevicetester_report.xml 文件中的显示方式。

```
<feature name="core-freertos" value="not-supported" type="required"></feature>
```

标记为 optional 的功能不是资格认证所必需的。以下代码段显示了可选功能。

```
<feature name="ota-dataplane-mqtt" value="not-supported" type="optional"></feature>
  <feature name="ota-dataplane-http" value="not-supported" type="optional"></
feature>
```

如果没有针对所需功能的测试失败或错误，则设备满足运行 FreeRTOS 的技术要求并可以与 AWS IoT 服务互操作。如果您想要在 AWS 合作伙伴设备目录中列出您的设备，则可以使用此报告作为资格证明。

如果出现测试失败或错误，则可以通过检查 <testsuites> XML 标签来确定失败的测试。<testsuites> 标签内的 <testsuite> XML 标签显示了测试组的测试结果摘要。例如：

```
<testsuite name="FreeRTOSVersion" package="" tests="1" failures="1" time="2"
 disabled="0" errors="0" skipped="0">
```

其格式与 <testsuites> 标签类似，但有一个未使用并可忽略的 skipped 属性。在每个 <testsuite> XML 标签内部，对于一个测试组，所执行的每个测试都有 <testcase> 标签。例如：

```
<testcase classname="FreeRTOSVersion" name="FreeRTOSVersion"></testcase>
```

<testcase> 标签中使用的属性

name

测试用例的名称。

attempts

适用于 FreeRTOS 的 IDT 执行测试用例的次数。

当测试失败或出现错误时，将会在 <testcase> 标签中添加包含用于故障排除的信息的 <failure> 或 <error> 标签。例如：

```
<testcase classname="FRQ FreeRTOSVersion" name="FreeRTOSVersion">
      <failure type="Failure">Reason for the test case failure</failure>
      <error>Reason for the test case execution error</error>
  </testcase>
```

有关更多信息，请参阅 排查 错误。

免费查看 IDT RTOSlogs

您可以在 *devicetester-extract-location*/results/*execution-id*/logs 中找到适用于 FreeRTOS 的 IDT 从测试执行生成的日志。它会生成两组日志：

- test_manager.log

包含从适用于 FreeRTOS 的 IDT 生成的日志（例如，与配置和报告生成相关的日志）。

- *test_group_id*/*test_case_id*/*test_case_id*.log

  测试用例的日志文件，包括来自所测设备的输出。日志文件根据运行的测试组和测试用例命名。

带有 FreeRTOS 资格套件 1.0 的 IDT (FRQ 1.0)

> **Important**
>
> 截至 2022 年 10 月， AWS IoT Device Tester AWS IoT FreeRTOS 资格认证 (FRQ) 1.0 不会生成已签署的资格报告。使用 IDT AWS IoT FRQ 1.0 版本，您无法通过AWS 设备资格认证计划将新 FreeRTOS 设备列入合作伙伴设备目录。AWS 虽然您无法使用 IDT FRQ 1.0 确定 FreeRTOS 设备的资格，但您可以继续使用 FRQ 1.0 测试 FreeRTOS 设备。我们建议您使用 IDT FRQ 2.0 来进行资格认证，并在 AWS 合作伙伴设备目录中列出 FreeRTOS 设备。

您可以使用 IDT 进行 FreeRTOS 资格认证，以验证 FreeRTOS 操作系统是否在您的设备上本地运行并且可以与之通信。 AWS IoT具体而言，它会验证正确实施了 FreeRTOS 库的移植层接口。它还使用执行 end-to-end测试 AWS IoT Core。例如，它验证您的主板是否能够发送和接收 MQTT 消息并正确处理它们。IDT 为 FreeRTOS 运行的测试是在 FreeRTOS 存储库中定义的。 GitHub

测试作为刷写到主板中的嵌入应用程序运行。应用程序二进制映像包括 FreeRTOS、半导体供应商移植的 FreeRTOS 接口以及主板设备驱动程序。测试的目的是验证移植的 FreeRTOS 接口在设备驱动程序上正常工作。

IDT for FreeRTOS 会生成测试报告，您可以提交这些报告 AWS IoT 以将您的硬件添加到 AWS 合作伙伴设备目录中。有关更多信息，请参阅 AWS 设备资格认证计划。

适用于 FreeRTOS 的 IDT 在与待测试主板连接的主机（Windows、Mac 或 Linux）上运行。IDT 执行测试用例并聚合结果。它还提供命令行界面来管理测试执行。

除了测试设备外，适用于 FreeRTOS 的 IDT 还会创建资源（例如， AWS IoT 东西、FreeRTOS 组、Lambda 函数等）来简化认证过程。要创建这些资源，适用于 FreeRTOS 的 IDT 使用中`config.json`配置 AWS 的凭据代表您进行 API 调用。这些资源将在测试过程的不同时间进行预置。

当您在主机上运行适用于 FreeRTOS 的 IDT 时，它将执行以下步骤：

1. 加载和验证您的设备和凭证配置。
2. 使用所需的本地资源和云资源执行选定测试。
3. 清除本地资源和云资源。
4. 生成测试报告，指明您的主板是否已通过资格认证所需的测试。

主题

- 设置 1.0 资格先决条件
- 首次测试您的微控制器板
- 使用 IDT 用户界面运行 FreeRTOS 资格套件
- 运行蓝牙低功耗测试
- 运行 FreeRTOS 资格套件
- 查看 IDT for FreeRTOS 结果
- 解读 FreeRTOS 结果的 IDT
- 查看 FreeRTOS 日志的 IDT

设置 1.0 资格先决条件

本节介绍使用测试微控制器的 AWS IoT Device Tester先决条件。

下载 FreeRTOS

你可以使用以下命令GitHub从中下载 FreeRTOS 的发行版：

```
git clone --branch <FREERTOS_RELEASE_VERSION> --recurse-submodules https://github.com/
aws/amazon-freertos.git
cd amazon-freertos
git submodule update --checkout --init --recursive
```

其中 <FREERTOS_RELEASE_VERSION> 是与 支持的版本 AWS IoT Device Tester 中列出的 IDT 版本对应的 FreeRTOS 版本（例如 202007.00）。这样可以确保您拥有完整的源代码，包括子模块，并且使用适用于您的 FreeRTOS 版本的 IDT 的正确版本，反之亦然。

Windows 的路径长度限制为 260 个字符。FreeRTOS 的路径结构是多级深层结构，因此如果您使用 Windows，请确保文件路径不超过 260 个字符的限制。例如，将 FreeRTOS 克隆到 `C:\FreeRTOS` 而不是 `C:\Users\username\programs\projects\myproj\FreeRTOS\`。

使用 LTS 库获得 FreeRTOS 资格

- 要在 AWS 合作伙伴设备目录中将您的微控制器指定为支持基于长期支持 (LTS) 的 FreeRTOS 版本，您必须提供清单文件。有关更多信息，请参阅《FreeRTOS 资格认证指南》中的 FreeRTOS 资格认证检查清单。
- 为了验证您的微控制器是否支持基于 LTS 的 FreeRTOS 版本并使其有资格提交到 AWS 合作伙伴设备目录，您必须使用 (AWS IoT Device Tester IDT) 和 FreeRTOS 资格认证 (FRQ) 测试套件版本 v1.4.x。
- 对基于 LTS 的 FreeRTOS 版本的支持仅限于 202012.xx 版本的 FreeRTOS。

下载适用于 FreeRTOS 的 IDT

为了执行资格认证测试，FreeRTOS 的每个版本都有对应的适用于 FreeRTOS 的 IDT 版本。从 支持的版本 AWS IoT Device Tester 下载适用于 FreeRTOS 的 ITD 的相应版本。

将适用于 FreeRTOS 的 IDT 提取到文件系统上您具有读写权限的位置。由于 Microsoft Windows 对路径长度具有字符数限制，因此将适用于 FreeRTOS 的 IDT 提取到根目录，如 `C:\` 或 `D:\`。

> **Note**
>
> 我们不建议多个用户从共享位置（如 NFS 目录或 Windows 网络共享文件夹）运行 IDT。这可能会导致崩溃或数据损坏。我们建议您将 IDT 包解压缩到本地驱动器。

创建和配置 AWS 账户

注册获取 AWS 账户

如果您没有 AWS 账户，请完成以下步骤来创建一个。

要注册 AWS 账户

1. 打开https://portal.aws.amazon.com/billing/注册。
2. 按照屏幕上的说明操作。

   注册过程的一部分涉及接听电话或短信，并在电话键盘上输入验证码。

当您注册时 AWS 账户，就会创建AWS 账户根用户一个。根用户有权访问该账户中的所有 AWS 服务 和资源。作为最佳安全实践，请为用户分配管理访问权限，并且只使用根用户来执行需要根用户访问权限的任务。

AWS 注册过程完成后会向您发送一封确认电子邮件。您可以随时前往 https://aws.amazon.com/并选择“我的账户”，查看您当前的账户活动并管理您的账户。

创建具有管理访问权限的用户

注册后，请保护您的安全 AWS 账户 AWS 账户根用户 AWS IAM Identity Center，启用并创建管理用户，这样您就不会使用 root 用户执行日常任务。

保护你的 AWS 账户根用户

1. 选择 Root 用户并输入您的 AWS 账户 电子邮件地址，以账户所有者的身份登录。AWS Management Console在下一页上，输入您的密码。

   要获取使用根用户登录方面的帮助，请参阅《AWS 登录 用户指南》中的 Signing in as the root user。
2. 为您的根用户启用多重身份验证（MFA）。

   有关说明，请参阅 I A M 用户指南中的为 AWS 账户 根用户启用虚拟 MFA 设备（控制台）。

创建具有管理访问权限的用户

1. 启用 IAM Identity Center。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的 Enabling AWS IAM Identity Center。
2. 在 IAM Identity Center 中，为用户授予管理访问权限。

   有关使用 IAM Identity Center 目录 作为身份源的教程，请参阅《用户指南》 IAM Identity Center 目录中的使用默认设置配置AWS IAM Identity Center 用户访问权限。

以具有管理访问权限的用户身份登录

- 要使用您的 IAM Identity Center 用户身份登录，请使用您在创建 IAM Identity Center 用户时发送到您的电子邮件地址的登录网址。

有关使用 IAM Identity Center 用户登录的帮助，请参阅AWS 登录 用户指南中的登录 AWS 访问门户。

将访问权限分配给其他用户

1. 在 IAM Identity Center 中，创建一个权限集，该权限集遵循应用最低权限的最佳做法。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的 Create a permission set。
2. 将用户分配到一个组，然后为该组分配单点登录访问权限。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的 Add groups。

AWS IoT Device Tester 托管策略

`AWSIoTDeviceTesterForFreeRTOSFullAccess`托管策略包含版本检查、auto update 功能和指标收集的以下 AWS IoT Device Tester 权限。

- `iot-device-tester:SupportedVersion`

  授 AWS IoT Device Tester 予获取受支持产品、测试套件和 IDT 版本列表的权限。
- `iot-device-tester:LatestIdt`

  授 AWS IoT Device Tester 予获取可供下载的最新 IDT 版本的权限。
- `iot-device-tester:CheckVersion`

  授 AWS IoT Device Tester 予检查 IDT、测试套件和产品的版本兼容性的权限。
- `iot-device-tester:DownloadTestSuite`

  授 AWS IoT Device Tester 予下载测试套件更新的权限。
- `iot-device-tester:SendMetrics`

  授 AWS 予收集有关 AWS IoT Device Tester 内部使用情况的指标的权限。

（可选）安装 AWS Command Line Interface

您可能更喜欢使用 AWS CLI 来执行某些操作。如果您没有安装 AWS CLI ，请按照安装 AWS CLI 中的说明执行操作。

通过aws configure从命令行运行来为要使用的 AWS 区域进行配置。 AWS CLI 有关支持 IDT for FreeRTOS 的 AWS 区域的信息，AWS 请参阅区域和终端节点。有关 aws configure 的更多信息，请参阅使用 aws configure 进行快速配置。

首次测试您的微控制器板

移植 FreeRTOS 接口时，您可以使用适用于 FreeRTOS 的 IDT 进行测试。在为主板的设备驱动程序移植了 FreeRTOS 接口后，您就需要在微 AWS IoT Device Tester 控制器主板上运行资格测试。

添加库移植层

要为您的设备移植 FreeRTOS，请按照《FreeRTOS 移植指南》中的说明操作。

配置您的 AWS 凭证

您需要配置您的 AWS 凭据才能与 AWS 云端通信。 AWS IoT Device Tester 有关更多信息，请参阅设置用于开发的 AWS 凭证和区域。必须在*devicetester_extract_location*/devicetester_afreertos_*[win|mac|linux]*/configs/config.json配置文件中指定有效的 AWS 凭据。

主题

- 在适用于 FreeRTOS 的 IDT 中创建设备池
- 配置构建、刷写和测试设置

在适用于 FreeRTOS 的 IDT 中创建设备池

要测试的设备排列在设备池中。每个设备池包含一个或多个相同的设备。您可以配置适用于 FreeRTOS 的 IDT 来测试某个池中的单个设备或多个设备。为了加快资格认证测试过程，适用于 FreeRTOS 的 IDT 可以并行测试具有相同规格的设备。它使用轮询方法，在设备池中的各个设备上执行不同的测试组。

您可以通过编辑 `configs` 文件夹中 `device.json` 模板的 `devices` 部分，将一个或多个设备添加到设备池中。

> **Note**
>
> 同一个池中的所有设备必须具有相同的技术规格和 SKU。

要允许针对不同测试组并行构建源代码，适用于 FreeRTOS 的 IDT 需要将源代码复制到适用于 FreeRTOS 的 IDT 提取文件夹的结果文件夹中。您的构建或刷写命令中的源代码路径必须使用 `testdata.sourcePath` 或 `sdkPath` 变量来引用。适用于 FreeRTOS 的 IDT 使用所复制源代码的临时路径来替换此变量。有关更多信息，请参阅适用于 FreeRTOS 变量的 IDT。

以下 `device.json` 示例文件用于创建具有多个设备的设备池：

```
[
    {
        "id": "pool-id",
        "sku": "sku",
        "features": [
            {
                "name": "WIFI",
                "value": "Yes | No"
            },
            {
                "name": "Cellular",
                "value": "Yes | No"
            },
            {
                "name": "OTA",
                "value": "Yes | No",
                "configs": [
                    {
                        "name": "OTADataPlaneProtocol",
                        "value": "HTTP | MQTT"
                    }
                ]
            },
            {
                "name": "BLE",
                "value": "Yes | No"
            },
            {
                "name": "TCP/IP",
                "value": "On-chip | Offloaded | No"
            },
            {
                "name": "TLS",
                "value": "Yes | No"
            },
            {
```

```
                "name": "PKCS11",
                "value": "RSA | ECC | Both | No"
            },
            {
                "name": "KeyProvisioning",
                "value": "Import | Onboard | No"
            }
        ],

        "devices": [
          {
            "id": "device-id",
            "connectivity": {
              "protocol": "uart",
              "serialPort": "/dev/tty*"
            },
            **********Remove the section below if the device does not support onboard
 key generation***************
            "secureElementConfig" : {
              "publicKeyAsciiHexFilePath": "absolute-path-to/public-key-txt-file:
 contains-the-hex-bytes-public-key-extracted-from-onboard-private-key",
              "secureElementSerialNumber": "secure-element-serialNo-value",
              "preProvisioned"           : "Yes | No"
            },

 ********************************************************************************************
            "identifiers": [
              {
                "name": "serialNo",
                "value": "serialNo-value"
              }
            ]
          }
        ]
    }
]
```

在 device.json 文件中使用以下属性：

id

用户定义的字母数字 ID，用于唯一地标识设备池。属于同一个池的设备必须具有相同的类型。运行一组测试时，池中的设备将用于对工作负载进行并行化处理。

sku

唯一标识您正在测试的主板的字母数字值。该 SKU 用于跟踪符合条件的主板。

> **Note**
>
> 如果您想在 AWS 合作伙伴设备目录中发布您的主板，则在此处指定的 SKU 必须与您在发布过程中使用的 SKU 相匹配。

features

包含设备支持的功能的阵列。 AWS IoT Device Tester 使用此信息来选择要运行的资格测试。

支持的值为：

TCP/IP

表示您的主板是否支持堆栈，以及该 TCP/IP 堆栈是片上支持 (MCU) 还是支持卸载到其他模块。 TCP/IP 是资格认证所必需的。

WIFI

指示您的主板是否具有 Wi-Fi 功能。如果将 `Cellular` 设置为 `Yes`，则必须设置为 `No`。

Cellular

指示您的主板是否具有蜂窝功能。如果将 `WIFI` 设置为 `Yes`，则必须设置为 `No`。当此功能设置为时`Yes`，将使用 AWS t2.micro EC2 实例执行 FullSecureSockets 测试，这可能会给您的账户带来额外费用。有关更多信息，请参阅 Amazon EC2 定价。

TLS

指示您的主板是否支持 TLS。资格认证需要使用 TLS。

PKCS11

表示主板支持的公钥加密算法。 PKCS11 是资格认证所必需的。支持的值为 `ECC`、`RSA`、`Both` 和 `No`。`Both` 表示主板同时支持 `ECC` 和 `RSA` 算法。

KeyProvisioning

指示将受信任的 X.509 客户端证书写入主板的方法。有效值为 `Import`、`Onboard` 和 `No`。资格认证需要进行密钥预置。

- 如果您的主板允许导入私有密钥，请使用 `Import`。IDT 将创建一个私有密钥并将其构建为 FreeRTOS 源代码。
- 如果您的主板支持生成板载私有密钥（例如，如果您的设备具有安全元件，或者如果您希望生成自己的设备密钥对和证书），请使用 `Onboard`。确保您在每个设备部分中添加一个 `secureElementConfig` 元素，并将公有密钥文件的绝对路径放在 `publicKeyAsciiHexFilePath` 字段中。
- 如果您的主板不支持密钥预置，请使用 `No`。

OTA

表示您的主板是否支持 over-the-air (OTA) 更新功能。`OtaDataPlaneProtocol` 属性指示设备支持哪个 OTA 数据平面协议。如果设备不支持 OTA 功能，则忽略此属性。在选择 "Both" 后，由于同时运行 MQTT、HTTP 和混合测试，因此 OTA 测试执行时间会延长。

> **Note**
>
> 从 IDT v4.1.0 开始，`OtaDataPlaneProtocol` 仅接受 `HTTP` 和 `MQTT` 作为支持的值。

BLE

指示您的主板是否支持低功耗蓝牙 (BLE) 功能。

devices.id

用户定义的测试的设备的唯一标识符。

devices.connectivity.protocol

用于与此设备通信的通信协议。支持的值为：`uart`。

devices.connectivity.serialPort

主机连接到所测试设备时使用的串行端口。

devices.secureElementConfig.PublicKeyAsciiHexFilePath

该文件的绝对路径，包含从板载私有密钥中提取的十六进制字节公有密钥。

示例格式：

```
3059 3013 0607 2a86 48ce 3d02 0106 082a
```

```
8648 ce3d 0301 0703 4200 04cd 6569 ceb8
1bb9 1e72 339f e8cf 60ef 0f9f b473 33ac
6f19 1813 6999 3fa0 c293 5fae 08f1 1ad0
41b7 345c e746 1046 228e 5a5f d787 d571
dcb2 4e8d 75b3 2586 e2cc 0c
```

如果您的公有密钥是 .der 格式，则可以直接对公有密钥进行十六进制编码以生成十六进制文件。

.der 公有密钥生成十六进制文件的命令示例：

```
xxd -p pubkey.der > outFile
```

如果您的公有密钥是 .pem 格式，则可以提取 base64 编码部分，将其解码为二进制格式，然后对其进行十六进制编码以生成十六进制文件。

例如，使用以下命令可为 .pem 公有密钥生成十六进制文件：

1. 提取密钥的 base64 编码部分（去掉页眉和页脚）并将其存储在文件中，例如，将其命名为 `base64key`，然后运行以下命令将其转换为 .der 格式：

   ```
   base64 –decode base64key > pubkey.der
   ```

2. 运行 `xxd` 命令以将其转换为十六进制格式。

   ```
   xxd -p pubkey.der > outFile
   ```

`devices.secureElementConfig.SecureElementSerialNumber`

（可选）安全元件的序列号。运行 FreeRTOS demo/test 项目时，当序列号与设备公钥一起打印出来时，请提供此字段。

`devices.secureElementConfig.preProvisioned`

（可选）如果设备预配置了带有锁定凭证的安全元件，无法导入、创建或销毁对象，则设置为“是”。只有在 `features` 将 `KeyProvisioning` 设置为“注册”，并且将 `PKCS11` 设置为“ECC”时，此配置才会生效。

`identifiers`

（可选）任意名称/值对的数组。您可以在下一部分所述的构建和刷写命令中使用这些值。

配置构建、刷写和测试设置

要让适用于 FreeRTOS 的 IDT 自动构建并刷写主板，您必须配置 IDT 针对硬件运行构建和刷写命令。构建和刷写设置在位于 config 文件夹的 userdata.json 模板文件中配置。

为测试设备配置设置

构建、刷写和测试设置在 configs/userdata.json 文件中进行。我们通过在 customPath 中加载客户端和服务器证书和密钥来支持 Echo 服务器配置。有关更多信息，请参阅《FreeRTOS 移植指南》中的设置 Echo 服务器。以下 JSON 示例显示如何配置适用于 FreeRTOS 的 IDT 来测试多个设备：

```
{
    "sourcePath": "/absolute-path-to/freertos",
    "vendorPath": "{{testData.sourcePath}}/vendors/vendor-name/boards/board-name",
    // ***********The sdkConfiguration block below is needed if you are not using the
 default, unmodified FreeRTOS repo.
    // In other words, if you are using the default, unmodified FreeRTOS repo then
 remove this block***************
    "sdkConfiguration": {
        "name": "sdk-name",
        "version": "sdk-version",
        "path": "/absolute-path-to/sdk"
    },
    "buildTool": {
        "name": "your-build-tool-name",
        "version": "your-build-tool-version",
        "command": [
            "{{config.idtRootPath}}/relative-path-to/build-parallel.sh
 {{testData.sourcePath}} {{enableTests}}"
        ]
    },
    "flashTool": {
        "name": "your-flash-tool-name",
        "version": "your-flash-tool-version",
        "command": [
            "/{{config.idtRootPath}}/relative-path-to/flash-parallel.sh
 {{testData.sourcePath}} {{device.connectivity.serialPort}} {{buildImageName}}"
        ],
        "buildImageInfo" : {
            "testsImageName": "tests-image-name",
            "demosImageName": "demos-image-name"
        }
    },
```

```
    "testStartDelayms": 0,
    "clientWifiConfig": {
        "wifiSSID": "ssid",
        "wifiPassword": "password",
        "wifiSecurityType": "eWiFiSecurityOpen | eWiFiSecurityWEP | eWiFiSecurityWPA |
eWiFiSecurityWPA2 | eWiFiSecurityWPA3"
    },
    "testWifiConfig": {
        "wifiSSID": "ssid",
        "wifiPassword": "password",
        "wifiSecurityType": "eWiFiSecurityOpen | eWiFiSecurityWEP | eWiFiSecurityWPA |
eWiFiSecurityWPA2 | eWiFiSecurityWPA3"
    },
    //**********
    //This section is used to start echo server based on server certificate generation
method,
    //When certificateGenerationMethod is set as Automatic specify the eccCurveFormat
to generate certifcate and key based on curve format,
    //When certificateGenerationMethod is set as Custom specify the certificatePath and
PrivateKeyPath to be used to start echo server
    //**********
    "echoServerCertificateConfiguration": {
      "certificateGenerationMethod": "Automatic | Custom",
      "customPath": {
          "clientCertificatePath":"/path/to/clientCertificate",
          "clientPrivateKeyPath": "/path/to/clientPrivateKey",
          "serverCertificatePath":"/path/to/serverCertificate",
          "serverPrivateKeyPath": "/path/to/serverPrivateKey"
      },
    "eccCurveFormat": "P224 | P256 | P384 | P521"
    },
    "echoServerConfiguration": {
        "securePortForSecureSocket": 33333, // Secure tcp port used by SecureSocket
test. Default value is 33333. Ensure that the port configured isn't blocked by the
firewall or your corporate network
        "insecurePortForSecureSocket": 33334, // Insecure tcp port used by SecureSocket
test. Default value is 33334. Ensure that the port configured isn't blocked by the
firewall or your corporate network
        "insecurePortForWiFi": 33335 // Insecure tcp port used by Wi-Fi test. Default
value is 33335. Ensure that the port configured isn't blocked by the firewall or your
corporate network
    },
    "otaConfiguration": {
```

```
        "otaFirmwareFilePath": "{{testData.sourcePath}}/relative-path-to/ota-image-
generated-in-build-process",
        "deviceFirmwareFileName": "ota-image-name-on-device",
        "otaDemoConfigFilePath": "{{testData.sourcePath}}/relative-path-to/ota-demo-
config-header-file",
        "codeSigningConfiguration": {
            "signingMethod": "AWS | Custom",
            "signerHashingAlgorithm": "SHA1 | SHA256",
            "signerSigningAlgorithm": "RSA | ECDSA",
            "signerCertificate": "arn:partition:service:region:account-
id:resource:qualifier | /absolute-path-to/signer-certificate-file",
            "signerCertificateFileName": "signerCertificate-file-name",
            "compileSignerCertificate": boolean,
            // ***********Use signerPlatform if you choose aws for
 signingMethod***************
            "signerPlatform": "AmazonFreeRTOS-Default | AmazonFreeRTOS-TI-CC3220SF",
            "untrustedSignerCertificate": "arn:partition:service:region:account-
id:resourcetype:resource:qualifier",
            // ***********Use signCommand if you choose custom for
 signingMethod***************
            "signCommand": [
                "/absolute-path-to/sign.sh {{inputImageFilePath}}
 {{outputSignatureFilePath}}"
            ]
        }
    },
    // ***********Remove the section below if you're not configuring
 CMake***************
    "cmakeConfiguration": {
        "boardName": "board-name",
        "vendorName": "vendor-name",
        "compilerName": "compiler-name",
        "frToolchainPath": "/path/to/freertos/toolchain",
        "cmakeToolchainPath": "/path/to/cmake/toolchain"
    },
    "freertosFileConfiguration": {
        "required": [
            {
                "configName": "pkcs11Config",
                "filePath": "{{testData.sourcePath}}/vendors/vendor-name/boards/board-
name/aws_tests/config_files/core_pkcs11_config.h"
            },
            {
                "configName": "pkcs11TestConfig",
```

```
                "filePath": "{{testData.sourcePath}}/vendors/vendor-name/boards/board-
name/aws_tests/config_files/iot_test_pkcs11_config.h"
            }
        ],
        "optional": [
            {
                "configName": "otaAgentTestsConfig",
                "filePath": "{{testData.sourcePath}}/vendors/vendor-name/boards/board-
name/aws_tests/config_files/ota_config.h"
            },
            {
                "configName": "otaAgentDemosConfig",
                "filePath": "{{testData.sourcePath}}/vendors/vendor-name/boards/board-
name/aws_demos/config_files/ota_config.h"
            },
            {
                "configName": "otaDemosConfig",
                "filePath": "{{testData.sourcePath}}/vendors/vendor-name/boards/board-
name/aws_demos/config_files/ota_demo_config.h"
            }
        ]
    }
}
```

下面列出了在 userdata.json 中使用的属性：

sourcePath

移植的 FreeRTOS 源代码的根目录的路径。对于使用开发工具包的并行测试，可使用 {{userData.sdkConfiguration.path}} 占位符对 sourcePath 进行设置。例如：

```
{ "sourcePath":"{{userData.sdkConfiguration.path}}/freertos" }
```

vendorPath

供应商特定 FreeRTOS 代码的路径。对于串行测试，vendorPath 可以设置为绝对路径。例如：

```
{ "vendorPath":"C:/path-to-freertos/vendors/espressif/boards/esp32" }
```

对于并行测试，vendorPath 可以使用 {{testData.sourcePath}} 占位符进行设置。例如：

```
{ "vendorPath":"{{testData.sourcePath}}/vendors/espressif/boards/esp32" }
```

只有在不使用开发工具包的情况下运行时才需要 vendorPath 变量，否则可将其删除。

> **Note**
>
> 在不使用开发工具包的情况下并行运行测试时，vendorPath、buildTool、flashTool 字段中必须使用 {{testData.sourcePath}} 占位符。使用单个设备运行测试时，vendorPath、buildTool、flashTool 字段中必须使用绝对路径。使用开发工具包运行时，必须在 sourcePath、buildTool 和 flashTool 命令中使用 {{sdkPath}} 占位符。

sdkConfiguration

除了移植所需之外，如果您要对 FreeRTOS 的其他文件和文件夹结构的任何修改进行资格认证，则需要在此数据块中配置开发工具包信息。如果您不符合在开发工具包中使用移植的 FreeRTOS 的资格，则应完全省略该数据块。

sdkConfiguration.name

在 FreeRTOS 中使用的开发工具包的名称。如果您没有使用开发工具包，则应省略整个 sdkConfiguration 数据块。

sdkConfiguration.version

在 FreeRTOS 中使用的开发工具包的版本。如果您没有使用开发工具包，则应省略整个 sdkConfiguration 数据块。

sdkConfiguration.path

包含 FreeRTOS 代码的开发工具包目录的绝对路径。如果您没有使用开发工具包，则应省略整个 sdkConfiguration 数据块。

buildTool

您的生成脚本（.bat 或 .sh）的完整路径，该脚本包含用于生成源代码的命令。构建命令中对源代码路径的所有引用都必须替换为 AWS IoT Device Tester 变量{{testdata.sourcePath}}，对 SDK 路径的引用应替换为{{sdkPath}}。使用 {{config.idtRootPath}} 占位符引用绝对或相对 IDT 路径。

testStartDelayms

指定 FreeRTOS 测试运行器在开始运行测试之前将等待多少毫秒。如果由于网络或其他延迟，待测试设备在 IDT 有机会连接并开始日志记录之前就开始输出重要的测试信息，这可能很有用。允许的

最大值为 30000 毫秒（30 秒）。此值仅适用于 FreeRTOS 测试组，不适用于不使用 FreeRTOS 测试运行器的其他测试组，例如 OTA 测试。

flashTool

您的刷入脚本（.sh 或 .bat）的完整路径，该脚本应该包含您设备的刷入命令。在刷写命令中，必须将对源代码路径的所有引用替换为适用于 FreeRTOS 的 IDT 的变量 {{testdata.sourcePath}}，并将对开发工具包路径的所有引用替换为适用于 FreeRTOS 的 IDT 变量 {{sdkPath}}。使用 {{config.idtRootPath}} 占位符引用绝对或相对 IDT路径。

buildImageInfo

testsImageName

从 *freertos-source*/tests 文件夹构建测试时由构建命令生成的文件的名称。

demosImageName

从 *freertos-source*/demos 文件夹构建测试时由构建命令生成的文件的名称。

clientWifiConfig

客户端 Wi-Fi 配置。Wi-Fi 库测试要求 MCU 主板连接到两个接入点。（两个接入点可以是相同的。）此属性配置第一个接入点的 Wi-Fi 设置。一些 Wi-Fi 测试用例需要接入点有一定的安全性，不能处于开放状态。确保两个接入点与运行 IDT 的主机位于同一个子网中。

wifi_ssid

Wi-Fi SSID。

wifi_password

Wi-Fi 密码。

wifiSecurityType

使用的 Wi-Fi 安全类型。以下值之一：

- eWiFiSecurityOpen
- eWiFiSecurityWEP
- eWiFiSecurityWPA
- eWiFiSecurityWPA2
- eWiFiSecurityWPA3

> **Note**
>
> 如果您的主板不支持 Wi-Fi，您仍须在 `device.json` 文件中包含 `clientWifiConfig` 部分，但您可以忽略这些属性的值。

`testWifiConfig`

测试 Wi-Fi 配置。Wi-Fi 库测试要求 MCU 主板连接到两个接入点。（两个接入点可以是相同的。）此属性配置第二个接入点的 Wi-Fi 设置。一些 Wi-Fi 测试用例需要接入点有一定的安全性，不能处于开放状态。确保两个接入点与运行 IDT 的主机位于同一个子网中。

`wifiSSID`

Wi-Fi SSID。

`wifiPassword`

Wi-Fi 密码。

`wifiSecurityType`

使用的 Wi-Fi 安全类型。以下值之一：

- `eWiFiSecurityOpen`
- `eWiFiSecurityWEP`
- `eWiFiSecurityWPA`
- `eWiFiSecurityWPA2`
- `eWiFiSecurityWPA3`

> **Note**
>
> 如果您的主板不支持 Wi-Fi，您仍须在 `device.json` 文件中包含 `testWifiConfig` 部分，但您可以忽略这些属性的值。

`echoServerCertificateConfiguration`

用于安全套接字测试的可配置 Echo 服务器证书生成占位符。该字段为必填。

`certificateGenerationMethod`

指定服务器证书是自动生成的，还是手动提供的。

customPath

如果 certificateGenerationMethod 为“自定义”，则 certificatePath 和 privateKeyPath 为必需项。

certificatePath

指定服务器证书的文件路径。

privateKeyPath

指定私有密钥的文件路径。

eccCurveFormat

指定主板支持的曲线格式。在 device.json 中将 PKCS11 设置为“ECC”时必需。有效值为“P224”、“P256”、“P384”或“P521”。

echoServerConfiguration

用于 WiFi 和安全套接字测试的可配置回声服务器端口。该字段是可选的。

securePortForSecureSocket

用于设置使用 TLS 的 Echo 服务器以进行安全套接字测试的端口。默认值是 33333。确保配置的端口未被防火墙或公司网络阻止。

insecurePortForSecureSocket

用于设置不使用 TLS 的 Echo 服务器以进行安全套接字测试的端口。测试中使用的默认值为 33334。确保配置的端口未被防火墙或公司网络阻止。

insecurePortForWiFi

用于设置不带 TLS 的回声服务器进行 WiFi 测试的端口。测试中使用的默认值为 33335。确保配置的端口未被防火墙或公司网络阻止。

otaConfiguration

OTA 配置。[可选]

otaFirmwareFilePath

在生成之后，所创建 OTA 映像的完整路径。例如 {{testData.sourcePath}}/*relative-path/to/ota/image/from/source/root*。

deviceFirmwareFileName

MCU 设备上 OTA 固件所在的完整文件路径。有些设备不使用此字段，但您仍必须提供一个值。

otaDemoConfigFilePath

位于 *afr-source*/vendors/vendor/boards/board/aws_demos/config_files/ 中的 aws_demo_config.h 的完整路径。这些文件包含在 FreeRTOS 提供的移植代码模板中。

codeSigningConfiguration

代码签名配置。

signingMethod

代码签名方法。可能的值为 AWS 或 Custom。

> **Note**
>
> 对于北京和宁夏区域，请使用 Custom。这些区域不支持 AWS 代码签名。

signerHashingAlgorithm

设备所支持的哈希算法。可能的值为 SHA1 或 SHA256。

signerSigningAlgorithm

设备所支持的签名算法。可能的值为 RSA 或 ECDSA。

signerCertificate

用于 OTA 的可信证书。

对于 AWS 代码签名方法，请使用上传到的可信证书的 Amazon 资源名称 (ARN)。 AWS Certificate Manager

对于自定义代码签名方法，请使用签署人证书文件的绝对路径。

有关创建可信证书的更多信息，请参阅 创建代码签名证书。

signerCertificateFileName

设备上代码签署证书的文件名。此值必须与您在运行 aws acm import-certificate 命令时提供的文件名相匹配。

有关更多信息，请参阅 创建代码签名证书。

compileSignerCertificate

true如果未配置或刷新代码签名者签名验证证书，则设置为，因此必须将其编译到项目中。AWS IoT Device Tester 获取可信证书并将其编译为。aws_codesigner_certifiate.h

untrustedSignerCertificate

在某些 OTA 测试中用作不可信证书的第二个证书的 ARN 或文件路径。有关创建证书的更多信息，请参阅创建代码签名证书。

signerPlatform

C AWS ode Signer 在创建 OTA 更新任务时使用的签名和哈希算法。目前，此字段的可能值为 AmazonFreeRTOS-TI-CC3220SF 和 AmazonFreeRTOS-Default。

- 如果为 SHA1 和 RSA，则选择 AmazonFreeRTOS-TI-CC3220SF。
- 如果为 SHA256 和 ECDSA，则选择 AmazonFreeRTOS-Default。

如果您的配置需要 SHA256 | RSA 或 SHA1 | ECDSA，请与我们联系以获取进一步的支持。

如果您为 signingMethod 选择 Custom，则配置 signCommand。

signCommand

用于执行自定义代码签名的命令。您可以在 /configs/script_templates 目录中找到模板。

命令中需要两个占位符 {{inputImageFilePath}} 和 {{outputSignatureFilePath}}。{{inputImageFilePath}} 是要签名的由 IDT 构建的映像的文件路径。{{outputSignatureFilePath}} 是将由脚本生成的签名的文件路径。

cmakeConfiguration

CMake 配置 [可选]

> **Note**
>
> 要执行 CMake 测试用例，必须提供主板名称、供应商名称和frToolchainPath或compilerName。cmakeToolchainPath如果您有 CMake工具链的自定义路径，也可以提供。

boardName

要测试的主板的名称。主板名称应该与 *path/to/afr/source/code*/vendors/*vendor*/boards/*board* 下的文件夹名称相同。

vendorName

所测试主板的供应商的名称。供应商名称应该与 *path/to/afr/source/code*/vendors/*vendor* 下的文件夹名称相同。

compilerName

编译器名称。

frToolchainPath

编译器工具链的全限定路径

cmakeToolchainPath

CMake 工具链的完全限定路径。此字段为可选项

freertosFileConfiguration

IDT 在运行测试之前修改的 FreeRTOS 文件的配置。

required

本节指定了您已移动其配置文件的必需测试，例如 PKCS11，TLS 等。

configName

正在配置的组件的名称。

filePath

freertos 存储库中配置文件的绝对路径。使用 {{testData.sourcePath}} 变量来定义路径。

optional

本节指定了您已移动其配置文件的可选测试 WiFi，例如 OTA 等。

configName

正在配置的组件的名称。

filePath

freertos 存储库中配置文件的绝对路径。使用 {{testData.sourcePath}} 变量来定义路径。

> **Note**
>
> 要执行 CMake 测试用例，必须提供主板名称、供应商名称和afrToolchainPath或compilerName。cmakeToolchainPath如果您有 CMake工具链的自定义路径，也可以提供。

适用于 FreeRTOS 变量的 IDT

生成代码和刷新设备的命令可能需要连接或有关设备的其他信息才能成功运行。 AWS IoT Device Tester 允许您在 Flash 中引用设备信息并使用生成命令JsonPath。通过使用简单的 JsonPath表达式，您可以获取device.json文件中指定的所需信息。

路径变量

适用于 FreeRTOS 的 IDT 定义了可在命令行和配置文件中使用的以下路径变量：

{{testData.sourcePath}}

扩展到到源代码路径。如果使用该变量，则必须在刷写和构建命令中使用该变量。

{{sdkPath}}

在构建和命令中使用时，扩展为您的 userData.sdkConfiguration.path 中的值。

{{device.connectivity.serialPort}}

扩展到串行端口。

{{device.identifiers[?(@.name == 'serialNo')].value[0]}}

扩展到您设备的序列号。

{{enableTests}}

整数值，指明构建适用于测试（值为 1）还是演示（值为 0）。

{{buildImageName}}

构建命令构建的映像的文件名。

{{otaCodeSignerPemFile}}

OTA 代码签署者的 PEM 文件。

{{config.idtRootPath}}

扩展到 AWS IoT Device Tester 根路径。当编译和刷写命令使用时，此变量取代 IDT 的绝对路径。

使用 IDT 用户界面运行 FreeRTOS 资格套件

从 IDT v4.3.0 开始，for AWS IoT Device Tester FreeRTOS (IDT-FreeRTOS) 包括一个基于 Web 的用户界面，使您可以与 IDT 命令行可执行文件和相关的配置文件进行交互。您可以使用 IDT-FreeRTOS UI 创建用于运行 IDT 测试的新配置，也可以修改现有配置。您也可以使用该 UI 调用 IDT 可执行文件并运行测试。

IDT-FreeRTOS UI 提供以下函数：

- 简化适用于 IDT-FreeRTOS 测试的配置文件设置。
- 简化使用 IDT-FreeRTOS 进行资格认证测试的过程。

有关使用命令行运行资格认证测试的信息，请参阅首次测试您的微控制器板。

本节介绍使用 IDT-FreeRTOS UI 的先决条件，并介绍如何开始在 UI 中运行资格认证测试。

主题

- 设置运行 FreeRTOS 资格套件的先决条件
- 开始使用 IDT-freeRTOS 用户界面

设置运行 FreeRTOS 资格套件的先决条件

本节介绍通过 AWS IoT Device Tester测试微控制器的先决条件。

主题

- 使用支持的 Web 浏览器
- 下载 FreeRTOS
- 下载适用于 FreeRTOS 的 IDT
- 创建和配置 AWS 账户

- AWS IoT Device Tester 托管策略

使用支持的 Web 浏览器

IDT-FreeRTOS UI 支持以下 Web 浏览器。

浏览器	版本
Google Chrome	最新的三个主要版本
Mozilla Firefox	最新的三个主要版本
Microsoft Edge	最新的三个主要版本
Apple Safari for macOS	最新的三个主要版本

为了获得更好的体验，我们建议您使用 Google Chrome 或 Mozilla Firefox。

> **Note**
>
> IDT-FreeRTOS UI 不支持 Microsoft Internet Explorer。

下载 FreeRTOS

你可以使用以下命令GitHub从中下载 FreeRTOS 的发行版：

```
git clone --branch <FREERTOS_RELEASE_VERSION> --recurse-submodules https://github.com/
aws/amazon-freertos.git
cd amazon-freertos
git submodule update --checkout --init --recursive
```

其中 <FREERTOS_RELEASE_VERSION> 是与 支持的版本 AWS IoT Device Tester 中列出的 IDT 版本对应的 FreeRTOS 版本（例如 202007.00）。这样可以确保您拥有完整的源代码，包括子模块，并且使用适用于您的 FreeRTOS 版本的 IDT 的正确版本，反之亦然。

Windows 的路径长度限制为 260 个字符。FreeRTOS 的路径结构是多级深层结构，因此，如果您使用 Windows，请确保文件路径不超过 260 个字符的限制。例如，将 FreeRTOS 克隆到 `C:\FreeRTOS` 而不是 `C:\Users\username\programs\projects\myproj\FreeRTOS\`。

LTS 资格认证的注意事项（使用 LTS 库的 FreeRTOS 的资格认证）

- 要在 AWS 合作伙伴设备目录中将您的微控制器指定为支持基于长期支持 (LTS) 的 FreeRTOS 版本，您必须提供清单文件。有关更多信息，请参阅《FreeRTOS 资格认证指南》中的 FreeRTOS 资格认证检查清单。
- 为了验证您的微控制器是否支持基于 LTS 的 FreeRTOS 版本并使其有资格提交到 AWS 合作伙伴设备目录，您必须使用（AWS IoT Device Tester IDT) 和 FreeRTOS 资格认证 (FRQ) 测试套件版本 v1.4.x。
- 对基于 LTS 的 FreeRTOS 版本的支持仅限于 202012.xx 版本的 FreeRTOS。

下载适用于 FreeRTOS 的 IDT

为了执行资格认证测试，FreeRTOS 的每个版本都有对应的适用于 FreeRTOS 的 IDT 版本。从 支持的版本 AWS IoT Device Tester 下载适用于 FreeRTOS 的 ITD 的相应版本。

将适用于 FreeRTOS 的 IDT 提取到文件系统上您具有读写权限的位置。由于 Microsoft Windows 对路径长度具有字符数限制，因此将适用于 FreeRTOS 的 IDT 提取到根目录，如 `C:\` 或 `D:\`。

> **Note**
>
> 我们建议您将 IDT 程序包提取到本地驱动器。允许多个用户从共享位置（例如 NFS 目录或 Windows 网络共享文件夹）运行 IDT 可能会导致系统无响应或数据损坏。

创建和配置 AWS 账户

注册获取 AWS 账户

如果您没有 AWS 账户，请完成以下步骤来创建一个。

报名参加 AWS 账户

1. 打开https://portal.aws.amazon.com/billing/注册。
2. 按照屏幕上的说明操作。

   注册过程的一部分涉及接听电话或短信，并在电话键盘上输入验证码。

   当您注册时 AWS 账户，就会创建AWS 账户根用户一个。根用户有权访问该账户中的所有 AWS 服务 和资源。作为最佳安全实践，请为用户分配管理访问权限，并且只使用根用户来执行需要根用户访问权限的任务。

AWS 注册过程完成后会向您发送一封确认电子邮件。您可以随时前往 https://aws.amazon.com/并选择“我的账户”，查看您当前的账户活动并管理您的账户。

创建具有管理访问权限的用户

注册后，请保护您的安全 AWS 账户 AWS 账户根用户 AWS IAM Identity Center，启用并创建管理用户，这样您就不会使用 root 用户执行日常任务。

保护你的 AWS 账户根用户

1. 选择 Root 用户并输入您的 AWS 账户 电子邮件地址，以账户所有者的身份登录。AWS Management Console在下一页上，输入您的密码。

   要获取使用根用户登录方面的帮助，请参阅《AWS 登录 用户指南》中的 Signing in as the root user。
2. 为您的根用户启用多重身份验证（MFA）。

   有关说明，请参阅 I A M 用户指南中的为 AWS 账户 根用户启用虚拟 MFA 设备（控制台）。

创建具有管理访问权限的用户

1. 启用 IAM Identity Center。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的 Enabling AWS IAM Identity Center。
2. 在 IAM Identity Center 中，为用户授予管理访问权限。

   有关使用 IAM Identity Center 目录 作为身份源的教程，请参阅《用户指南》 IAM Identity Center 目录中的使用默认设置配置AWS IAM Identity Center 用户访问权限。

以具有管理访问权限的用户身份登录

- 要使用您的 IAM Identity Center 用户身份登录，请使用您在创建 IAM Identity Center 用户时发送到您的电子邮件地址的登录网址。

  有关使用 IAM Identity Center 用户登录的帮助，请参阅AWS 登录 用户指南中的登录 AWS 访问门户。

将访问权限分配给其他用户

1. 在 IAM Identity Center 中，创建一个权限集，该权限集遵循应用最低权限的最佳做法。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的 Create a permission set。
2. 将用户分配到一个组，然后为该组分配单点登录访问权限。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的 Add groups。

AWS IoT Device Tester 托管策略

为了支持设备测试程序运行和收集指标，`AWSIoTDeviceTesterForFreeRTOSFullAccess` 托管策略包含了以下权限：

- `iot-device-tester:SupportedVersion`

  授予权限以获取 IDT 支持的 FreeRTOS 版本和测试套件版本的列表，以便它们可以从 AWS CLI获取。
- `iot-device-tester:LatestIdt`

  授予获取可供下载的最新 AWS IoT Device Tester 版本的权限。
- `iot-device-tester:CheckVersion`

  授予权限以检查产品、测试套件和 AWS IoT Device Tester 版本组合是否兼容。
- `iot-device-tester:DownloadTestSuite`

  授 AWS IoT Device Tester 予下载测试套件的权限。
- `iot-device-tester:SendMetrics`

  授予发布 AWS IoT Device Tester 使用率指标数据的权限。

开始使用 IDT-freeRTOS 用户界面

本节介绍如何使用 IDT-FreeRTOS UI 来创建或修改配置，然后介绍了如何运行测试。

主题

- 配置 AWS 凭证
- 打开 IDT-FreeRTOS UI

- 创建新的配置
- 修改现有配置
- 运行资格认证测试

配置 AWS 凭证

您必须为在中创建的 AWS 用户配置凭据创建和配置 AWS 账户。您可以采用以下两种方法之一来指定凭证：

- 在凭证文件中
- 作为环境变量

使用 AWS 凭证文件配置凭证

IDT 使用与 AWS CLI相同的凭证文件。有关更多信息，请参阅配置和凭证文件。

凭证文件的位置因您使用的操作系统而异：

- macOS、Linux：`~/.aws/credentials`
- Windows：`C:\Users\UserName\.aws\credentials`

按以下格式将您的 AWS 凭证添加到`credentials`文件中：

```
[default]
aws_access_key_id = <your_access_key_id>
aws_secret_access_key = <your_secret_access_key>
```

> **Note**
>
> 如果您不使用`default` AWS 配置文件，请务必在 IDT-FreeRTOS 用户界面中指定配置文件名称。有关配置文件的更多信息，请参阅配置和凭据文件设置。

使用环境变量配置 AWS 凭证

环境变量是由操作系统维护且由系统命令使用的变量。如果您关闭 SSH 会话，则不会保存。IDT-FreeRTOS UI 使用 `AWS_ACCESS_KEY_ID` 和 `AWS_SECRET_ACCESS_KEY` 环境变量来存储您的 AWS 凭证。

要在 Linux、macOS 或 Unix 上设置这些变量，请使用 export：

```
export AWS_ACCESS_KEY_ID=<your_access_key_id>
export AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

要在 Windows 上设置这些变量，请使用 set：

```
set AWS_ACCESS_KEY_ID=<your_access_key_id>
set AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

打开 IDT-FreeRTOS UI

打开 IDT-FreeRTOS UI

1. 下载支持的 IDT-FreeRTOS 版本，并将下载的存档提取到文件系统上您有读写权限的位置。
2. 运行以下命令以导航到 IDT-FreeRTOS 安装目录：

   ```
   cd devicetester-extract-location/bin
   ```

3. 运行以下命令以打开 IDT-FreeRTOS UI：

   Linux

   ```
   .devicetestergui_linux_x86-64.exe
   ```

   Windows

   ```
   ./devicetestergui_win_x64-64
   ```

   macOS

   ```
   ./devicetestergui_mac_x86-64
   ```

   > **Note**
   >
   > 在 Mac 上，要允许您的系统运行此 UI，请转到系统首选项 -> 安全和隐私。当您运行测试时，可能需要再执行三次。

IDT-FreeRTOS UI 将在您的默认浏览器中打开。有关支持的浏览器的信息，请参阅使用支持的 Web 浏览器。

创建新的配置

如果您是首次使用的用户，则必须创建一个新配置来设置 IDT-FreeRTOS 运行测试所需的 JSON 配置文件。然后，您可以运行测试或修改已创建的配置。

有关 `config.json`、`device.json` 和 `userdata.json` 文件的示例，请参阅首次测试您的微控制器板。有关仅用于运行低功耗蓝牙 (BLE) 测试的 `resource.json` 文件的示例，请参阅运行蓝牙低功耗测试。

创建新的配置

1. 在 IDT-FreeRTOS UI 中，打开导航菜单，然后选择创建新配置。

   > **Important**
   >
   > 在打开 UI 之前，您必须配置您的 AWS 凭据。如果您尚未配置凭证，请关闭 IDT-FreeRTOS UI 浏览器窗口，按照配置 AWS 凭证中的步骤操作，然后重新打开 IDT-FreeRTOS UI。

2. 按照配置向导输入用于运行资格认证测试的 IDT 配置设置。该向导在 *`devicetester-extract-location`*`/config` 目录中的 JSON 配置文件中配置以下设置。

   - AWS 设置 — IDT-FreeRTOS 在测试运行期间用于创建 AWS 资源 AWS 账户 的信息。这些设置在 `config.json` 文件中进行配置。
   - FreeRTOS 存储库 – FreeRTOS 存储库和移植代码的绝对路径，以及您要执行的资格认证类型。这些设置在 `userdata.json` 文件中进行配置。

     在运行资格认证测试之前，必须为设备移植 FreeRTOS。有关更多信息，请参阅《FreeRTOS 移植指南》。
   - 构建和刷写 – 适用于您的硬件的构建和刷写命令，允许 IDT 在您的主板上自动构建和刷写测试。这些设置在 `userdata.json` 文件中进行配置。
   - 设备 – 要测试的设备的设备池设置。这些设置在 `id` 和 `sku` 字段中配置，设备池的 `devices` 数据块位于 `device.json` 文件中。

- 网络 – 用于测试您的设备是否支持网络通信的设置。这些设置在 `device.json` 文件的 `features` 数据块中，以及 `userdata.json` 文件的 `clientWifiConfig` 和 `testWifiConfig` 数据块中配置。
- Echo 服务器 – 用于安全套接字测试的 Echo 服务器配置设置。这些设置在 `userdata.json` 文件中进行配置。

  有关 Echo 服务器配置的更多信息，请参阅 https://docs.aws.amazon.com/freertos/latest/portingguide/afr-echo-server.html。
- CMake—（可选）运行 CMake 生成功能测试的设置。只有当您使用 CMake 作为构建系统时，才需要此配置。这些设置在 `userdata.json` 文件中进行配置。
- BLE – 运行低功耗蓝牙功能测试的设置。这些设置在 `device.json` 文件和 `resource.json` 文件的 `features` 数据块中配置。
- OTA – 运行 OTA 功能测试的设置。这些设置在 `device.json` 文件和 `userdata.json` 文件的 `features` 数据块中配置。

3. 在审核页面上，验证您的配置信息。

审核配置完成后，要运行资格认证测试，请选择运行测试。

修改现有配置

如果您已经为 IDT 设置了配置文件，则可以使用 IDT-FreeRTOS UI 修改现有配置。确保您的现有配置文件在 *devicetester-extract-location*`/config` 目录中可用。

修改新的配置

1. 在 IDT-FreeRTOS UI 中，打开导航菜单，然后选择编辑现有配置。

   配置控制面板会显示有关现有配置设置的信息。如果配置不正确或不可用，则配置的状态为 `Error validating configuration`。
2. 要修改现有配置设置，请完成以下步骤：

   a. 选择配置设置的名称以打开其设置页面。

   b. 修改设置，然后选择保存，以便重新生成相应的配置文件。

修改配置完成后，请确认所有配置设置均可通过验证。如果每个配置设置的状态为 `Valid`，则可以使用此配置运行资格认证测试。

运行资格认证测试

为 IDT-FreeRTOS 创建配置后，即可开始运行资格认证测试。

运行资格认证测试

1. 验证配置。
2. 在导航菜单中，选择运行测试。
3. 要开始运行测试，请选择开始测试。

IDT-FreeRTOS 运行资格认证测试，并在 Test Runner 控制台中显示测试运行摘要和所有错误。测试运行完成后，您可以从以下位置查看测试结果和日志：

- 测试结果位于 *devicetester-extract-location*/results/*execution-id* 目录中。
- 测试日志位于 *devicetester-extract-location*/results/*execution-id*/logs 目录中。

有关测试结果和日志的更多信息，请参阅查看 IDT for FreeRTOS 结果和查看 FreeRTOS 日志的 IDT。

运行蓝牙低功耗测试

本节介绍如何使用 AWS IoT Device Tester FreeRTOS设置和运行低功耗蓝牙测试。

核心资格认证不需要蓝牙测试。如果您不想使用 FreeRTOS 蓝牙支持测试您的设备，则可跳过此设置，确保将 device.json 中的 BLE 功能设置为 No。

先决条件

- 按照首次测试您的微控制器板中的说明进行操作。
- 一个 Raspberry Pi 4B 或 3B+。（运行 Raspberry Pi BLE 配套应用程序时必需）
- 一个适用于 Raspberry Pi 软件的 microSD 卡和 SD 卡适配器。

Raspberry Pi 设置

要测试受测设备 (DUT) 的 BLE 功能，您必须拥有 Raspberry Pi Model 4B 或 3B+。

设置 Raspberry Pi 以运行 BLE 测试

1. 下载包含执行测试所需的软件的自定义 Yocto 映像之一。

   - Raspberry Pi 4B 的映像
   - Raspberry Pi 3B+ 的映像

   > **Note**
   >
   > Yocto 镜像只能用于 AWS IoT Device Tester FreeRTOS 进行测试，不能用于任何其他目的。

2. 将 yocto 映像刷写到 Raspberry Pi 的 SD 卡。

   - 通过使用 SD 卡写入工具（例如 Etcher），将下载的 *image-name*`.rpi-sd.img` 文件刷写到 SD 卡中。由于操作系统映像很大，因此，该步骤可能需要一些时间。然后，从您的计算机中弹出 SD 卡，然后将 microSD 卡插入您的 Raspberry Pi 上。

3. 配置您的 Raspberry Pi。

   a. 对于第一次启动，我们建议您将 Raspberry Pi 连接到显示器、键盘和鼠标。

   b. 将您的 Raspberry Pi 连接到微型 USB 电源。

   c. 使用默认凭证登录。对于用户 ID，请输入 **`root`**。对于密码，请输入 **`idtafr`**。

   d. 通过使用以太网或 Wi-Fi 连接，将 Raspberry Pi 连接到您的网络。

      i. 要通过 Wi-Fi 连接您的 Raspberry Pi，请在 Raspberry Pi 上打开 `/etc/wpa_supplicant.conf`，并将您的 Wi-Fi 凭证添加到 `Network` 配置。

```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=0
update_config=1

network={
        scan_ssid=1
        ssid="your-wifi-ssid"
        psk="your-wifi-password"
        }
```

      ii. 运行 `ifup wlan0` 以启动 Wi-Fi 连接。连接到 Wi-Fi 网络可能需要一分钟。

e. 对于以太网连接，请运行 `ifconfig eth0`。对于 Wi-Fi 连接，请运行 `ifconfig wlan0`。记下 IP 地址，它在命令输出中显示为 `inet addr`。您在本过程的后面部分需要该 IP 地址。

f. （可选）测试使用 yocto 映像的默认凭证通过 SSH 在 Raspberry Pi 上执行命令。为了提高安全性，我们建议您为 SSH 设置公有密钥身份验证并禁用基于密码的 SSH。

i. 使用 OpenSSL `ssh-keygen` 命令创建 SSH 密钥。如果你的主机上已经有一个 SSK 密钥对，那么最好创建一个新的 SSK 密钥对，让 FreeRTOS 登录你 AWS IoT Device Tester 的 Raspberry Pi。

> **Note**
>
> Windows 未预装 SSH 客户端。有关如何在 Windows 上安装 SSH 客户端的信息，请参阅下载 SSH 软件。

ii. `ssh-keygen` 命令会提示您输入要存储密钥对的名称和路径。默认情况下，密钥对文件的名称为 `id_rsa`（私有密钥）和 `id_rsa.pub`（公有密钥）。在 macOS 和 Linux 上，这些文件的默认位置为 `~/.ssh/`。在 Windows 上，默认位置为 `C:\Users\`*`user-name`*。

iii. 当系统提示您输入密钥短语时，请按 Enter 以继续。

iv. 要将你的 SSH 密钥添加到 Raspberry Pi 上，让 FreeRTOS 可以登录设备，请使用`ssh-copy-id`主机上的命令。 AWS IoT Device Tester 此命令会将您的公有密钥添加到 Raspberry Pi 上的 `~/.ssh/authorized_keys` 文件中。

```
ssh-copy-id root@raspberry-pi-ip-address
```

v. 当系统提示您输入密码时，请输入 **`idtafr`**。这是 yocto 映像的默认密码。

> **Note**
>
> `ssh-copy-id` 命令假定公有密钥的名称为 `id_rsa.pub`。在 macOS 和 Linux 上，默认位置为 `~/.ssh/`。在 Windows 上，默认位置为 `C:\Users\`*`user-name`*`\.ssh`。如果公有密钥采用其他名称或存储在其他位置，则必须使用 `-i` 选项与 `ssh-copy-id` 指定 SSH 公有密钥的完全限定路径（例如，`ssh-copy-id -i ~/my/path/myKey.pub`）。有关创建 SSH 密钥和复制公有密钥的更多信息，请参阅 SSH-COPY-ID。

vi. 要测试公有密钥身份验证是否有效，请运行 `ssh -i` */my/path/myKey* `root@`*`raspberry-pi-device-ip`*。

如果系统未提示您输入密码，则公有密钥身份验证有效。

vii. 验证您是否能使用公有密钥登录 Raspberry Pi，然后禁用基于密码的 SSH。

A. 在 Raspberry Pi 中，编辑 `/etc/ssh/sshd_config` 文件。

B. 将 `PasswordAuthentication` 属性设置为 `no`。

C. 保存并关闭 `sshd_config`文件。

D. 通过运行 `/etc/init.d/sshd reload` 重新加载 SSH 服务器。

g. 创建 `resource.json` 文件。

i. 在提取 AWS IoT 设备测试器的目录中，创建一个名为的文件`resource.json`。

ii. 将以下有关树莓派的信息添加到文件中，*`rasp-pi-ip-address`*替换为 Raspberry Pi 的 IP 地址。

```
[
    {
        "id": "ble-test-raspberry-pi",
        "features": [
            {"name":"ble", "version":"4.2"}
        ],
        "devices": [
            {
                "id": "ble-test-raspberry-pi-1",
                "connectivity": {
                    "protocol": "ssh",
                    "ip": "rasp-pi-ip-address"
                }
            }
        ]
    }
]
```

iii. 如果您未选择对 SSH 使用公有密钥身份验证，请将以下内容添加到 `resource.json` 文件的 `connectivity` 部分。

```
"connectivity": {
    "protocol": "ssh",
    "ip": "rasp-pi-ip-address",
    "auth": {
        "method": "password",
```

```
        "credentials": {
            "user": "root",
            "password": "idtafr"
        }
    }
}
```

iv. （可选）如果您对 SSH 使用公有密钥身份验证，请将以下内容添加到 resource.json 文件的 connectivity 部分。

```
"connectivity": {
    "protocol": "ssh",
    "ip": "rasp-pi-ip-address",
    "auth": {
        "method": "pki",
        "credentials": {
            "user": "root",
            "privKeyPath": "location-of-private-key"
        }
    }
}
```

FreeRTOS 设备设置

在 device.json 文件中，将 BLE 功能设置为 Yes。如果您在蓝牙测试可用前开始使用 device.json 文件，您需要将 BLE 功能添加到 features 数组：

```
{
    ...
    "features": [
        {
            "name": "BLE",
            "value": "Yes"
        },
    ...
}
```

运行 BLE 测试

在 device.json 中启用 BLE 功能后，在不指定群组 ID 的情况下运行 devicetester_*[linux | mac | win_x86-64]* run-suite 时，BLE 测试将运行。

如果要单独运行 BLE 测试，可以为 BLE 指定组 ID：`devicetester_[linux | mac | win_x86-64] run-suite --userdata path-to-userdata/userdata.json --group-id FullBLE`。

要获得最可靠的性能，请将 Raspberry Pi 放置在受测设备 (DUT) 附近 。

BLE 测试故障排除

确保您已执行首次测试您的微控制器板中的步骤。如果 BLE 以外的测试失败，则问题很可能与蓝牙配置无关。

运行 FreeRTOS 资格套件

你可以使用 for FreeRTOS 的可执行文件与 FreeRTOS 的 IDT 进行交互。 AWS IoT Device Tester 以下命令行示例向您显示如何针对某个设备池（一组相同的设备）运行资格测试。

IDT v3.0.0 and later

```
devicetester_[linux | mac | win] run-suite  \
    --suite-id suite-id  \
    --group-id group-id  \
    --pool-id your-device-pool \
    --test-id test-id  \
    --upgrade-test-suite y|n  \
    --update-idt y|n  \
    --update-managed-policy y|n  \
    --userdata userdata.json
```

对某个设备池运行一组测试。`userdata.json` 文件必须位于 *devicetester_extract_location*`/devicetester_afreertos_[win|mac|linux]/configs/` 目录中。

> **Note**
>
> 如果您在 Windows 上运行适用于 FreeRTOS 的 IDT，请使用正斜杠 (/) 指定 `userdata.json` 文件的路径。

使用以下命令运行特定测试组：

```
devicetester_[linux | mac | win] run-suite  \
```

```
    --suite-id FRQ_1.0.1  \
    --group-id group-id  \
    --pool-id pool-id  \
    --userdata userdata.json
```

如果您在单个设备池上（即仅在 device.json 文件中定义了一个设备池）运行单个测试套件，则 suite-id 和 pool-id 参数为可选。

使用以下命令运行测试组中的特定测试用例：

```
devicetester_[linux | mac | win_x86-64] run-suite  \
    --group-id group-id  \
    --test-id test-id
```

您可以使用 list-test-cases 命令列出测试组中的测试用例。

适用于 FreeRTOS 的 IDT 命令行选项

group-id

（可选）要以逗号分隔的列表形式运行的测试组。如果未指定，IDT 将运行测试套件中的所有测试组。

pool-id

（可选）要测试的设备池。如果您在 device.json 中定义多个设备池，则需要执行此操作。如果您只有一个设备池，则可以省略此选项。

suite-id

（可选）要运行的测试套件版本。如果未指定，IDT 将在系统上使用测试目录中的最新版本。

> **Note**
>
> 从 IDT v3.0.0 开始，IDT 在线检查较新的测试套件。有关更多信息，请参阅 测试套件版本。

test-id

（可选）要以逗号分隔的列表形式运行的测试。如果指定，则 group-id 必须指定单个组。

Example

```
devicetester_[linux | mac | win_x86-64] run-suite --group-id mqtt --test-id
 mqtt_test
```

update-idt

（可选）如果未设置此参数且有较新的 IDT 版本可用，则系统将提示您更新 IDT。如果将此参数设置为 Y，则 IDT 将在检测到有更新版本可用时停止执行测试。如果将此参数设置为 N，IDT 将继续执行测试。

update-managed-policy

（可选）如果未使用此参数且 IDT 检测到您的托管策略未使用 up-to-date，则系统将提示您更新托管策略。如果将此参数设置为Y，则 IDT 将在检测到您的托管策略未 up-to-date被执行时停止测试执行。如果将此参数设置为 N，IDT 将继续执行测试。

upgrade-test-suite

（可选）如果未使用，并且较新的测试套件版本可用，系统将提示您下载该版本。要隐藏提示，请指定 y 以始终下载最新的测试套件，或者指定 n 以使用系统上指定的测试套件或最新版本。

Example

示例

要始终下载并使用最新的测试套件，请使用以下命令。

```
devicetester_[linux | mac | win_x86-64] run-suite --userdata userdata file --
group-id group ID --upgrade-test-suite y
```

要使用系统上的最新测试套件，请使用以下命令。

```
devicetester_[linux | mac | win_x86-64] run-suite --userdata userdata file --
group-id group ID --upgrade-test-suite n
```

h

使用帮助选项了解有关 `run-suite` 选项的更多信息。

Example

示例

```
devicetester_[linux | mac | win_x86-64] run-suite -h
```

IDT v1.7.0 and earlier

```
devicetester_[linux | mac | win] run-suite  \
    --suite-id suite-id  \
    --pool-id your-device-pool  \
    --userdata userdata.json
```

userdata.json 文件应位于 *devicetester_extract_location*/devicetester_afreertos_*[win|mac|linux]*/configs/ 目录中。

> **Note**
>
> 如果您在 Windows 上运行适用于 FreeRTOS 的 IDT，请使用正斜杠 (/) 指定 userdata.json 文件的路径。

使用以下命令运行特定测试组。

```
devicetester_[linux | mac | win] run-suite  \
    --suite-id FRQ_1 --group-id group-id  \
    --pool-id pool-id  \
    --userdata userdata.json
```

如果您在单个设备池上（即仅在 device.json 文件中定义了一个设备池）运行单个测试套件，则 suite-id 和 pool-id 为可选。

适用于 FreeRTOS 的 IDT 命令行选项

group-id

（可选）指定测试组。

pool-id

指定要测试的设备池。如果您只有一个设备池，则可以省略此选项。

suite-id

（可选）指定要运行的测试套件。

适用于 FreeRTOS 命令的 IDT

适用于 FreeRTOS 的 IDT 命令支持以下操作：

IDT v3.0.0 and later

help

列出有关指定命令的信息。

list-groups

列出给定套件中的组。

list-suites

列出可用套件。

list-supported-products

列出支持的产品和测试套件版本。

list-supported-versions

列出当前 IDT 版本支持的 FreeRTOS 和测试套件版本。

list-test-cases

列出指定组中的测试用例。

run-suite

对某个设备池运行一组测试。

使用 `--suite-id` 选项可以指定测试套件版本，省略它可以使用系统上的最新版本。

使用 `--test-id` 运行单个测试用例。

Example

```
devicetester_[linux | mac | win_x86-64] run-suite --group-id mqtt --test-id
 mqtt_test
```

有关选项的完整列表，请参阅 运行 FreeRTOS 资格套件。

> **Note**
>
> 从 IDT v3.0.0 开始，IDT 在线检查较新的测试套件。有关更多信息，请参阅 测试套件版本。

IDT v1.7.0 and earlier

`help`

列出有关指定命令的信息。

`list-groups`

列出给定套件中的组。

`list-suites`

列出可用套件。

`run-suite`

对某个设备池运行一组测试。

用于重新确定资格的测试

随着适用于 FreeRTOS 的 IDT 资格认证测试的新版本发布，或者在您更新特定于主板的程序包或设备驱动程序时，您可以使用适用于 FreeRTOS 的 IDT 来测试微控制器主板。对于后续资格，请确保您具有 FreeRTOS 和适用于 FreeRTOS 的 IDT 的最新版本并再次运行资格认证测试。

查看 IDT for FreeRTOS 结果

在运行时，IDT 会将错误写入控制台、日志文件和测试报告中。IDT 在完成资格测试套件后，会将测试运行摘要写入控制台并生成两个测试报告。可在 *devicetester-extract-location*/results/*execution-id*/ 中找到这些报告。两个报告都捕获资格测试套件执行的结果。

awsiotdevicetester_report.xml这是您提交给的资格测试报告， AWS 用于在 AWS 合作伙伴设备目录中列出您的设备。该报告包含以下元素：

- 适用于 FreeRTOS 版本的 IDT。

- 所测试的 FreeRTOS 版本。
- 设备所支持的 FreeRTOS 功能基于所通过的测试。
- 在 `device.json` 文件中指定的 SKU 和设备名称。
- 在 `device.json` 文件中指定的设备的功能。
- 测试用例结果的摘要汇总。
- 按基于设备功能（例如 FullmQTT 等 FullWiFi）测试的库对测试用例结果进行细分。
- FreeRTOS 的这种资格认证是否适用于使用 LTS 库的 202012.00 版本。

`FRQ_Report.xml`是标准 JUnit XML 格式的报告。你可以将其集成到 Jenkins、Bamb oo 等 CI/CD 平台中。该报告包含以下元素：

- 测试用例结果的摘要汇总。
- 基于设备功能，按照所测试的库细分的测试用例结果。

解读 FreeRTOS 结果的 IDT

`awsiotdevicetester_report.xml` 或 `FRQ_Report.xml` 中的报告部分列出了所执行的测试的结果。

第一个 XML 标签 `<testsuites>` 包含测试执行的整体摘要。例如：

```
<testsuites name="FRQ results" time="5633" tests="184" failures="0"
errors="0" disabled="0">
```

`<testsuites>` 标签中使用的属性

`name`

测试套件的名称。

`time`

运行资格套件所用的时间（以秒为单位）。

`tests`

执行的测试用例数量。

failures

已运行但未通过的测试用例数。

errors

适用于 FreeRTOS 的 IDT 无法执行的测试用例数。

disabled

此属性未使用，可以忽略。

如果没有测试用例失败或错误，则您的设备符合运行 FreeRTOS 的技术要求，并且可以与服务互操作。 AWS IoT 如果您选择在 AWS 合作伙伴设备目录中列出您的设备，则可以使用此报告作为资格证据。

如果出现测试用例失败或错误，可以通过检查 <testsuites> XML 标签来确定失败的测试用例。<testsuites> 标签内部的 <testsuite> XML 标签显示测试组的测试用例结果摘要。

```
<testsuite name="FullMQTT" package="" tests="16" failures="0" time="76"
disabled="0" errors="0" skipped="0">
```

其格式类似于 <testsuites> 标签，但具有名为 skipped 的属性，此属性未使用，可以忽略。在每个 <testsuite> XML 标签内部，对于一个测试组，所执行的每个测试用例都有 <testcase> 标签。例如：

```
<testcase classname="mcu.Full_MQTT" name="AFQP_MQTT_Connect_HappyCase"
attempts="1"></testcase>
```

<awsproduct> 标签中使用的属性

name

所测试的产品的名称。

version

所测试的产品的版本。

sdk

如果使用开发工具包运行 IDT，则此数据块包含您的开发工具包的名称和版本。如果您没有使用开发工具包运行 IDT，则此数据块包含：

```
<sdk>
    <name>N/A</vame>
    <version>N/A</version>
</sdk>
```

features

验证的功能。标记为 `required` 的功能需要提交您的主板信息以供资格审核。以下代码段演示了它在 `awsiotdevicetester_report.xml` 文件中的显示方式。

```
<feature name="core-freertos" value="not-supported" type="required"></feature>
```

标记为 `optional` 的功能不是资格认证所必需的。以下代码段显示了可选功能。

```
<feature name="ota-dataplane-mqtt" value="not-supported" type="optional"></feature>
<feature name="ota-dataplane-http" value="not-supported" type="optional"></feature>
```

如果没有针对所需功能的测试失败或错误，则设备满足运行 FreeRTOS 的技术要求并可以与 AWS IoT 服务互操作。如果您想要在 AWS 合作伙伴设备目录中列出您的设备，则可以使用此报告作为资格证明。

如果出现测试失败或错误，则可以通过检查 `<testsuites>` XML 标签来确定失败的测试。`<testsuites>` 标签内的 `<testsuite>` XML 标签显示了测试组的测试结果摘要。例如：

```
<testsuite name="FreeRTOSVersion" package="" tests="1" failures="1" time="2"
 disabled="0" errors="0" skipped="0">
```

其格式与 `<testsuites>` 标签类似，但有一个未使用并可忽略的 `skipped` 属性。在每个 `<testsuite>` XML 标签内部，对于一个测试组，所执行的每个测试都有 `<testcase>` 标签。例如：

```
<testcase classname="FreeRTOSVersion" name="FreeRTOSVersion"></testcase>
```

lts

如果您符合使用 LTS 库的 FreeRTOS 版本的资格，则为 true，否则为 false。

<testcase> 标签中使用的属性

name

测试用例的名称。

attempts

适用于 FreeRTOS 的 IDT 执行测试用例的次数。

当测试失败或出现错误时，将会在 <testcase> 标签中添加包含用于故障排除的信息的 <failure> 或 <error> 标签。例如：

```
<testcase classname="mcu.Full_MQTT" name="AFQP_MQTT_Connect_HappyCase">
    <failure type="Failure">Reason for the test case failure</failure>
    <error>Reason for the test case execution error</error>
</testcase>
```

有关更多信息，请参阅 排查 错误。

查看 FreeRTOS 日志的 IDT

您可以在 *devicetester-extract-location*/results/*execution-id*/logs 中找到适用于 FreeRTOS 的 IDT 从测试执行生成的日志。它会生成两组日志：

test_manager.log

包含从适用于 FreeRTOS 的 IDT 生成的日志（例如，与配置和报告生成相关的日志）。

***test_group_id*__*test_case_id*.log**（例如，**FullMQTT__Full_MQTT.log**）

测试用例的日志文件，包括来自所测设备的输出。日志文件根据运行的测试组和测试用例命名。

开发和运行自己的 IDT 测试套件

从 IDT v4.0.0 开始，适用于 FreeRTOS 的 IDT 将标准化配置设置和结果格式与测试套件环境相结合，使您能够为设备和设备软件开发自定义测试套件。您可以添加自定义测试来用于自己的内部验证，也可以将其提供给客户进行设备验证。

使用 IDT 开发和运行自定义测试套件，如下所示：

开发自定义测试套件

- 使用自定义测试逻辑为要测试的设备创建测试套件。
- 向 IDT 提供您的自定义测试套件以供测试运行器使用。包括有关测试套件的特定设置配置的信息。

运行自定义测试套件

- 设置要测试的设备。
- 根据要使用的测试套件的要求实现设置配置。
- 使用 IDT 运行您的自定义测试套件。
- 查看 IDT 运行的测试的测试结果和执行日志。

下载最新版本的 Free AWS IoT RTOS 设备测试器

下载 IDT 的最新版本并将软件提取到文件系统中您具有读取和写入权限的位置。

> **Note**
>
> IDT 不支持由多个用户从共享位置（如 NFS 目录或 Windows 网络共享文件夹）运行。建议您将 IDT 包解压缩到本地驱动器，并在本地工作站上运行 IDT 二进制文件。
> Windows 的路径长度限制为 260 个字符。如果您使用的是 Windows，请将 IDT 提取到根目录（如 `C:\` 或 `D:\`）以使路径长度不超过 260 个字符的限制。

测试套件工作流程

测试套件由三种类型的文件组成：

- 提供有关如何执行测试套件的信息的配置文件。
- 测试 IDT 用来运行测试用例的可执行文件。
- 运行测试所需的其他文件。

完成以下基本步骤来创建自定义 IDT 测试：

1. 为测试套件创建配置文件。
2. 创建包含测试套件测试逻辑的测试用例可执行文件。
3. 验证并记录测试运行器运行测试套件所需的配置信息。

4. 验证 IDT 能否按预期运行您的测试套件并生成测试结果。

要快速构建示例自定义套件并运行它，请按照 教程：构建和运行示例 IDT 测试套件 中的说明进行操作。

要开始使用 Python 创建自定义测试套件，请参阅教程：开发一个简单的 IDT 测试套件。

教程：构建和运行示例 IDT 测试套件

AWS IoT Device Tester 下载内容包括示例测试套件的源代码。您可以完成本教程来构建和运行示例测试套件，以了解如何使用 AWS IoT Device Tester FreeRTOS 来运行自定义测试套件。尽管本教程使用 SSH，但学习如何在 FreeRTOS 设备上使用还是很有 AWS IoT Device Tester 用的。

在本教程中，您将完成以下步骤：

1. 构建示例测试套件
2. 使用 IDT 运行示例测试套件

主题

- 为示例测试套件设置先决条件
- 配置 IDT 的设备信息
- 构建示例测试套件
- 使用 IDT 运行示例测试套件
- 排查错误

为示例测试套件设置先决条件

要完成本教程，您需要：

- 主机要求
  - 最新版本的 AWS IoT Device Tester
  - Python 3.7 或更高版本

    要检查您计算机安装的 Python 版本，请运行以下命令：

    ```
    python3 --version
    ```

在 Windows 上，如果运行此命令时返回错误，则可改用 `python --version`。如果返回的版本号为 3.7 或更高版本，则可通过在 Powershell 终端中运行以下命令将 `python3` 设置为 `python` 命令的别名。

```
Set-Alias -Name "python3" -Value "python"
```

如果没有返回版本信息，或者版本号小于 3.7，则按照下载 Python 中的说明安装 Python 3.7+。有关更多信息，请参阅 Python 文档。

- urllib3

  要验证 `urllib3` 是否已正确安装，请运行以下命令：

  ```
  python3 -c 'import urllib3'
  ```

  如果未安装 `urllib3`，请运行以下命令进行安装：

  ```
  python3 -m pip install urllib3
  ```

- 设备要求
  - 一种运行 Linux 操作系统的设备，其网络连接到与您主机相同的网络。

    我们建议您使用搭载 Raspberry Pi 操作系统的 Raspberry Pi。请确保您设置 Raspberry Pi 上的 SSH 才能远程连接到它。

配置 IDT 的设备信息

配置您的设备信息，以便 IDT 运行测试。您必须使用以下信息，更新位于 *<device-tester-extract-location>*`/configs` 文件夹中的 `device.json` 模板。

```
[
  {
    "id": "pool",
    "sku": "N/A",
    "devices": [
      {
        "id": "<device-id>",
        "connectivity": {
          "protocol": "ssh",
```

```
          "ip": "<ip-address>",
          "port": "<port>",
          "auth": {
            "method": "pki | password",
            "credentials": {
              "user": "<user-name>",
              "privKeyPath": "/path/to/private/key",
              "password": "<password>"
            }
          }
        }
      }
    ]
  }
]
```

在 devices 对象中，提供以下信息：

id

专属于您设备的用户定义唯一标识符。

connectivity.ip

您设备的 IP 地址。

connectivity.port

可选。用于通过 SSH 连接到您的设备的端口号。

connectivity.auth

连接的身份验证信息。

此属性仅在 connectivity.protocol 设置为 ssh 时适用。

connectivity.auth.method

用于通过给定的连接协议访问设备的身份验证方法。

支持的值为：

- pki
- password

connectivity.auth.credentials

用于身份验证的凭证。

connectivity.auth.credentials.user

用于登录您的设备的用户名。

connectivity.auth.credentials.privKeyPath

用于登录您设备的私有密钥的完整路径。

此值仅在 connectivity.auth.method 设置为 pki 时适用。

devices.connectivity.auth.credentials.password

该密码用于登录到您的设备。

此值仅在 connectivity.auth.method 设置为 password 时适用。

> **Note**
>
> 只有当 method 设置为 pki 时才指定 privKeyPath。
> 只有当 method 设置为 password 时才指定 password。

构建示例测试套件

<device-tester-extract-location>/samples/python 文件夹包含示例配置文件、源代码和 IDT 客户端软件开发工具包，您可以使用提供的构建脚本将其组合成一个测试套件。以下目录树显示了这些示例文件的位置：

```
<device-tester-extract-location>
### ...
### tests
### samples
#   ### ...
#   ### python
#       ### configuration
#       ### src
#       ### build-scripts
#           ### build.sh
#           ### build.ps1
### sdks
    ### ...
    ### python
```

```
        ### idt_client
```

要构建测试套件，请在主机上运行以下命令：

Windows

```
cd <device-tester-extract-location>/samples/python/build-scripts
./build.ps1
```

Linux, macOS, or UNIX

```
cd <device-tester-extract-location>/samples/python/build-scripts
./build.sh
```

这将在该 *<device-tester-extract-location>*/tests 文件夹下的 IDTSampleSuitePython_1.0.0 文件夹中创建示例测试套件。检查 IDTSampleSuitePython_1.0.0文件夹中的文件，以了解示例测试套件的结构，并查看测试用例可执行文件和测试配置文件的各种示例。

> **Note**
>
> 示例测试套件包含 python 源代码。请勿在测试套件代码中包含敏感信息。

下一步：使用 IDT 运行您创建的示例测试套件。

使用 IDT 运行示例测试套件

要运行示例测试套件，请在主机上运行以下命令：

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id IDTSampleSuitePython
```

IDT 会运行示例测试套件，并将结果流式传输到控制台。测试运行完毕后，您会看到以下信息：

```
========== Test Summary ==========
Execution Time:         5s
Tests Completed:        4
```

```
Tests Passed:           4
Tests Failed:           0
Tests Skipped:          0
---------------------------------
Test Groups:
    sample_group:       PASSED
---------------------------------
Path to AWS IoT Device Tester Report: /path/to/devicetester/
results/87e673c6-1226-11eb-9269-8c8590419f30/awsiotdevicetester_report.xml
Path to Test Execution Logs: /path/to/devicetester/
results/87e673c6-1226-11eb-9269-8c8590419f30/logs
Path to Aggregated JUnit Report: /path/to/devicetester/
results/87e673c6-1226-11eb-9269-8c8590419f30/IDTSampleSuitePython_Report.xml
```

排查错误

使用以下信息，以帮助解决在完成本教程时遇到的任何问题。

测试用例未成功运行

- 如果测试运行失败，IDT 会将错误日志流式传输到控制台，以帮助您对测试运行进行故障排除。请确保满足本教程的所有先决条件。

无法连接到被测设备

请验证以下内容：

- 您的 `device.json` 文件包含正确的 IP 地址、端口和身份验证信息。
- 您可以通过 SSH 从主机连接到您的设备。

教程：开发一个简单的 IDT 测试套件

测试套件结合了以下内容：

- 包含测试逻辑的测试可执行文件
- 描述测试套件的配置文件

本教程向您展示如何使用适用于 FreeRTOS 的 IDT 来开发包含单个测试用例的 Python 测试套件。尽管本教程使用 SSH，但学习如何在 FreeRTOS 设备上使用还是很有 AWS IoT Device Tester 用的。

在本教程中，您将完成以下步骤：

1. 创建测试套件目录
2. 创建配置文件
3. 创建测试用例可执行文件
4. 运行测试套件

按照以下步骤完成开发简单 IDT 测试套件的教程。

主题

- 为简单的 IDT 测试套件设置先决条件
- 创建测试套件目录
- 创建配置文件
- 获取 IDT 客户端 SDK
- 创建测试用例可执行文件
- 配置 IDT 的设备信息
- 运行测试套件
- 排查错误
- 创建 IDT 测试套件配置文件
- 配置 IDT 测试编排工具
- 配置 IDT 状态机
- 创建 IDT 测试用例可执行文件
- 使用 IDT 上下文
- 为测试运行器配置设置
- 调试和运行自定义测试套件
- 查看 IDT 测试结果和日志
- 提交 IDT 使用量指标

为简单的 IDT 测试套件设置先决条件

要完成本教程，您需要：

- 主机要求

- 最新版本的 AWS IoT Device Tester
- Python 3.7 或更高版本

  要检查您计算机安装的 Python 版本，请运行以下命令：

  ```
  python3 --version
  ```

  在 Windows 上，如果运行此命令时返回错误，则可改用 `python --version`。如果返回的版本号为 3.7 或更高版本，则可通过在 Powershell 终端中运行以下命令将 `python3` 设置为 `python` 命令的别名。

  ```
  Set-Alias -Name "python3" -Value "python"
  ```

  如果没有返回版本信息，或者版本号小于 3.7，则按照下载 Python 中的说明安装 Python 3.7+。有关更多信息，请参阅 Python 文档。
- urllib3

  要验证 `urllib3` 是否已正确安装，请运行以下命令：

  ```
  python3 -c 'import urllib3'
  ```

  如果未安装 `urllib3`，请运行以下命令进行安装：

  ```
  python3 -m pip install urllib3
  ```

- 设备要求
  - 一种运行 Linux 操作系统的设备，其网络连接到与您主机相同的网络。

    我们建议您使用搭载 Raspberry Pi 操作系统的 Raspberry Pi。请确保您设置 Raspberry Pi 上的 SSH 才能远程连接到它。

创建测试套件目录

IDT 在逻辑上将测试用例分成每个测试套件中的测试组。每个测试用例都必须位于测试组中。在本教程中，创建一个名为 `MyTestSuite_1.0.0` 的文件夹，并在此文件夹中创建以下目录树：

```
MyTestSuite_1.0.0
### suite
```

```
    ### myTestGroup
        ### myTestCase
```

创建配置文件

您的测试套件必须包含以下必需的配置文件：

所需的文件

suite.json

包含有关测试套件的信息。请参阅配置 suite.json。

group.json

包含有关测试组的信息。您必须为测试套件中的每个测试组创建一个 group.json 文件。请参阅配置 group.json。

test.json

包含有关测试用例的信息。您必须为测试套件中的每个测试用例创建一个 test.json 文件。请参阅配置 test.json。

1. 在 MyTestSuite_1.0.0/suite 文件夹中，创建以下文件夹结构的 suite.json：

```
{
    "id": "MyTestSuite_1.0.0",
    "title": "My Test Suite",
    "details": "This is my test suite.",
    "userDataRequired": false
}
```

2. 在 MyTestSuite_1.0.0/myTestGroup 文件夹中，创建以下文件夹结构的 group.json：

```
{
    "id": "MyTestGroup",
    "title": "My Test Group",
    "details": "This is my test group.",
    "optional": false
}
```

3. 在 MyTestSuite_1.0.0/myTestGroup/myTestCase 文件夹中，创建以下文件夹结构的 test.json：

```
{
    "id": "MyTestCase",
    "title": "My Test Case",
    "details": "This is my test case.",
    "execution": {
        "timeout": 300000,
        "linux": {
            "cmd": "python3",
            "args": [
                "myTestCase.py"
            ]
        },
        "mac": {
            "cmd": "python3",
            "args": [
                "myTestCase.py"
            ]
        },
        "win": {
            "cmd": "python3",
            "args": [
                "myTestCase.py"
            ]
        }
    }
}
```

您的 `MyTestSuite_1.0.0` 文件夹应类似于以下内容：

```
MyTestSuite_1.0.0
### suite
    ### suite.json
    ### myTestGroup
        ### group.json
        ### myTestCase
            ### test.json
```

获取 IDT 客户端 SDK

您可以使用 IDT 客户端 SDK 让 IDT 与被测设备进行交互并报告测试结果。在本教程中，您将使用 Python 版本的软件开发工具包。

从 *<device-tester-extract-location>*/sdks/python/ 文件夹，将 idt_client 文件夹复制到您的 MyTestSuite_1.0.0/suite/myTestGroup/myTestCase 文件夹。

要验证 SDK 是否复制，可以运行以下命令。

```
cd MyTestSuite_1.0.0/suite/myTestGroup/myTestCase
python3 -c 'import idt_client'
```

创建测试用例可执行文件

测试用例可执行文件包含要运行的测试逻辑。一个测试套件可以包含多个测试用例可执行文件。在本教程中，您将只创建一个测试用例可执行文件。

1. 创建测试套件文件。

   在 MyTestSuite_1.0.0/suite/myTestGroup/myTestCase 文件夹中，创建使用以下内容的 myTestCase.py 文件：

   ```
   from idt_client import *

   def main():
       # Use the client SDK to communicate with IDT
       client = Client()

   if __name__ == "__main__":
       main()
   ```

2. 使用客户端 SDK 函数将以下测试逻辑添加到您的 myTestCase.py 文件中：

   a. 在被测设备上运行 SSH 命令。

   ```
   from idt_client import *

   def main():
       # Use the client SDK to communicate with IDT
       client = Client()

       # Create an execute on device request
       exec_req = ExecuteOnDeviceRequest(ExecuteOnDeviceCommand("echo 'hello
    world'"))

       # Run the command
   ```

```
    exec_resp = client.execute_on_device(exec_req)

    # Print the standard output
    print(exec_resp.stdout)

if __name__ == "__main__":
    main()
```

b. 将测试结果发送给 IDT。

```
from idt_client import *

def main():
    # Use the client SDK to communicate with IDT
    client = Client()

    # Create an execute on device request
    exec_req = ExecuteOnDeviceRequest(ExecuteOnDeviceCommand("echo 'hello
 world'"))

    # Run the command
    exec_resp = client.execute_on_device(exec_req)

    # Print the standard output
    print(exec_resp.stdout)

    # Create a send result request
    sr_req = SendResultRequest(TestResult(passed=True))

    # Send the result
    client.send_result(sr_req)

if __name__ == "__main__":
    main()
```

配置 IDT 的设备信息

配置您的设备信息，以便 IDT 运行测试。您必须使用以下信息，更新位于 *<device-tester-extract-location>*/configs 文件夹中的 device.json 模板。

```
[
  {
```

```
    "id": "pool",
    "sku": "N/A",
    "devices": [
      {
        "id": "<device-id>",
        "connectivity": {
          "protocol": "ssh",
          "ip": "<ip-address>",
          "port": "<port>",
          "auth": {
            "method": "pki | password",
            "credentials": {
              "user": "<user-name>",
              "privKeyPath": "/path/to/private/key",
              "password": "<password>"
            }
          }
        }
      }
    ]
  }
]
```

在 devices 对象中，提供以下信息：

id

专属于您设备的用户定义唯一标识符。

connectivity.ip

您设备的 IP 地址。

connectivity.port

可选。用于通过 SSH 连接到您的设备的端口号。

connectivity.auth

连接的身份验证信息。

此属性仅在 connectivity.protocol 设置为 ssh 时适用。

connectivity.auth.method

用于通过给定的连接协议访问设备的身份验证方法。

支持的值为：

- pki
- password

connectivity.auth.credentials

用于身份验证的凭证。

connectivity.auth.credentials.user

用于登录您的设备的用户名。

connectivity.auth.credentials.privKeyPath

用于登录您设备的私有密钥的完整路径。

此值仅在 connectivity.auth.method 设置为 pki 时适用。

devices.connectivity.auth.credentials.password

该密码用于登录到您的设备。

此值仅在 connectivity.auth.method 设置为 password 时适用。

> **Note**
>
> 只有当 method 设置为 pki 时才指定 privKeyPath。
> 只有当 method 设置为 password 时才指定 password。

运行测试套件

创建测试套件后，您需要确保其按预期运行。要使用现有设备池运行测试套件，请完成以下步骤。

1. 将您的 MyTestSuite_1.0.0 文件夹复制到 *<device-tester-extract-location>*/tests。
2. 运行以下 命令：

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id MyTestSuite
```

IDT 会运行您的测试套件，并将结果流式传输到控制台。测试运行完毕后，您会看到以下信息：

```
time="2020-10-19T09:24:47-07:00" level=info msg=Using pool: pool
time="2020-10-19T09:24:47-07:00" level=info msg=Using test suite "MyTestSuite_1.0.0"
 for execution
time="2020-10-19T09:24:47-07:00" level=info msg=b'hello world\n'
 suiteId=MyTestSuite groupId=myTestGroup testCaseId=myTestCase deviceId=my-device
 executionId=9a52f362-1227-11eb-86c9-8c8590419f30
time="2020-10-19T09:24:47-07:00" level=info msg=All tests finished.
 executionId=9a52f362-1227-11eb-86c9-8c8590419f30
time="2020-10-19T09:24:48-07:00" level=info msg=

========== Test Summary ==========
Execution Time:         1s
Tests Completed:        1
Tests Passed:           1
Tests Failed:           0
Tests Skipped:          0
----------------------------------
Test Groups:
    myTestGroup:        PASSED
----------------------------------
Path to AWS IoT Device Tester Report: /path/to/devicetester/
results/9a52f362-1227-11eb-86c9-8c8590419f30/awsiotdevicetester_report.xml
Path to Test Execution Logs: /path/to/devicetester/
results/9a52f362-1227-11eb-86c9-8c8590419f30/logs
Path to Aggregated JUnit Report: /path/to/devicetester/
results/9a52f362-1227-11eb-86c9-8c8590419f30/MyTestSuite_Report.xml
```

排查错误

使用以下信息，以帮助解决在完成本教程时遇到的任何问题。

测试用例未成功运行

如果测试运行失败，IDT 会将错误日志流式传输到控制台，以帮助您对测试运行进行故障排除。检查错误日志之前，请验证以下内容：

- IDT 客户端 SDK 位于正确的文件夹中，如中所述。获取 IDT 客户端 SDK
- 您已满足本教程的所有先决条件。有关更多信息，请参阅 为简单的 IDT 测试套件设置先决条件。

无法连接到被测设备

请验证以下内容：

- 您的 device.json 文件包含正确的 IP 地址、端口和身份验证信息。
- 您可以通过 SSH 从主机连接到您的设备。

创建 IDT 测试套件配置文件

本节介绍创建配置文件时使用的格式，您在编写自定义测试套件时包含了这些文件。

必需配置文件

suite.json

包含有关测试套件的信息。请参阅配置 suite.json。

group.json

包含有关测试组的信息。您必须为测试套件中的每个测试组创建一个 group.json 文件。请参阅配置 group.json。

test.json

包含有关测试用例的信息。您必须为测试套件中的每个测试用例创建一个 test.json 文件。请参阅配置 test.json。

可选配置文件

test_orchestrator.yaml 或 state_machine.json

定义 IDT 运行测试套件时如何运行测试。 SSe 配置 test_orchestrator.yaml。

> **Note**
>
> 从 IDT v4.5.2 开始，您可以使用 test_orchestrator.yaml 文件来定义测试工作流程。在以前版本的 IDT 中，请使用 state_machine.json 文件。有关状态机的信息，请参阅配置 IDT 状态机。

userdata_schema.json

定义测试运行器可以在其设置配置中包含的userdata.json文件架构。userdata.json 文件用于存储运行测试所需但 device.json 文件中不存在的任何其他配置信息。请参阅配置 userdata_schema.json。

配置文件放置在您的 *<custom-test-suite-folder>* 中，如下所示。

```
<custom-test-suite-folder>
### suite
    ### suite.json
    ### test_orchestrator.yaml
    ### userdata_schema.json
    ### <test-group-folder>
        ### group.json
        ### <test-case-folder>
            ### test.json
```

配置 suite.json

suite.json 文件设置环境变量并确定运行测试套件是否需要用户数据。使用以下模板来配置您的 *<custom-test-suite-folder>*/suite/suite.json 文件：

```
{
    "id": "<suite-name>_<suite-version>",
    "title": "<suite-title>",
    "details": "<suite-details>",
    "userDataRequired": true | false,
    "environmentVariables": [
        {
            "key": "<name>",
            "value": "<value>",
        },
        ...
        {
            "key": "<name>",
            "value": "<value>",
        }
    ]
}
```

包含值的所有字段都为必填字段，如下所述：

id

测试套件的唯一用户定义 ID。id 的值必须与 suite.json 文件所在的测试套件文件夹的名称相匹配。套件名称和套件版本还必须满足以下要求：

- *<suite-name>* 不可以包含下划线。

- *<suite-version>* 表示为 *x.x.x*，其中 x 为数字。

ID 显示在 IDT 生成的测试报告中。

title

此测试套件正在测试的产品或功能的用户定义名称。该名称显示在 IDT CLI 中供测试运行器使用。

details

测试套件用途的简短描述。

userDataRequired

定义测试运行器是否需要在 userdata.json 文件中包含自定义信息。如果将此值设置为 true，还必须将userdata_schema.json 文件包含在测试套件文件夹中。

environmentVariables

可选。一组要为此测试套件设置的环境变量。

environmentVariables.key

环境变量的名称。

environmentVariables.value

环境变量的值。

配置 group.json

group.json 文件定义测试组是必需的还是可选的。使用以下模板来配置您的 *<custom-test-suite-folder>*/suite/*<test-group>*/group.json 文件：

```
{
    "id": "<group-id>",
    "title": "<group-title>",
    "details": "<group-details>",
    "optional": true | false,
}
```

包含值的所有字段都为必填字段，如下所述：

id

测试组的唯一用户定义 ID。id 的值必须与 group.json 文件所在的测试组文件夹的名称相匹配，并且不得包含下划线 (_)。ID 用于 IDT 生成的测试报告中。

title

测试组的描述性名称。该名称显示在 IDT CLI 中供测试运行器使用。

details

测试组用途的简短描述。

optional

可选。设置为 true 以便在 IDT 完成运行所需测试后，将此测试组显示为可选组。默认值为 false。

配置 test.json

test.json 文件确定测试用例的可执行文件和测试用例使用的环境变量。有关创建测试用例可执行文件的更多信息，请参阅 创建 IDT 测试用例可执行文件。

使用以下模板来配置您的 *<custom-test-suite-folder>*/suite/*<test-group>*/*<test-case>*/test.json 文件：

```
{
    "id": "<test-id>",
    "title": "<test-title>",
    "details": "<test-details>",
    "requireDUT": true | false,
    "requiredResources": [
        {
            "name": "<resource-name>",
            "features": [
                {
                    "name": "<feature-name>",
                    "version": "<feature-version>",
                    "jobSlots": <job-slots>
                }
            ]
        }
    ],
```

```
    "execution": {
        "timeout": <timeout>,
        "mac": {
            "cmd": "/path/to/executable",
            "args": [
                "<argument>"
            ],
        },
        "linux": {
            "cmd": "/path/to/executable",
            "args": [
                "<argument>"
            ],
        },
        "win": {
            "cmd": "/path/to/executable",
            "args": [
                "<argument>"
            ]
        }
    },
    "environmentVariables": [
        {
            "key": "<name>",
            "value": "<value>",
        }
    ]
}
```

包含值的所有字段都为必填字段，如下所述：

id

测试用例的唯一用户定义 ID。id 的值必须与 test.json 文件所在的测试用例文件夹的名称相匹配，并且不得包含下划线 (_)。ID 用于 IDT 生成的测试报告中。

title

测试用例的描述性名称。该名称显示在 IDT CLI 中供测试运行器使用。

details

测试用例用途的简短描述。

requireDUT

可选。如果需要设备才能运行此测试，则设置为 `true`，否则设置为 `false`。默认值为 `true`。测试运行器将在其 `device.json` 文件中配置将用来运行测试的设备。

requiredResources

可选。一个阵列，提供有关运行此测试所需资源设备的信息。

requiredResources.name

运行此测试时为资源设备提供的唯一名称。

requiredResources.features

一系列用户定义的资源设备功能。

requiredResources.features.name

功能的名称。您要使用此设备的设备功能。此名称与 `resource.json` 文件中测试运行器提供的功能名称相匹配。

requiredResources.features.version

可选。功能的版本。此值与 `resource.json` 文件中测试运行器提供的功能版本相匹配。如果未提供版本，则不检查该功能。如果功能不需要版本号，请将此字段留为空白。

requiredResources.features.jobSlots

可选。此功能可以支持的同时测试的数量。默认值为 1。如果您希望 IDT 使用不同的设备来实现各项功能，我们建议您将此值设置为 1。

execution.timeout

IDT 等待测试完成运行的时间长度（以毫秒为单位）。有关设置该值的更多信息，请参阅 创建 IDT 测试用例可执行文件。

execution.*os*

要运行的测试用例可执行文件基于运行 IDT 的主机操作系统。支持的值有`linux`、`mac`和`win`。

execution.*os*.cmd

要在指定操作系统上运行的测试用例可执行文件的路径。此位置必须位于系统路径中。

execution.*os*.args

可选。为运行测试用例可执行文件而提供的参数。

environmentVariables

可选。一组要为此测试用例设置的环境变量。

environmentVariables.key

环境变量的名称。

environmentVariables.value

环境变量的值。

> **Note**
>
> 如果在 test.json 文件和 suite.json 文件中指定相同的环境变量，则 test.json 文件中的值优先。

配置 test_orchestrator.yaml

测试编排工具是一种控制测试套件执行流程的构造。它决定测试套件的起始状态，根据用户定义的规则管理状态转换，并继续在这些状态之间进行转换，直到达到结束状态。

如果您的测试套件不包含用户定义的测试编排工具，IDT 将为您生成一个测试编排工具。

默认测试编排工具执行以下功能：

- 使测试运行器能够选择和运行特定的测试组，而不是整个测试套件。
- 如果未选择特定的测试组，则按随机顺序运行测试套件中的每个测试组。
- 生成报告并打印控制台摘要，其中显示每个测试组和测试用例的测试结果。

有关 IDT 测试编排工具工作原理的更多信息，请参阅配置 IDT 测试编排工具。

配置 userdata_schema.json

userdata_schema.json 文件确定了测试运行器提供用户数据的架构。如果您的测试套件需要 device.json 文件中不存在的信息，则需要用户数据。例如，您的测试可能需要 Wi-Fi 网络凭证、特定的开放端口或用户必须提供的证书。此信息可提供给 IDT，作为用户在其 *<device-tester-extract-location>*/config 文件夹中创建的输入参数，称为 userdata，其值是一个 userdata.json 文件。userdata.json 文件的格式取决于您在测试套件中包含的 userdata_schema.json 文件。

要指示测试运行器必须提供 userdata.json 文件：

1. 在 suite.json 文件中设置 userDataRequired 为 true。
2. 在您的 *<custom-test-suite-folder>* 中，创建一个 userdata_schema.json 文件。
3. 编辑 userdata_schema.json 文件以创建有效的 IETF 草稿 v4 JSON 架构。

当 IDT 运行您的测试套件时，它会自动读取架构并使用它来验证测试运行器提供的 userdata.json 文件。如果有效，则 userdata.json 文件的内容在 IDT 上下文和测试编排工具上下文中均可用。

配置 IDT 测试编排工具

从 IDT v4.5.2 开始，IDT 包括一个新的测试编排工具组件。测试编排工具是一个 IDT 组件，用于控制测试套件的执行流程，并在 IDT 完成运行所有测试后生成测试报告。测试编排工具根据用户定义的规则确定测试选择和测试的运行顺序。

如果您的测试套件不包含用户定义的测试编排工具，IDT 将为您生成一个测试编排工具。

默认测试编排工具执行以下功能：

- 使测试运行器能够选择和运行特定的测试组，而不是整个测试套件。
- 如果未选择特定的测试组，则按随机顺序运行测试套件中的每个测试组。
- 生成报告并打印控制台摘要，其中显示每个测试组和测试用例的测试结果。

测试编排工具用于取代 IDT 状态机。我们强烈建议您使用测试编排工具来开发测试套件，而不是使用 IDT 状态机。测试编排工具提供了以下改进功能：

- IDT 状态机使用命令式格式，而该工具使用声明式格式。这允许您指定要运行哪些测试以及何时运行它们。
- 管理特定的组处理、报告生成、错误处理和结果跟踪，让您无需手动管理这些操作。
- 使用 YAML 格式，该格式默认支持注释。
- 定义相同的工作流程所需的磁盘空间比测试编排工具少 80%。
- 添加测试前验证，以验证您的工作流程定义不包含不正确的测试 IDs 或循环依赖关系。

测试编排工具格式

您可以使用以下模板来配置自己的*custom-test-suite-folder*/suite/test_orchestrator.yaml文件：

```
Aliases:
  string: context-expression

ConditionalTests:
  - Condition: context-expression
    Tests:
      - test-descriptor

Order:
  - - group-descriptor
    - group-descriptor

Features:
  - Name: feature-name
    Value: support-description
    Condition: context-expression
    Tests:
        - test-descriptor
    OneOfTests:
        - test-descriptor
    IsRequired: boolean
```

包含值的所有字段都为必填字段，如下所述：

`Aliases`

可选。映射到上下文表达式的用户定义字符串。别名允许您生成友好的名称，以便识别测试编排工具配置中的上下文表达式。如果您要创建复杂的上下文表达式或在多个位置使用的表达式，则此功能特别有用。

您可以使用上下文表达式来存储上下文查询，从而允许您访问其他 IDT 配置中的数据。有关更多信息，请参阅 在上下文中访问数据。

Example

示例

```
Aliases:
    FizzChosen: "'{{$pool.features[?(@.name == 'Fizz')].value[0]}}' == 'yes'"
    BuzzChosen: "'{{$pool.features[?(@.name == 'Buzz')].value[0]}}' == 'yes'"
    FizzBuzzChosen: "'{{$aliases.FizzChosen}}' && '{{$aliases.BuzzChosen}}'"
```

`ConditionalTests`

可选。条件列表以及满足每个条件时运行的相应测试用例。每个条件可以有多个测试用例；但是，您只能将给定测试用例分配给一个条件。

默认情况下，IDT 会运行任何未分配给此列表中的条件的测试用例。如果您未指定此部分，IDT 将运行测试套件中的所有测试组。

`ConditionalTests` 列表中的每个项目都包含以下参数：

`Condition`

计算结果为布尔值的上下文字符串。如果计算值为 true，IDT 将运行 `Tests` 参数中指定的测试用例。

`Tests`

测试描述符列表。

每个测试描述符使用测试组 ID 和一个或多个测试用例 IDs 来标识要从特定测试组运行的单个测试。测试描述符使用以下格式：

```
GroupId: group-id
CaseIds: [test-id, test-id] # optional
```

Example

示例

以下示例使用可定义为 `Aliases` 的通用上下文表达式。

```
ConditionalTests:
    - Condition: "{{$aliases.Condition1}}"
      Tests:
          - GroupId: A
          - GroupId: B
    - Condition: "{{$aliases.Condition2}}"
      Tests:
          - GroupId: D
    - Condition: "{{$aliases.Condition1}} || {{$aliases.Condition2}}"
      Tests:
          - GroupId: C
```

IDT 根据定义的条件选择测试组，如下所示：

- 如果 Condition1 为 true，IDT 将在测试组 A、B 和 C 中运行测试。
- 如果 Condition2 为 true，IDT 将在测试组 C 和 D 中运行测试。

`Order`

可选。运行测试的顺序。您可以在测试组级别指定测试顺序。如果未指定此部分，IDT 将按随机顺序运行所有适用的测试组。`Order` 的值是组描述符列表的列表。未在 `Order` 中列出的任何测试组都可以与列出的任何其他测试组并行运行。

每个组描述符列表都包含一个或多个组描述符，并标识每个描述符中指定的组的运行顺序。您可以使用以下格式定义单个组描述符：

- *group-id* – 现有测试组的组 ID。
- [*group-id*, *group-id*] – 可按相对于彼此的任意顺序运行的测试组列表。
- "*" – 通配符。这等同于所有尚未在当前组描述符列表中指定的测试组的列表。

`Order` 的值必须满足以下要求：

- 您在组 IDs 描述符中指定的测试组必须存在于您的测试套件中。
- 每个组描述符列表必须包含至少一个测试组。
- 每个组描述符列表必须包含唯一的组 IDs。不能在单个组描述符中使用重复的测试组 ID。
- 一个组描述符列表最多可以有一个通配符组描述符。通配符组描述符必须是列表中的第一项或最后一项。

Example

示例

对于包含测试组 A、B、C、D 和 E 的测试套件，以下示例列表显示了指定 IDT 应先运行测试组 A，然后运行测试组 B，然后按任意顺序运行测试组 C、D 和 E 的不同方法。

- ```
  Order:
      - - A
        - B
        - [C, D, E]
  ```
- ```
  Order:
      - - A
        - B
        - "*"
  ```

- ```
  Order:
      - - A
        - B

      - - B
        - C

      - - B
        - D

      - - B
        - E
  ```

Features

可选。您希望将 IDT 添加到 awsiotdevicetester_report.xml 文件中的产品功能列表。如果未指定此部分，则 IDT 不会将任何产品功能添加到报告中。

产品功能是关于设备可能符合的特定标准的用户定义信息。例如，MQTT 产品功能可以指定设备正确发布 MQTT 消息。在 awsiotdevicetester_report.xml 中，根据指定的测试是否通过，将产品功能设置为 supported、not-supported 或自定义的用户定义值。

Features 列表中的每个项目都包含以下参数：

Name

特征的名称。

Value

可选。要在报告中使用的自定义值，而不是 supported。如果未指定此值，则根据测试结果，IDT 将特征值设置为 supported 或 not-supported。如果您使用不同的条件测试相同的功能，则可以在 Features 列表中为该特征的各个实例使用自定义值，而 IDT 会将支持条件的特征值串联起来。有关更多信息，请参阅

Condition

计算结果为布尔值的上下文字符串。如果计算值为 true，IDT 将在运行完测试套件后将该功能添加到测试报告中。如果计算值为 false，则不会将测试包含在报告中。

Tests

可选。测试描述符列表。必须通过在此列表中指定的所有测试才能支持该功能。
  ```

此列表中的每个测试描述符都使用测试组 ID 和一个或多个测试用例 IDs 来标识要从特定测试组运行的单个测试。测试描述符使用以下格式：

```
GroupId: group-id
CaseIds: [test-id, test-id] # optional
```

必须为 Features 列表中的每个特征指定 Tests 或 OneOfTests。

OneOfTests

可选。测试描述符列表。必须通过在此列表中指定的至少一个测试才能支持该功能。

此列表中的每个测试描述符都使用测试组 ID 和一个或多个测试用例 IDs 来标识要从特定测试组运行的单个测试。测试描述符使用以下格式：

```
GroupId: group-id
CaseIds: [test-id, test-id] # optional
```

必须为 Features 列表中的每个特征指定 Tests 或 OneOfTests。

IsRequired

用于定义测试报告中是否需要该功能的布尔值。默认值为 false。

测试编排工具上下文

测试编排工具上下文是一个只读 JSON 文档，其中包含在执行期间可供测试编排工具使用的数据。测试编排工具上下文只能从测试编排工具访问，其中包含决定测试流程的信息。例如，您可以使用 userdata.json 文件中测试运行器配置的信息来确定是否需要运行特定的测试。

测试编排工具上下文使用以下格式：

```
{
    "pool": {
        <device-json-pool-element>
    },
    "userData": {
        <userdata-json-content>
    },
    "config": {
        <config-json-content>
    }
```

```
}
```

`pool`

有关为测试运行选择的设备池的信息。对于选定的设备池，此信息将从 `device.json` 文件中定义的相应顶级设备池阵列元素中检索。

`userData`

`userdata.json` 文件中的信息

`config`

`config.json` 文件中的信息

您可以使用 JSONPath 符号查询上下文。状态定义中 JSONPath查询的语法是`{{query}}`。从测试编排工具上下文访问数据时，请确保每个值的计算结果都为字符串、数字或布尔值。

有关使用 JSONPath 符号从上下文访问数据的更多信息，请参阅使用 IDT 上下文。

配置 IDT 状态机

> **Important**
>
> 从 IDT v4.5.2 开始，此状态机已弃用。我们强烈建议您使用新的测试编排工具。有关更多信息，请参阅 配置 IDT 测试编排工具。

状态机是一种控制测试套件执行流程的构造。它决定测试套件的起始状态，根据用户定义的规则管理状态转换，并继续在这些状态之间进行转换，直到达到结束状态。

如果您的测试套件不包含用户定义的状态机，IDT 将为您生成状态机。默认状态机执行以下功能：

- 使测试运行器能够选择和运行特定的测试组，而不是整个测试套件。
- 如果未选择特定的测试组，则按随机顺序运行测试套件中的每个测试组。
- 生成报告并打印控制台摘要，其中显示每个测试组和测试用例的测试结果。

IDT 测试套件的状态机必须满足以下条件：

- 每个状态都对应于 IDT 要采取的操作，例如运行测试组或产品报告文件。
- 过渡到状态会执行与该状态关联的操作。

- 每个状态都定义了下一个状态的过渡规则。
- 结束状态必须为 Succeed 或 Fail。

状态机格式

您可以使用以下模板来配置自己的*<custom-test-suite-folder>*/suite/state_machine.json文件：

```
{
  "Comment": "<description>",
  "StartAt": "<state-name>",
  "States": {
    "<state-name>": {
      "Type": "<state-type>",
      // Additional state configuration
    }

    // Required states
    "Succeed": {
      "Type": "Succeed"
    },
    "Fail": {
      "Type": "Fail"
    }
  }
}
```

包含值的所有字段都为必填字段，如下所述：

Comment

状态机的描述。

StartAt

IDT 开始运行测试套件的状态名称。StartAt 的值必须设置为 States 对象中列出的其中一个状态。

States

将用户定义的状态名称映射到有效的 IDT 状态的对象。每个州。 *state-name*对象包含映射到. 的有效状态的定义*state-name*。

States 对象必须包含 Succeed 和 Fail 状态。有关有效状态的信息，请参阅有效状态和状态定义。

有效状态和状态定义

本节介绍可在 IDT 状态机中使用的所有有效状态的状态定义。以下某些状态支持测试用例级别的配置。但是，除非绝对必要，否则我们建议您在测试组级别而不是测试用例级别配置状态转换规则。

状态定义

- RunTask
- Choice
- Parallel
- AddProductFeatures
- 报告
- LogMessage
- SelectGroup
- Fail
- Succeed

RunTask

RunTask 状态运行测试套件中定义的测试组中的测试用例。

```
{
    "Type": "RunTask",
    "Next": "<state-name>",
    "TestGroup": "<group-id>",
    "TestCases": [
        "<test-id>"
    ],
    "ResultVar": "<result-name>"
}
```

包含值的所有字段都为必填字段，如下所述：

Next

在当前状态下执行操作后要过渡到的状态的名称。

TestGroup

可选。要运行的测试组的 ID。如果未指定此值，则 IDT 将运行测试运行器选择的测试组。

TestCases

可选。 IDs 来自中指定组的测试用例数组TestGroup。IDT 根据 TestGroup 和 TestCases 的值确定测试执行行为，如下所示：

- 同时指定 TestGroup 和 TestCases 时，IDT 会运行测试组中的指定测试用例。
- 如果 TestCases 已指定但 TestGroup 未指定，IDT 将运行指定的测试用例。
- 如果 TestGroup 已指定但 TestCases 未指定，则 IDT 将运行指定测试组中的所有测试用例。
- 如果未指定 TestGroup 或 TestCases，IDT 将运行测试运行器从 IDT CLI 中选择的测试组中的所有测试用例。要为测试运行器启用组选择，必须在 statemachine.json 文件中同时包含 RunTask 和 Choice 状态。有关其工作原理的示例，请参阅状态机示例：运行用户选择的测试组。

  有关为测试运行器启用 IDT CLI 命令的更多信息，请参阅 the section called “启用 IDT CLI 命令”。

ResultVar

要与测试运行结果一起设置的上下文变量的名称。如果您没有为指定值，请不要指定此值 TestGroup。基于以下内容，IDT 将您在 ResultVar 中定义的变量的值设置为 true 或 false：

- 如果变量名称的形式为 *text*_*text*_passed，则该值设置为第一个测试组中的所有测试均已通过或跳过。
- 在所有其他情况下，该值设置为所有测试组中的所有测试已通过或跳过。

通常，您将使用RunTask状态来指定测试组 ID，而不指定单个测试用例 IDs，这样 IDT 将运行指定测试组中的所有测试用例。在此状态下运行的所有测试用例均按随机顺序并行运行。但是，如果所有测试用例都需要一台设备运行，并且只有一台设备可用，则测试用例将按顺序运行。

错误处理

如果任何指定的测试组或测试用例 IDs 无效，则此状态会发出RunTaskError执行错误。如果状态遇到执行错误，则它还会将状态机上下文中的 hasExecutionError 变量设置为 true。

Choice

Choice 状态允许您根据用户定义的条件动态设置要过渡到的下一个状态。

```
{
    "Type": "Choice",
    "Default": "<state-name>",
    "FallthroughOnError": true | false,
    "Choices": [
        {
            "Expression": "<expression>",
            "Next": "<state-name>"
        }
    ]
}
```

包含值的所有字段都为必填字段，如下所述：

Default

如果 Choices 中定义的表达式都无法评估到 true，则设置要过渡到的默认状态。

FallthroughOnError

可选。指定状态在评估表达式时遇到错误时的行为。如果要在评估结果出错时跳过表达式，则设置为 true。如果没有匹配的表达式，则状态机将过渡到 Default 状态。如果 FallthroughOnError 的值未指定，将默认为 false。

Choices

一组表达式和状态，用于确定在当前状态下执行操作后要过渡到哪个状态。

Choices.Expression

计算结果为布尔值的表达式字符串。如果表达式的评估结果为 true，则状态机将过渡到 Choices.Next 中定义的状态。表达式字符串从状态机上下文中检索值，然后对它们执行操作以得出布尔值。有关访问状态机上下文的信息，请参阅状态机上下文。

Choices.Next

如果 Choices.Expression 中定义的表达式的评估结果为 true，则设置要过渡到的状态的名称。

错误处理

在以下情况下，Choice 状态可能需要错误处理：

- 选择表达式中的某些变量在状态机上下文中不存在。

- 表达式的结果不是布尔值。
- JSON 查询的结果不是字符串、数字或布尔值。

在这种状态下，不能使用 Catch 数据块来处理错误。如果要在状态机遇到错误时停止执行它，则必须将 FallthroughOnError 设置为 false。但是，建议您将 FallthroughOnError 设置为 true，并根据您的用例，执行以下操作之一：

- 如果您正在访问的变量预计在某些情况下不存在，则使用的值 Default 和其他 Choices 数据块来指定下一个状态。
- 如果您正在访问的变量应始终存在，则将 Default 状态设置为 Fail。

Parallel

Parallel 状态允许您并行地定义和运行新的状态机。

```
{
    "Type": "Parallel",
    "Next": "<state-name>",
    "Branches": [
        <state-machine-definition>
    ]
}
```

包含值的所有字段都为必填字段，如下所述：

Next

在当前状态下执行操作后要过渡到的状态的名称。

Branches

要运行的状态机定义阵列。每个状态机定义都必须包含自己的StartAt、Succeed 和 Fail 状态。此阵列中的状态机定义不能引用其自身定义之外的状态。

> **Note**
>
> 由于每个分支状态机共享相同的状态机上下文，因此在一个分支中设置变量然后从另一个分支读取这些变量可能会导致意外行为。

只有在 Parallel 运行所有分支状态机之后，该状态才会移动到下一个状态。每种需要设备的状态都将等到设备可用后才运行。如果有多个设备可用，则此状态会并行运行来自多个组的测试用例。如果没有足够的设备可用，则测试用例将按顺序运行。由于测试用例在并行运行时按随机顺序运行，因此可能会使用不同的设备来运行来自同一个测试组的测试。

错误处理

确保分支状态机和父状态机都过渡到 Fail 状态以处理执行错误。

由于分支状态机不会将执行错误传输到父状态机，因此您不能使用 Catch 数据块来处理分支状态机中的执行错误。相反，在共享状态机上下文中使用 hasExecutionErrors 值。有关如何执行此操作的示例，请参阅 状态机示例：并行运行两个测试组。

AddProductFeatures

AddProductFeatures 状态允许您将产品功能添加到 IDT 生成的 awsiotdevicetester_report.xml 文件中。

产品功能是关于设备可能符合的特定标准的用户定义信息。例如，MQTT 产品功能可以指定设备正确发布 MQTT 消息。在报告中，根据指定的测试是否通过，将产品功能设置为 supported、not-supported 或自定义值。

> **Note**
>
> AddProductFeatures 状态本身不生成报告。此状态必须过渡到Report状态才能生成报告。

```
{
    "Type": "Parallel",
    "Next": "<state-name>",
    "Features": [
        {
            "Feature": "<feature-name>",
            "Groups": [
                "<group-id>"
            ],
            "OneOfGroups": [
                "<group-id>"
            ],
```

```
            "TestCases": [
                "<test-id>"
            ],
            "IsRequired": true | false,
            "ExecutionMethods": [
                "<execution-method>"
            ]
        }
    ]
}
```

包含值的所有字段都为必填字段，如下所述：

Next

在当前状态下执行操作后要过渡到的状态的名称。

Features

要在 awsiotdevicetester_report.xml 文件中显示的一系列产品功能。

Feature

功能的名称

FeatureValue

可选。要在报告中使用的自定义值，而不是 supported。如果未指定此值，则根据测试结果，将特征值设置为 supported 或 not-supported。

如果您使用 FeatureValue 自定义值，则可以在不同的条件下测试相同的功能，IDT 会将支持条件的特征值串联起来。例如，以下摘录显示具有两个独立 MyFeature 特征值的要素：

```
...
{
    "Feature": "MyFeature",
    "FeatureValue": "first-feature-supported",
    "Groups": ["first-feature-group"]
},
{
    "Feature": "MyFeature",
    "FeatureValue": "second-feature-supported",
    "Groups": ["second-feature-group"]
},
```

```
...
```

如果两个测试组都通过，则特征值将设置为 `first-feature-supported, second-feature-supported`。

Groups

可选。一组测试组 IDs。每个指定测试组中的所有测试都必须通过才能支持该功能。

OneOfGroups

可选。一组测试组 IDs。至少一个指定测试组中的所有测试都必须通过才能支持该功能。

TestCases

可选。一系列测试用例 IDs。如果您指定此值，则适用以下条件：

- 必须通过所有指定的测试用例才能支持该功能。
- `Groups` 必须仅包含一个测试组 ID。
- `OneOfGroups` 不得指定。

IsRequired

可选。设置为 `false` 可在报告中将此功能标记为可选功能。默认值为 `true`。

ExecutionMethods

可选。与 `device.json` 文件中指定的 `protocol` 值相匹配的执行方法阵列。如果指定了此值，则测试运行器必须指定与该阵列中的一个 `protocol` 值相匹配的值，才能将该功能包含在报告中。如果未指定此值，则该要素将始终包含在报告中。

要使用 `AddProductFeatures` 状态，必须将 `RunTask` 状态 `ResultVar` 中的值设置为以下值之一：

- 如果您指定了单个测试用例 IDs，则将设置`ResultVar`为*`group-id_test-id`*_passed。
- 如果您未指定单个测试用例 IDs，则将设置`ResultVar`为*`group-id`*_passed。

`AddProductFeatures` 状态通过以下方式检查测试结果：

- 如果您未指定任何测试用例 IDs，则每个测试组的结果将根据状态机上下文中的*`group-id`*_passed变量值确定。
- 如果您确实指定了测试用例 IDs，则每个测试的结果都是根据状态机上下文中*`group-id_test-id`*_passed变量的值确定的。

错误处理

如果在此状态下提供的群组 ID 不是有效的组 ID，则此状态会导致 AddProductFeaturesError 执行错误。如果状态遇到执行错误，则它还会将状态机上下文中的 hasExecutionErrors 变量设置为 true。

报告

Report 状态会生成 *suite-name*_Report.xml 和 awsiotdevicetester_report.xml 文件。此状态还会将报告流式传输到控制台。

```
{
    "Type": "Report",
    "Next": "<state-name>"
}
```

包含值的所有字段都为必填字段，如下所述：

Next

在当前状态下执行操作后要过渡到的状态的名称。

您应始终过渡到测试执行流程即将结束时的 Report 状态，以便测试运行器可以查看测试结果。通常，此状态之后的下一个状态是 Succeed。

错误处理

如果此状态在生成报告时遇到问题，则会发出 ReportError 执行错误。

LogMessage

LogMessage 状态生成 test_manager.log 文件并将日志消息流式传输到控制台。

```
{
    "Type": "LogMessage",
    "Next": "<state-name>"
    "Level": "info | warn | error"
    "Message": "<message>"
}
```

包含值的所有字段都为必填字段，如下所述：

Next

在当前状态下执行操作后要过渡到的状态的名称。

Level

创建日志消息的错误级别。如果您指定的级别无效，则此状态会生成一条错误消息并将其丢弃。

Message

要记入日志的消息。

SelectGroup

`SelectGroup` 状态会更新状态机上下文以指示选择了哪些组。任何后续 `Choice` 状态都使用此状态设置的值。

```
{
    "Type": "SelectGroup",
    "Next": "<state-name>"
    "TestGroups": [
        <group-id>"
    ]
}
```

包含值的所有字段都为必填字段，如下所述：

Next

在当前状态下执行操作后要过渡到的状态的名称。

TestGroups

一组将被标记为选中的测试组。对于此阵列中的每个测试组 ID，*group-id*_selected 变量在上下文中都设置为 `true`。请确保提供有效的测试组， IDs 因为 IDT 不会验证指定的组是否存在。

Fail

`Fail` 状态表示状态机未正确执行。这是状态机的结束状态，每个状态机定义都必须包含此状态。

```
{
    "Type": "Fail"
```

```
}
```

Succeed

Succeed 状态表示状态机已正确执行。这是状态机的结束状态，每个状态机定义都必须包含此状态。

```
{
    "Type": "Succeed"
}
```

状态机上下文

状态机上下文是一个只读 JSON 文档，其中包含在执行期间可供状态机使用的数据。状态机上下文只能从状态机访问，并且包含决定测试流程的信息。例如，您可以使用 userdata.json 文件中测试运行器配置的信息来确定是否需要运行特定的测试。

状态机上下文使用以下格式：

```
{
    "pool": {
        <device-json-pool-element>
    },
    "userData": {
        <userdata-json-content>
    },
    "config": {
        <config-json-content>
    },
    "suiteFailed": true | false,
    "specificTestGroups": [
        "<group-id>"
    ],
    "specificTestCases": [
        "<test-id>"
    ],
    "hasExecutionErrors": true
}
```

pool

有关为测试运行选择的设备池的信息。对于选定的设备池，此信息将从 device.json 文件中定义的相应顶级设备池阵列元素中检索。

userData

userdata.json 文件中的信息

config

信息会锁定 config.json 文件。

suiteFailed

该值在状态机启动时设置为 false。如果测试组在某种 RunTask 状态下失败，则在状态机执行的剩余时间内，该值将设置为 true。

specificTestGroups

如果测试运行者选择特定的测试组而不是整个测试套件来运行，则会创建此密钥并包含特定测试组的列表 IDs。

specificTestCases

如果测试运行器选择要运行的特定测试用例而不是整个测试套件，则会创建此密钥并包含特定测试用例的列表 IDs。

hasExecutionErrors

状态机启动时不退出。如果任何状态遇到执行错误，则会在状态机执行的剩余时间内创建此变量并将其设置为 true。

您可以使用 JSONPath 符号查询上下文。状态定义中 JSONPath查询的语法是{{$.*query*}}。在某些状态下，您可以将 JSONPath查询用作占位符字符串。IDT 将占位符字符串替换为来自上下文的已评估 JSONPath 查询的值。您可以对占位符使用以下值：

- RunTask状态的 TestCases 值。
- Expression 值 Choice 状态。

当您访问状态机上下文中的数据时，请确保满足以下条件：

- JSON 路径必须以 $. 开头
- 每个值的计算结果必须为字符串、数字或布尔值。

有关使用 JSONPath 符号从上下文访问数据的更多信息，请参阅使用 IDT 上下文。

执行错误

执行错误是状态机在执行状态时遇到的状态机定义中的错误。IDT 在 test_manager.log 文件中记录有关每个错误的信息，并将日志消息流式传输到控制台。

您可以使用以下方法来处理执行错误：

- 在状态定义中添加一个Catch 数据块。
- 在状态机上下文中检查 hasExecutionErrors值。

捕获

要使用 Catch，请将以下内容添加到您的状态定义中：

```
"Catch": [
    {
        "ErrorEquals": [
            "<error-type>"
        ]
        "Next": "<state-name>"
    }
]
```

包含值的所有字段都为必填字段，如下所述：

Catch.ErrorEquals

要捕获的错误类型的数组。如果执行错误与其中一个指定值匹配，则状态机将转换为 Catch.Next 中指定的状态。有关其产生的错误类型的信息，请参阅每个状态定义。

Catch.Next

当前状态遇到与 Catch.ErrorEquals 中指定值之一相匹配的执行错误时，设置要过渡到的下一个状态。

捕获数据块按顺序处理，直到匹配。如果没有错误与捕获数据块中列出的错误相匹配，则状态机将继续执行。由于执行错误是由状态定义不正确造成的，因此我们建议您在状态遇到执行错误时过渡到失败状态。

hasExecutionError

当某些状态遇到执行错误时，除了发出错误外，它们还会在状态机上下文中将 hasExecutionError 值设置为 true。您可以使用此值来检测何时发生错误，然后使用 Choice 状态将状态机过渡到 Fail 状态。

该方式具有以下特征。

- 状态机不以分配给 hasExecutionError 的任何值启动，并且在特定状态设置该值之前，该值不可用。这意味着必须将访问此值的 Choice 状态的 FallthroughOnError 明确设置为 false，以防止在没有发生执行错误时状态机停止。
- 一旦将其设置为 true，hasExecutionError 就永远不会设置为 false 或将其从上下文中删除。这意味着该值仅在首次设置为 true 时才有用，并且对于所有后续状态，它不会提供有意义的值。
- hasExecutionError 值与处于 Parallel 状态的所有分支状态机共享，这可能会导致意想不到的结果，具体取决于访问该值的顺序。

由于这些特性，如果您可以改用捕获数据块，我们不建议您使用此方法。

示例状态机

本部分提供了一些状态机配置示例。

示例

- 状态机示例：运行单个测试组
- 状态机示例：运行用户选择的测试组
- 状态机示例：使用产品功能运行单个测试组
- 状态机示例：并行运行两个测试组

状态机示例：运行单个测试组

该状态机：

- 运行带有 id GroupA 的测试组，该组必须以 group.json 文件形式存在于套件中。
- 检查是否存在执行错误，如果发现任何错误，则过渡到 Fail。
- 生成报告，如果没有错误，则过渡到 Succeed，否则过渡到 Fail。

```
{
```

```
    "Comment": "Runs a single group and then generates a report.",
    "StartAt": "RunGroupA",
    "States": {
        "RunGroupA": {
            "Type": "RunTask",
            "Next": "Report",
            "TestGroup": "GroupA",
            "Catch": [
                {
                    "ErrorEquals": [
                        "RunTaskError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "Report": {
            "Type": "Report",
            "Next": "Succeed",
            "Catch": [
                {
                    "ErrorEquals": [
                        "ReportError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "Succeed": {
            "Type": "Succeed"
        },
        "Fail": {
            "Type": "Fail"
        }
    }
}
```

状态机示例：运行用户选择的测试组

该状态机：

- 检查测试运行器是否选择了特定的测试组。状态机不检查特定的测试用例，因为如果不选择测试组，测试运行器就无法选择测试用例。

- 如果选择了测试组：
  - 在选定的测试组中运行测试用例。为此，状态机不会明确指定处于 RunTask 状态中的任何测试组或测试用例。
  - 运行所有测试后生成报告并退出。
- 如果未选择测试组：
  - 在测试组 GroupA 中运行测试。
  - 生成报告并退出。

```
{
    "Comment": "Runs specific groups if the test runner chose to do that, otherwise
 runs GroupA.",
    "StartAt": "SpecificGroupsCheck",
    "States": {
        "SpecificGroupsCheck": {
            "Type": "Choice",
            "Default": "RunGroupA",
            "FallthroughOnError": true,
            "Choices": [
                {
                    "Expression": "{{$.specificTestGroups[0]}} != ''",
                    "Next": "RunSpecificGroups"
                }
            ]
        },
        "RunSpecificGroups": {
            "Type": "RunTask",
            "Next": "Report",
            "Catch": [
                {
                    "ErrorEquals": [
                        "RunTaskError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "RunGroupA": {
            "Type": "RunTask",
            "Next": "Report",
            "TestGroup": "GroupA",
```

```
            "Catch": [
                {
                    "ErrorEquals": [
                        "RunTaskError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "Report": {
            "Type": "Report",
            "Next": "Succeed",
            "Catch": [
                {
                    "ErrorEquals": [
                        "ReportError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "Succeed": {
            "Type": "Succeed"
        },
        "Fail": {
            "Type": "Fail"
        }
    }
}
```

状态机示例：使用产品功能运行单个测试组

该状态机：

- 运行测试组 GroupA。
- 检查是否存在执行错误，如果发现任何错误，则过渡到 Fail。
- 将 FeatureThatDependsOnGroupA 功能添加到 awsiotdevicetester_report.xml 文件中：
  - 如果 GroupA 通过，则该功能将设置为 supported。
  - 报告中未将该功能标记为可选。
- 生成报告，如果没有错误，则过渡到 Succeed，否则过渡到 Fail

```
{
    "Comment": "Runs GroupA and adds product features based on GroupA",
    "StartAt": "RunGroupA",
    "States": {
        "RunGroupA": {
            "Type": "RunTask",
            "Next": "AddProductFeatures",
            "TestGroup": "GroupA",
            "ResultVar": "GroupA_passed",
            "Catch": [
                {
                    "ErrorEquals": [
                        "RunTaskError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "AddProductFeatures": {
            "Type": "AddProductFeatures",
            "Next": "Report",
            "Features": [
                {
                    "Feature": "FeatureThatDependsOnGroupA",
                    "Groups": [
                        "GroupA"
                    ],
                    "IsRequired": true
                }
            ]
        },
        "Report": {
            "Type": "Report",
            "Next": "Succeed",
            "Catch": [
                {
                    "ErrorEquals": [
                        "ReportError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
```

```
        "Succeed": {
            "Type": "Succeed"
        },
        "Fail": {
            "Type": "Fail"
        }
    }
}
```

状态机示例：并行运行两个测试组

该状态机：

- 并行运行 GroupA 和 GroupB 测试组。通过分支状态机中的 RunTask 状态存储在上下文中的 ResultVar 变量可供 AddProductFeatures 状态使用。
- 检查是否存在执行错误，如果发现任何错误，则过渡到 Fail。此状态机不使用 Catch 数据块，因为该方法不会检测分支状态机中的执行错误。
- 根据通过的群组向 awsiotdevicetester_report.xml 文件添加功能
  - 如果 GroupA 通过，则该功能将设置为 supported。
  - 报告中未将该功能标记为可选。
- 生成报告，如果没有错误，则过渡到 Succeed，否则过渡到 Fail

如果在设备池中配置了两个设备，则 GroupA 和 GroupB 可以同时运行。但是，如果其中一个 GroupA 或 GroupB 有多个测试，则两个设备都可能分配给这些测试。如果只配置了一台设备，则测试组将按顺序运行。

```
{
    "Comment": "Runs GroupA and GroupB in parallel",
    "StartAt": "RunGroupAAndB",
    "States": {
        "RunGroupAAndB": {
            "Type": "Parallel",
            "Next": "CheckForErrors",
            "Branches": [
                {
                    "Comment": "Run GroupA state machine",
                    "StartAt": "RunGroupA",
                    "States": {
                        "RunGroupA": {
                            "Type": "RunTask",
```

```
                    "Next": "Succeed",
                    "TestGroup": "GroupA",
                    "ResultVar": "GroupA_passed",
                    "Catch": [
                        {
                            "ErrorEquals": [
                                "RunTaskError"
                            ],
                            "Next": "Fail"
                        }
                    ]
                },
                "Succeed": {
                    "Type": "Succeed"
                },
                "Fail": {
                    "Type": "Fail"
                }
            }
        },
        {
            "Comment": "Run GroupB state machine",
            "StartAt": "RunGroupB",
            "States": {
                "RunGroupA": {
                    "Type": "RunTask",
                    "Next": "Succeed",
                    "TestGroup": "GroupB",
                    "ResultVar": "GroupB_passed",
                    "Catch": [
                        {
                            "ErrorEquals": [
                                "RunTaskError"
                            ],
                            "Next": "Fail"
                        }
                    ]
                },
                "Succeed": {
                    "Type": "Succeed"
                },
                "Fail": {
                    "Type": "Fail"
                }
```

```
                }
            }
        ]
    },
    "CheckForErrors": {
        "Type": "Choice",
        "Default": "AddProductFeatures",
        "FallthroughOnError": true,
        "Choices": [
            {
                "Expression": "{{$.hasExecutionErrors}} == true",
                "Next": "Fail"
            }
        ]
    },
    "AddProductFeatures": {
        "Type": "AddProductFeatures",
        "Next": "Report",
        "Features": [
            {
                "Feature": "FeatureThatDependsOnGroupA",
                "Groups": [
                    "GroupA"
                ],
                "IsRequired": true
            },
            {
                "Feature": "FeatureThatDependsOnGroupB",
                "Groups": [
                    "GroupB"
                ],
                "IsRequired": true
            }
        ]
    },
    "Report": {
        "Type": "Report",
        "Next": "Succeed",
        "Catch": [
            {
                "ErrorEquals": [
                    "ReportError"
                ],
                "Next": "Fail"
```

```
                }
            ]
        },
        "Succeed": {
            "Type": "Succeed"
        },
        "Fail": {
            "Type": "Fail"
        }
    }
}
```

创建 IDT 测试用例可执行文件

您可以通过以下方式，创建测试用例可执行文件并将其放在测试套件文件夹中：

- 如果测试套件使用 `test.json` 文件中的参数或环境变量来确定要运行哪些测试，您可以为整个测试套件创建一个测试用例可执行文件，也可为测试套件中的每个测试组分别创建一个测试可执行文件。
- 对于要基于指定命令运行特定测试的测试套件，您可以为测试套件中的每个测试用例创建一个测试用例可执行文件。

作为测试编写者，您可以决定哪一种方法适合您的用例，并相应地构建测试用例可执行文件。请确保在每个 `test.json` 文件中提供正确的测试用例可执行文件路径，并且指定的可执行文件能够正确地运行。

当所有设备都准备好运行测试用例时，IDT 将会读取以下文件：

- 所选测试用例的 `test.json` 决定了要启动的进程以及要设置的环境变量。
- 测试套件的 `suite.json` 决定了要设置的环境变量。

IDT 根据 `test.json` 文件中指定的命令和参数启动所需的测试可执行文件进程，并将必要的环境变量传递给这个进程。

使用 IDT 客户端软件开发工具包

IDT 客户端 SDKs 允许您使用 API 命令简化在测试可执行文件中编写测试逻辑的方式，这些命令可用于与 IDT 和被测设备交互。IDT 目前提供以下内容： SDKs

- 适用于 Python 的 IDT 客户端软件开发工具包

- 适用于 Go 的 IDT 客户端软件开发工具包
- 适用于 Java 的 IDT 客户端软件开发工具包

SDKs 它们位于*<device-tester-extract-location>*/sdks文件夹中。创建新的测试用例可执行文件时，您必须将要使用的软件开发工具包复制到包含测试用例可执行文件的文件夹中，并在代码中引用这个软件开发工具包。本节简要描述了可以在测试用例可执行文件中使用的 API 命令。

本节内容

- 设备交互
- IDT 交互
- 主机交互

设备交互

通过运行以下命令，您无需实施任何其他设备交互和连接管理功能，即可与被测设备通信。

`ExecuteOnDevice`

允许测试套件在支持 SSH 或 Docker shell 连接的设备上运行 shell 命令。

`CopyToDevice`

允许测试套件将本地文件从运行 IDT 的主机复制到支持 SSH 或 Docker shell 连接的设备上的指定位置。

`ReadFromDevice`

允许测试套件从支持 UART 连接的设备的串行端口进行读取。

> **Note**
>
> 由于 IDT 不管理使用上下文中设备访问信息与设备建立的直接连接，因此我们建议在测试用例可执行文件中使用这些设备交互 API 命令。但是，如果这些命令不满足测试用例要求，您可以从 IDT 环境中检索设备访问信息，并使用该信息从测试套件建立与设备的直接连接。
> 要建立直接连接，请分别在 `device.connectivity` 和 `resource.devices.connectivity` 字段中检索被测设备和资源设备的信息。有关使用 IDT 上下文的更多信息，请参阅 使用 IDT 上下文。

IDT 交互

您可以通过运行以下命令，使测试套件与 IDT 进行通信。

`PollForNotifications`

允许测试套件检查来自 IDT 的通知。

`GetContextValue` 和 **`GetContextString`**

允许测试套件从 IDT 上下文中检索值。有关更多信息，请参阅 使用 IDT 上下文。

`SendResult`

允许测试套件向 IDT 报告测试用例结果。此命令必须在测试套件中每个测试用例结束时进行调用。

主机交互

您可以通过运行以下命令，使测试套件与主机进行通信。

`PollForNotifications`

允许测试套件检查来自 IDT 的通知。

`GetContextValue` 和 **`GetContextString`**

允许测试套件从 IDT 上下文中检索值。有关更多信息，请参阅 使用 IDT 上下文。

`ExecuteOnHost`

允许测试套件在本地计算机上运行命令，并使 IDT 能够管理测试用例可执行文件的生命周期。

启用 IDT CLI 命令

IDT CLI `run-suite` 命令提供了多个选项，允许测试运行器自定义测试执行。要允许测试运行器使用这些选项来运行您的自定义测试套件，您需要实施对 IDT CLI 的支持。如果您不实施支持，测试运行器仍然可以运行测试，但是某些 CLI 选项将无法正常运作。为了提供理想的客户体验，我们建议您在 IDT CLI 中实施对以下 `run-suite` 命令参数的支持：

`timeout-multiplier`

指定一个大于 1.0 的值，该值将应用于测试运行期间的所有超时。

测试运行器可以使用此参数来延长他们想要运行的测试用例的超时时间。测试运行器在其 `run-suite` 命令中指定此参数后，IDT 会使用它来计算 IDT_TEST_TIMEOUT 环境变量的值，并在 IDT 上下文中设置 `config.timeoutMultiplier` 字段。要支持这个参数，您必须执行以下操作：

- 读取 IDT_TEST_TIMEOUT 环境变量来获取正确计算的超时值，而不是直接使用 `test.json` 文件中的超时值。
- 从 IDT 上下文中检索 `config.timeoutMultiplier` 值，并将其应用于长时间运行的超时。

有关因超时事件而提前退出的更多信息，请参阅指定退出行为。

`stop-on-first-failure`

指定 IDT 在遇到故障时应当停止运行所有测试。

测试运行器在其 `run-suite` 命令中指定此参数后，IDT 会在遇到故障时立即停止运行测试。但是，如果测试用例是并行运行的，可能会导致意外的结果。要实施支持，请确保当 IDT 遇到此事件时，您的测试逻辑会指示所有运行中的测试用例停止运行、清理临时资源并向 IDT 报告测试结果。有关失败时提早退出的更多信息，请参阅 指定退出行为。

`group-id` 和 `test-id`

指定 IDT 应当仅运行选定的测试组或测试用例。

测试运行器可以在 `run-suite` 命令中使用这些参数来指定以下测试执行行为：

- 在指定的测试组中运行所有测试。
- 运行一组来自指定测试组的测试。

为了支持这些参数，测试套件的状态机必须在状态机中包含一组特定的 `RunTask` 和 `Choice` 状态。如果您不使用自定义状态机，默认 IDT 状态机包含了所需的状态，您无需采取其他操作。但是，如果您要使用自定义状态机，则可以将 状态机示例：运行用户选择的测试组 作为示例，在状态机中添加所需的状态。

有关 IDT CLI 命令的更多信息，请参阅 调试和运行自定义测试套件。

写入事件日志

在测试运行期间，您可以通过向 `stdout` 和 `stderr` 发送数据，将事件日志和错误消息写入控制台。有关控制台消息格式的信息，请参阅 控制台消息格式。

IDT 运行完测试套件后，也可以在 *`<devicetester-extract-location>`*`/results/`*`<execution-id>`*`/logs` 文件夹下的 `test_manager.log` 文件中找到此信息。

您可以将每个测试用例配置为将其测试运行的日志（包括来自被测设备的日志）写入 *<device-tester-extract-location>*/results/*execution-id*/logs 文件夹下的 *<group-id>*_*<test-id>* 文件中。为此，请使用 testData.logFilePath 查询从 IDT 上下文中检索日志文件的路径，在该路径上创建一个文件，然后将所需的内容写入这个文件中。IDT 会根据正在运行的测试用例来自动更新路径。如果您选择不为测试用例创建日志文件，则不会为该测试用例生成任何文件。

此外，也可将文本可执行文件设置为按需在 *<device-tester-extract-location>*/logs 文件夹中创建其他日志文件。我们建议您为日志文件名指定唯一的前缀，以避免您的文件被覆盖。

向 IDT 报告结果

IDT 将测试结果写入 awsiotdevicetester_report.xml 和 *suite-name*_report.xml 文件中。这些报告文件位于 *<device-tester-extract-location>*/results/*<execution-id>*/ 下。两个报告都捕获测试套件执行的结果。有关 IDT 用于这些报告的架构的更多信息，请参阅 查看 IDT 测试结果和日志

要在 *suite-name*_report.xml 文件中填充内容，您必须在测试执行结束前使用 SendResult 命令向 IDT 报告测试结果。如果 IDT 找不到测试结果，则会针对该测试用例发出错误。以下 Python 摘录显示了用于向 IDT 发送测试结果的命令：

```
request-variable = SendResultRequest(TestResult(result))
client.send_result(request-variable)
```

如果您不通过 API 报告结果，IDT 会在测试构件文件夹中查找测试结果。此文件夹的路径存储在 IDT 上下文中的 testData.testArtifactsPath 文件中。在此文件夹中，IDT 将找到的第一个按字母顺序排序的 XML 文件用作测试结果。

如果您的测试逻辑生成 JUnit XML 结果，则可以将测试结果写入 artifacts 文件夹中的 XML 文件中，直接向 IDT 提供结果，而不必解析结果然后使用 API 将其提交给 IDT。

如果使用此方法，请确保您的测试逻辑准确汇总了测试结果，并将结果文件格式化为与 *suite-name*_report.xml 文件相同的格式。IDT 不会对您提供的数据进行任何验证，但以下情况除外：

- IDT 会忽略 testsuites 标签的所有属性。相反，它会从其他报告的测试组结果计算标签属性。
- testsuites 中必须至少存在一个 testsuite 标签。

IDT 对所有测试用例使用相同的构件文件夹，并且不会在两次测试运行之间删除结果文件。因此，如果 IDT 读取了错误的文件，此方法也可能会导致错误的报告。我们建议您在所有测试用例中对生成的 XML 结果文件使用相同的名称，以覆盖每个测试用例的结果，并将正确的结果提供给 IDT 使用。您可

以在测试套件中使用混合方法来进行报告，即：对某些测试用例使用 XML 结果文件，同时通过 API 提交其他测试用例的结果。但是，我们不建议采用这种方法。

指定退出行为

将您的文本可执行文件配置为始终以退出代码 0 退出，即使测试用例报告失败或错误结果时也不例外。仅使用非零退出代码来表示测试用例未运行，或者表示测试用例可执行文件无法向 IDT 传达任何结果。当 IDT 收到非零退出代码时，这表示测试用例遇到了阻碍其运行的错误。

在以下事件中，IDT 可能会请求或期望测试用例在完成之前停止运行。您可以使用此信息来配置测试用例可执行文件，以检测测试用例中的每个事件：

超时

当测试用例的运行时间超过 `test.json` 文件中指定的超时值时发生。如果测试运行器使用 `timeout-multiplier` 参数来指定超时乘数，则 IDT 会使用该乘数来计算超时值。

要检测此事件，请使用 IDT_TEST_TIMEOUT 环境变量。当测试运行器启动测试时，IDT 会将 IDT_TEST_TIMEOUT 环境变量的值设置为计算得出的超时值（以秒为单位），并将该变量传递给测试用例可执行文件。您可以读取变量值来设置相应的计时器。

中断

在测试运行器中断 IDT 时发生。例如，按下 Ctrl+C。

终端会将信号传播到所有子进程，因此您只需在测试用例中配置信号处理程序来检测中断信号即可。

或者，也可以定期轮询 API 以检查 `PollForNotifications` API 响应中 `CancellationRequested` 布尔值的值。当 IDT 收到中断信号时，它会将 `CancellationRequested` 布尔值设置为 `true`。

首次失败时停止

发生的条件为：与当前测试用例并行运行的测试用例失败，并且测试运行器使用了 `stop-on-first-failure` 参数来指定 IDT 应当在遇到任何失败时停止运行。

要检测此事件，您可以定期轮询 API 以检查 `PollForNotifications` API 响应中 `CancellationRequested` 布尔值的值。如果 IDT 遇到故障并且已配置为在首次失败时停止，它会将 `CancellationRequested` 布尔值设置为 `true`。

发生其中任何一个事件时，IDT 会等待 5 分钟，让当前正在运行的测试用例完成运行。如果所有正在运行的测试用例未能在 5 分钟内退出，IDT 会强制停止它们的每个进程。如果 IDT 没有在进程结束之前

收到测试结果，它会将测试用例标记为已超时。作为一种最佳实践，您应确保测试用例在遇到其中一个事件时执行以下操作：

1. 停止运行正常的测试逻辑。
2. 清理所有的临时资源，例如被测设备上的测试构件。
3. 向 IDT 报告测试结果，例如测试失败或错误。
4. 退出。

使用 IDT 上下文

IDT 运行测试套件时，测试套件可以访问一组数据，这些数据可用于确定每个测试的运行方式。这些数据被称为 IDT 上下文。例如，测试运行器在 `userdata.json` 文件中提供的用户数据配置可用于 IDT 环境中的测试套件。

IDT 上下文可以被视为只读 JSON 文档。测试套件可以使用对象、阵列、数字等标准 JSON 数据类型从上下文中检索数据并将数据写入上下文。

上下文架构

IDT 上下文采用以下格式：

```
{
    "config": {
        <config-json-content>
        "timeoutMultiplier": timeout-multiplier,
        "idtRootPath": <path/to/IDT/root>
    },
    "device": {
        <device-json-device-element>
    },
    "devicePool": {
        <device-json-pool-element>
    },
    "resource": {
        "devices": [
            {
                <resource-json-device-element>
                "name": "<resource-name>"
            }
        ]
    },
```

```
    "testData": {
        "awsCredentials": {
            "awsAccessKeyId": "<access-key-id>",
            "awsSecretAccessKey": "<secret-access-key>",
            "awsSessionToken": "<session-token>"
        },
        "logFilePath": "/path/to/log/file"
    },
    "userData": {
        <userdata-json-content>
    }
}
```

config

config.json文件中的信息。config 字段还包含以下附加字段：

config.timeoutMultiplier

测试套件使用的任何超时值的乘数。此值由 IDT CLI 中的测试运行器指定。默认值为 1。

config.idRootPath

此值是配置 userdata.json 文件时 IDT 的绝对路径值的占位符。编译命令和刷写命令会使用此值。

device

有关为测试运行选择的设备的信息。此信息等同于所选设备device.json文件中的 devices 阵列元素。

devicePool

有关为测试运行选择的设备池的信息。此信息等同于 device.json 文件中为所选设备池定义的顶级设备池阵列元素。

resource

resource.json 文件中有关资源设备的信息。

resource.devices

此信息等同于 resource.json 文件中定义的 devices 阵列。每个 devices 元素都包括以下附加字段：

resource.device.name

资源的名称。此值设置为 test.json 文件中的 requiredResource.name 值。

testData.awsCredentials

测试用于连接到 AWS 云的 AWS 凭据。此信息是从 config.json 文件中获得的。

testData.logFilePath

测试用例写入日志消息的日志文件的路径。如果日志文件不存在，则会创建。

userData

userdata.json文件中由测试运行器提供的信息。

在上下文中访问数据

您可以使用配置文件中的 JSONPath 符号以及带有和的文本可执行文件中的符号来查询上下文GetContextString APIs。GetContextValue用于访问 IDT 上下文的 JSONPath 字符串的语法各不相同，如下所示：

- 在 suite.json 和test.json 中，使用 {{*query*}}。也就是说，不要使用根元素 $. 开始表达式。
- 在 statemachine.json 中，使用 {{$.*query*}}。
- 在 API 命令中，根据命令的不同，您可以使用 *query* 或 {{$.*query*}}。有关更多信息，请参阅中的内联文档 SDKs。

下表描述了典型的 foobar JSONPath 表达式中的运算符：

运算符	描述
$	根元素。由于 IDT 的顶级上下文值是一个对象，因此，您通常会使用 $. 来启动查询。
.childName	使用来自对象的名称 childName 访问子元素。如果应用到数组，则生成一个新数组，并将此运算符应用到每个元素。该元素名称区分大小写。例如，访问 config 对象中 awsRegion 值的查询是 $.config.awsRegion 。
[start:end]	筛选数组中的元素，检索从 start 索引开始直到 end 索引的项目（包括首尾）。

运算符	描述
`[index1, index2, ... , indexN]`	筛选数组中的元素，仅从指定索引检索项目。
`[?(expr)]`	使用 `expr` 表达式筛选数组中的元素。该表达式的计算结果必须为布尔值。

要创建筛选表达式，请使用以下语法：

```
<jsonpath> | <value> operator <jsonpath> | <value>
```

在此语法中：

- `jsonpath`是 JSONPath 使用标准 JSON 语法的。
- `value` 是使用标准 JSON 语法的任何自定义值。
- `operator` 下列运算符之一：
  - < (小于)
  - <= (小于或等于)
  - == (等于)

    如果表达式中的 JSONPath 或值是数组、布尔值或对象值，则这是您可以使用的唯一支持的二进制运算符。
  - >= (大于或等于)
  - > (大于)
  - =~ (正则表达式匹配)。要在筛选表达式中使用此运算符，表达式左侧的 JSONPath 或值必须计算为字符串，而右侧的值必须是遵循该RE2语法的模式值。

您可以使用 {{*query*}} 形式的 JSONPath 查询作为文件和字段中`args`以及`test.json`文件中`environmentVariables`字段中的占位符字符串。`environmentVariables` `suite.json`IDT 执行上下文查找，并使用查询的评估值填充字段。例如，在 `suite.json` 文件中，您可以使用占位符字符串来指定随每个测试用例而变化的环境变量值，IDT 将使用每个测试用例的正确值填充环境变量。但是，当您在 `test.json` 和 `suite.json` 文件中使用占位符字符串时，以下注意事项适用于您的查询：

- 查询中每次出现的 `devicePool` 密钥都必须全部使用小写字母。也就是说，改用 `devicepool`。

- 对于数组，只能使用字符串数组。此外，数组使用非标准 item1, item2,...,itemN 格式。如果数组仅包含一个元素，则将其序列化为 item，使其与字符串字段没有区别。
- 不能使用占位符从上下文中检索对象。

出于这些考虑，我们建议您尽可能使用 API 来访问测试逻辑中的上下文，而不是 test.json 和 suite.json 文件中的占位符字符串。但是，在某些情况下，使用 JSONPath 占位符检索要设置为环境变量的单个字符串可能会更方便。

为测试运行器配置设置

要运行自定义测试套件，测试运行器必须根据他们要运行的测试套件配置设置。设置是根据位于该 *<device-tester-extract-location>*/configs/ 文件夹中的配置文件模板指定的。如果需要，测试运行者还必须设置 IDT 用于连接 AWS 云的 AWS 凭据。

作为测试编写者，您需要配置这些文件来调试您的测试套件。您必须向测试运行器提供说明，以便他们可以根据需要配置以下设置来运行您的测试套件。

配置 device.json

device.json 文件包含有关运行测试的设备的信息（例如，IP 地址、登录信息、操作系统和 CPU 架构）。

测试运行器可以使用位于 *<device-tester-extract-location>*/configs/ 文件夹中的以下模板 device.json 文件来提供此信息。

```
[
    {
        "id": "<pool-id>",
        "sku": "<pool-sku>",
        "features": [
            {
                "name": "<feature-name>",
                "value": "<feature-value>",
                "configs": [
                    {
                        "name": "<config-name>",
                        "value": "<config-value>"
                    }
                ],
            }
```

```
        ],
        "devices": [
            {
                "id": "<device-id>",
                "pairedResource": "<device-id>", //used for no-op protocol
                "connectivity": {
                    "protocol": "ssh | uart | docker | no-op",
                    // ssh
                    "ip": "<ip-address>",
                    "port": <port-number>,
                    "publicKeyPath": "<public-key-path>",
                    "auth": {
                        "method": "pki | password",
                        "credentials": {
                            "user": "<user-name>",
                            // pki
                            "privKeyPath": "/path/to/private/key",

                            // password
                            "password": "<password>",
                        }
                    },

                    // uart
                    "serialPort": "<serial-port>",

                    // docker
                    "containerId": "<container-id>",
                    "containerUser": "<container-user-name>",
                }
            }
        ]
    }
]
```

包含值的所有字段都为必填字段，如下所述：

id

一个用户定义的字母数字 ID，用于唯一地标识称作设备池的设备集合。属于池的设备必须具有相同的硬件。运行一组测试时，池中的设备将用于对工作负载进行并行化处理。多个设备用于运行不同测试。

sku

唯一标识所测试设备的字母数字值。该 SKU 用于跟踪符合条件的设备。

> **Note**
>
> 如果您想在 AWS 合作伙伴设备目录中发布您的主板，则在此处指定的 SKU 必须与您在发布过程中使用的 SKU 相匹配。

features

可选。包含设备支持的功能的数组。设备功能是您在测试套件中配置的用户定义值。您必须向测试运行器提供有关要包含在 `device.json` 文件中的功能名称和值的信息。例如，如果您想测试一台可充当其他设备的 MQTT 服务器的设备，则可以配置测试逻辑来验证名为 `MQTT_QoS` 的功能的特定支持级别。测试运行器提供此功能名称，并将该功能值设置为其设备支持的 QoS 级别。您可以通过查询从 IDT 上下文中检索所提供的信息，也可以通过`devicePool.features`查询从状态机上下文中检索所`pool.features`提供的信息。

features.name

功能的名称。

features.value

支持的功能值。

features.configs

该功能的配置设置（如果需要）。

features.config.name

配置设置的名称。

features.config.value

支持的设置值。

devices

池中待测试的设备阵列。至少需要选择一个设备。

devices.id

用户定义的测试的设备的唯一标识符。

devices.pairedResource

用户定义的资源设备的唯一标识符。使用 no-op 连接协议测试设备时，此值是必需的。

connectivity.protocol

用于与此设备通信的通信协议。池中的每台设备都必须使用相同的协议。

当前，支持的值只包括适用于物理设备的 ssh 和 uart、适用于 Docker 容器的 docker 以及适用于设备的 no-op，这些设备与 IDT 主机没有直接连接，但需要将资源设备作为物理中间件才能与主机通信。

对于无操作设备，您可以在 devices.pairedResource 中配置资源设备 ID。您还必须在 resource.json 文件中指定此 ID。已配对的设备必须是与待测设备进行物理配对的设备。IDT 识别并连接到配对的资源设备后，根据 test.json 文件中描述的功能，IDT 将不会连接到其他资源设备。

connectivity.ip

测试的设备 IP 地址。

此属性仅在 connectivity.protocol 设置为 ssh 时适用。

connectivity.port

可选。用于 SSH 连接的端口号。

默认值为 22。

此属性仅在 connectivity.protocol 设置为 ssh 时适用。

connectivity.publicKeyPath

可选。用于验证待测设备连接的公有密钥的完整路径。如果指定 publicKeyPath，IDT 会在与待测设备建立 SSH 连接时验证设备的公有密钥。如果未指定此值，IDT 将创建 SSH 连接，但不验证设备的公有密钥。

我们强烈建议您指定公有密钥的路径，并使用安全的方法来获取此公有密钥。对于基于标准命令行的 SSH 客户端，known_hosts 文件中提供了公有密钥。如果您指定单独的公有密钥文件，则该文件必须使用与 known_hosts 文件相同的格式，即 ip-address key-type public-key。

connectivity.auth

连接的身份验证信息。

此属性仅在 connectivity.protocol 设置为 ssh 时适用。

connectivity.auth.method

用于通过给定的连接协议访问设备的身份验证方法。

支持的值为：

- pki
- password

connectivity.auth.credentials

用于身份验证的凭证。

connectivity.auth.credentials.password

该密码用于登录到正在测试的设备。

此值仅在 connectivity.auth.method 设置为 password 时适用。

connectivity.auth.credentials.privKeyPath

用于登录所测试设备的私有密钥的完整路径。

此值仅在 connectivity.auth.method 设置为 pki 时适用。

connectivity.auth.credentials.user

用于登录所测试设备的用户名。

connectivity.serialPort

可选。设备所连接的串行端口。

此属性仅在 connectivity.protocol 设置为 uart 时适用。

connectivity.containerId

所测试的 Docker 容器的容器 ID 或名称。

此属性仅在 connectivity.protocol 设置为 docker 时适用。

connectivity.containerUser

可选。容器内用户对用户的名称。默认值为 Dockerfile 中提供的用户。

默认值为 22。

此属性仅在 connectivity.protocol 设置为 docker 时适用。

> **Note**
>
> 要检查测试运行器是否为测试配置了错误的设备连接，可以从状态机上下文中检索 pool.Devices[0].Connectivity.Protocol，并将其与 Choice 状态下的预期值进行比较。如果使用的协议不正确，则使用 LogMessage 状态打印一条消息并过渡到 Fail 状态。
> 或者，您可以使用错误处理代码来报告错误设备类型的测试失败。

（可选）配置 userdata.json

userdata.json 文件包含测试套件所需但 device.json 文件中未指定的任何其他信息。此文件的格式取决于测试套件中定义的userdata_scheme.json文件。如果您是测试编写者，请务必将此信息提供给将运行您编写的测试套件的用户。

（可选）配置 resource.json

resource.json 文件包含有关将用作资源设备的所有设备的信息。资源设备是测试被测设备的某些功能所需的设备。例如，要测试设备的蓝牙功能，您可以使用资源设备来测试您的设备能否成功连接到该设备。资源设备是可选的，您可以根据需要任意数量的资源设备。作为测试编写者，您可以使用test.json 文件来定义测试所需的资源设备功能。然后，测试运行器使用 resource.json 文件提供具有所需功能的资源设备池。请务必将此信息提供给将运行您编写的测试套件的用户。

测试运行器可以使用位于 *<device-tester-extract-location>*/configs/ 文件夹中的以下模板 resource.json 文件来提供此信息。

```
[
    {
        "id": "<pool-id>",
        "features": [
            {
                "name": "<feature-name>",
                "version": "<feature-value>",
                "jobSlots": <job-slots>
            }
        ],
        "devices": [
            {
                "id": "<device-id>",
```

```
                "connectivity": {
                    "protocol": "ssh | uart | docker",
                    // ssh
                    "ip": "<ip-address>",
                    "port": <port-number>,
                    "publicKeyPath": "<public-key-path>",
                    "auth": {
                        "method": "pki | password",
                        "credentials": {
                            "user": "<user-name>",
                            // pki
                            "privKeyPath": "/path/to/private/key",

                            // password
                            "password": "<password>",
                        }
                    },

                    // uart
                    "serialPort": "<serial-port>",

                    // docker
                    "containerId": "<container-id>",
                    "containerUser": "<container-user-name>",
                }
            }
        ]
    }
]
```

包含值的所有字段都为必填字段，如下所述：

id

一个用户定义的字母数字 ID，用于唯一地标识称作设备池的设备集合。属于池的设备必须具有相同的硬件。运行一组测试时，池中的设备将用于对工作负载进行并行化处理。多个设备用于运行不同测试。

features

可选。包含设备支持的功能的数组。此字段中所需的信息在测试套件的 test.json 文件中定义，这些信息用于确定要运行哪些测试以及如何运行这些测试。如果测试套件不需要任何功能，则此字段不是必填字段。

features.name

功能的名称。

features.version

功能版本。

features.jobSlots

用于指明可以同时使用该设备的测试次数的设置。默认值为 1。

devices

池中待测试的设备阵列。至少需要选择一个设备。

devices.id

用户定义的测试的设备的唯一标识符。

connectivity.protocol

用于与此设备通信的通信协议。池中的每台设备都必须使用相同的协议。

目前，唯一支持的值，对于物理设备为 ssh 和 uart，对于 Docker 容器为 docker。

connectivity.ip

测试的设备 IP 地址。

此属性仅在 connectivity.protocol 设置为 ssh 时适用。

connectivity.port

可选。用于 SSH 连接的端口号。

默认值为 22。

此属性仅在 connectivity.protocol 设置为 ssh 时适用。

connectivity.publicKeyPath

可选。用于验证待测设备连接的公有密钥的完整路径。如果指定 publicKeyPath，IDT 会在与待测设备建立 SSH 连接时验证设备的公有密钥。如果未指定此值，IDT 将创建 SSH 连接，但不验证设备的公有密钥。

我们强烈建议您指定公有密钥的路径，并使用安全的方法来获取此公有密钥。对于基于标准命令行的 SSH 客户端，known_hosts 文件中提供了公有密钥。如果您指定单独的公有密钥文件，

则该文件必须使用与 known_hosts 文件相同的格式，即 ip-address key-type public-key。

connectivity.auth

连接的身份验证信息。

此属性仅在 connectivity.protocol 设置为 ssh 时适用。

connectivity.auth.method

用于通过给定的连接协议访问设备的身份验证方法。

支持的值为：

- pki
- password

connectivity.auth.credentials

用于身份验证的凭证。

connectivity.auth.credentials.password

该密码用于登录到正在测试的设备。

此值仅在 connectivity.auth.method 设置为 password 时适用。

connectivity.auth.credentials.privKeyPath

用于登录所测试设备的私有密钥的完整路径。

此值仅在 connectivity.auth.method 设置为 pki 时适用。

connectivity.auth.credentials.user

用于登录所测试设备的用户名。

connectivity.serialPort

可选。设备所连接的串行端口。

此属性仅在 connectivity.protocol 设置为 uart 时适用。

connectivity.containerId

所测试的 Docker 容器的容器 ID 或名称。

此属性仅在 `connectivity.protocol` 设置为 `docker` 时适用。

`connectivity.containerUser`

可选。容器内用户对用户的名称。默认值为 Dockerfile 中提供的用户。

默认值为 22。

此属性仅在 `connectivity.protocol` 设置为 `docker` 时适用。

（可选）配置 config.json

`config.json` 文件包含 IDT 的配置信息。通常，测试运行者无需修改此文件，除非提供他们的 IDT AWS 用户凭证和 AWS 区域（可选）。如果提供了具有所需权限的 AWS 证书，则 AWS IoT Device Tester 收集使用情况指标并将其提交给 AWS。这是一项可选功能，用来改进 IDT 功能。有关更多信息，请参阅 提交 IDT 使用量指标。

测试运行者可以通过以下方式之一配置其 AWS 凭证：

- 凭证文件

  IDT 使用与 AWS CLI相同的凭证文件。有关更多信息，请参阅配置和凭证文件。

  凭证文件的位置因您使用的操作系统而异：

  - macOS、Linux：`~/.aws/credentials`
  - Windows：`C:\Users\UserName\.aws\credentials`
- 环境变量

  环境变量是由操作系统维护且由系统命令使用的变量。在 SSH 会话期间定义的变量在该会话关闭后不可用。IDT 可使用 `AWS_ACCESS_KEY_ID` 和 `AWS_SECRET_ACCESS_KEY` 环境变量来存储 AWS 凭证

  要在 Linux、macOS 或 Unix 上设置这些变量，请使用 export：

  ```
  export AWS_ACCESS_KEY_ID=<your_access_key_id>
  export AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
  ```

  要在 Windows 上设置这些变量，请使用 set：

  ```
  set AWS_ACCESS_KEY_ID=<your_access_key_id>
  ```

```
set AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

要配置 IDT 的 AWS 凭据，测试运行者需要编辑该*<device-tester-extract-location>*/configs/文件夹中config.json文件中的auth部分。

```
{
    "log": {
        "location": "logs"
    },
    "configFiles": {
        "root": "configs",
        "device": "configs/device.json"
    },
    "testPath": "tests",
    "reportPath": "results",
    "awsRegion": "<region>",
    "auth": {
        "method": "file | environment",
        "credentials": {
            "profile": "<profile-name>"
        }
    }
}
]
```

包含值的所有字段都为必填字段，如下所述：

> **Note**
>
> 此文件中的所有路径都是相对于定义的*<device-tester-extract-location>*。

log.location

中日志文件夹的路径*<device-tester-extract-location>*。

configFiles.root

包含配置文件的文件夹的路径。

configFiles.device

device.json 文件的路径。

testPath

包含测试套件的文件夹的路径。

reportPath

IDT 运行测试套件后将包含测试结果的文件夹的路径。

awsRegion

可选。测试套件将使用的 AWS 区域。如果未设置，则测试套件将使用每个测试套件中指定的默认区域。

auth.method

IDT 用于检索 AWS 凭证的方法。支持的值是用于从凭证文件中检索凭证的 `file`，以及使用环境变量检索凭证的 `environment`。

auth.credentials.profile

要从凭证文件中使用的凭证配置文件。此属性仅在 `auth.method` 设置为 `file` 时适用。

调试和运行自定义测试套件

设置所需的配置后，IDT 就可以运行您的测试套件了。完整测试套件的运行时取决于硬件和测试套件的组成。作为参考，在 Raspberry Pi 3B 上完成完整的 FreeRTOS 资格认证测试套件大约需要 30 分钟。

在编写测试套件时，您可以使用 IDT 在调试模式下运行测试套件，以便在运行代码之前检查代码或将其提供给测试运行器。

在调试模式下运行 IDT

由于测试套件依赖于 IDT 来与设备交互、提供上下文和接收结果，因此如果没有任何 IDT 交互，您就无法在 IDE 中简单地调试测试套件。为此，IDT CLI 提供了 `debug-test-suite` 命令，供您用于在调试模式下运行 IDT。运行以下命令以查看 `debug-test-suite` 的可用选项：

```
devicetester_[linux | mac | win_x86-64] debug-test-suite -h
```

当您在调试模式下运行 IDT 时，IDT 实际上并不启动测试套件或运行编排工具；相反，它会与您的 IDE 交互以响应在 IDE 中运行的测试套件所发出的请求，并将日志输出到控制台。IDT 不会超时，而会等待退出，直到手动中断。在调试模式下，IDT 也不运行测试编排工具，也不会生成任何报告文件。要调试测试套件，您必须使用 IDE 来提供 IDT 通常从配置文件中获得的一些信息。务必提供以下信息：

- 每个测试的环境变量和参数。IDT 不会从 `test.json` 或 `suite.json` 中读取此信息。

- 用于选择资源设备的参数。IDT 不会从 `test.json` 中读取此信息。

要调试您的测试套件，请完成以下步骤：

1. 创建运行测试套件所需的设置配置文件。例如，如果您的测试套件需要 `device.json`、`resource.json` 和 `user data.json`，请确保根据需要来配置所有测试套件。
2. 运行以下命令将 IDT 置于调试模式，然后选择运行测试需要的所有设备。

   ```
   devicetester_[linux | mac | win_x86-64] debug-test-suite [options]
   ```

   运行此命令后，IDT 会等待来自测试套件的请求，然后响应这些请求。IDT 还会生成 IDT 客户端软件开发工具包案例处理所需的环境变量。
3. 在您的 IDE 中，使用 `run` 或 `debug` 配置来执行以下操作：
   a. 设置 IDT 生成的环境变量的值。
   b. 设置您在 `test.json` 和 `suite.json` 文件中指定的任何环境变量或参数的值。
   c. 根据需要设置断点。
4. 在 IDE 中运行测试套件。

   您可以根据需要多次调试和重新运行测试套件。IDT 在调试模式下不会超时。
5. 完成调试后，请中断 IDT 以退出调试模式。

用于运行测试的 IDT CLI 命令

以下小节介绍了 IDT CLI 命令。

IDT v4.0.0

`help`

列出有关指定命令的信息。

`list-groups`

列出给定测试套件中的组。

`list-suites`

列出可用的测试套件。

list-supported-products

列出您的 IDT 版本的受支持产品（在本例中为 FreeRTOS 版本）和当前 IDT 版本的 FreeRTOS 资格认证测试套件版本。

list-test-cases

列出给定测试组中的测试用例。支持以下选项：

- group-id. 要搜索的测试组。此选项是必需的，必须指定单个组。

run-suite

对某个设备池运行一组测试。以下是一些常用的选项：

- suite-id. 要运行的测试套件版本。如果未指定，IDT 将使用 tests 文件夹中的最新版本。
- group-id. 要以逗号分隔的列表形式运行的测试组。如果未指定，IDT 将运行测试套件中的所有测试组。
- test-id. 要以逗号分隔的列表形式运行的测试用例。指定后，group-id 必须指定单个组。
- pool-id. 要测试的设备池。如果您在 device.json 文件中定义了多个设备池，则测试运行器必须指定一个池。
- timeout-multiplier。将 IDT 配置为使用用户定义的乘数来修改 test.json 文件中为测试指定的测试执行超时。
- stop-on-first-failure. 将 IDT 配置为在第一次失败时停止执行。应将此选项与 group-id 结合使用来调试指定的测试组。
- userdata。设置包含运行测试套件所需用户数据信息的文件。只有在测试套件的 suite.json 文件中 userdataRequired 被设置为 true 时，才需要这样做。

有关 run-suite 选项的更多信息，请使用 help 选项：

```
devicetester_[linux | mac | win_x86-64] run-suite -h
```

debug-test-suite

在调试模式下运行测试套件。有关更多信息，请参阅 在调试模式下运行 IDT。

查看 IDT 测试结果和日志

本节介绍 IDT 生成控制台日志和测试报告的格式。

控制台消息格式

AWS IoT Device Tester 使用标准格式在启动测试套件时向控制台打印消息。以下摘录显示了一个由 IDT 生成的控制台消息示例。

```
[INFO] [2000-01-02 03:04:05]: Using suite: MyTestSuite_1.0.0
 executionId=9a52f362-1227-11eb-86c9-8c8590419f30
```

大多数控制台消息都包含以下字段：

time

所记录事件的完整 ISO 8601 时间戳。

level

所记录事件的消息级别。通常，记录的消息级别为 `info`、`warn` 或 `error`。如果遇到导致其提前退出的预期事件，IDT 会发出 `fatal` 或 `panic` 消息。

msg

记录的消息。

executionId

当前 IDT 流程的唯一 ID 字符串。此 ID 用于区分各个 IDT 运行。

测试套件生成的控制台消息提供了有关被测设备以及 IDT 运行的测试套件、测试组和测试用例的更多信息。以下摘录显示了从测试套件生成的控制台消息的示例。

```
[INFO] [2000-01-02 03:04:05]: Hello world! suiteId=MyTestSuitegroupId=myTestGroup
 testCaseId=myTestCase deviceId=my-
deviceexecutionId=9a52f362-1227-11eb-86c9-8c8590419f30
```

控制台消息中特定于测试套件的部分包含以下字段：

suiteId

当前正在运行的测试套件的名称。

groupId

当前正在运行的测试组的 ID。

testCaseId

当前正在运行的测试用例的 ID。

deviceId

当前测试用例正在使用的被测设备的 ID。

测试摘要包含有关测试套件、每个已运行组的测试结果以及生成的日志和报告文件位置的信息。以下示例显示了测试摘要消息。

```
========== Test Summary ==========
Execution Time:     5m00s
Tests Completed:    4
Tests Passed:       3
Tests Failed:       1
Tests Skipped:      0
---------------------------------
Test Groups:
    GroupA:         PASSED
    GroupB:         FAILED
---------------------------------
Failed Tests:
    Group Name: GroupB
        Test Name: TestB1
            Reason: Something bad happened
---------------------------------
Path to AWS IoT Device Tester Report: /path/to/awsiotdevicetester_report.xml
Path to Test Execution Logs: /path/to/logs
Path to Aggregated JUnit Report: /path/to/MyTestSuite_Report.xml
```

AWS IoT Device Tester 报告架构

`awsiotdevicetester_report.xml` 是一份包含以下信息的签名报告：

- IDT 版本。
- 测试套件版本。
- 用于对报告进行签名的报告签名和密钥。
- `device.json` 文件中指定的设备 SKU 和设备池名称。
- 经过测试的产品版本和设备功能。
- 测试结果的摘要汇总。此信息与 *`suite-name`*`_report.xml` 文件中包含的信息相同。

```
<apnreport>
    <awsiotdevicetesterversion>idt-version</awsiotdevicetesterversion>
    <testsuiteversion>test-suite-version</testsuiteversion>
    <signature>signature</signature>
    <keyname>keyname</keyname>
    <session>
        <testsession>execution-id</testsession>
        <starttime>start-time</starttime>
        <endtime>end-time</endtime>
    </session>
    <awsproduct>
        <name>product-name</name>
        <version>product-version</version>
        <features>
            <feature name="<feature-name>" value="supported | not-supported | <feature-
value>" type="optional | required"/>
        </features>
    </awsproduct>
    <device>
        <sku>device-sku</sku>
        <name>device-name</name>
        <features>
            <feature name="<feature-name>" value="<feature-value>"/>
        </features>
        <executionMethod>ssh | uart | docker</executionMethod>
    </device>
    <devenvironment>
        <os name="<os-name>"/>
    </devenvironment>
    <report>
        <suite-name-report-contents>
    </report>
</apnreport>
```

awsiotdevicetester_report.xml 文件包含一个 <awsproduct> 标签，其中包含有关正测试的产品以及在运行测试套件后验证的产品功能的信息。

<awsproduct> 标签中使用的属性

name

所测试的产品的名称。

version

所测试的产品的版本。

features

验证的功能。标记为 required 的功能是测试套件验证设备所必需的。以下代码段演示了此信息在 awsiotdevicetester_report.xml 文件中的显示方式。

```
<feature name="ssh" value="supported" type="required"></feature>
```

标记为 optional 的功能不是验证所必需的。以下代码段显示了可选功能。

```
<feature name="hsi" value="supported" type="optional"></feature>
<feature name="mqtt" value="not-supported" type="optional"></feature>
```

测试套件报告架构

该*suite-name*_Result.xml报告采用 JUnit XML 格式。您可以将它集成到持续集成和开发平台，例如 Jenkins、Bamboo 等。报告包含测试结果的摘要汇总。

```
<testsuites name="<suite-name> results" time="<run-duration>" tests="<number-of-test>"
 failures="<number-of-tests>" skipped="<number-of-tests>" errors="<number-of-tests>"
 disabled="0">
    <testsuite name="<test-group-id>" package="" tests="<number-of-tests>"
 failures="<number-of-tests>" skipped="<number-of-tests>" errors="<number-of-tests>"
 disabled="0">
        <!--success-->
        <testcase classname="<classname>" name="<name>" time="<run-duration>"/>
        <!--failure-->
        <testcase classname="<classname>" name="<name>" time="<run-duration>">
            <failure type="<failure-type>">
                reason
            </failure>
        </testcase>
        <!--skipped-->
        <testcase classname="<classname>" name="<name>" time="<run-duration>">
            <skipped>
                reason
            </skipped>
        </testcase>
        <!--error-->
```

```
        <testcase classname="<classname>" name="<name>" time="<run-duration>">
            <error>
                reason
            </error>
        </testcase>
    </testsuite>
</testsuites>
```

awsiotdevicetester_report.xml 或 *suite-name*_report.xml 中的报告部分列出了运行的测试以及结果。

第一个 XML 标签 <testsuites> 包含测试执行情况的摘要。例如：

```
<testsuites name="MyTestSuite results" time="2299" tests="28" failures="0" errors="0"
 disabled="0">
```

<testsuites> 标签中使用的属性

name

测试套件的名称。

time

运行测试套件所用的时间（以秒为单位）。

tests

执行的测试数。

failures

已运行但未通过的测试数。

errors

IDT 无法执行的测试数。

disabled

此属性未使用，可以忽略。

如果出现测试失败或错误，则可以通过检查 <testsuites> XML 标签来确定失败的测试。<testsuites> 标签内的 <testsuite> XML 标签显示了测试组的测试结果摘要。例如：

```
<testsuite name="combination" package="" tests="1" failures="0" time="161" disabled="0"
 errors="0" skipped="0">
```

其格式与 <testsuites> 标签类似，但包含一个未使用并可忽略的 skipped 属性。在每个 <testsuite> XML 标签内部，对于一个测试组，所执行的每个测试都有 <testcase> 标签。例如：

```
<testcase classname="Security Test" name="IP Change Tests" attempts="1"></testcase>
```

<testcase> 标签中使用的属性

name

测试的名称。

attempts

IDT 执行测试用例的次数。

当测试失败或出现错误时，将会在 <testcase> 标签中添加包含用于故障排除的信息的 <failure> 或 <error> 标签。例如：

```
<testcase classname="mcu.Full_MQTT" name="MQTT_TestCase" attempts="1">
 <failure type="Failure">Reason for the test failure</failure>
 <error>Reason for the test execution error</error>
</testcase>
```

提交 IDT 使用量指标

如果您提供具有所需权限的 AWS 证书，则 AWS IoT Device Tester 会收集使用情况指标并将其提交给 AWS。这是一项可选功能，用来改进 IDT 功能。IDT 将收集以下信息：

- 用于运行 IDT 的 AWS 账户 ID
- 用于运行测试的 IDT CLI 命令
- 正在运行的测试套件
- *<device-tester-extract-location>*文件夹中的测试套件
- 设备池中配置的设备数量
- 测试用例名称和运行时间
- 测试结果信息，例如测试是通过、失败、遇到错误，还是已被跳过

- 测试的产品功能
- IDT 退出行为，例如意外退出或提前退出

IDT 发送的所有信息也会记录到 *<device-tester-extract-location>*/results/*<execution-id>*/ 文件夹下的 metrics.log 文件中。您可以查看日志文件以检查在测试运行期间收集的信息。只有选择了收集使用量指标后，才会生成此文件。

要禁用指标收集，您无需采取其他操作。只需不要存储您的 AWS 凭证，如果您确实存储了 AWS 凭据，也不要将config.json文件配置为访问它们。

注册获取 AWS 账户

如果您没有 AWS 账户，请完成以下步骤来创建一个。

要注册 AWS 账户

1. 打开https://portal.aws.amazon.com/billing/注册。
2. 按照屏幕上的说明操作。

   注册过程的一部分涉及接听电话或短信，并在电话键盘上输入验证码。

   当您注册时 AWS 账户，就会创建AWS 账户根用户一个。根用户有权访问该账户中的所有 AWS 服务 和资源。作为最佳安全实践，请为用户分配管理访问权限，并且只使用根用户来执行需要根用户访问权限的任务。

AWS 注册过程完成后会向您发送一封确认电子邮件。您可以随时前往 https://aws.amazon.com/并选择“我的账户”，查看您当前的账户活动并管理您的账户。

创建具有管理访问权限的用户

注册后，请保护您的安全 AWS 账户 AWS 账户根用户 AWS IAM Identity Center，启用并创建管理用户，这样您就不会使用 root 用户执行日常任务。

保护你的 AWS 账户根用户

1. 选择 Root 用户并输入您的 AWS 账户 电子邮件地址，以账户所有者的身份登录。AWS Management Console在下一页上，输入您的密码。

   要获取使用根用户登录方面的帮助，请参阅《AWS 登录 用户指南》中的 Signing in as the root user。

2. 为您的根用户启用多重身份验证（MFA）。

   有关说明，请参阅 I A M 用户指南中的为 AWS 账户 根用户启用虚拟 MFA 设备（控制台）。

创建具有管理访问权限的用户

1. 启用 IAM Identity Center。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的 Enabling AWS IAM Identity Center。

2. 在 IAM Identity Center 中，为用户授予管理访问权限。

   有关使用 IAM Identity Center 目录 作为身份源的教程，请参阅《用户指南》 IAM Identity Center 目录中的使用默认设置配置AWS IAM Identity Center 用户访问权限。

以具有管理访问权限的用户身份登录

- 要使用您的 IAM Identity Center 用户身份登录，请使用您在创建 IAM Identity Center 用户时发送到您的电子邮件地址的登录网址。

  有关使用 IAM Identity Center 用户登录的帮助，请参阅AWS 登录 用户指南中的登录 AWS 访问门户。

将访问权限分配给其他用户

1. 在 IAM Identity Center 中，创建一个权限集，该权限集遵循应用最低权限的最佳做法。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的 Create a permission set。

2. 将用户分配到一个组，然后为该组分配单点登录访问权限。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的 Add groups。

要提供访问权限，请为您的用户、组或角色添加权限：

- 中的用户和群组 AWS IAM Identity Center：

  创建权限集合。按照《AWS IAM Identity Center 用户指南》中创建权限集的说明进行操作。

- 通过身份提供商在 IAM 中托管的用户：

创建适用于身份联合验证的角色。按照《IAM 用户指南》中针对第三方身份提供商创建角色（联合身份验证）的说明进行操作。

- IAM 用户：
  - 创建您的用户可以担任的角色。按照《IAM 用户指南》中为 IAM 用户创建角色的说明进行操作。
  - （不推荐使用）将策略直接附加到用户或将用户添加到用户组。按照《IAM 用户指南》中向用户添加权限（控制台）中的说明进行操作。

向 ID AWS T 提供凭证

要允许 IDT 访问您的 AWS 凭证并向其提交指标 AWS，请执行以下操作：

1. 将您的 IAM 用户的 AWS 证书存储为环境变量或存储在证书文件中：

   a. 要使用环境变量，请运行以下命令：

   ```
   AWS_ACCESS_KEY_ID=access-key
   AWS_SECRET_ACCESS_KEY=secret-access-key
   ```

   b. 要使用凭证文件，请将以下信息添加到 `.aws/credentials file:`：

   ```
   [profile-name]
   aws_access_key_id=access-key
   aws_secret_access_key=secret-access-key
   ```

2. 配置 `config.json` 文件的 `auth` 部分。有关更多信息，请参阅（可选）配置 config.json。

维护测试套件版本

适用于 FreeRTOS 的 IDT 将测试资源组织到测试套件和测试组中：

- 测试套件是一组测试组，用于验证设备运行的是否为特定版本的 FreeRTOS。
- 测试组是与特定功能（如 BLE 和 MQTT 消息收发）相关的一组单独测试。

从 IDT v3.0.0 开始，测试套件使用 `major.minor.patch` 格式进行版本化，格式从 1.0.0 开始。当您下载 IDT 时，数据包中包含最新的测试套件版本。

当您在命令行界面中启动 IDT 时，IDT 会检查是否有较新的测试套件版本。如果有，它会提示您更新到新版本。您可以选择更新或继续进行当前测试。

> **Note**
>
> IDT 支持三个最新的测试套件版本以获得资格认证。有关更多信息，请参阅 了解以下方面的支持政策 AWS IoT Device Tester。

您可以使用 `upgrade-test-suite` 命令下载测试套件。或者，您可以在启动 IDT `-upgrade-test-suite` *`flag`* 时使用可选参数，其中*`flag`*可以是 “y” 以始终下载最新版本，或者 “n” 表示使用现有版本。

您还可以运行 `list-supported-versions` 命令以列出当前 IDT 版本支持的 FreeRTOS 和测试套件版本。

新测试可能会引入新的 IDT 配置设置。如果设置是可选的，IDT 会通知您并继续运行测试。如果需要这些设置，IDT 会通知您并停止运行。配置这些设置后，您可以继续运行测试。

排查 错误

每个测试套件执行都有一个唯一的执行 ID，用于在 `results` 目录中创建名为 `results/`*`execution-id`* 的文件夹。单个测试组日志位于 `results/`*`execution-id`*`/logs` 目录下。使用适用于 FreeRTOS 的 IDT 控制台输出以查找失败的测试用例的执行 ID、测试用例 ID 和测试组 ID，然后打开名为 `results/`*`execution-id`*`/logs/`*`test_group_id__test_case_id`*`.log` 的测试用例的日志文件。此文件中的信息包括：

- 完整的 build 和 flash 命令输出。
- 测试执行输出。
- 适用于 FreeRTOS 的 IDT 控制台更为详细的输出。

建议采用以下工作流程进行故障排除：

1. 如果您看到错误 “*`user/role`*无权访问此资源”，请确保按照中的规定配置权限创建和配置 AWS 账户。
2. 阅读控制台输出以查找信息，例如执行 UUID 和当前正在执行的任务。
3. 在 `FRQ_Report.xml` 文件中查找各个测试的错误语句。此目录包含每个测试组执行日志。

4. 查看 /results/*execution-id*/logs 下的日志文件。
5. 调查以下问题领域之一：

   - 设备配置，如 /configs/ 文件夹中的 JSON 配置文件。
   - 设备接口。检查日志以确定哪些接口失败。
   - 设备工具。确保已正确安装和配置用于生成和刷写设备的工具链。
   - 对于 FRQ 1.x.x，请确保有干净的 FreeRTOS 克隆版本源代码可用。FreeRTOS 版本根据 FreeRTOS 版本进行标记。要克隆代码的特定版本，请使用以下命令。

     ```
     git clone --branch version-number https://github.com/aws/amazon-freertos.git
     cd amazon-freertos
     git submodule update --checkout --init --recursive
     ```

对设备配置进行故障排除

使用适用于 FreeRTOS 的 IDT 时，在执行二进制文件之前必须获取正确的配置文件。如果您收到了解析和配置错误，第一步应该是找到并使用适合您环境的配置模板。这些模板位于 *IDT_ROOT*/configs 目录中。

如果仍有问题，请参阅以下调试过程。

在哪里查找问题？

首先读取控制台输出以查找信息，例如在此文档中作为 execution-id 引用的执行 UUID。

接下来，在 /results/*execution-id* 目录中查找 FRQ_Report.xml 文件。此文件包含已运行的所有测试用例以及每次失败的错误代码片段。要获取所有执行日志，请查找每个测试用例的文件 /results/*execution-id*/logs/*test_group_id*__*test_case_id*.log。

IDT 错误代码

下表说明了适用于 FreeRTOS 的 IDT 生成的错误代码：

错误代码	错误代码名称	可能的根源	故障排除
201	InvalidInputError	device.js on 、config.js	确保在列出的文件中具有所需的字段，并

错误代码	错误代码名称	可能的根源	故障排除
		on 或 userdata.json 中的字段缺失或格式不正确。	且它们具有所需的格式。有关更多信息，请参阅 首次测试您的微控制器板。
202	ValidationError	device.json 、config.json 或 userdata.json 中的字段包含无效的值。	检查报告中的错误代码右侧的错误消息： • 无效 AWS 区域-在config.json 文件中指定有效 AWS 区域。有关 AWS 区域的更多信息，请参阅区域和终端节点。 • AWS 凭据无效-在测试计算机上设置有效的 AWS 凭据（通过环境变量或凭据文件）。验证是否正确配置了身份验证字段。有关更多信息，请参阅 创建和配置 AWS 账户。

错误代码	错误代码名称	可能的根源	故障排除
203	CopySourceCodeError	无法将 FreeRTOS 源代码复制到指定的目录中。	验证以下内容： • 检查是否在 `userdata.json` 文件中指定了有效的 `sourcePath` 。 • 删除 FreeRTOS 源代码目录中的 `build` 文件夹（如果存在）。有关更多信息，请参阅 配置构建、刷写和测试设置。 • Windows 对文件路径名称有字符数限制。文件路径名称过长会引发错误。

错误代码	错误代码名称	可能的根源	故障排除
204	BuildSourceError	无法编译 FreeRTOS 源代码。	验证以下内容： • 检查 `userdata.json` 文件中的 `buildTool` 下面的信息是否正确。 • 如果将 `cmake` 作为构建工具，请确保在 `buildTool` 命令中指定了 `{{enableTests}}` 。有关更多信息，请参阅 配置构建、刷写和测试设置。 • 如果您已将 FreeRTOS 的 IDT 提取到系统上包含空格的文件路径中，例如 `C:\Users\My Name\Desktop\`，则可能需要在构建命令中添加额外的引号才能确保正确解析路径。刷写命令可能需要执行同样的操作。

错误代码	错误代码名称	可能的根源	故障排除
205	FlashOrRunTestError	IDT FreeRTOS 无法在 DUT 上刷写或运行 FreeRTOS。	验证 `userdata.json` 文件中的 `flashTool` 下面的信息是否正确。有关更多信息，请参阅 配置构建、刷写和测试设置。
206	StartEchoServerError	IDT FreeRTOS 无法启动回声服务器进行或安全套接字测试 WiFi 。	确保 `userdata.json` 文件中 `echoServerConfiguration` 下配置的端口未被使用或未被防火墙或网络设置阻止。

调试配置文件解析错误

有时候，JSON 配置中的输入错误会导致解析错误。大部分情况下，问题是因 JSON 文件中漏掉括号、逗号或引号所导致。适用于 FreeRTOS 的 IDT 执行 JSON 验证并输出调试信息。它输出发生错误的行、行号以及语法错误的列号。这些信息应该足以帮助您修复错误，但是如果您仍然无法找到错误，则可以在IDE、Atom或Sublime等文本编辑器中或通过在线工具（例如Atom）或Sublime中手动进行验证，也可以通过诸如之类 JSONLint的在线工具进行验证。

调试测试结果解析错误

在从 FullTransportInterfaceTLS、Full _C PKCS11 ore FreeRTOS-Libraries-Integration-Tests、Full _onboard_ecc、Full PKCS11 _onboard_rsa、Full _ PKCS11 _ECC、Full PKCS11 _ _RSA 或 OTACoreFre PreProvisioned e PreProvisioned RTO S 的 IDT 等运行测试组时，使用串行连接解析来自测试设备的测试结果。PKCS11有时，设备上的额外串行输出可能会干扰测试结果的解析。

在上述情况下，系统会输出奇怪的测试用例失败原因，例如，输出来自不相关设备输出的字符串。适用于 FreeRTOS 的 IDT 测试用例日志文件（包括适用于 FreeRTOS 的 IDT 在测试过程中收到的所有串行输出）可能显示以下内容：

```
<unrelated device output>
TEST(Full_PKCS11_Capabilities, PKCS11_Capabilities)<unrelated device output>
<unrelated device output>
 PASS
```

在上述示例中，不相关的设备输出会阻止适用于 FreeRTOS 的 IDT 检测到 PASS 测试结果。

查看以下内容以确保实现最佳测试。

- 确保设备上使用的日志宏是线程安全的。有关更多信息，请参阅实现库日志记录宏。
- 在测试过程中，请确保串行连接的输出尽量少。即使您的日志记录宏是线程安全的，其他设备输出也可能存在问题，因为在测试过程中，测试结果会在单独的调用中输出。

理想情况下，适用于 FreeRTOS 的 IDT 测试用例日志会显示不间断的测试结果输出，如下所示：

```
---------STARTING TESTS---------
TEST(Full_OTA_PAL, otaPal_CloseFile_ValidSignature) PASS
TEST(Full_OTA_PAL, otaPal_CloseFile_InvalidSignatureBlockWritten) PASS
-----------------------
2 Tests 0 Failures 0 Ignored
```

调试完整性检查失败

如果使用 FRQ 1.x.x 版本的 FreeRTOS，则需要进行以下完整性检查。

在运行 Free RTOSIntegrity 测试组时遇到失败时，请先确保没有修改任何*freertos*目录文件。如果您没有修改，但仍然遇到问题，请确保您使用的是正确的分支。如果您运行 IDT 的 `list-supported-products` 命令，您可以查找应该使用 *freertos* 存储库的哪个标记分支。

如果您克隆的是 `freertos` 存储库的正确标记分支，但仍然存在问题，请确保您还运行了 `submodule update` 命令。`freertos` 存储库的克隆工作流程如下。

```
git clone --branch version-number https://github.com/aws/amazon-freertos.git
cd amazon-freertos
git submodule update --checkout —init —recursive
```

完整性检查程序查找的文件列表位于您的 *freertos* 目录中的 checksums.json 文件中。要在不修改文件和文件夹结构的情况下限定 FreeRTOS 移植，请确保未修改 checksums.json 文件的“exhaustive”和“minimal”部分中列出的文件。要在已配置开发工具包的情况下运行，请确认未修改“minimal”部分下的所有文件。

如果您使用开发工具包运行 IDT 且已修改 *freertos* 目录中的某些文件，请确保在 userdata 文件中正确配置开发工具包。否则，完整性检查程序将验证 *freertos* 目录中的所有文件。

调试 FullWiFi 测试组故障

如果您使用的是 FRQ 1.x.x，但在 FullWiFi 测试组中遇到故障，而“AFQP_WiFiConnectMultipleAP” 测试失败，则可能是因为两个接入点与运行 IDT 的主机不在同一个子网中。确保两个接入点与运行 IDT 的主机位于同一个子网中。

调试 “缺少必填参数” 错误

由于在适用于 FreeRTOS 的 IDT 中添加了新功能，所以配置文件可能会发生更改。使用旧配置文件可能会破坏您的配置。如果出现这种情况，results/*execution-id*/logs 目录下的 *test_group_id*__*test_case_id*.log 文件明确列出了所有缺少的参数。适用于 FreeRTOS 的 IDT 会验证您的 JSON 配置文件架构，确保使用了支持的最新版本。

调试 “测试无法启动” 错误

在测试启动期间，您可能看到指示失败的错误。由于有多种可能的原因，请检查以下地方正确与否：

- 确保您包括在执行命令中的池名称实际存在。这会从您的 device.json 文件直接引用。
- 确保池中的设备具有正确的配置参数。

调试 “找不到测试结果的开始” 错误

当 IDT 尝试解析被测试的设备输出的结果时，您可能会看到错误。这可能有多种原因，请检查以下方面是否正确：

- 确保被测试的设备与您的主机之间有稳定的连接。您可以查看日志文件中是否有显示这些错误的测试，了解 IDT 收到了哪些数据。
- 如果使用 FRQ 1.x.x，并且被测试的设备通过慢速网络或其他接口进行连接，或者您在 FreeRTOS 测试组日志以及其他 FreeRTOS 测试组输出中看不到“---------STARTING TESTS--------”标记，则可以尝试在用户数据配置中增加 testStartDelayms 的值。有关更多信息，请参阅 配置构建、刷写和测试设置。

调试 “测试失败：预期 __ 结果但看到 ___” 错误

在测试过程中，您可能看到指示测试失败的错误。该测试期望获得一定数量的结果，但在测试过程中没有看到。某些 FreeRTOS 测试会在 IDT 看到设备的输出之前运行。如果您看到此错误，可以尝试在用户数据配置中增加 `testStartDelayms` 的值。有关更多信息，请参阅 配置构建、刷写和测试设置。

调试 “由于限制而未选择 _____” 错误 ConditionalTests

这意味着您正在与测试不兼容的设备池上运行测试。OTA E2E 测试可能会发生这种情况。例如，在运行 `OTADataplaneMQTT` 测试组和 `device.json` 配置文件中，您选择 OTA 作为 No，或者选择 `OTADataPlaneProtocol` 作为 HTTP。选择运行的测试组必须与您选择的 `device.json` 的能力相匹配。

在设备输出监控期间调试 IDT 超时

IDT 可能由于多种原因而超时。如果在测试的设备输出监控阶段发生超时，并且您可以在 IDT 测试用例日志中看到结果，则表示 IDT 错误地解析了结果。原因之一可能是测试结果中间有交错的日志消息。如果是这种情况，请参阅《FreeRTOS 移植指南》，了解有关如何设置 UNITY 日志的更多详细信息。

设备输出监控过程中出现超时的另一个原因可能是，设备在单个 TLS 测试用例失败后重新启动。然后，设备会运行刷写的映像并导致无限循环，这在日志中可以看到。如果发生这种情况，请确保您的设备在测试失败后不会重新启动。

调试 “未授权访问资源” 错误

您可能会在终端输出或下的`test_manager.log`文件中看到 “*`user/role`*无权访问此资源” 错误`/results/`*`execution-id`*`/logs`。要解决此问题，请将 AWS `IoTDeviceTesterForFreeRTOSFullAccess` 托管策略附加到您的测试用户。有关更多信息，请参阅 创建和配置 AWS 账户。

调试网络测试错误

对于基于网络的测试，IDT 启动绑定到主机上非预留端口的 Echo 服务器。如果由于超时 WiFi 或安全套接字测试中的连接不可用而遇到错误，请确保您的网络配置为允许流量流向1024-49151范围内的已配置端口。

安全套接字测试默认使用端口 33333 和 33334。默认情况下， WiFi 测试使用端口 33335。如果这三个端口正在使用或被防火墙或网络阻止，您可以选择使用 userdata.json 中的其他端口进行测试。有关更多信息，请参阅 配置构建、刷写和测试设置。您可以使用以下命令来检查特定端口是否正在使用：

- Windows：`netsh advfirewall firewall show rule name=all | grep port`
- Linux：`sudo netstat -pan | grep port`
- macOS：`netstat -nat | grep port`

由于版本有效载荷相同，OTA 更新失败

如果 OTA 测试用例因在执行 OTA 后设备上存在相同版本而失败，则可能是由于您的构建系统（例如 cmake）未注意到 IDT 对 FreeRTOS 源代码所做的更改并且未构建更新后的二进制文件。这将导致使用当前位于设备上的相同二进制文件执行 OTA，并且会导致测试失败。要对 OTA 更新失败进行故障排除，首先请确保您使用的是受支持的最新版本的构建系统。

`PresignedUrlExpired` 测试用例上的 OTA 测试失败

此测试的一个先决条件是 OTA 更新时间应多于 60 秒，否则测试将失败。如果发生此情况，日志将包含以下错误消息：“Test takes less than 60 seconds (url expired time) to finish。Please reach out to us. (测试在 60 秒(URL 过期时间)内完成。请与我们联系。)”

调试设备接口和端口错误

此部分包含有 IDT 连接到设备所用设备接口的信息。

支持的平台

IDT 支持 Linux、macOS 和 Windows。对于连接到其上的串行设备，所有三种平台都有不同的命名方案：

- Linux：`/dev/tty*`
- macOS： `/dev/tty.*` 或 `/dev/cu.*`
- Windows：COM*

要查看您的设备端口，请执行以下操作：

- 对于 Linux/macOS，打开终端并运行 `ls /dev/tty*`。
- 对于 macOS，打开终端并运行 `ls /dev/tty.*` 或 `ls /dev/cu.*`。
- 对于 Windows，打开设备管理器并展开串行设备组。

要验证连接到端口的设备，请执行以下操作：

- 对于 Linux，请确保 udev 程序包已安装，然后运行 `udevadm info –name=PORT`。此实用程序打印设备驱动程序信息，帮助您验证使用了正确的端口。
- 对于 macOS，请打开 Launchpad 并搜索 **System Information**。
- 对于 Windows，打开设备管理器并展开串行设备组。

设备接口

每台嵌入式设备都不相同，这意味着它们可以有一个或多个串行端口。设备在连接到计算机时有两个端口是常见情况：

- 一个数据端口，用于刷写设备。
- 一个读取端口，用于读取输出。

  您必须在 `device.json` 文件中设置正确的读取端口。否则，从设备中读取输出可能会失败。

  在有多个端口时，请确保在您的 `device.json` 文件中使用设备的读取端口。例如，如果您插入 Espressif WRover 设备并且分配给它的两个端口是`/dev/ttyUSB0`和`/dev/ttyUSB1`，则在您的文件`/dev/ttyUSB1`中使用。`device.json`

对于 Windows，请按照相同的逻辑操作。

读取设备数据

适用于 FreeRTOS 的 IDT 使用单独的设备构建和刷写工具来指定端口配置。如果您测试设备但未获得输出，请尝试以下默认设置：

- 波特率：115200
- 数据位：8
- 奇偶校验：无
- 停止位：1
- 流控制：无

这些设置通过适用于 FreeRTOS 的 IDT 处理。您无需设置它们。不过，您可以使用相同的方法手动读取设备输出。在 Linux 或 macOS 上，您可以使用 `screen` 命令完成此操作。在 Windows 上，你可以使用诸如这样的程序 TeraTerm。

```
Screen: screen /dev/cu.usbserial 115200
```

TeraTerm: Use the above-provided settings to set the fields explicitly in the GUI.

开发工具链问题

此部分讨论您的工具链中可能出现的问题。

Ubuntu 上的 Code Composer Studio

Ubuntu 的较新版本（17.10 和 18.04）具有的 glibc 程序包版本与 Code Composer Studio 7.x 版本不兼容。建议您安装 Code Composer Studio 版本 8.2 或更高版本。

不兼容症状可能包括：

- FreeRTOS 无法构建或刷写到您的设备。
- Code Composer Studio 安装程序可能会冻结。
- 在生成或刷写过程中，控制台未显示任何日志输出。
- 即使以无管控模式调用，构建命令尝试以 GUI 模式启动。

日志记录

适用于 FreeRTOS 的 IDT 日志放在一个位置。从根 IDT 目录中，这些文件在 results/*execution-id*/ 下提供：

- FRQ_Report.xml
- awsiotdevicetester_report.xml
- logs/*test_group_id*__*test_case_id*.log

FRQ_Report.xml 和 logs/*test_group_id*__*test_case_id*.log 是要检查的最重要的日志。FRQ_Report.xml 包含有关哪些测试用例失败并显示特定错误消息的信息。然后，您可以使用 logs/*test_group_id*__*test_case_id*.log 来深入挖掘问题以更好地了解上下文。

控制台错误

运行 AWS IoT Device Tester 时，会向控制台报告失败并显示简短消息。查看 results/*execution-id*/logs/*test_group_id*__*test_case_id*.log 以了解有关错误的更多信息。

日志错误

每个测试套件执行都有一个唯一的执行 ID，用于创建名为 `results/execution-id` 的文件夹。单个测试用例日志位于 `results/execution-id/logs` 目录下。使用适用于 FreeRTOS 的 IDT 控制台的输出以查找失败的测试用例的执行 ID、测试用例 ID 和测试组 ID。然后使用此信息查找并打开名`results/execution-id/logs/test_group_id__test_case_id.log`为的测试用例的日志文件。此文件中的信息包括完整的编译和 Flash 命令输出、测试执行输出以及更详细的 AWS IoT Device Tester 控制台输出。

S3 存储桶问题

如果在运行 IDT 时按下 CTRL+C，IDT 将启动清理过程。清理工作的一部分是移除在 IDT 测试中创建的 Amazon S3 资源。如果清理无法完成，则可能会遇到已创建的 Amazon S3 存储桶过多的问题。这意味着下次运行 IDT 时，测试将开始失败。

如果您按下 CTRL+C 以停止 IDT，则必须让它完成清理过程才能避免出现此问题。您也可以从您的账户中删除手动创建的 Amazon S3 存储桶。

超时错误疑难解答

如果您在运行测试套件时看到超时错误，请指定超时乘数系数来增大超时值。该系数将应用于默认超时值。为此标志配置的任何值都必须大于或等于 1.0。要使用超时乘数，请在运行测试套件时使用 `--timeout-multiplier` 标志。

Example

IDT v3.0.0 and later

```
./devicetester_linux run-suite --suite-id FRQ_1.0.1 --pool-id DevicePool1 --timeout-multiplier 2.5
```

IDT v1.7.0 and earlier

```
./devicetester_linux run-suite --suite-id FRQ_1 --pool-id DevicePool1 --timeout-multiplier 2.5
```

蜂窝功能和 AWS 费用

当您的`device.JSON`文件`Yes`中将该`Cellular`功能设置为时， FullSecureSockets 将使用 t.micro EC2 实例运行测试，这可能会给您的 AWS 账户带来额外费用。有关更多信息，请参阅 Amazon EC2 定价。

资格报告生成策略

资格报告仅由最近两年内发布的支持 FreeRTOS 版本的 AWS IoT Device Tester (IDT) 版本生成。如果您对支持策略有疑问，请联系 AWS 支持。

了解以下各项的 AWS 托管策略 AWS IoT Device Tester

AWS 托管策略是由创建和管理的独立策略 AWS。 AWS 托管策略旨在为许多常见用例提供权限，以便您可以开始为用户、组和角色分配权限。

请记住， AWS 托管策略可能不会为您的特定用例授予最低权限权限，因为它们可供所有 AWS 客户使用。我们建议通过定义特定于您的使用场景的客户管理型策略来进一步减少权限。

您无法更改 AWS 托管策略中定义的权限。如果 AWS 更新 AWS 托管策略中定义的权限，则更新会影响该策略所关联的所有委托人身份（用户、组和角色）。 AWS 最有可能在启动新的 API 或现有服务可以使用新 AWS 服务 的 API 操作时更新 AWS 托管策略。

有关更多信息，请参阅《IAM 用户指南》中的 AWS 托管策略。

主题

- AWS 托管策略： AWS Io TDevice TesterForFree RTOSFull 访问权限
- AWS 托管策略的更新

AWS 托管策略： AWS Io TDevice TesterForFree RTOSFull 访问权限

`AWSIoTDeviceTesterForFreeRTOSFullAccess`托管策略包含版本检查、auto update 功能和指标收集的以下 AWS IoT Device Tester 权限。

权限详细信息

该策略包含以下权限：

- `iot-device-tester:SupportedVersion`

授 AWS IoT Device Tester 予获取受支持产品、测试套件和 IDT 版本列表的权限。

- `iot-device-tester:LatestIdt`

  授 AWS IoT Device Tester 予获取可供下载的最新 IDT 版本的权限。

- `iot-device-tester:CheckVersion`

  授 AWS IoT Device Tester 予检查 IDT、测试套件和产品的版本兼容性的权限。

- `iot-device-tester:DownloadTestSuite`

  授 AWS IoT Device Tester 予下载测试套件更新的权限。

- `iot-device-tester:SendMetrics`

  授 AWS 予收集 AWS IoT Device Tester 内部使用情况指标的权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::*:role/idt-*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "iot.amazonaws.com"
                }
            }
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "iot:DeleteThing",
                "iot:AttachThingPrincipal",
                "iot:DeleteCertificate",
                "iot:GetRegistrationCode",
                "iot:CreatePolicy",
                "iot:UpdateCACertificate",
                "s3:ListBucket",
```

```
                "iot:DescribeEndpoint",
                "iot:CreateOTAUpdate",
                "iot:CreateStream",
                "signer:ListSigningJobs",
                "acm:ListCertificates",
                "iot:CreateKeysAndCertificate",
                "iot:UpdateCertificate",
                "iot:CreateCertificateFromCsr",
                "iot:DetachThingPrincipal",
                "iot:RegisterCACertificate",
                "iot:CreateThing",
                "iam:ListRoles",
                "iot:RegisterCertificate",
                "iot:DeleteCACertificate",
                "signer:PutSigningProfile",
                "s3:ListAllMyBuckets",
                "signer:ListSigningPlatforms",
                "iot-device-tester:SendMetrics",
                "iot-device-tester:SupportedVersion",
                "iot-device-tester:LatestIdt",
                "iot-device-tester:CheckVersion",
                "iot-device-tester:DownloadTestSuite"
            ],
            "Resource": "*"
        },
        {
            "Sid": "VisualEditor2",
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "signer:StartSigningJob",
                "acm:GetCertificate",
                "signer:DescribeSigningJob",
                "s3:CreateBucket",
                "execute-api:Invoke",
                "s3:DeleteBucket",
                "s3:PutBucketVersioning",
                "signer:CancelSigningProfile"
            ],
            "Resource": [
                "arn:aws:execute-api:us-east-1:098862408343:9xpmnvs5h4/prod/POST/
metrics",
                "arn:aws:signer:*:*:/signing-profiles/*",
                "arn:aws:signer:*:*:/signing-jobs/*",
```

```
                "arn:aws:iam::*:role/idt-*",
                "arn:aws:acm:*:*:certificate/*",
                "arn:aws:s3:::idt-*",
                "arn:aws:s3:::afr-ota*"
            ]
        },
        {
            "Sid": "VisualEditor3",
            "Effect": "Allow",
            "Action": [
                "iot:DeleteStream",
                "iot:DeleteCertificate",
                "iot:AttachPolicy",
                "iot:DetachPolicy",
                "iot:DeletePolicy",
                "s3:ListBucketVersions",
                "iot:UpdateCertificate",
                "iot:GetOTAUpdate",
                "iot:DeleteOTAUpdate",
                "iot:DescribeJobExecution"
            ],
            "Resource": [
                "arn:aws:s3:::afr-ota*",
                "arn:aws:iot:*:*:thinggroup/idt*",
                "arn:aws:iam::*:role/idt-*"
            ]
        },
        {
            "Sid": "VisualEditor4",
            "Effect": "Allow",
            "Action": [
                "iot:DeleteCertificate",
                "iot:AttachPolicy",
                "iot:DetachPolicy",
                "s3:DeleteObjectVersion",
                "iot:DeleteOTAUpdate",
                "s3:PutObject",
                "s3:GetObject",
                "iot:DeleteStream",
                "iot:DeletePolicy",
                "s3:DeleteObject",
                "iot:UpdateCertificate",
                "iot:GetOTAUpdate",
                "s3:GetObjectVersion",
```

```
                "iot:DescribeJobExecution"
            ],
            "Resource": [
                "arn:aws:s3:::afr-ota*/*",
                "arn:aws:s3:::idt-*/*",
                "arn:aws:iot:*:*:policy/idt*",
                "arn:aws:iam::*:role/idt-*",
                "arn:aws:iot:*:*:otaupdate/idt*",
                "arn:aws:iot:*:*:thing/idt*",
                "arn:aws:iot:*:*:cert/*",
                "arn:aws:iot:*:*:job/*",
                "arn:aws:iot:*:*:stream/*"
            ]
        },
        {
            "Sid": "VisualEditor5",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::afr-ota*/*",
                "arn:aws:s3:::idt-*/*"
            ]
        },
        {
            "Sid": "VisualEditor6",
            "Effect": "Allow",
            "Action": [
                "iot:CancelJobExecution"
            ],
            "Resource": [
                "arn:aws:iot:*:*:job/*",
                "arn:aws:iot:*:*:thing/idt*"
            ]
        },
        {
            "Sid": "VisualEditor7",
            "Effect": "Allow",
            "Action": [
                "ec2:TerminateInstances"
            ],
            "Resource": [
```

```
                "arn:aws:ec2:*:*:instance/*"
            ],
            "Condition": {
                "StringEquals": {
                    "ec2:ResourceTag/Owner": "IoTDeviceTester"
                }
            }
        },
        {
            "Sid": "VisualEditor8",
            "Effect": "Allow",
            "Action": [
                "ec2:AuthorizeSecurityGroupIngress",
                "ec2:DeleteSecurityGroup"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:security-group/*"
            ],
            "Condition": {
                "StringEquals": {
                    "ec2:ResourceTag/Owner": "IoTDeviceTester"
                }
            }
        },
        {
            "Sid": "VisualEditor9",
            "Effect": "Allow",
            "Action": [
                "ec2:RunInstances"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:instance/*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/Owner": "IoTDeviceTester"
                }
            }
        },
        {
            "Sid": "VisualEditor10",
            "Effect": "Allow",
            "Action": [
                "ec2:RunInstances"
```

```
            ],
            "Resource": [
                "arn:aws:ec2:*:*:image/*",
                "arn:aws:ec2:*:*:security-group/*",
                "arn:aws:ec2:*:*:volume/*",
                "arn:aws:ec2:*:*:key-pair/*",
                "arn:aws:ec2:*:*:placement-group/*",
                "arn:aws:ec2:*:*:snapshot/*",
                "arn:aws:ec2:*:*:network-interface/*",
                "arn:aws:ec2:*:*:subnet/*"
            ]
        },
        {
            "Sid": "VisualEditor11",
            "Effect": "Allow",
            "Action": [
                "ec2:CreateSecurityGroup"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:security-group/*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/Owner": "IoTDeviceTester"
                }
            }
        },
        {
            "Sid": "VisualEditor12",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeInstances",
                "ec2:DescribeSecurityGroups",
                "ssm:DescribeParameters",
                "ssm:GetParameters"
            ],
            "Resource": "*"
        },
        {
            "Sid": "VisualEditor13",
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags"
            ],
```

```
            "Resource": [
                "arn:aws:ec2:*:*:security-group/*",
                "arn:aws:ec2:*:*:instance/*"
            ],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "aws:TagKeys": [
                        "Owner"
                    ]
                },
                "StringEquals": {
                    "ec2:CreateAction": [
                        "RunInstances",
                        "CreateSecurityGroup"
                    ]
                }
            }
        }
    ]
}
```

AWS 托管策略的更新

您可以查看自该服务开始跟踪这些更改之时 AWS IoT Device Tester 起的 AWS 托管策略更新的详细信息。

版本	更改	描述	日期
7（最新）	重构了 `ec2:CreateTags` 条件。	移除了 `ForAnyValues` 的使用。	2023 年 6 月 14 日
6	从策略中移除了 `freertos:ListHardwarePlatforms` 。	移除权限，因为自 2023 年 3 月 1 日起，此操作已弃用。	2023 年 2 月 6 日
5	添加了使用运行 echo 服务器测试的权限 EC2。	这用于启动和停止客户 AWS 账户中的 EC2 实例。	12/15/2020

版本	更改	描述	日期
4	增加了 `iot:CancelJobExecution` 。	此权限会取消 OTA 作业。	7/17/2020

版本	更改	描述	日期
3	添加了以下权限： • iot-device-tester:DownloadTestSuite， • iot-device-tester:CheckVersion， • iot-device-tester:LatestIdt， • iot-device-tester:SupportedVersion。	• iot-device-tester:DownloadTestSuite — 授予下载测试套件更新的 AWS IoT Device Tester 权限， • iot-device-tester:CheckVersion — 授 AWS IoT Device Tester 予检查 IDT、测试套件和产品的版本兼容性的权限， • iot-device-tester:LatestIdt — 授 AWS IoT Device Tester 予获取可供下载的最新 IDT 版本的权限， • iot-device-tester:SupportedVersion — 授 AWS IoT Device Tester 予获取受支持产品、测试套件和 IDT 版本列表的权限。	2020 年 3 月 23 日

版本	更改	描述	日期
2	添加了 `iot-device-tester: SendMetrics` 权限。	授 AWS 予收集 AWS IoT Device Tester 内部使用情况指标的权限。	2/18/2020
1	初始版本。		2/12/2020

了解以下方面的支持政策 AWS IoT Device Tester

> **Important**
>
> 截至 2022 年 10 月， AWS IoT Device Tester AWS IoT FreeRTOS 资格认证 (FRQ) 1.0 不会生成已签署的资格报告。使用 IDT AWS IoT FRQ 1.0 版本，您无法通过AWS 设备资格认证计划将新 FreeRTOS 设备列入合作伙伴设备目录。AWS 虽然您无法使用 IDT FRQ 1.0 确定 FreeRTOS 设备的资格，但您可以继续使用 FRQ 1.0 测试 FreeRTOS 设备。我们建议您使用 IDT FRQ 2.0 来进行资格认证，并在 AWS 合作伙伴设备目录中列出 FreeRTOS 设备。

AWS IoT Device Tester for FreeRTOS 是一款测试自动化工具，用于验证 FreeRTOS 是否移植到设备。此外，您还可以对您的 FreeRTOS 设备进行资格认证，并将其列在 AWS 合作伙伴设备目录中。for AWS IoT Device Tester FreeRTOS 支持对 FreeRTOS 长期支持 (LTS) 库进行验证和认证，在 FreeRTOS/FreerTOS 上可用 FreeRTOS 主线，FreeRTOS 主线可在 FreerTOS/ FreeRTOS GitHub 上线。我们建议您使用最新版本的 FreeRTOS 和 for FreeRTOS 来验证和鉴 AWS IoT Device Tester 定您的设备。

对于 FreeRTOS-LTS，IDT 支持 FreeRTOS 202210 LTS 版本的验证和资格认证。有关 FreeRTOS LTS 版本及其维护时间表的更多信息，请参阅此处。这些 LTS 版本的支持期限结束后，您仍然可以继续进行验证，但是 IDT 不会生成允许您提交设备进行资格认证的报告。

对于 FreeRTOS/FreeRTOS 上提供的主线 FreeRTOS，我们支持对过去六个月内发布的所有版本进行验证和认证，或者，如果发布时间相隔超过六个月，则支持对之前两个版本的 FreeRTOS 进行验证和认证。有关当前支持的版本的列表，请参阅此处。对于不支持的 FreeRTOS 版本，您仍然可以继续验证，但是 IDT 不会生成允许您提交设备进行资格认证的报告。

有关支持的最新 IDT 和 FreeRTOS 版本，请参阅支持的版本 AWS IoT Device Tester。您可以将任何支持的版本与相应版本的 FreeRTOS AWS IoT Device Tester 配合使用来测试或鉴定您的设备。您可以继续使用 不受支持的适用于 FreeRTOS 的 IDT 版本，但不会不会收到最新的错误修复或更新。

如果您对支持策略有疑问，请联系 AWS 客户支持。

安全性 AWS

云安全 AWS 是重中之重。作为 AWS 客户，您可以受益于专为满足大多数安全敏感型组织的要求而构建的数据中心和网络架构。

安全是双方共同承担 AWS 的责任。责任共担模式将其描述为云的 安全性和云中 的安全性：

- 云安全 — AWS 负责保护在 AWS 云中运行 AWS 服务的基础架构。 AWS 还为您提供可以安全使用的服务。作为 AWS 合规性计划的一部分，我们的安全措施的有效性定期由第三方审计员进行测试和验证。要了解适用于某项 AWS 服务的合规计划，请参阅按合规性计划划分的范围内的AWS 服务。
- 云端安全-您的责任由您使用的 AWS 服务决定。您还需要对其他因素负责，包括您的数据的敏感性、您组织的要求以及适用的法律法规。

本文档将帮助您了解在使用时如何应用分担责任模型 AWS。以下主题向您展示如何进行配置 AWS 以满足您的安全和合规性目标。您还将学习如何使用可以帮助您监控和保护 AWS 资源的 AWS 服务。

有关 AWS IoT 安全的更多深入信息，请参阅 “安全和身份” AWS IoT。

主题

- 适用于 FreeRTOS 的身份和访问权限管理
- 合规性验证
- 韧性在 AWS
- FreeRTOS 中的基础设施安全性

适用于 FreeRTOS 的身份和访问权限管理

AWS Identity and Access Management (IAM) AWS 服务 可帮助管理员安全地控制对 AWS 资源的访问权限。IAM 管理员控制可以通过身份验证（登录）和授权（具有权限）使用 FreeRTOS 资源的人员。您可以使用 IAM AWS 服务 ，无需支付额外费用。

主题

- 受众
- 使用身份进行身份验证
- 使用策略管理访问

- 如何将 FreeRTOS 与 IAM 配合使用
- 适用于 FreeRTOS 的基于身份的策略示例
- 对 FreeRTOS 身份和访问进行故障排除

受众

您的使用方式 AWS Identity and Access Management (IAM) 会有所不同，具体取决于您在 FreeRTOS 中所做的工作。

服务用户 – 如果使用 FreeRTOS 服务来完成任务，则您的管理员会为您提供所需的凭证和权限。当您使用更多 FreeRTOS 特征来完成工作时，您可能需要额外权限。了解如何管理访问权限有助于您向管理员请求适合的权限。如果您无法访问 FreeRTOS 中的特征，请参阅对 FreeRTOS 身份和访问进行故障排除。

服务管理员 – 如果您在公司负责管理 FreeRTOS 资源，则您可能具有 FreeRTOS 的完全访问权限。您有责任确定您的服务用户应访问哪些 FreeRTOS 特征和资源。然后，您必须向 IAM 管理员提交请求以更改服务用户的权限。请查看该页面上的信息以了解 IAM 的基本概念。要了解有关您的公司如何将 IAM 与 FreeRTOS 配合使用的更多信息，请参阅如何将 FreeRTOS 与 IAM 配合使用。

IAM 管理员 – 如果您是 IAM 管理员，您可能希望了解有关如何编写策略以管理对 FreeRTOS 的访问权限的详细信息。要查看您可在 IAM 中使用的 FreeRTOS 基于身份的策略示例，请参阅适用于 FreeRTOS 的基于身份的策略示例。

使用身份进行身份验证

身份验证是您 AWS 使用身份凭证登录的方式。您必须以 IAM 用户身份或通过担 AWS 账户根用户任 IAM 角色进行身份验证（登录 AWS）。

您可以使用通过身份源提供的凭据以 AWS 联合身份登录。 AWS IAM Identity Center （IAM Identity Center）用户、贵公司的单点登录身份验证以及您的 Google 或 Facebook 凭据就是联合身份的示例。当您以联合身份登录时，您的管理员以前使用 IAM 角色设置了身份联合验证。当你使用联合访问 AWS 时，你就是在间接扮演一个角色。

根据您的用户类型，您可以登录 AWS Management Console 或 AWS 访问门户。有关登录的更多信息 AWS，请参阅《AWS 登录 用户指南》中的如何登录到您 AWS 账户的。

如果您 AWS 以编程方式访问，则会 AWS 提供软件开发套件 (SDK) 和命令行接口 (CLI)，以便使用您的凭据对请求进行加密签名。如果您不使用 AWS 工具，则必须自己签署请求。有关使用推荐的方法自行签署请求的更多信息，请参阅《IAM 用户指南》中的用于签署 API 请求的AWS 签名版本 4。

无论使用何种身份验证方法，您都可能需要提供其他安全信息。例如， AWS 建议您使用多重身份验证 (MFA) 来提高账户的安全性。要了解更多信息，请参阅《AWS IAM Identity Center 用户指南》中的多重身份验证和《IAM 用户指南》中的 IAM 中的AWS 多重身份验证。

AWS 账户 root 用户

创建时 AWS 账户，首先要有一个登录身份，该身份可以完全访问账户中的所有资源 AWS 服务 和资源。此身份被称为 AWS 账户 root 用户，使用您创建账户时使用的电子邮件地址和密码登录即可访问该身份。强烈建议您不要使用根用户执行日常任务。保护好根用户凭证，并使用这些凭证来执行仅根用户可以执行的任务。有关要求您以根用户身份登录的任务的完整列表，请参阅 IAM 用户指南中的需要根用户凭证的任务。

联合身份

作为最佳实践，要求人类用户（包括需要管理员访问权限的用户）使用与身份提供商的联合身份验证 AWS 服务 通过临时证书进行访问。

联合身份是指您的企业用户目录、Web 身份提供商、Identity C enter 目录中的用户，或者任何使用 AWS 服务 通过身份源提供的凭据进行访问的用户。 AWS Directory Service当联合身份访问时 AWS 账户，他们将扮演角色，角色提供临时证书。

要集中管理访问权限，建议您使用 AWS IAM Identity Center。您可以在 IAM Identity Center 中创建用户和群组，也可以连接并同步到您自己的身份源中的一组用户和群组，以便在您的所有 AWS 账户 和应用程序中使用。有关 IAM Identity Center 的信息，请参阅 AWS IAM Identity Center 用户指南中的什么是 IAM Identity Center？。

IAM 用户和群组

I AM 用户是您 AWS 账户 内部对个人或应用程序具有特定权限的身份。在可能的情况下，我们建议使用临时凭证，而不是创建具有长期凭证（如密码和访问密钥）的 IAM 用户。但是，如果您有一些特定的使用场景需要长期凭证以及 IAM 用户，建议您轮换访问密钥。有关更多信息，请参阅《IAM 用户指南》中的对于需要长期凭证的用例，应在需要时更新访问密钥。

IAM 组是一个指定一组 IAM 用户的身份。您不能使用组的身份登录。您可以使用组来一次性为多个用户指定权限。如果有大量用户，使用组可以更轻松地管理用户权限。例如，您可以拥有一个名为的群组，IAMAdmins并向该群组授予管理 IAM 资源的权限。

用户与角色不同。用户唯一地与某个人员或应用程序关联，而角色旨在让需要它的任何人代入。用户具有永久的长期凭证，而角色提供临时凭证。要了解更多信息，请参阅《IAM 用户指南》中的 IAM 用户的使用案例。

IAM 角色

I AM 角色是您内部具有特定权限 AWS 账户 的身份。它类似于 IAM 用户，但与特定人员不关联。要在中临时担任 IAM 角色 AWS Management Console，您可以从用户切换到 IAM 角色（控制台）。您可以通过调用 AWS CLI 或 AWS API 操作或使用自定义 URL 来代入角色。有关使用角色的方法的更多信息，请参阅《IAM 用户指南》中的代入角色的方法。

具有临时凭证的 IAM 角色在以下情况下很有用：

- 联合用户访问：要向联合身份分配权限，请创建角色并为角色定义权限。当联合身份进行身份验证时，该身份将与角色相关联并被授予由此角色定义的权限。有关用于联合身份验证的角色的信息，请参阅《IAM 用户指南》中的针对第三方身份提供商创建角色（联合身份验证）。如果您使用 IAM Identity Center，则需要配置权限集。为控制您的身份在进行身份验证后可以访问的内容，IAM Identity Center 将权限集与 IAM 中的角色相关联。有关权限集的信息，请参阅《AWS IAM Identity Center 用户指南》中的权限集。
- 临时 IAM 用户权限：IAM 用户可代入 IAM 用户或角色，以暂时获得针对特定任务的不同权限。
- 跨账户存取：您可以使用 IAM 角色以允许不同账户中的某个人（可信主体）访问您的账户中的资源。角色是授予跨账户访问权限的主要方式。但是，对于某些资源 AWS 服务，您可以将策略直接附加到资源（而不是使用角色作为代理）。要了解用于跨账户访问的角色和基于资源的策略之间的差别，请参阅 IAM 用户指南中的 IAM 中的跨账户资源访问。
- 跨服务访问 — 有些 AWS 服务 使用其他 AWS 服务服务中的功能。例如，当您在服务中拨打电话时，该服务通常会在 Amazon 中运行应用程序 EC2 或在 Amazon S3 中存储对象。服务可能会使用发出调用的主体的权限、使用服务角色或使用服务相关角色来执行此操作。
  - 转发访问会话 (FAS) — 当您使用 IAM 用户或角色在中执行操作时 AWS，您被视为委托人。使用某些服务时，您可能会执行一个操作，然后此操作在其他服务中启动另一个操作。FAS 使用调用委托人的权限以及 AWS 服务 向下游服务发出请求的请求。 AWS 服务只有当服务收到需要与其他 AWS 服务 或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两项操作的权限。有关发出 FAS 请求时的策略详情，请参阅转发访问会话。
  - 服务角色 - 服务角色是服务代表您在您的账户中执行操作而分派的 IAM 角色。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的创建向 AWS 服务委派权限的角色。
  - 服务相关角色-服务相关角色是一种与服务相关联的服务角色。 AWS 服务服务可以代入代表您执行操作的角色。服务相关角色出现在您的中 AWS 账户 ，并且归服务所有。IAM 管理员可以查看但不能编辑服务相关角色的权限。
- 在 A@@ mazon 上运行的应用程序 EC2 — 您可以使用 IAM 角色管理在 EC2 实例上运行并发出 AWS CLI 或 AWS API 请求的应用程序的临时证书。这比在 EC2 实例中存储访问密钥更可取。要

为 EC2 实例分配 AWS 角色并使其可供其所有应用程序使用，您需要创建一个附加到该实例的实例配置文件。实例配置文件包含该角色，并允许在 EC2 实例上运行的程序获得临时证书。有关更多信息，请参阅 IAM 用户指南中的使用 IAM 角色向在 A mazon EC2 实例上运行的应用程序授予权限。

使用策略管理访问

您可以 AWS 通过创建策略并将其附加到 AWS 身份或资源来控制中的访问权限。策略是其中的一个对象 AWS ，当与身份或资源关联时，它会定义其权限。 AWS 在委托人（用户、root 用户或角色会话）发出请求时评估这些策略。策略中的权限确定是允许还是拒绝请求。大多数策略都以 JSON 文档的 AWS 形式存储在中。有关 JSON 策略文档的结构和内容的更多信息，请参阅 IAM 用户指南中的 JSON 策略概览。

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

默认情况下，用户和角色没有权限。要授予用户对所需资源执行操作的权限，IAM 管理员可以创建 IAM 策略。管理员随后可以向角色添加 IAM 策略，用户可以代入角色。

IAM 策略定义操作的权限，无关乎您使用哪种方法执行操作。例如，假设您有一个允许 `iam:GetRole` 操作的策略。拥有该策略的用户可以从 AWS Management Console AWS CLI、或 AWS API 获取角色信息。

基于身份的策略

基于身份的策略是可附加到身份（如 IAM 用户、用户组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的使用客户托管策略定义自定义 IAM 权限。

基于身份的策略可以进一步归类为内联策略或托管式策略。内联策略直接嵌入单个用户、组或角色中。托管策略是独立的策略，您可以将其附加到中的多个用户、群组和角色 AWS 账户。托管策略包括 AWS 托管策略和客户托管策略。要了解如何在托管策略和内联策略之间进行选择，请参阅《IAM 用户指南》中的在托管策略与内联策略之间进行选择。

基于资源的策略

基于资源的策略是附加到资源的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定主体可以对该资源执行哪些操作以及在什么条件下执行。您必须在基于资源的策略中指定主体。委托人可以包括账户、用户、角色、联合用户或 AWS 服务。

基于资源的策略是位于该服务中的内联策略。您不能在基于资源的策略中使用 IAM 中的 AWS 托管策略。

访问控制列表 (ACLs)

访问控制列表 (ACLs) 控制哪些委托人（账户成员、用户或角色）有权访问资源。 ACLs 与基于资源的策略类似，尽管它们不使用 JSON 策略文档格式。

Amazon S3 和 Amazon VPC 就是支持的服务示例 ACLs。 AWS WAF要了解更多信息 ACLs，请参阅《亚马逊简单存储服务开发者指南》中的访问控制列表 (ACL) 概述。

其他策略类型

AWS 支持其他不太常见的策略类型。这些策略类型可以设置更常用的策略类型向您授予的最大权限。

- 权限边界：权限边界是一个高级特征，用于设置基于身份的策略可以为 IAM 实体（IAM 用户或角色）授予的最大权限。您可为实体设置权限边界。这些结果权限是实体基于身份的策略及其权限边界的交集。在 `Principal` 中指定用户或角色的基于资源的策略不受权限边界限制。任一项策略中的显式拒绝将覆盖允许。有关权限边界的更多信息，请参阅IAM 用户指南中的 IAM 实体的权限边界。
- 服务控制策略 (SCPs)- SCPs 是指定组织或组织单位 (OU) 的最大权限的 JSON 策略 AWS Organizations。 AWS Organizations 是一项用于对您的企业拥有的多 AWS 账户 项进行分组和集中管理的服务。如果您启用组织中的所有功能，则可以将服务控制策略 (SCPs) 应用于您的任何或所有帐户。SCP 限制成员账户中的实体（包括每个 AWS 账户根用户实体）的权限。有关 Organization SCPs s 和的更多信息，请参阅《AWS Organizations 用户指南》中的服务控制策略。
- 资源控制策略 (RCPs) — RCPs 是 JSON 策略，您可以使用它来设置账户中资源的最大可用权限，而无需更新附加到您拥有的每个资源的 IAM 策略。RCP 限制成员账户中资源的权限，并可能影响身份（包括身份）的有效权限 AWS 账户根用户，无论这些身份是否属于您的组织。有关 Organizations 的更多信息 RCPs，包括 AWS 服务 该支持的列表 RCPs，请参阅《AWS Organizations 用户指南》中的资源控制策略 (RCPs)。
- 会话策略：会话策略是当您以编程方式为角色或联合用户创建临时会话时作为参数传递的高级策略。结果会话的权限是用户或角色的基于身份的策略和会话策略的交集。权限也可以来自基于资源的策略。任一项策略中的显式拒绝将覆盖允许。有关更多信息，请参阅 IAM 用户指南中的会话策略。

多个策略类型

当多个类型的策略应用于一个请求时，生成的权限更加复杂和难以理解。要了解在涉及多种策略类型时如何 AWS 确定是否允许请求，请参阅 IAM 用户指南中的策略评估逻辑。

如何将 FreeRTOS 与 IAM 配合使用

在使用 IAM 管理对 FreeRTOS 的访问之前，您应该了解哪些 IAM 功能可用于 FreeRTOS。

您可以与 FreeRTOS 配合使用的 IAM 特征

IAM 特征	FreeRTOS 支持
基于身份的策略	是
基于资源的策略	否
策略操作	是
策略资源	是
策略条件键（特定于服务）	是
ACLs	否
ABAC（策略中的标签）	部分
临时凭证	是
主体权限	是
服务角色	是
服务相关角色	否

要全面了解 FreeRTOS AWS 和其他服务如何与大多数 IAM 功能配合使用，AWS 请参阅 IAM 用户指南中与 IAM 配合使用的服务。

适用于 FreeRTOS 的基于身份的策略

支持基于身份的策略：是

基于身份的策略是可附加到身份（如 IAM 用户、用户组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的使用客户管理型策略定义自定义 IAM 权限。

通过使用 IAM 基于身份的策略，您可以指定允许或拒绝的操作和资源以及允许或拒绝操作的条件。您无法在基于身份的策略中指定主体，因为它适用于其附加的用户或角色。要了解可在 JSON 策略中使用的所有元素，请参阅《IAM 用户指南》中的 IAM JSON 策略元素引用。

适用于 FreeRTOS 的基于身份的策略示例

要查看 FreeRTOS 基于身份的策略的示例，请参阅适用于 FreeRTOS 的基于身份的策略示例。

FreeRTOS 内基于资源的策略

支持基于资源的策略：否

基于资源的策略是附加到资源的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定主体可以对该资源执行哪些操作以及在什么条件下执行。您必须在基于资源的策略中指定主体。委托人可以包括账户、用户、角色、联合用户或 AWS 服务。

要启用跨账户访问，您可以将整个账户或其他账户中的 IAM 实体指定为基于资源的策略中的主体。将跨账户主体添加到基于资源的策略只是建立信任关系工作的一半而已。当委托人和资源处于不同位置时 AWS 账户，可信账户中的 IAM 管理员还必须向委托人实体（用户或角色）授予访问资源的权限。他们通过将基于身份的策略附加到实体以授予权限。但是，如果基于资源的策略向同一个账户中的主体授予访问权限，则不需要额外的基于身份的策略。有关更多信息，请参阅《IAM 用户指南》中的 IAM 中的跨账户资源访问。

适用于 FreeRTOS 的策略操作

支持策略操作：是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

JSON 策略的 `Action` 元素描述可用于在策略中允许或拒绝访问的操作。策略操作通常与关联的 AWS API 操作同名。有一些例外情况，例如没有匹配 API 操作的仅限权限 操作。还有一些操作需要在策略中执行多个操作。这些附加操作称为相关操作。

在策略中包含操作以授予执行关联操作的权限。

要查看 FreeRTOS 操作的列表，请参阅《服务授权参考》中的 FreeRTOS 定义的操作。

FreeRTOS 中的策略操作在操作前使用以下前缀：

```
awes
```

要在单个语句中指定多项操作，请使用逗号将它们隔开。

```
"Action": [
      "awes:action1",
      "awes:action2"
         ]
```

要查看 FreeRTOS 基于身份的策略的示例，请参阅适用于 FreeRTOS 的基于身份的策略示例。

FreeRTOS 的策略资源

支持策略资源：是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

`Resource` JSON 策略元素指定要向其应用操作的一个或多个对象。语句必须包含 `Resource` 或 `NotResource` 元素。作为最佳实践，请使用其 Amazon 资源名称（ARN）指定资源。对于支持特定资源类型（称为资源级权限）的操作，您可以执行此操作。

对于不支持资源级权限的操作（如列出操作），请使用通配符（*）指示语句应用于所有资源。

```
"Resource": "*"
```

要查看 FreeRTOS 资源类型 ARNs及其列表，请参阅《服务授权参考》中的 FreeRTOS 定义的资源。要了解您可以在哪些操作中指定每个资源的 ARN，请参阅 FreeRTOS 定义的操作。

要查看 FreeRTOS 基于身份的策略的示例，请参阅适用于 FreeRTOS 的基于身份的策略示例。

FreeRTOS 的策略条件键

支持特定于服务的策略条件键：是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

在 Condition 元素（或 Condition 块）中，可以指定语句生效的条件。Condition 元素是可选的。您可以创建使用条件运算符（例如，等于或小于）的条件表达式，以使策略中的条件与请求中的值相匹配。

如果您在一个语句中指定多个 Condition 元素，或在单个 Condition 元素中指定多个键，则 AWS 使用逻辑 AND 运算评估它们。如果您为单个条件键指定多个值，则使用逻辑OR运算来 AWS 评估条件。在授予语句的权限之前必须满足所有的条件。

在指定条件时，您也可以使用占位符变量。例如，只有在使用 IAM 用户名标记 IAM 用户时，您才能为其授予访问资源的权限。有关更多信息，请参阅《IAM 用户指南》中的 IAM 策略元素：变量和标签。

AWS 支持全局条件密钥和特定于服务的条件密钥。要查看所有 AWS 全局条件键，请参阅 IAM 用户指南中的AWS 全局条件上下文密钥。

有关 FreeRTOS 条件键的列表，请参阅《服务授权参考》中 FreeRTOS 的条件键。要了解您可以对哪些操作和资源使用条件键，请参阅 FreeRTOS 定义的操作。

要查看 FreeRTOS 基于身份的策略的示例，请参阅适用于 FreeRTOS 的基于身份的策略示例。

ACLs 在 FreeRTOS 中

支持 ACLs：否

访问控制列表 (ACLs) 控制哪些委托人（账户成员、用户或角色）有权访问资源。 ACLs 与基于资源的策略类似，尽管它们不使用 JSON 策略文档格式。

ABAC 及 FreeRTOS

支持 ABAC（策略中的标签）：部分支持

基于属性的访问控制（ABAC）是一种授权策略，该策略基于属性来定义权限。在中 AWS，这些属性称为标签。您可以向 IAM 实体（用户或角色）和许多 AWS 资源附加标签。标记实体和资源是 ABAC 的第一步。然后设计 ABAC 策略，以在主体的标签与他们尝试访问的资源标签匹配时允许操作。

ABAC 在快速增长的环境中非常有用，并在策略管理变得繁琐的情况下可以提供帮助。

要基于标签控制访问，您需要使用 aws:ResourceTag/*key-name*、aws:RequestTag/*key-name* 或 aws:TagKeys 条件键在策略的条件元素中提供标签信息。

如果某个服务对于每种资源类型都支持所有这三个条件键，则对于该服务，该值为是。如果某个服务仅对于部分资源类型支持所有这三个条件键，则该值为部分。

有关 ABAC 的更多信息，请参阅《IAM 用户指南》中的使用 ABAC 授权定义权限。要查看设置 ABAC 步骤的教程，请参阅《IAM 用户指南》中的使用基于属性的访问权限控制（ABAC）。

将临时凭证与 FreeRTOS 配合使用

支持临时凭证：是

当你使用临时证书登录时，有些 AWS 服务 不起作用。有关更多信息，包括哪些 AWS 服务 适用于临时证书，请参阅 IAM 用户指南中的AWS 服务 与 IAM 配合使用的信息。

如果您使用除用户名和密码之外的任何方法登录，则 AWS Management Console 使用的是临时证书。例如，当您 AWS 使用公司的单点登录 (SSO) 链接进行访问时，该过程会自动创建临时证书。当您以用户身份登录控制台，然后切换角色时，您还会自动创建临时凭证。有关切换角色的更多信息，请参阅《IAM 用户指南》中的从用户切换到 IAM 角色（控制台）。

您可以使用 AWS CLI 或 AWS API 手动创建临时证书。然后，您可以使用这些临时证书进行访问 AWS。 AWS 建议您动态生成临时证书，而不是使用长期访问密钥。有关更多信息，请参阅 IAM 中的临时安全凭证。

FreeRTOS 的跨服务主体权限

支持转发访问会话（FAS）：是

当您使用 IAM 用户或角色在中执行操作时 AWS，您被视为委托人。使用某些服务时，您可能会执行一个操作，然后此操作在其他服务中启动另一个操作。FAS 使用调用委托人的权限以及 AWS 服务 向下游服务发出请求的请求。 AWS 服务只有当服务收到需要与其他 AWS 服务 或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两项操作的权限。有关发出 FAS 请求时的策略详情，请参阅转发访问会话。

FreeRTOS 的服务角色

支持服务角色：是

服务角色是由一项服务担任、代表您执行操作的 IAM 角色。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的创建向 AWS 服务委派权限的角色。

> **Warning**
>
> 更改服务角色的权限可能会破坏 FreeRTOS 的功能。仅当 FreeRTOS 提供相关指导时才编辑服务角色。

FreeRTOS 的服务相关角色

支持服务相关角色：否

服务相关角色是一种与服务相关联的 AWS 服务服务角色。服务可以代入代表您执行操作的角色。服务相关角色出现在您的中 AWS 账户 ，并且归服务所有。IAM 管理员可以查看但不能编辑服务相关角色的权限。

有关创建或管理服务相关角色的详细信息，请参阅能够与 IAM 搭配使用的AWS 服务。在表中查找服务相关角色列中包含 Yes 的表。选择是链接以查看该服务的服务相关角色文档。

适用于 FreeRTOS 的基于身份的策略示例

默认情况下，用户和角色没有创建或修改 FreeRTOS 资源的权限。他们也无法使用 AWS Management Console、 AWS Command Line Interface (AWS CLI) 或 AWS API 执行任务。要授予用户对所需资源执行操作的权限，IAM 管理员可以创建 IAM 策略。管理员随后可以向角色添加 IAM 策略，用户可以代入角色。

要了解如何使用这些示例 JSON 策略文档创建基于 IAM 身份的策略，请参阅《IAM 用户指南》中的创建 IAM 策略（控制台）。

有关 FreeRTOS 定义的操作和资源类型（包括每种资源类型的 ARNs 格式）的详细信息，请参阅《服务授权参考》中的 FreeRTOS 的操作、资源和条件键。

主题

- 策略最佳实践
- 使用 FreeRTOS 控制台
- 允许用户查看他们自己的权限

策略最佳实践

基于身份的策略确定某个人是否可以创建、访问或删除您账户中的 FreeRTOS 资源。这些操作可能会使 AWS 账户产生成本。创建或编辑基于身份的策略时，请遵循以下指南和建议：

- 开始使用 AWS 托管策略并转向最低权限权限 — 要开始向用户和工作负载授予权限，请使用为许多常见用例授予权限的AWS 托管策略。它们在你的版本中可用 AWS 账户。我们建议您通过定义针对您的用例的 AWS 客户托管策略来进一步减少权限。有关更多信息，请参阅《IAM 用户指南》中的AWS 托管式策略或工作职能的AWS 托管式策略。

- 应用最低权限：在使用 IAM 策略设置权限时，请仅授予执行任务所需的权限。为此，您可以定义在特定条件下可以对特定资源执行的操作，也称为最低权限许可。有关使用 IAM 应用权限的更多信息，请参阅《IAM 用户指南》中的 IAM 中的策略和权限。
- 使用 IAM 策略中的条件进一步限制访问权限：您可以向策略添加条件来限制对操作和资源的访问。例如，您可以编写策略条件来指定必须使用 SSL 发送所有请求。如果服务操作是通过特定 AWS 服务的（例如）使用的，则也可以使用条件来授予对服务操作的访问权限 AWS CloudFormation。有关更多信息，请参阅《IAM 用户指南》中的 IAM JSON 策略元素：条件。
- 使用 IAM Access Analyzer 验证您的 IAM 策略，以确保权限的安全性和功能性 – IAM Access Analyzer 会验证新策略和现有策略，以确保策略符合 IAM 策略语言（JSON）和 IAM 最佳实践。IAM Access Analyzer 提供 100 多项策略检查和可操作的建议，以帮助您制定安全且功能性强的策略。有关更多信息，请参阅《IAM 用户指南》中的使用 IAM Access Analyzer 验证策略。
- 需要多重身份验证 (MFA)-如果 AWS 账户您的场景需要 IAM 用户或根用户，请启用 MFA 以提高安全性。若要在调用 API 操作时需要 MFA，请将 MFA 条件添加到您的策略中。有关更多信息，请参阅《IAM 用户指南》中的使用 MFA 保护 API 访问。

有关 IAM 中的最佳实操的更多信息，请参阅《IAM 用户指南》中的 IAM 中的安全最佳实践。

使用 FreeRTOS 控制台

要访问 FreeRTOS 控制台，您必须拥有一组最低的权限。这些权限必须允许您列出和查看有关您的 FreeRTOS 资源的详细信息。 AWS 账户如果创建比必需的最低权限更为严格的基于身份的策略，对于附加了该策略的实体（用户或角色），控制台将无法按预期正常运行。

对于仅调用 AWS CLI 或 AWS API 的用户，您无需为其设置最低控制台权限。相反，只允许访问与其尝试执行的 API 操作相匹配的操作。

为确保用户和角色仍然可以使用 FreeRTOS 控制台，还要将 FreeRTOS 或托管策略附加到*ConsoleAccess*实*ReadOnly* AWS 体。有关更多信息，请参阅《IAM 用户指南》中的为用户添加权限。

允许用户查看他们自己的权限

该示例说明了您如何创建策略，以允许 IAM 用户查看附加到其用户身份的内联和托管式策略。此策略包括在控制台上或使用 AWS CLI 或 AWS API 以编程方式完成此操作的权限。

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

对 FreeRTOS 身份和访问进行故障排除

使用以下信息可帮助您诊断和修复在使用 FreeRTOS 和 IAM 时可能遇到的常见问题。

主题

- 我无权在 FreeRTOS 中执行操作
- 我无权执行 iam：PassRole
- 允许我以外的人访问我的 AWS 账户 FreeRTOS 资源

我无权在 FreeRTOS 中执行操作

如果您收到错误提示，指明您无权执行某个操作，则必须更新策略以允许执行该操作。

当 mateojackson IAM 用户尝试使用控制台查看有关虚构 *my-example-widget* 资源的详细信息，但不拥有虚构 awes:*GetWidget* 权限时，会发生以下示例错误。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 awes:GetWidget on resource: my-example-widget
```

在此情况下，必须更新 mateojackson 用户的策略，以允许使用 awes:*GetWidget* 操作访问 *my-example-widget* 资源。

如果您需要帮助，请联系您的 AWS 管理员。您的管理员是提供登录凭证的人。

我无权执行 iam：PassRole

如果您收到一个错误，表明您无权执行 iam:PassRole 操作，则必须更新策略以允许您将角色传递给 FreeRTOS。

有些 AWS 服务 允许您将现有角色传递给该服务，而不是创建新的服务角色或服务相关角色。为此，您必须具有将角色传递到服务的权限。

当名为 marymajor 的 IAM 用户尝试使用控制台在 FreeRTOS 中执行操作时，会发生以下示例错误。但是，服务必须具有服务角色所授予的权限才可执行此操作。Mary 不具有将角色传递到服务的权限。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

在这种情况下，必须更新 Mary 的策略以允许她执行 iam:PassRole 操作。

如果您需要帮助，请联系您的 AWS 管理员。您的管理员是提供登录凭证的人。

允许我以外的人访问我的 AWS 账户 FreeRTOS 资源

您可以创建一个角色，以便其他账户中的用户或您组织外的人员可以使用该角色来访问您的资源。您可以指定谁值得信赖，可以代入角色。对于支持基于资源的策略或访问控制列表 (ACLs) 的服务，您可以使用这些策略向人们授予访问您的资源的权限。

要了解更多信息，请参阅以下内容：

- 要了解 FreeRTOS 是否支持这些特征，请参阅如何将 FreeRTOS 与 IAM 配合使用。

- 要了解如何提供对您拥有的资源的访问权限 AWS 账户 ，请参阅 IAM 用户指南中的向您拥有 AWS 账户 的另一个 IAM 用户提供访问权限。
- 要了解如何向第三方提供对您的资源的访问权限 AWS 账户，请参阅 IAM 用户指南中的向第三方提供访问权限。 AWS 账户
- 要了解如何通过身份联合验证提供访问权限，请参阅《IAM 用户指南》中的为经过外部身份验证的用户（身份联合验证）提供访问权限。
- 要了解使用角色和基于资源的策略进行跨账户访问之间的差别，请参阅《IAM 用户指南》中的 IAM 中的跨账户资源访问。

合规性验证

要了解是否属于特定合规计划的范围，请参阅AWS 服务 “按合规计划划分的范围” ”，然后选择您感兴趣的合规计划。 AWS 服务 有关一般信息，请参阅AWS 合规计划AWS。

您可以使用下载第三方审计报告 AWS Artifact。有关更多信息，请参阅中的 “下载报告” 中的 “ AWS Artifact。

您在使用 AWS 服务 时的合规责任取决于您的数据的敏感性、贵公司的合规目标以及适用的法律和法规。 AWS 提供了以下资源来帮助实现合规性：

- Security Compliance & Governance：这些解决方案实施指南讨论了架构考虑因素，并提供了部署安全性和合规性功能的步骤。
- 符合 HIPAA 要求的服务参考：列出符合 HIPAA 要求的服务。并非所有 AWS 服务 人都符合 HIPAA 资格。
- AWS 合规资源AWS — 此工作簿和指南集可能适用于您所在的行业和所在地区。
- AWS 客户合规指南 — 从合规角度了解责任共担模式。这些指南总结了保护的最佳实践， AWS 服务并将指南映射到跨多个框架（包括美国国家标准与技术研究院 (NIST)、支付卡行业安全标准委员会 (PCI) 和国际标准化组织 (ISO)）的安全控制。
- 使用AWS Config 开发人员指南中的规则评估资源 — 该 AWS Config 服务评估您的资源配置在多大程度上符合内部实践、行业准则和法规。
- AWS Security Hub— 这 AWS 服务 提供了您内部安全状态的全面视图 AWS。Security Hub 通过安全控制措施评估您的 AWS 资源并检查其是否符合安全行业标准和最佳实践。有关受支持服务及控制措施的列表，请参阅 Security Hub 控制措施参考。
- Amazon GuardDuty — 它通过监控您的 AWS 账户环境中是否存在可疑和恶意活动，来 AWS 服务检测您的工作负载、容器和数据面临的潜在威胁。 GuardDuty 通过满足某些合规性框架规定的入侵检测要求，可以帮助您满足各种合规性要求，例如 PCI DSS。

- AWS Audit Manager— 这 AWS 服务 可以帮助您持续审计 AWS 使用情况，从而简化风险管理以及对法规和行业标准的合规性。

韧性在 AWS

AWS 全球基础设施是围绕 AWS 区域和可用区构建的。 AWS 区域提供多个在物理上独立且隔离的可用区，这些可用区通过延迟低、吞吐量高且冗余性高的网络连接在一起。利用可用区，您可以设计和操作在可用区之间无中断地自动实现故障转移的应用程序和数据库。与传统的单个或多个数据中心基础架构相比，可用区具有更高的可用性、容错性和可扩展性。

有关 AWS 区域和可用区的更多信息，请参阅AWS 全球基础设施。

FreeRTOS 中的基础设施安全性

AWS 托管服务受《Amazon Web Services：安全流程概述》白皮书中描述的 AWS 全球网络安全程序的保护。

您可以使用 AWS 已发布的 API 调用通过网络访问 AWS 服务。客户端必须支持传输层安全性（TLS）1.2 或更高版本。我们建议使用 TLS 1.3 或更高版本。客户端还必须支持具有完全向前保密（PFS）的密码套件，例如 Ephemeral Diffie-Hellman（DHE）或 Elliptic Curve Ephemeral Diffie-Hellman（ECDHE）。大多数现代系统（如 Java 7 及更高版本）都支持这些模式。

此外，必须使用访问密钥 ID 和与 IAM 主体关联的秘密访问密钥来对请求进行签名。或者，您可以使用 AWS Security Token Service（AWS STS）生成临时安全凭证来对请求进行签名。

Amazon-FreeRTOS Github 存储库迁移指南

如果您有一个基于现已弃用的 Amazon FreeRTOS 存储库的 FreeRTOS 项目，请按照以下步骤操作：

1. 随时了解最新的公开可用安全修复程序。查看 FreeRTOS LTS 库页面以获取更新，或者订阅 GitHub FreerTOS-LTS 存储库以接收包含关键和安全错误修复的最新 LTS 补丁。您可以直接从各个存储库下载或克隆所需的最新 FreeRTOS LTS 补丁。 GitHub
2. 考虑重构网络传输接口实现以优化您的硬件平台。最新的 CoreMQTT 库不需要 APIs 诸如安全套接字和 Wifi APIs 之类的摘要。有关更多详细信息，请参阅传输接口。

附录

下表提供了对 Amazon-FreerTOS 存储库 APIs 中所有演示项目、旧库和摘要的建议。

迁移的库和演示

名称	类型	建议
coreHTTP	演示和库	直接从 FreeRTOS Github 组织中的 coreHTTP 存储库（如果使用 git，则为子模块）克隆或下载 coreHTTP 库。coreHTTP 演示在 FreeRTOS 的主分发中。有关更多详细信息，请参考 coreHTTP 页面。
coreMQTT	演示和库	直接从 FreeRTOS Github 组织中的 coreMQTT 存储库（如果使用 git，则为子模块）克隆或下载 coreMQTT 库。coreMQTT 演示在 FreeRTOS 主分发中。有关更多详细信息，请参考 coreMQTT 页面。
coreMQTT-Agent	演示和库	直接从 FreeRTOS Github 组织中的 coreMQTT 代理存储库（如果使用 git，则为子模块）克隆或下载 coreMQTT 代理库。coreMQTT 代理演示在 coreMQTT 代理演示存储

名称	类型	建议
		库中。有关更多详细信息，请参考 coreMQTT 代理页面。
device_defender_for_aws	演示和库	De AWS IoT vice Defender 库位于AWS GitHub 组织中的存储库中。直接从 AWS IoT Device Defender 存储库中克隆或下载该库（如果使用 git，则为子模块）。Dev AWS IoT ice Defender 演示在 FreeR TOS 的主发行版中。有关更多详细信息，请参阅 AWS IoT Device Defender 页面。
device_shadow_for_aws	演示和库	AWS IoT Device Shadow 库位于AWS GitHub 组织中的存储库中。直接从 AWS IoT Device Shadow 存储库中克隆或下载该库（如果使用 git，则为子模块）。Dev AWS IoT ice Shadow 演示在 FreeR TOS 的主发行版中。有关更多详细信息，请参阅 AWS IoT Device Shadow 页面。
jobs_for_aws	演示和库	AWS IoT 作业库位于AWS GitHub 组织中的存储库中。直接从 AWS IoT Jobs 存储库中克隆或下载该库（如果使用 git，则为子模块）。 AWS IoT 任务演示位于主要的 FreeRT OS 发行版中。有关更多详细信息，请参考 AWS IoT Jobs 页面。

名称	类型	建议	
OTA	演示和库	AWS IoT Over-The-Air (OTA) 更新库位于AWS GitHub 组织中的存储库中。直接从 AWS IoT OTA 存储库中克隆或下载该库（如果使用 git，则为子模块）。 AWS IoT OTA 演示在 FreeR TOS 的主发行版中。有关更多详细信息，请参考 AWS IoT OTA 页面。	
CLI 和 FreeRTOS_Plus_CLI	演示和库	上面有一个 CLI 示例 WinSim。有关更多详细信息，请参阅 FreeRTOS Plus 命令行界面页面。 RT1恩智浦 i.MX STM32060和 U5平台上的精选FreeRTOS物联网参考集成还提供了有关实际硬件的CLI示例。	
logging	宏	某些 FreeRTOS 库提供了适用于特定硬件平台的日志宏的实现。有关如何实现日志宏的信息，请参阅日志记录页面。有关在实际硬件上运行的示例，请参阅 FreeRTOS 精选 IoT 参考之一。	
greengrass_connectivity	演示	[迁移正在进行中] 此演示项目假设在连接到 AWS IoT Greengrass 设备之前可以进行云连接。一个演示本地身份验证和发现能力的新项目正在开发中。预计新演示项目将很快会在 FreeRTOS Github 组织中发布。	

已弃用的库和演示

名称	类型	建议
BLE	演示和库	FreeRTOS BLE 库可实现专有的 MQTT 协议并支持通过代理设备（例如手机）使用低功耗蓝牙 (BLE) 功能发布和订阅 MQTT 主题。这不再是强制性的。您可以使用自己的 BLE 堆栈或第三方选项（例如 NimBLE）来最大限度优化项目。
dev_mode_key_provisioning	演示	RT1恩智浦i.MX_STM32060、U5或-C3平台上的精选FreeRTOS物联网参考集成提供了使用CLI进行 ESP32关键配置的示例。
posix	抽象和演示	不建议使用。
wifi_provisioning	示例	此示例演示了如何使用 Amazon-FreerTOS BLE 库在设备上配置 WiFi 凭证。有关通过 BLE 进行配置的 WiFi 示例，请参阅 C3 平台ESP32上的 FreeRTOS 精选物联网参考资料。
遗产摘要 APIs	code	它们 APIs 旨在为来自不同供应商的各种第三方软件堆栈、连接模块和 MCU 平台提供抽象接口。例如，有 WiFi 抽象接口、安全套接字等。它们在 Amazon-FreeRTOS 存储库中受支持，并且位于 `/libraries/abstractions/` 文件夹中。 APIs 在使用 FreeRTOS LTS 库时，这些都不是必需的。

上表中的库和演示不会获得安全补丁或错误修复。

第三方库

当 Amazon-FreeRTOS 中的演示使用第三方库时，我们建议您直接使用第三方存储库中的子模块。

- CMock: 直接从 C moc k 存储库中克隆它（如果你使用 git 则为子模块）。
- jsmn：不推荐，也不再受支持。
- lwip：直接从 lwip-tcpip 存储库中克隆（如果使用 git，则为子模块）。
- lwip_osal：有关如何在硬件平台/主板上实现 lwip_osal 的信息，请参阅 i.MX RT1 060 或 U5 上的 FreeRTOS 精选参考集成。STM32
- mbedtls：直接从 Mbed-TLS 存储库中克隆（如果使用 git，则为子模块）。mbedtls 配置和实用工具可重复使用；在这种情况下，请创建本地副本。
- pkcs11：直接从核心库或 O ASIS PK CS 11 存储PKCS11库中克隆它（如果你使用 git 则为子模块）。
- tinycbor：直接从 tinycbor 存储库中克隆（如果使用 git，则为子模块）。
- tinycrypt：我们建议您使用 MCU 平台上的加密加速器（如果有）。如果您想继续使用 tinycrypt，请直接从 tinycrypt 存储库中克隆（如果使用 git，则为子模块）。
- tracealyzer_recorder：直接从 Percepio 的 trace recorder 存储库中克隆（如果使用 git，则为子模块）。
- unity：直接从 ThrowTheSwitch/Unit y 存储库中克隆它（如果你使用 git 则为子模块）。
- win_pcap：win_pcap 已停止维护。我们建议您使用 libslirp、libpcap (posix) 或 npcap。

移植测试和集成测试

该`/tests`文件夹下验证FreeRTOS库集成所需的所有测试都已迁移到存储库中。 FreeRTOS-Libraries-Integration-Tests它们可用于测试 PAL 实现和库集成。 AWS IoT 设备测试器 (IDT) 在 FreeRTOS 的AWS 设备认证计划中使用相同的测试。

FreeRTOS 存档文档

此内容已存档，不再更新。它可以引用过时的信息或过时的功能。有关 FreeRTOS 的最新指南，请参阅本 FreeRTOS 用户指南或其他相关文档中的最新信息。 AWS 有关最新信息，请参见什么是 FreeRTOS？。

主题

- FreeRTOS 用户指南存档
- FreeRTOS 用户指南之前的内容

FreeRTOS 用户指南存档

这些先前版本的 FreeRTOS 用户指南适用于 FreeRTOS LTS（长期支持）版本。

- 适用于 FreeRTOS 版本 202210.00 的《FreeRTOS 用户指南》
- 适用于 FreeRTOS 版本 202012.00 的《FreeRTOS 用户指南》

FreeRTOS 用户指南之前的内容

此内容已过时，仅供参考。

有关最新内容，请参阅开始使用 FreeRTOS上提供的链接。

开始使用 FreeRTOS

> **Important**
>
> 此页面指向已弃用的 Amazon FreeRTOS 存储库。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

本 FreeRTOS 入门教程介绍如何在主机上下载和配置 FreeRTOS，然后在符合条件的微控制器主板上编译和运行简单的演示应用程序。

在本教程中，我们假设您熟悉 AWS IoT AWS IoT 控制台。如果不熟悉，建议您先完成 AWS IoT 入门教程，然后再继续。

主题：

- FreeRTOS 演示应用程序
- 初始步骤
- 问题排查入门
- CMake 与 FreeRTOS 一起使用
- 开发人员模式密钥预置
- 主板特定的入门指南
- FreeRTOS 的后续步骤

FreeRTOS 演示应用程序

> **Important**
>
> 此页面指向已弃用的 Amazon FreeRTOS 存储库。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

本教程中的演示应用程序是在 *freertos*/demos/coreMQTT_Agent/mqtt_agent_task.c 文件中定义的 coreMQTT 代理演示。它使用coreMQTT 库连接到 AWS 云端，然后定期向 MQTT 代理托管的 MQTT 主题发布消息。AWS IoT

一次只能运行一个 FreeRTOS 演示应用程序。在构建 FreeRTOS 演示项目时，*freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h 标头文件中启用的第一个演示为运行的应用程序。在本教程中，您无需启用或禁用任何演示。coreMQTT 代理演示默认为启用状态。

有关 FreeRTOS 附带的演示应用程序的更多信息，请参阅 FreeRTOS 演示。

初始步骤

> **Important**
>
> 此页面指向已弃用的 Amazon FreeRTOS 存储库。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

要开始使用 FreeRTOS，你必须拥有 AWS 一个帐户、一个有权访问权限的用户 AWS IoT 和 FreeR AWS IoT TOS 云服务。您还必须下载 FreeRTOS 并配置开发板的 FreeRTOS 演示项目才能使用。AWS IoT以下各节将引导您了解这些要求。

> **Note**
>
> - 如果你使用的是 Espressif ESP32-DevKit C ESP-WROVER-KIT、或 ESP32-WROOM-32SE，请跳过这些步骤转到。开始使用 Espressif ESP32-C 和 DevKit ESP-WROVER-KIT
> - 如果你使用的是 Nordic n RF5284 0-DK，请跳过这些步骤并转到。Nordic n RF5284 0-DK 入门

1. 设置您的 AWS 账户和权限
2. 注册您的 MCU 主板 AWS IoT
3. 下载 FreeRTOS
4. 配置 FreeRTOS 演示

设置您的 AWS 账户和权限

注册获取 AWS 账户

如果您没有 AWS 账户，请完成以下步骤来创建一个。

报名参加 AWS 账户

1. 打开https://portal.aws.amazon.com/billing/注册。
2. 按照屏幕上的说明操作。

   注册过程的一部分涉及接听电话或短信，并在电话键盘上输入验证码。

   当您注册时 AWS 账户，就会创建AWS 账户根用户一个。根用户有权访问该账户中的所有 AWS 服务 和资源。作为最佳安全实践，请为用户分配管理访问权限，并且只使用根用户来执行需要根用户访问权限的任务。

AWS 注册过程完成后会向您发送一封确认电子邮件。您可以随时前往 https://aws.amazon.com/并选择“我的账户”，查看您当前的账户活动并管理您的账户。

创建具有管理访问权限的用户

注册后，请保护您的安全 AWS 账户 AWS 账户根用户 AWS IAM Identity Center，启用并创建管理用户，这样您就不会使用 root 用户执行日常任务。

保护你的 AWS 账户根用户

1. 选择 Root 用户并输入您的 AWS 账户 电子邮件地址，以账户所有者的身份登录。AWS Management Console在下一页上，输入您的密码。

   要获取使用根用户登录方面的帮助，请参阅《AWS 登录 用户指南》中的 Signing in as the root user。
2. 为您的根用户启用多重身份验证（MFA）。

   有关说明，请参阅 I A M 用户指南中的为 AWS 账户 根用户（控制台）启用虚拟 MFA 设备。

创建具有管理访问权限的用户

1. 启用 IAM Identity Center。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的 Enabling AWS IAM Identity Center。
2. 在 IAM Identity Center 中，为用户授予管理访问权限。

   有关使用 IAM Identity Center 目录 作为身份源的教程，请参阅《用户指南》 IAM Identity Center 目录中的使用默认设置配置AWS IAM Identity Center 用户访问权限。

以具有管理访问权限的用户身份登录

- 要使用您的 IAM Identity Center 用户身份登录，请使用您在创建 IAM Identity Center 用户时发送到您的电子邮件地址的登录网址。

  有关使用 IAM Identity Center 用户登录的帮助，请参阅AWS 登录 用户指南中的登录 AWS 访问门户。

将访问权限分配给其他用户

1. 在 IAM Identity Center 中，创建一个权限集，该权限集遵循应用最低权限的最佳做法。

有关说明，请参阅《AWS IAM Identity Center 用户指南》中的 Create a permission set。

2. 将用户分配到一个组，然后为该组分配单点登录访问权限。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的 Add groups。

要提供访问权限，请为您的用户、组或角色添加权限：

- 中的用户和群组 AWS IAM Identity Center：

  创建权限集合。按照《AWS IAM Identity Center 用户指南》中创建权限集的说明进行操作。
- 通过身份提供商在 IAM 中托管的用户：

  创建适用于身份联合验证的角色。按照《IAM 用户指南》中针对第三方身份提供商创建角色（联合身份验证）的说明进行操作。
- IAM 用户：
  - 创建您的用户可以担任的角色。按照《IAM 用户指南》中为 IAM 用户创建角色的说明进行操作。
  - （不推荐使用）将策略直接附加到用户或将用户添加到用户组。按照《IAM 用户指南》中向用户添加权限（控制台）中的说明进行操作。

注册您的 MCU 主板 AWS IoT

您的董事会必须注册 AWS IoT 才能与 AWS 云端通信。要注册您的董事会 AWS IoT，您必须：

一项 AWS IoT 政策

该 AWS IoT 政策授予您的设备访问 AWS IoT 资源的权限。它存储在 AWS 云端。

一 AWS IoT 件事

一个 AWS IoT 东西可以让你在中管理你的设备 AWS IoT。它存储在 AWS 云端。

私有密钥和 X.509 证书

私钥和证书允许您的设备进行身份验证 AWS IoT。

要注册主板，请按照以下过程操作。

创建 AWS IoT 策略

1. 要创建 IAM 策略，您必须知道您的 AWS 地区和 AWS 账号。

要查找您的 AWS 账号，请打开AWS 管理控制台，找到并展开右上角账户名称下方的菜单，然后选择我的账户。您的账户 ID 显示在 Account Settings (账户设置) 下。

要查找您 AWS 账户所在 AWS 的地区，请使用 AWS Command Line Interface。要安装 AWS CLI，请按照《AWS Command Line Interface 用户指南》中的说明进行操作。安装后 AWS CLI，打开命令提示符窗口并输入以下命令：

```
aws iot describe-endpoint --endpoint-type=iot:Data-ATS
```

输出应该如下所示：

```
{
    "endpointAddress": "xxxxxxxxxxxxxx-ats.iot.us-west-2.amazonaws.com"
}
```

在此示例中，区域为 us-west-2。

> **Note**
>
> 我们建议使用 ATS 端点，如示例所示。

2. 浏览至 AWS IoT 控制台。
3. 在导航窗格中依次选择安全、策略和创建。
4. 输入用于标识您的策略的名称。
5. 在添加语句部分中，选择高级模式。将以下 JSON 复制并粘贴到策略编辑器窗口中。*aws-account*用您的 AWS 地区*aws-region*和账户 ID 替换和。

```
{
    "Version": "2012-10-17",
    "Statement": [
    {
        "Effect": "Allow",
        "Action": "iot:Connect",
        "Resource":"arn:aws:iot:aws-region:aws-account-id:*"
    },
    {
        "Effect": "Allow",
        "Action": "iot:Publish",
```

```
        "Resource": "arn:aws:iot:aws-region:aws-account-id:*"
    },
    {
         "Effect": "Allow",
         "Action": "iot:Subscribe",
         "Resource": "arn:aws:iot:aws-region:aws-account-id:*"
    },
    {
         "Effect": "Allow",
         "Action": "iot:Receive",
         "Resource": "arn:aws:iot:aws-region:aws-account-id:*"
    }
    ]
}
```

此策略授予以下权限：

iot:Connect

向您的设备授予使用任何客户端 ID 连接 AWS IoT 消息代理的权限。

iot:Publish

授予设备在任何 MQTT 主题上发布 MQTT 消息的权限。

iot:Subscribe

授予设备订阅到任何 MQTT 主题筛选条件的权限。

iot:Receive

授予设备从 AWS IoT 消息代理接收有关任何 MQTT 主题的消息的权限。

6. 选择创建。

为设备创建 IoT 事物、私有密钥和证书

1. 浏览至 AWS IoT 控制台。
2. 在导航窗格中，选择管理，然后选择事物。
3. 如果您的账户中未注册任何 IoT 事物，此时将显示您还没有任何事物页面。如果您看到此页面，请选择注册事物。否则，选择 创建。
4. 在创建 AWS IoT 事物页面上，选择创建单个事物。
5. 在将设备添加到事物注册表页面上，输入您事物的名称，然后选择下一步。

6. 在添加事物的证书页面上的一键式创建证书下，选择创建证书。
7. 选择各项的下载链接来下载私有密钥和证书。
8. 选择激活来激活您的证书。必须先激活证书，然后才能使用它们 。
9. 选择 Att ach a policy，将策略附加到您的证书，该策略允许您的设备访问 AWS IoT 操作。
10. 选择您刚刚创建的策略，然后选择注册事物。

在您的董事会注册后 AWS IoT，您可以继续下载 FreeRTOS。

下载 FreeRTOS

你可以从 FreeRTOS 存储库中下载 FreeRTOS。 GitHub

下载 FreeRTOS 后，您可以继续配置 FreeRTOS 演示。

配置 FreeRTOS 演示

您需要在 FreeRTOS 目录中编辑一些配置文件，然后才能在主板上编译和运行演示。

配置您的 AWS IoT 终端节点

您必须向FreeRTOS提供终端节点，以便在 AWS IoT 您的主板上运行的应用程序可以向正确的端点发送请求。

1. 浏览至 AWS IoT 控制台。
2. 在左侧导航窗格中，选择 设置。

   您的 AWS IoT 终端节点显示在设备数据端点中。它应该类似于 *1234567890123*-ats.iot.*us-east-1*.amazonaws.com。记下此终端节点。
3. 在导航窗格中，选择管理，然后选择事物。

   您的设备应该有一个 AWS IoT 事物名称。记下此名称。
4. 打开 demos/include/aws_clientcredential.h。
5. 为以下常量指定值：

   - #define clientcredentialMQTT_BROKER_ENDPOINT "*Your AWS IoT endpoint*";
   - #define clientcredentialIOT_THING_NAME "*The AWS IoT thing name of your board*"

配置 Wi-Fi

如果您的主板通过 Wi-Fi 连接到 Internet，则必须向 FreeRTOS 提供 Wi-Fi 凭证才能连接到网络。如果主板不支持 Wi-Fi，您可以跳过这些步骤。

1. demos/include/aws_clientcredential.h.
2. 为以下 #define 常量指定值：

   - #define clientcredentialWIFI_SSID "*The SSID for your Wi-Fi network*"
   - #define clientcredentialWIFI_PASSWORD "*The password for your Wi-Fi network*"
   - #define clientcredentialWIFI_SECURITY *The security type of your Wi-Fi network*

     有效安全类型为：

     - eWiFiSecurityOpen（开放，不安全）
     - eWiFiSecurityWEP（WEP 安全性）
     - eWiFiSecurityWPA（WPA 安全性）
     - eWiFiSecurityWPA2（WPA2 安全）

格式化您的 AWS IoT 凭证

FreeRTOS 必须 AWS IoT 将证书和私钥与您注册的内容及其权限策略相关联，才能代表您的设备成功 AWS IoT 与之通信。

> **Note**
>
> 要配置您的 AWS IoT 凭据，您必须拥有注册设备时从 AWS IoT 控制台下载的私钥和证书。将设备注册为 AWS IoT 事物后，您可以从 AWS IoT 控制台检索设备证书，但无法检索私钥。

FreeRTOS 是 C 语言项目，证书和私有密钥必须经过专门格式化才能添加到项目中。

1. 在浏览器窗口中，打开 tools/certificate_configuration/CertificateConfigurator.html。
2. 在 Certificate PEM file (证书 PEM 文件) 下，选择您从 AWS IoT 控制台下载的 *ID*-certificate.pem.crt。

3. 在 Private Key PEM file (私有密钥 PEM 文件) 下，选择您从 AWS IoT 控制台下载的 *ID*-private.pem.key。
4. 选择 Generate and save aws_clientcredential_keys.h (生成并保存 aws_clientcredential_keys.h)，然后将文件保存到 demos/include 中。这将覆盖目录中的现有文件。

> **Note**
>
> 仅出于演示目的对证书和私有密钥进行了硬编码。生产级应用程序应将这些文件存储在安全位置。

完成 FreeRTOS 配置后，您可以继续参阅适用于您主板的入门指南，以便设置平台的硬件及其软件开发环境，然后在主板上编译并运行演示。有关主板特定的说明，请参阅主板特定的入门指南。本入门教程中使用的演示应用程序为 coreMQTT 双向身份验证演示，位于 demos/coreMQTT/mqtt_demo_mutual_auth.c 中。

问题排查入门

> **Important**
>
> 此页面指向已弃用的 Amazon FreeRTOS 存储库。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

以下主题可帮助您排查开始使用 FreeRTOS 时遇到的问题：

主题

- 一般的入门问题排查提示
- 安装终端仿真器

有关主板特定的问题排查，请参阅该主板的开始使用 FreeRTOS指南。

一般的入门问题排查提示

在我运行 Hello World 演示项目后， AWS IoT 控制台中没有显示任何消息。我应该怎么办？

尝试以下操作：

1. 打开终端窗口，查看示例的日志记录输出。这可以帮助您确定发生了什么错误。
2. 核查您的网络凭证是否有效。

运行演示时，终端上显示的日志被截断。我怎样才能增加它们的长度？

将正在运行的演示的 FreeRTOSconfig.h 文件中的 configLOGGING_MAX_MESSAGE_LENGTH 值增加到 255：

```
#define configLOGGING_MAX_MESSAGE_LENGTH    255
```

安装终端仿真器

终端仿真器可以帮助您诊断问题，或者验证设备代码是否正确运行。有多种终端仿真器可用于 Windows、macOS 和 Linux。

您必须先将主板连接到计算机，然后再尝试通过终端仿真器建立与主板的串行连接。

使用以下设置配置终端仿真器：

终端设置	值
波特率	115200
数据	8 位
奇偶校验	none
停止	1 位
流控制	none

查找主板的串行端口

如果不知道主板的串行端口，您可以从命令行或终端发布以下命令之一，返回连接到您主机的所有设备的串行端口：

Windows

```
chgport
```

Linux

```
ls /dev/tty*
```

macOS

```
ls /dev/cu.*
```

CMake 与 FreeRTOS 一起使用

> **Important**
>
> 此页面指向已弃用的 Amazon FreeRTOS 存储库。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

您可以使用 CMake 从 FreeRTOS 应用程序源代码生成项目生成文件，以及生成和运行源代码。

您还可以使用 IDE，在符合 FreeRTOS 要求的设备上编辑、调试、编译、刷写和运行代码。每个主板特定的入门指南提供了针对特定平台设置 IDE 的指南。如果您更喜欢在没有 IDE 的情况下工作，则可以使用其他第三方代码编辑和调试工具来开发和调试代码，然后使用 CMake 来生成和运行应用程序。

以下主板支持 CMake：

- Espressif-C ESP32 DevKit
- Espressif ESP-WROVER-KIT
- 英飞凌 00 XMC48 物联网连接套件
- Marvell MW32 0 AWS IoT 入门套件
- Marvell MW322 AWS IoT 入门套件
- 微芯片好奇号 PIC32 MZEF 捆绑包
- Nordic n RF5284 0 DK 开发套件
- STMicroelectronicsSTM32L4 探索套件物联网节点
- 德州仪器 CC322 0SF-LAUNCHXL
- Microsoft Windows Simulator

有关使用 FreeRTOS 的更多信息 CMake ，请参阅以下主题。

主题

- 先决条件
- 使用第三方代码编辑器和调试工具开发 FreeRTOS 应用程序
- 使用 FreeRTOS 构建 CMake

先决条件

请先确保主机符合以下先决条件，然后再继续：

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

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

  您可以 CMake 从 CMake.org 下载的二进制发行版。

  > **Note**
  >
  > 如果您下载的二进制发行版 CMake，请务必先将 CMake 可执行文件添加到 PATH 环境变量中，然后再 CMake 从命令行使用。

  你也可以 CMake 使用软件包管理器下载和安装，比如 macOS 上的 h omebrew，在 Windows 上使用 scoop 或 chocolatey。

  > **Note**
  >
  > 许多 Linux 发行版的软件包管理器中提供的软件包版本是 out-of-date。 CMake 如果你的发行版的包管理器没有提供最新版本的 CMake，你可以尝试其他包管理器，比如`linuxbrew`或`nix`。
- 必须具有兼容的本机构建系统。

  CMake 可以针对许多原生构建系统，包括 GNU Make 或 N inja。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。

使用第三方代码编辑器和调试工具开发 FreeRTOS 应用程序

您可以使用代码编辑器和调试扩展或者第三方调试工具来为 FreeRTOS 开发应用程序。

例如，如果您使用 Visual Studio Code 作为代码编辑器，则可以安装 Cortex-Debug VS Code 扩展作为调试程序。完成应用程序开发后，可以调用 CMake 命令行工具从 VS Code 中构建项目。有关使用 CMake 构建 FreeRTOS 应用程序的更多信息，请参阅。使用 FreeRTOS 构建 CMake

对于调试，您可以向 VS Code 提供类似于下文的调试配置：

```
"configurations": [
    {
        "name": "Cortex Debug",
        "cwd": "${workspaceRoot}",
        "executable": "./build/st/stm32l475_discovery/aws_demos.elf",
        "request": "launch",
        "type": "cortex-debug",
        "servertype": "stutil"
    }
]
```

使用 FreeRTOS 构建 CMake

CMake 默认情况下，将您的主机操作系统作为目标系统。要将其用于交叉编译， CMake 需要一个工具链文件，该文件指定要使用的编译器。在 FreeRTOS 中，我们在 *freertos*/tools/cmake/toolchains 中提供了默认工具链文件。向其提供此文件的方法 CMake 取决于您使用的是 CMake命令行界面还是 GUI。有关更多详细信息，请按照以下生成生成文件（CMake 命令行工具）说明进行操作。有关中交叉编译的更多信息 CMake，请参见CrossCompiling官方 CMake 维基。

生成 CMake基于项目的方法

1. 运行 CMake 为原生编译系统（例如 Make 或 Ninja）生成生成文件。

您可以使用CMake 命令行工具或 CMake GUI 为原生构建系统生成生成文件。

有关生成 FreeRTOS 构建文件的信息，请参阅生成生成文件（CMake 命令行工具）和生成生成文件 (CMake GUI)。

2. 调用本机构建系统，将项目制作为可执行文件。

有关如何创建 FreeRTOS 构建文件的信息，请参阅从生成的构建文件构建 FreeRTOS。

生成生成文件（CMake 命令行工具）

你可以使用 CMake 命令行工具 (cmake) 为 FreeRTOS 生成构建文件。要生成构建文件，您需要指定目标主板、编译器以及源代码和构建目录的位置。

您可以对 cmake 使用以下选项：

- `-DVENDOR` – 指定目标主板。
- `-DCOMPILER` – 指定编译器。
- `-S` – 指定源代码的位置。
- `-B` –指定生成的构建文件的位置。

> **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，如果供应商是德州仪器，主板是 CC322 0 Launchpad，编译器是 GCC for ARM，则可以发出以下命令将源文件从当前目录构建到名为的目录中：*build-directory*

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

> **Note**
>
> 如果您使用的是 Windows，则必须指定本机生成系统，因为默认情况下 CMake 使用 Visual Studio。例如：

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

或：

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

如果工具链文件没有为您的编译器使用绝对路径，并且您没有将编译器添加到PATH环境变量中，则 CMake 可能无法找到它。要确保 CMake 找到你的工具链文件，你可以使用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，在编译选项中 CMake 添加调试标志，并使用调试符号构建 FreeRTOS。

```
# Build with debug symbols
```

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

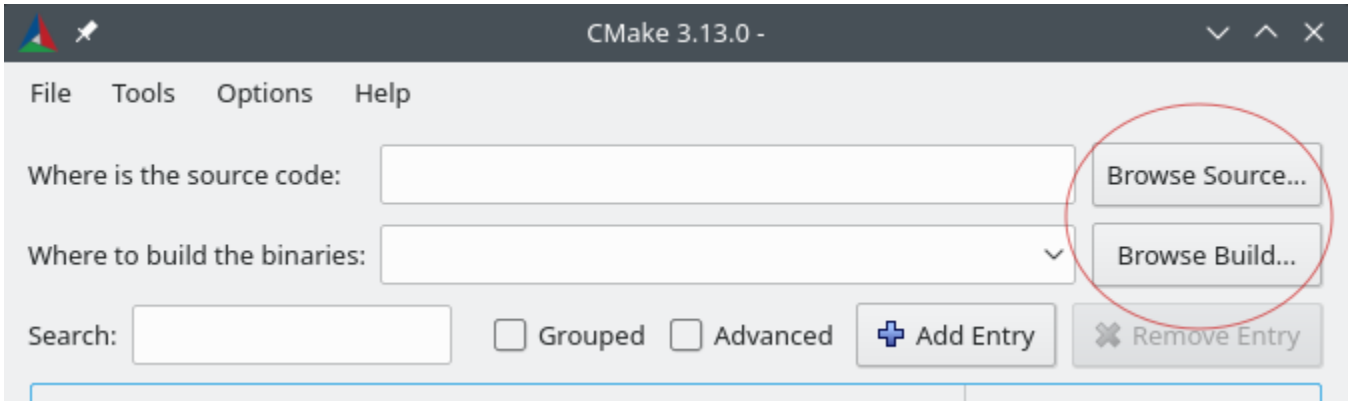
也可以将 CMAKE_BUILD_TYPE 设置为 release，将优化标志添加到编译选项。

生成生成文件 (CMake GUI)

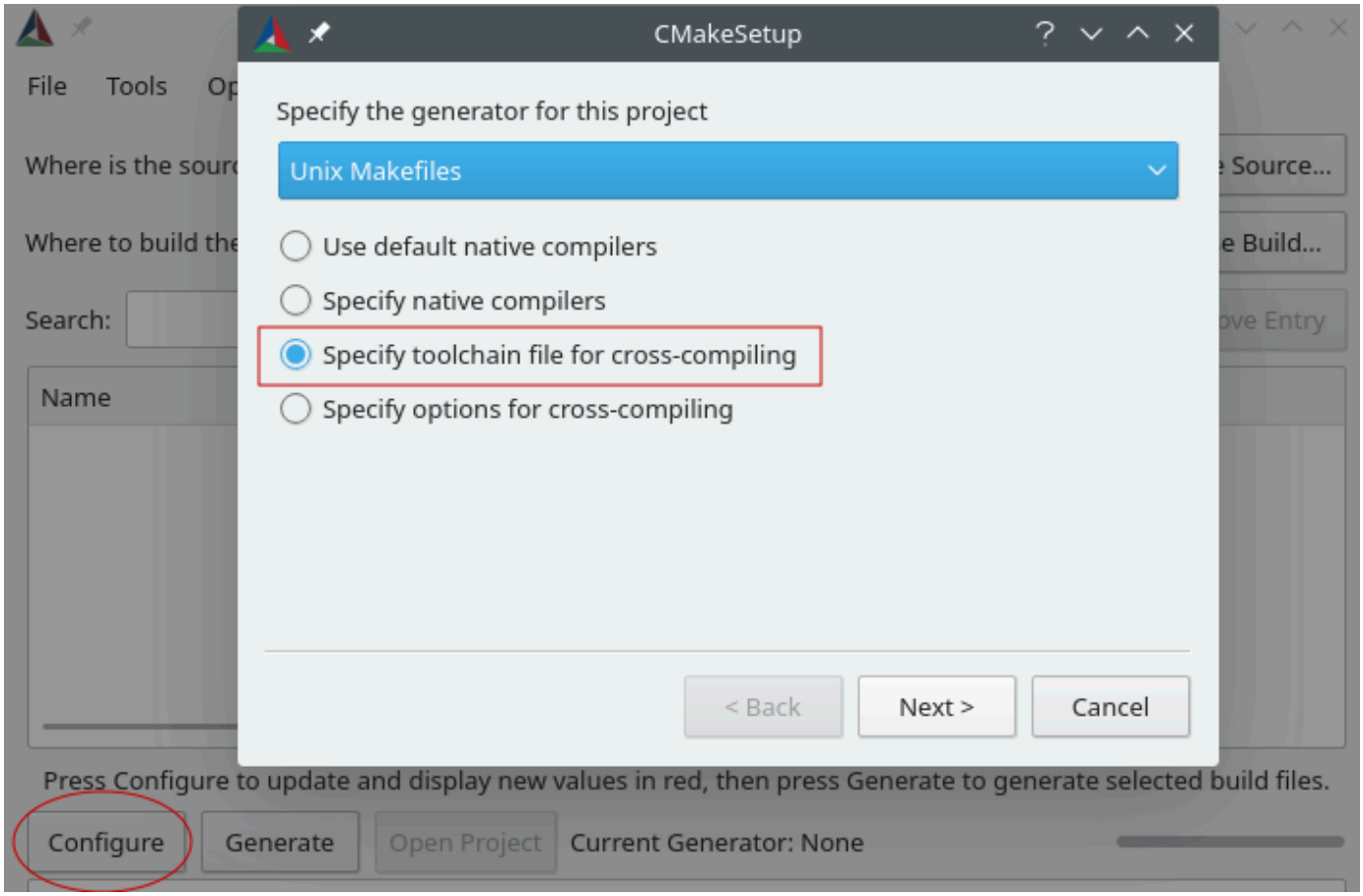
你可以使用 CMake GUI 来生成 FreeRTOS 编译文件。

使用 CMake GUI 生成生成文件

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，查找并选择要用于构建所生成的构建文件的构建系统。如果您未看到弹出窗口，则可能正在重用现有的构建目录。在这种情况下，请从 “文件” 菜单中 CMake 选择 “删除缓存” 来删除缓存。



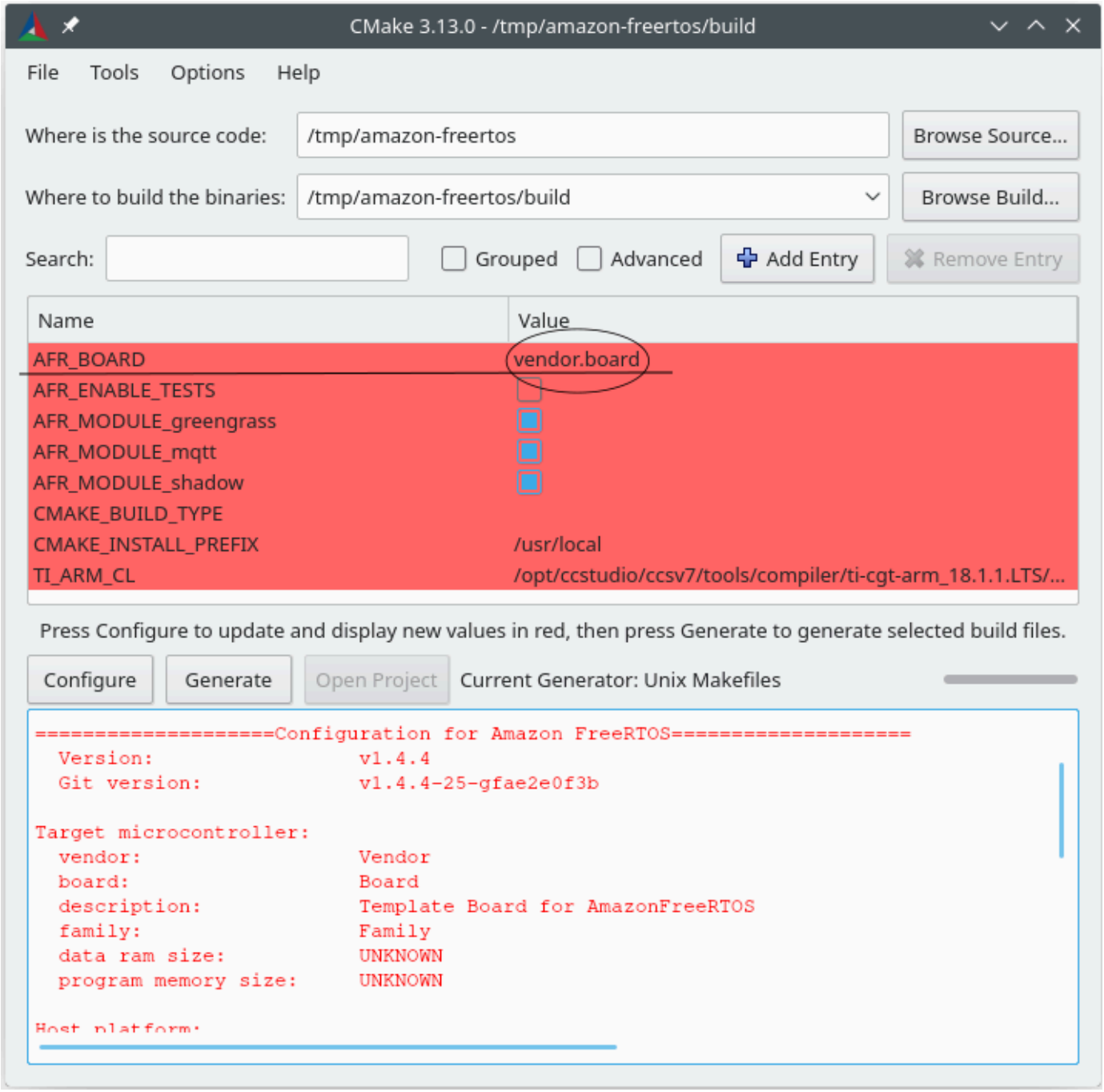
4. 选择 Specify toolchain file for cross-compiling (指定用于交叉编译的工具链文件)，然后选择 Next (下一步)。
5. 选择工具链文件（例如，*freertos*/tools/cmake/toolchains/arm-ti.cmake），然后选择 Finish (完成)。

   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。

> **Note**
>
> 如果您看到以下错误：
>
> ```
> CMake Error at tools/cmake/toolchains/find_compiler.cmake:23 (message):
> Compiler not found, you can specify search path with AFR_TOOLCHAIN_PATH.
> ```

这意味着编译器不在您的 PATH 环境变量中。你可以在 GUI 中设置AFR_TOOLCHAIN_PATH变量来告诉你 CMake 在哪里安装了编译器。如果您未看到 AFR_TOOLCHAIN_PATH 变量，请选择 Add Entry (添加条目)。在弹出窗口中，在 Name (名称) 下，键入 **AFR_TOOLCHAIN_PATH**。在 Compiler Path (编译器路径) 下，键入编译器的路径，例如 `C:/toolchains/arm-none-eabi-gcc`。

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 “生成”。 CMake 生成生成系统文件（例如 makefile 或 ninja 文件），这些文件将出现在您在第一步中指定的构建目录中。按照下一节中的说明生成二进制映像。

从生成的构建文件构建 FreeRTOS

使用本机构建系统构建

可以使用本机构建系统构建 FreeRTOS，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_dir>，并且您使用 Make 作为本机构建系统，则可运行以下命令：

```
cd <build_dir>
make -j4
```

搭建用 CMake

你也可以使用 CMake 命令行工具来构建 FreeRTOS。 CMake 为调用原生构建系统提供了一个抽象层。例如：

```
cmake --build build_dir
```

以下是 CMake 命令行工具编译模式的其他一些常见用法：

```
# 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
```

有关 CMake 构建模式的更多信息，请参阅CMake 文档。

开发人员模式密钥预置

> **Important**
>
> 此页面指向已弃用的 Amazon FreeRTOS 存储库。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

简介

本部分介绍了两个选项，可使 IoT 设备获得受信任的 X.509 客户端证书以进行实验室测试。由于设备的功能差异，可能会支持或不支持各种与预置相关的操作，包括板载 ECDSA 密钥生成、私有密钥导入和 X.509 证书注册。此外，不同的使用案例需要不同级别的密钥保护，从板载闪存存储到使用专用加密硬件都有涉及。本部分提供了在设备的加密功能范围内工作的逻辑。

选项 #1: 从中导入私钥 AWS IoT

出于实验室测试目的，如果您的设备允许导入私有密钥，请按照 配置 FreeRTOS 演示 中的说明操作。

选项 2：板载私有密钥生成

如果您的设备具有安全元件，或者如果您想要生成自己的设备密钥对和证书，请按照此处的说明操作。

初始配置

首先，执行中的步骤配置 FreeRTOS 演示，但跳过最后一步（也就是说，不要执行格式化 AWS IoT 凭证）。最终结果应该是 `demos/include/aws_clientcredential.h` 文件已使用您的设置完成了更新，但 `demos/include/aws_clientcredential_keys.h` 文件没有更新。

演示项目配置

按照主板特定的入门指南中有关您的主板的指南中所述，打开 coreMQTT 双向身份验证演示。在该项目中，打开文件 `aws_dev_mode_key_provisioning.c` 并将默认设置为零的 `keyprovisioningFORCE_GENERATE_NEW_KEY_PAIR` 定义更改为一：

```
#define keyprovisioningFORCE_GENERATE_NEW_KEY_PAIR 1
```

然后构建并运行演示项目并继续下一步。

公有密钥提取

由于设备尚未预置私有密钥和客户端证书，该演示将无法通过 AWS IoT的身份验证。但是，coreMQTT 双向身份验证演示首先会运行开发人员模式密钥预置，如果还没有私有密钥，则会创建一个。您应该在串行控制台输出的开头附近看到类似以下内容的信息：

```
7 910 [IP-task] Device public key, 91 bytes:
3059 3013 0607 2a86 48ce 3d02 0106 082a
8648 ce3d 0301 0703 4200 04cd 6569 ceb8
1bb9 1e72 339f e8cf 60ef 0f9f b473 33ac
6f19 1813 6999 3fa0 c293 5fae 08f1 1ad0
41b7 345c e746 1046 228e 5a5f d787 d571
dcb2 4e8d 75b3 2586 e2cc 0c
```

将六行密钥字节复制到名为 `DevicePublicKeyAsciiHex.txt` 的文件中。然后使用命令行工具“xxd”将十六进制字节解析为二进制：

```
xxd -r -ps DevicePublicKeyAsciiHex.txt DevicePublicKeyDer.bin
```

使用“openssl”将二进制编码 (DER) 设备公有密钥格式化为 PEM：

```
openssl ec -inform der -in DevicePublicKeyDer.bin -pubin -pubout -outform pem -out
 DevicePublicKey.pem
```

不要忘记禁用上面启用的临时密钥生成设置。否则，该设备将创建另一个密钥对，您将不得不重复前面的步骤：

```
#define keyprovisioningFORCE_GENERATE_NEW_KEY_PAIR 0
```

公有密钥基础设施设置

按照注册 CA 证书中的说明为设备实验室证书创建证书层次结构。在执行使用 CA 证书创建设备证书 部分中描述的序列之前停止。

在这种情况下，设备将不会签署证书请求（即证书服务请求或 CSR），因为创建和签署 CSR 所需的 X.509 编码逻辑已从 FreeRTOS 演示项目中排除，以减少 ROM 大小。出于实验室测试目的，请改为在您的工作站上创建一个私有密钥并使用它来签署 CSR。

```
openssl genrsa -out tempCsrSigner.key 2048
openssl req -new -key tempCsrSigner.key -out deviceCert.csr
```

创建证书颁发机构并向其注册后 AWS IoT，使用以下命令根据上一步中签署的设备 CSR 颁发客户端证书：

```
openssl x509 -req -in deviceCert.csr -CA rootCA.pem -CAkey rootCA.key
 -CAcreateserial -out deviceCert.pem -days 500 -sha256 -force_pubkey
 DevicePublicKey.pem
```

尽管 CSR 是使用临时私有密钥签署的，但颁发的证书只能与实际的设备私有密钥一起使用。如果您将 CSR 签署人密钥存储在单独的硬件中，并配置您的证书颁发机构，使其仅为已由该特定密钥签署的请求颁发证书，则可以在生产中使用相同的机制。该密钥也应继续由指定的管理员控制。

证书导入

颁发证书后，下一步是将其导入到您的设备中。您还需要导入您的证书颁发机构 (CA) 证书，因为在使用 JITP 时，首次进行身份验证是成功 AWS IoT 进行认证所必需的。在项目中的 `aws_clientcredential_keys.h` 文件中，将 `keyCLIENT_CERTIFICATE_PEM` 宏设置为 deviceCert.pem 的内容，并将 `keyJITR_DEVICE_CERTIFICATE_AUTHORITY_PEM` 宏设置为 `rootCA.pem` 的内容。

设备授权

按照使用自己的证书中所述导`deviceCert.pem`入到 AWS IoT 注册表。您必须创建一个新 AWS IoT 事物，将待处理的证书和策略附加到您的事物上，然后将该证书标记为已激活。所有这些步骤都可以在 AWS IoT 控制台中手动执行。

在新客户端证书处于活动状态并与某个事物和策略关联后，请再次运行 coreMQTT 双向身份验证演示。这次，与 AWS IoT MQTT 代理的连接将成功。

主板特定的入门指南

> **Important**
>
> 此页面指向已弃用的 Amazon FreeRTOS 存储库。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

在完成初始步骤后，请参阅主板指南以查看有关 FreeRTOS 入门的主板特定说明：

- 开始使用 Cypress CYW9439 07 AEVAL1 F 开发套件

- 开始使用 Cypress CYW9549 07 AEVAL1 F 开发套件
- 赛普拉斯 CY8 CKIT-064S0S2-4343W 套件入门
- 英飞凌 00 XMC48 物联网连接套件入门
- 入门套件 MW32x AWS IoT 入门
- MediaTek MT7697Hx 开发套件入门
- Microchip Curiosity PIC32 MZ EF 入门
- 新 NuMaker-IoT-M腾 487 入门
- 开始使用恩智浦 LPC54 018物联网模块
- 开始使用适用于 N-2MB 的瑞萨电子入门套件+ RX65
- STMicroelectronics STM32L4 探索套件物联网节点入门
- 德州仪器 CC322 0SF-LAUNCHXL 入门
- Windows 设备模拟器入门
- Xilinx Avnet MicroZed 工业物联网套件入门

Note

对于以下独立的 FreeRTOS 入门指南，您无需完成初始步骤：

- 带有 Windows 模拟器的 Microchip ATECC6 08A Secure Element 入门
- 开始使用 Espressif ESP32-C 和 DevKit ESP-WROVER-KIT
- 乐鑫-WROOM-32SE ESP32 入门
- 乐鑫-S ESP32 2 入门
- 开始使用英飞凌 OPTIGA Trust X 和 00 XMC48 物联网连接套件
- Nordic n RF5284 0-DK 入门

开始使用 Cypress CYW9439 07 AEVAL1 F 开发套件

Important

该参考集成托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

本教程提供了 Cypress CYW9439 07 AEVAL1 F 开发套件的入门说明。如果您没有 Cypress CYW9439 07 AEVAL1 F 开发套件，请访问 AWS 合作伙伴设备目录从我们的合作伙伴处购买。

> **Note**
>
> 本教程将指导您完成设置和运行 coreMQTT 双向身份验证演示。此主板的 FreeRTOS 移植当前不支持 TCP 服务器和客户端演示。

在开始之前，您必须进行配置 AWS IoT 并下载 FreeRTOS 才能将您的设备连接到云端。 AWS 有关说明，请参阅初始步骤：

> **Important**
>
> - 在本主题中，FreeRTOS 下载目录的路径称为 *`freertos`*。
> - *`freertos`* 路径中的空格字符可能会导致构建失败。克隆或复制存储库时，请确保您创建的路径不包含空格字符。
> - Microsoft Windows 上的文件路径最大长度为 260 个字符。FreeRTOS 下载目录路径过长可能会导致构建操作失败。
> - 由于源代码可能包含符号链接，因此，如果您使用 Windows 提取存档，则可能必须：
>   - 启用开发者模式，或者，
>   - 使用已提升为管理员的控制台。
>
>   这样，Windows 就可以在提取存档文件时正确创建符号链接。否则，符号链接将作为普通文件写入，该文件会将符号链接的路径作为文本，或者是空文件。有关更多信息，请参阅博客文章 Windows 10 中的符号链接。
>
>   如果您在 Windows 下使用 Git，则必须启用开发者模式，或者必须：
>   - 使用以下命令将 `core.symlinks` 设置为 true：
>
>     ```
>     git config --global core.symlinks true
>     ```
>   - 每当您使用写入系统的 git 命令（例如，git pull、git clone 和 git submodule update --init --recursive）时，请使用具有管理员权限的控制台。
> - 如中所述下载 FreeRTOS，赛普拉斯的 FreeRTOS 端口目前仅在上可用。 GitHub

概览

本教程包含有关以下入门步骤的说明：

1. 在主机上安装软件来开发和调试微控制器主板的嵌入式应用程序。
2. 将 FreeRTOS 演示应用程序交叉编译为二进制映像。
3. 将应用程序二进制映像加载到您的主板上，然后运行该应用程序。
4. 跨串行连接与主板上运行的应用程序进行交互，以便进行监视和调试。

设置开发环境

下载并安装 WICED Studio 开发工具包

在本入门指南中，您可以使用 Cypress WICED Studio 开发工具包通过 FreeRTOS 演示对主板进行编程。访问 WICED Software 网站，从 Cypress 下载 WICED Studio 开发工具包。您必须注册免费的 Cypress 账户才能下载该软件。WICED Studio 开发工具包与 Windows、macOS 和 Linux 操作系统兼容。

> **Note**
>
> 某些操作系统需要额外的安装步骤。确保您阅读并遵循所安装的操作系统和 WICED Studio 版本的所有安装说明。

设置环境变量

在使用 WICED Studio 对主板进行编程之前，必须为 WICED Studio 开发工具包安装目录创建一个环境变量。如果在创建变量时 WICED Studio 正在运行，则您需要在设置变量后重新启动该应用程序。

> **Note**
>
> WICED Studio 安装程序会在您的计算机上创建两个名为 `WICED-Studio-m.n` 的单独文件夹，其中 m 和 n 分别是主要版本号和次要版本号。本文档假定文件夹名称为 `WICED-Studio-6.2`，但请确保为您安装的版本使用正确的名称。定义 `WICED_STUDIO_SDK_PATH` 环境变量时，请确保指定 WICED Studio 开发工具包的完整安装路径，而不是 WICED Studio IDE 的安装路径。在 Windows 和 macOS 中，开发工具包的 `WICED-Studio-m.n` 文件夹默认情况下在 `Documents` 文件夹中创建。

在 Windows 上创建环境变量

1. 在控制面板上，依次选择系统和高级系统设置。
2. 在高级选项卡上，选择环境变量。
3. 在用户变量下，选择新建。
4. 对于变量名称，输入 **WICED_STUDIO_SDK_PATH**。对于变量值，输入 WICED Studio 开发工具包安装目录。

在 Linux 或 macOS 中创建环境变量

1. 打开您计算机上的 /etc/profile 文件，然后将以下内容添加到该文件最后一行：

```
export WICED_STUDIO_SDK_PATH=installation-path/WICED-Studio-6.2
```

2. 重启计算机。
3. 打开一个终端，并运行以下命令：

```
cd freertos/vendors/cypress/WICED_SDK
```

```
perl platform_adjust_make.pl
```

```
chmod +x make
```

建立串行连接

在主机和主板之间建立串行连接

1. 使用 USB Standard-A 到 Micro-B 线缆将主板连接到主机。
2. 标识与主机上主板的连接的 USB 串行端口号。
3. 启动一个串行终端，使用以下设置建立连接：

   - 波特率：115200
   - 数据：8 位
   - 奇偶校验：无
   - 停止位：1

- 流控制：无

有关安装终端和设置串行连接的更多信息，请参阅安装终端仿真器。

在云上监控 MQTT 消息

在运行 FreeRTOS 演示项目之前，您可以在控制台中 AWS IoT 设置 MQTT 客户端，以监控您的设备发送到云端的消息。 AWS

使用 MQTT 客户端订阅 M AWS IoT QTT 主题

1. 登录 AWS IoT 控制台。
2. 在导航窗格中选择测试，然后选择 MQTT 测试客户端，以便打开 MQTT 客户端。
3. 在 Subscription topic (订阅主题)中，输入 ***your-thing-name*/example/topic**，然后选择 Subscribe to topic (订阅主题)。

构建并运行 FreeRTOS 演示项目

建立与主板的串行连接后，您可以构建 FreeRTOS 演示项目，将演示刷入主板，然后运行演示。

在 WICED Studio 中构建并运行 FreeRTOS 演示项目

1. 启动 WICED Studio。
2. 从文件菜单中，选择导入。展开 `General` 文件夹，选择 Existing Projects into Workspace (现有项目到工作区)，然后选择 Next (下一步)。
3. 在 Select root directory (选择根目录) 中，选择 Browse... (浏览...)，导航到路径 *freertos*`/projects/cypress/CYW943907AEVAL1F/wicedstudio`，然后选择 OK (确定)。
4. 在 Projects (项目) 下，选中与 aws_demo 项目对应的复选框。选择 Finish (完成) 以导入项目。目标项目 aws_demo 应显示在 Make Target (制作目标) 窗口中。
5. 展开 WICED Platform (WICED 平台) 菜单，然后选择 WICED Filters off (WICED 滤除)。
6. 在 Make Target (制作目标) 窗口中，展开 aws_demo，右键单击 `demo.aws_demo` 文件，然后选择 Build Target (生成目标) 以生成演示并将其下载到您的主板。生成演示并下载到主板后，该演示应自动运行。

故障排除

- 如果您使用的是 Windows，则在构建和运行演示项目时可能会看到以下错误：

```
: recipe for target 'download_dct' failed
make.exe[1]: *** [download_dct] Error 1
```

要排查该错误，请执行以下操作：

1. 浏览找到 *WICED-Studio-SDK-PATH*\WICED-Studio-6.2\43xxx_Wi-Fi\tools\OpenOCD\Win32，然后双击 openocd-all-brcm-libftdi.exe。
2. 浏览找到 *WICED-Studio-SDK-PATH*\WICED-Studio-6.2\43xxx_Wi-Fi\tools\drivers\CYW9WCD1EVAL1，然后双击 InstallDriver.exe。

- 如果您使用的是 Linux 或 macOS，则在构建和运行演示项目时可能会看到以下错误：

```
make[1]: *** [download_dct] Error 127
```

要排查此错误，请使用以下命令更新 libusb-dev 程序包：

```
sudo apt-get install libusb-dev
```

有关 FreeRTOS 入门的常规故障排除信息，请参阅问题排查入门。

开始使用 Cypress CYW9549 07 AEVAL1 F 开发套件

> **Important**
>
> 该参考集成托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

本教程提供了 Cypress CYW9549 07 AEVAL1 F 开发套件的入门说明。如果您没有 Cypress CYW9549 07 AEVAL1 F 开发套件，请访问 AWS 合作伙伴设备目录从我们的合作伙伴处购买。

> **Note**
>
> 本教程将指导您完成设置和运行 coreMQTT 双向身份验证演示。此主板的 FreeRTOS 移植当前不支持 TCP 服务器和客户端演示。

在开始之前，您必须进行配置 AWS IoT 并下载 FreeRTOS 才能将您的设备连接到云端。 AWS 有关说明，请参阅初始步骤：在本教程中，FreeRTOS 下载目录的路径称为 *freertos*。

Important

- 在本主题中，FreeRTOS 下载目录的路径称为 *freertos*。
- *freertos* 路径中的空格字符可能会导致构建失败。克隆或复制存储库时，请确保您创建的路径不包含空格字符。
- Microsoft Windows 上的文件路径最大长度为 260 个字符。FreeRTOS 下载目录路径过长可能会导致构建操作失败。
- 由于源代码可能包含符号链接，因此，如果您使用 Windows 提取存档，则可能必须：
  - 启用开发者模式，或者，
  - 使用已提升为管理员的控制台。

  这样，Windows 就可以在提取存档文件时正确创建符号链接。否则，符号链接将作为普通文件写入，该文件会将符号链接的路径作为文本，或者是空文件。有关更多信息，请参阅博客文章 Windows 10 中的符号链接。

  如果您在 Windows 下使用 Git，则必须启用开发者模式，或者必须：
  - 使用以下命令将 `core.symlinks` 设置为 true：

    ```
    git config --global core.symlinks true
    ```
  - 每当您使用写入系统的 git 命令（例如，git pull、git clone 和 git submodule update --init --recursive）时，请使用具有管理员权限的控制台。
- 如中所述下载 FreeRTOS，赛普拉斯的 FreeRTOS 端口目前仅在上可用。 GitHub

概览

本教程包含有关以下入门步骤的说明：

1. 在主机上安装软件来开发和调试微控制器主板的嵌入式应用程序。
2. 将 FreeRTOS 演示应用程序交叉编译为二进制映像。
3. 将应用程序二进制映像加载到您的主板上，然后运行该应用程序。
4. 跨串行连接与主板上运行的应用程序进行交互，以便进行监视和调试。

设置开发环境

下载并安装 WICED Studio 开发工具包

在本入门指南中，您可以使用 Cypress WICED Studio 开发工具包通过 FreeRTOS 演示对主板进行编程。访问 WICED Software 网站，从 Cypress 下载 WICED Studio 开发工具包。您必须注册免费的 Cypress 账户才能下载该软件。WICED Studio 开发工具包与 Windows、macOS 和 Linux 操作系统兼容。

> **Note**
>
> 某些操作系统需要额外的安装步骤。确保您阅读并遵循所安装的操作系统和 WICED Studio 版本的所有安装说明。

设置环境变量

在使用 WICED Studio 对主板进行编程之前，必须为 WICED Studio 开发工具包安装目录创建一个环境变量。如果在创建变量时 WICED Studio 正在运行，则您需要在设置变量后重新启动该应用程序。

> **Note**
>
> WICED Studio 安装程序会在您的计算机上创建两个名为 `WICED-Studio-m.n` 的单独文件夹，其中 m 和 n 分别是主要版本号和次要版本号。本文档假定文件夹名称为 `WICED-Studio-6.2`，但请确保为您安装的版本使用正确的名称。定义 `WICED_STUDIO_SDK_PATH` 环境变量时，请确保指定 WICED Studio 开发工具包的完整安装路径，而不是 WICED Studio IDE 的安装路径。在 Windows 和 macOS 中，开发工具包的 `WICED-Studio-m.n` 文件夹默认情况下在 `Documents` 文件夹中创建。

在 Windows 上创建环境变量

1. 在控制面板上，依次选择系统和高级系统设置。
2. 在高级选项卡上，选择环境变量。
3. 在用户变量下，选择新建。
4. 对于变量名称，输入 **`WICED_STUDIO_SDK_PATH`**。对于变量值，输入 WICED Studio 开发工具包安装目录。

在 Linux 或 macOS 中创建环境变量

1. 打开您计算机上的 /etc/profile 文件，然后将以下内容添加到该文件最后一行：

```
export WICED_STUDIO_SDK_PATH=installation-path/WICED-Studio-6.2
```

2. 重启计算机。
3. 打开一个终端，并运行以下命令：

```
cd freertos/vendors/cypress/WICED_SDK
```

```
perl platform_adjust_make.pl
```

```
chmod +x make
```

建立串行连接

在主机和主板之间建立串行连接

1. 使用 USB Standard-A 到 Micro-B 线缆将主板连接到主机。
2. 标识与主机上主板的连接的 USB 串行端口号。
3. 启动一个串行终端，使用以下设置建立连接：
   - 波特率：115200
   - 数据：8 位
   - 奇偶校验：无
   - 停止位：1
   - 流控制：无

有关安装终端和设置串行连接的更多信息，请参阅安装终端仿真器。

在云上监控 MQTT 消息

在运行 FreeRTOS 演示项目之前，您可以在控制台中 AWS IoT 设置 MQTT 客户端，以监控您的设备发送到云端的消息。 AWS

使用 MQTT 客户端订阅 M AWS IoT QTT 主题

1. 登录 AWS IoT 控制台。
2. 在导航窗格中选择测试，然后选择 MQTT 测试客户端，以便打开 MQTT 客户端。
3. 在 Subscription topic (订阅主题)中，输入 ***your-thing-name*/example/topic**，然后选择 Subscribe to topic (订阅主题)。

构建并运行 FreeRTOS 演示项目

建立与主板的串行连接后，您可以构建 FreeRTOS 演示项目，将演示刷入主板，然后运行演示。

在 WICED Studio 中构建并运行 FreeRTOS 演示项目

1. 启动 WICED Studio。
2. 从文件菜单中，选择导入。展开 `General` 文件夹，选择 Existing Projects into Workspace (现有项目到工作区)，然后选择 Next (下一步)。
3. 在 Select root directory (选择根目录) 中，选择 Browse... (浏览...)，导航到路径 *freertos*`/projects/cypress/CYW954907AEVAL1F/wicedstudio`，然后选择 OK (确定)。
4. 在 Projects (项目) 下，选中与 aws_demo 项目对应的复选框。选择 Finish (完成) 以导入项目。目标项目 aws_demo 应显示在 Make Target (制作目标) 窗口中。
5. 展开 WICED Platform (WICED 平台) 菜单，然后选择 WICED Filters off (WICED 滤除)。
6. 在 Make Target (制作目标) 窗口中，展开 aws_demo，右键单击 `demo.aws_demo` 文件，然后选择 Build Target (生成目标) 以生成演示并将其下载到您的主板。生成演示并下载到主板后，该演示应自动运行。

故障排除

- 如果您使用的是 Windows，则在构建和运行演示项目时可能会看到以下错误：

```
: recipe for target 'download_dct' failed
make.exe[1]: *** [download_dct] Error 1
```

要排查该错误，请执行以下操作：

1. 浏览找到 *WICED-Studio-SDK-PATH*`\WICED-Studio-6.2\43xxx_Wi-Fi\tools\OpenOCD\Win32`，然后双击 `openocd-all-brcm-libftdi.exe`。

2. 浏览找到 *WICED-Studio-SDK-PATH*\WICED-Studio-6.2\43xxx_Wi-Fi\tools\drivers\CYW9WCD1EVAL1，然后双击 InstallDriver.exe。

- 如果您使用的是 Linux 或 macOS，则在构建和运行演示项目时可能会看到以下错误：

```
make[1]: *** [download_dct] Error 127
```

要排查此错误，请使用以下命令更新 libusb-dev 程序包：

```
sudo apt-get install libusb-dev
```

有关 FreeRTOS 入门的常规故障排除信息，请参阅问题排查入门。

赛普拉斯 CY8 CKIT-064S0S2-4343W 套件入门

> **Important**
>
> 该参考集成托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

本教程提供了 CY8CKIT-064S0S2-4343W 套件的入门说明。如果您还没有此工具包，可以使用该链接购买一个。您也可以使用该链接访问有关此工具包的用户指南。

入门

在开始之前，您必须配置 AWS IoT 和 FreeRTOS 以将您的设备连接到云端。 AWS 有关说明，请参阅初始步骤。完成先决条件后，您将获得一个带有凭据的 FreeRTOS 软件包。 AWS IoT Core

> **Note**
>
> 在本教程中，在“初始步骤”一节中创建的 FreeRTOS 下载目录的路径称为 *freertos*。

设置开发环境

FreeRTOS 可以与 CMake a 或 Make 构建流程配合使用。你可以用 ModusToolbox 于 Make 构建流程。你可以使用随附的 Eclipse IDE ModusToolbox 或合作伙伴 IDE，例如 IAR ew-arm、Arm MDK 或微软 Visual Studio Code。Eclipse IDE 与 Windows、macOS 和 Linux 操作系统兼容。

在开始之前，请下载并安装最新的ModusToolbox 软件。有关更多信息，请参阅《ModusToolbox 安装指南》。

更新 ModusToolbox 2.1 或更早版本的工具

如果你使用 ModusToolbox 2.1 Eclipse IDE 对该套件进行编程，则需要更新 OpenOCD 和固件加载器工具。

在以下步骤中，*ModusToolbox* 的默认路径为：

- 在 Windows 上为 `C:\Users\`*user_name*`\ModusToolbox`。
- 在 Linux 上为 *user_home*`/ModusToolbox`，或您选择的提取存档文件的路径。
- 在 MacOS 上为您在向导中选择的卷中的“应用程序”文件夹下。

更新 OpenOCD

该工具包需要 Cypress OpenOCD 4.0.0 或更高版本才能成功对芯片进行擦除和编程。

更新 Cypress OpenOCD

1. 转到 Cypress OpenOCD 发布页面。
2. 下载适用于您的操作系统的存档文件 (Windows/Mac/Linux)。
3. 删除 *ModusToolbox*`/tools_2.x/openocd` 中的现有文件。
4. 将 *ModusToolbox*`/tools_2.x/openocd` 中的文件替换为您在上一步下载的存档中提取的内容。

更新固件加载程序

此工具包需要 Cypress 固件加载程序 3.0.0 或更高版本。

更新 Cypress 固件加载程序

1. 转到 Cypress 固件加载程序发布页面。

2. 下载适用于您的操作系统的存档文件 (Windows/Mac/Linux)。
3. 删除 *ModusToolbox*/tools_2.x/fw-loader 中的现有文件。
4. 将 *ModusToolbox*/tools_2.x/fw-loader 中的文件替换为您在上一步下载的存档中提取的内容。

或者，您可以使用 CMake 从 FreeRTOS 应用程序源代码生成项目构建文件，使用首选的构建工具生成项目，然后使用 OpenOCD 对套件进行编程。如果您更喜欢使用 GUI 工具按照 CMake 流程进行编程，请从 Cypress 编程解决方案网页下载并安装 Cypress Program menter。有关更多信息，请参阅 CMake 与 FreeRTOS 一起使用。

设置硬件

按照以下步骤设置工具包的硬件。

1. 配置工具包

   按照 CY8CKIT-064S0S2-4343W 套件配置指南中的说明安全地为其配置套件 AWS IoT。

   此套件需要 CySecureTools 3.1.0 或更高版本。
2. 设置串行连接

   a. 将工具包连接到主机。
   b. 该工具包的 USB 串行端口会在主机上自动枚举。识别端口号。在 Windows 中，您可以使用端口（COM 和 LPT）下的设备管理器进行识别。
   c. 启动一个串行终端，使用以下设置建立连接：

      - 波特率：115200
      - 数据：8 位
      - 奇偶校验：无
      - 停止位：1
      - 流控制：无

构建并运行 FreeRTOS 演示项目

在本节中，您将构建并运行演示。

1. 请务必按照 CY8CKIT-064S0S2-4343W 套件配置指南中的步骤进行操作。

2. 构建 FreeRTOS 演示。

   a. 打开 Eclipse IDE， ModusToolbox 然后选择或创建工作区。

   b. 从文件菜单中，选择导入。

      展开常规，选择现有项目到工作区，然后选择下一步。

   c. 在根目录中，输入 *freertos*/projects/cypress/CY8CKIT-064S0S2-4343W/mtb/aws_demos 并选择项目名称 aws_demos。默认情况下会将其选中。

   d. 选择完成，以便将项目导入工作区。

   e. 要构建应用程序，请执行以下操作之一：

      - 在快速面板中，选择构建 aws_demos 应用程序。
      - 选择项目，然后选择全部构建。

      确保编译过程没有错误。

3. 在云上监控 MQTT 消息

   在运行演示之前，您可以在 AWS IoT 控制台中设置 MQTT 客户端，以监控您的设备发送到 AWS 云端的消息。要通过 MQTT 客户端订阅 M AWS IoT QTT 主题，请按照以下步骤操作。

   a. 登录 AWS IoT 控制台。

   b. 在导航窗格中选择测试，然后选择 MQTT 测试客户端，以便打开 MQTT 客户端。

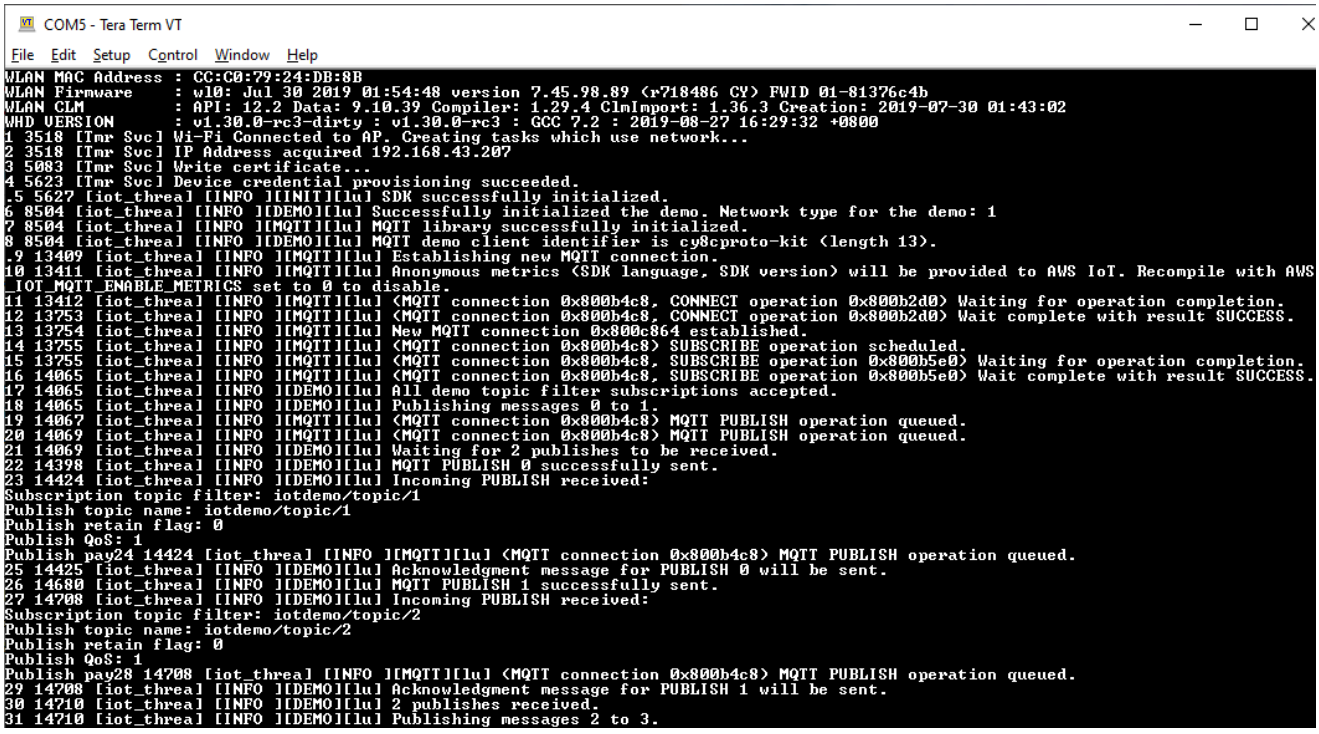
   c. 对于订阅主题，请输入 ***your-thing-name*/example/topic**，然后选择订阅主题。

4. 运行 FreeRTOS 演示项目

   a. 在工作区中选择项目 aws_demos。

   b. 在快速面板中，选择 aws_demos 程序 (3) KitProg。这会对开发主板进行编程，并在编程完成后开始运行演示应用程序。

   c. 您可以在串行终端上查看运行应用程序的状态。下图显示了终端输出的一部分。



MQTT 演示发布有关四个不同主题（`iotdemo/topic/`*n*，其中 n=1 到 4）的消息，并订阅所有这些主题以接收相同的消息。收到消息后，该演示会发布一条关于主题 `iotdemo/acknowledgements` 的确认消息。以下列表介绍终端输出中显示的调试消息，并引用了消息的序列号。在输出中，首先输出 WICED 主机驱动程序 (WHD) 的驱动程序详细信息（不带序列号）。

1. 1 到 4-设备连接到已配置的接入点 (AP)，并通过使用配置的端点和证书连接到 AWS 服务器进行配置。
2. 5 到 13 – CoreMQTT 库已初始化且设备已建立 MQTT 连接。
3. 14 到 17 – 设备订阅所有主题以接收发布的消息。
4. 18 到 30 – 设备发布两条消息并等待接收。收到每条消息后，设备都会发送一条确认消息。

相同的发布、接收和确认周期一直会持续到所有消息发布为止。在配置的周期数完成之前，每个周期都会发布两条消息。

5. CMake 与 FreeRTOS 一起使用

您也可以使用 CMake 来构建和运行演示应用程序。要设置 CMake 和原生构建系统，请参阅先决条件。

a. 使用以下命令可生成构建文件。使用 -DBOARD 选项指定目标主板。

```
cmake -DVENDOR=cypress -DBOARD=CY8CKIT_064S0S2_4343W -DCOMPILER=arm-gcc -
S freertos -B build_dir
```

如果你使用的是 Windows，则必须使用-G选项指定本机生成系统，因为默认情况下 CMake 使用 Visual Studio。

Example

```
cmake -DVENDOR=cypress -DBOARD=CY8CKIT_064S0S2_4343W -DCOMPILER=arm-gcc -
S freertos -B build_dir -G Ninja
```

如果arm-none-eabi-gcc不在你的 shell 路径中，你还需要设置AFR_TOOLCHAIN_PATH CMake 变量。

Example

```
 -DAFR_TOOLCHAIN_PATH=/home/user/opt/gcc-arm-none-eabi/bin
```

b. 使用以下命令使用生成项目 CMake。

```
cmake --build build_dir
```

c. 最后，使用 Cypress Programmer 对 *build_dir* 下生成的 cm0.hex 和 cm4.hex 文件进行编程。

运行其他演示

以下演示应用程序已经过测试和验证，可以与当前版本配合使用。您可以在 *freertos*/demos 目录下找到这些演示。有关如何运行这些演示的信息，请参阅 FreeRTOS 演示。

- 低功耗蓝牙演示
- Over-the-Air 更新演示
- 安全套接字 Echo 客户端演示
- AWS IoT Device Shadow 演示

调试

套件上的 KitProg 3 支持通过 SWD 协议进行调试。

- 要调试 FreeRTOS 应用程序，请在工作区中选择 aws_demos 项目，然后从快速面板中选择 aws_demo s Debug (3)。KitProg

OTA 更新

PSoC 64 MCUs 已通过所有必需的 FreeRTOS 资格测试。但是， PSoC 64 Standard Secure AWS 固件库中实现的可选 over-the-air (OTA) 功能仍有待评估。目前实现的 OTA 功能已通过所有 OTA 资格认证测试，但 aws_ota_test_case_rollback_if_unable_to_connect_after_update.py 除外。

当使用 PSo C64 Standard Secure — AWS MCU 将成功验证的 OTA 映像应用于设备并且该设备无法与之通信时 AWS IoT Core，该设备无法自动回滚到原始已知良好的映像。这可能会导致无法访问该设备 AWS IoT Core 以进行进一步的更新。Cypress团队仍在开发此功能。

有关更多信息，请参阅附带的 OTA 更新 AWS 和 CY8 CKIT-064S0S2-4343W 套件。如果您还有其他问题或需要技术支持，请联系Cypress 开发人员社区。

带有 Windows 模拟器的 Microchip ATECC6 08A Secure Element 入门

> **Important**
>
> 该参考集成托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

本教程提供了使用 Windows 模拟器开始使用 Microchip ATECC6 08A 安全元件的说明。

您需要以下硬件：

- Microchip ATECC6 08A 安全元件点击板
- SAMD21 XPlained Pro
- mikroBUS Xplained Pro 适配器

在开始之前，必须进行配置 AWS IoT 并下载 FreeRTOS 才能将设备连接到云端。 AWS 有关说明，请参阅初始步骤：在本教程中，FreeRTOS 下载目录的路径称为 *freertos*。

概览

本教程包含以下步骤：

1. 将主板连接到主机。
2. 在主机上安装软件来开发和调试微控制器主板的嵌入式应用程序。
3. 将 FreeRTOS 演示应用程序交叉编译为二进制映像。
4. 将应用程序二进制映像加载到您的主板中，然后运行该应用程序。

设置 Microchip ATECC6 08A 硬件

在与 Microchip ATECC6 08A 设备进行交互之前，必须先对设备进行编程。 SAMD21

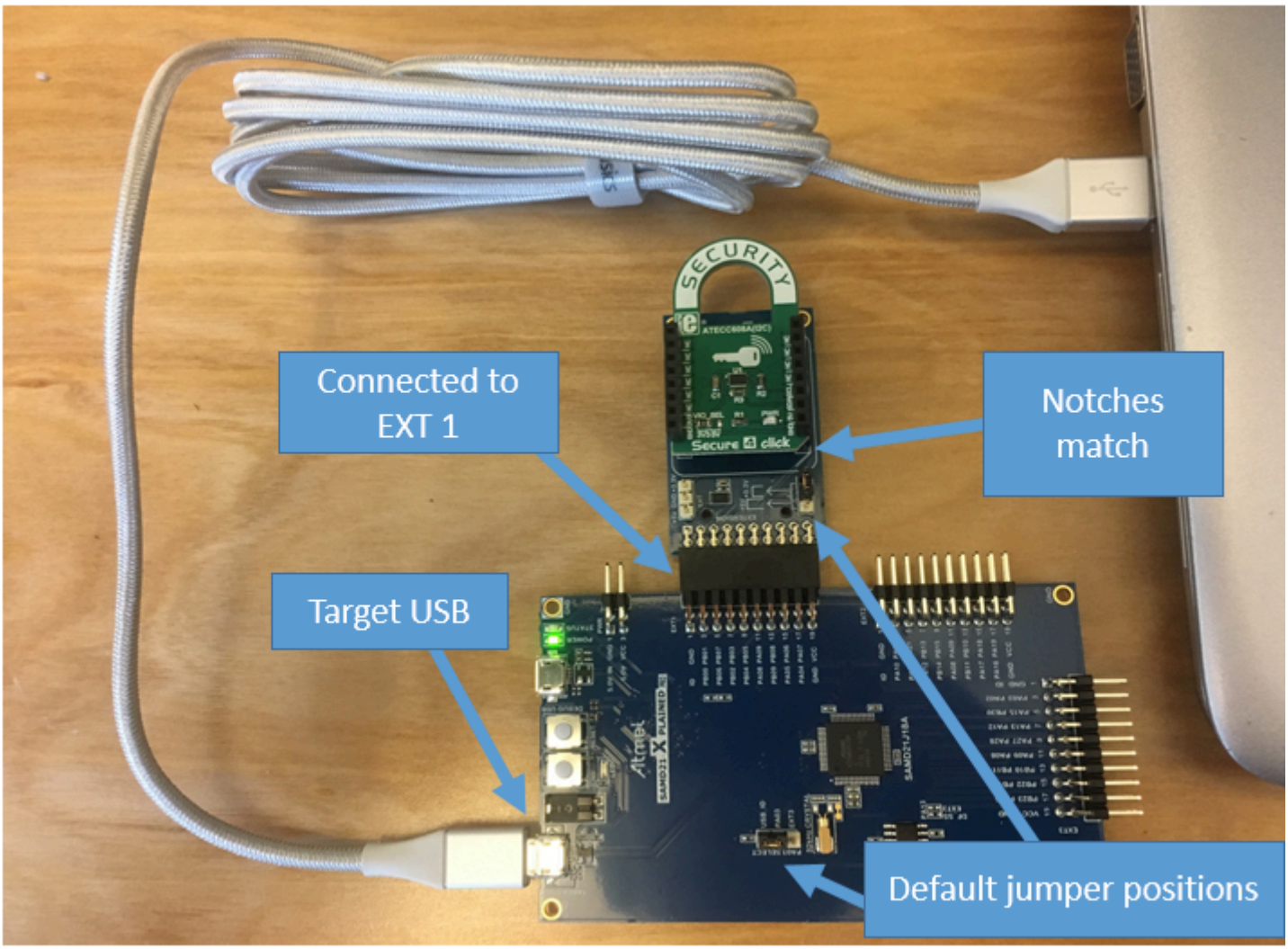
设置 P SAMD21 XPlained ro 主板

1. 点击 CryptoAuthSSH-XSTK (DM320109)-最新固件链接，下载包含指令 (PDF) 和可在 D21 上编程的二进制文件的.zip 文件。
2. 下载并安装 Atmel Studio 7 IDP。请确保在安装过程中选择 SMART ARM MCU 驱动程序架构。
3. 使用 USB 2.0 Micro B 电缆将“调试 USB”连接器连接到您的计算机，并按照 PDF 中的说明操作。（“调试 USB”连接器是最接近电源引线和引脚的 USB 端口。）

连接硬件

1. 从调试 USB 中拔下微型 USB 电缆。
2. 将 MikroBus XPlained Pro 适配器插入主 SAMD21 板所在位置。 EXT1
3. 将 ATECC6 08A Secure 4 Click 主板插入 XPlained MikrobusX Pro 适配器。确保 Click 主板的缺口角与适配器板上的缺口图标匹配。
4. 将微型 USB 电缆插入到目标 USB 中。

您的设置应如下所示。



设置开发环境

注册获取 AWS 账户

如果您没有 AWS 账户，请完成以下步骤来创建一个。

要注册 AWS 账户

1. 打开https://portal.aws.amazon.com/billing/注册。
2. 按照屏幕上的说明操作。

   注册过程的一部分涉及接听电话或短信，并在电话键盘上输入验证码。

   当您注册时 AWS 账户，就会创建AWS 账户根用户一个。根用户有权访问该账户中的所有 AWS 服务 和资源。作为最佳安全实践，请为用户分配管理访问权限，并且只使用根用户来执行需要根用户访问权限的任务。

AWS 注册过程完成后会向您发送一封确认电子邮件。您可以随时前往 https://aws.amazon.com/并选择“我的账户”，查看当前账户活动并管理您的账户。

创建具有管理访问权限的用户

注册后，请保护您的安全 AWS 账户 AWS 账户根用户 AWS IAM Identity Center，启用并创建管理用户，这样您就不会使用 root 用户执行日常任务。

保护你的 AWS 账户根用户

1. 选择 Root 用户并输入您的 AWS 账户 电子邮件地址，以账户所有者的身份登录。AWS Management Console在下一页上，输入您的密码。

   要获取使用根用户登录方面的帮助，请参阅《AWS 登录 用户指南》中的 Signing in as the root user。
2. 为您的根用户启用多重身份验证（MFA）。

   有关说明，请参阅 I A M 用户指南中的为 AWS 账户 根用户（控制台）启用虚拟 MFA 设备。

创建具有管理访问权限的用户

1. 启用 IAM Identity Center。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的 Enabling AWS IAM Identity Center。
2. 在 IAM Identity Center 中，为用户授予管理访问权限。

   有关使用 IAM Identity Center 目录 作为身份源的教程，请参阅《用户指南》 IAM Identity Center 目录中的使用默认设置配置AWS IAM Identity Center 用户访问权限。

以具有管理访问权限的用户身份登录

- 要使用您的 IAM Identity Center 用户身份登录，请使用您在创建 IAM Identity Center 用户时发送到您的电子邮件地址的登录网址。

  有关使用 IAM Identity Center 用户登录的帮助，请参阅AWS 登录 用户指南中的登录 AWS 访问门户。

将访问权限分配给其他用户

1. 在 IAM Identity Center 中，创建一个权限集，该权限集遵循应用最低权限的最佳做法。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的 Create a permission set。

2. 将用户分配到一个组，然后为该组分配单点登录访问权限。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的 Add groups。

要提供访问权限，请为您的用户、组或角色添加权限：

- 中的用户和群组 AWS IAM Identity Center：

  创建权限集合。按照《AWS IAM Identity Center 用户指南》中创建权限集的说明进行操作。

- 通过身份提供商在 IAM 中托管的用户：

  创建适用于身份联合验证的角色。按照《IAM 用户指南》中针对第三方身份提供商创建角色（联合身份验证）的说明进行操作。

- IAM 用户：
  - 创建您的用户可以担任的角色。按照《IAM 用户指南》中为 IAM 用户创建角色的说明进行操作。
  - （不推荐使用）将策略直接附加到用户或将用户添加到用户组。按照《IAM 用户指南》中向用户添加权限（控制台）中的说明进行操作。

设置

1. 从 FreeRTOS 存储库中下载 FreeRTOS 存储库。 GitHub

   要从以下网址下载 FreeRTOS： GitHub

   1. 浏览到 FreeRTOS 存储库 GitHub 。
   2. 选择 Clone or download (克隆或下载)。
   3. 从计算机的命令行中，将存储库克隆到主机上的一个目录中。

      ```
      git clone https://github.com/aws/amazon-freertos.git -\-recurse-submodules
      ```

> **Important**
>
> - 在本主题中，FreeRTOS 下载目录的路径称为 *freertos*。
> - *freertos* 路径中的空格字符可能会导致构建失败。克隆或复制存储库时，请确保您创建的路径不包含空格字符。
> - Microsoft Windows 上的文件路径最大长度为 260 个字符。FreeRTOS 下载目录路径过长可能会导致构建操作失败。
> - 由于源代码可能包含符号链接，因此，如果您使用 Windows 提取存档，则可能必须：
>   - 启用开发者模式，或者，
>   - 使用已提升为管理员的控制台。
>
>   这样，Windows 就可以在提取存档文件时正确创建符号链接。否则，符号链接将作为普通文件写入，该文件会将符号链接的路径作为文本，或者是空文件。有关更多信息，请参阅博客文章 Windows 10 中的符号链接。
>
>   如果您在 Windows 下使用 Git，则必须启用开发者模式，或者必须：
>   - 使用以下命令将 `core.symlinks` 设置为 true：
>
>     ```
>     git config -\-global core.symlinks true
>     ```
>
>   - 每当您使用写入系统的 git 命令（例如，git pull、git clone 和 git submodule update -\-init -\-recursive）时，请使用具有管理员权限的控制台。

4. 从 *freertos* 目录中，检查要使用的分支。

2. 设置开发环境。

   a. 安装最新版本的 Win PCap。

   b. 安装 Microsoft Visual Studio。

      Visual Studio 2017 和 2019 版已知可用。支持 Visual Studio 的所有版本（社区版、专业版或企业版）。

      在 IDE 之外，请安装 Desktop development with C++ (C++ 桌面开发)组件。然后，在 Optional (可选) 下安装最新的 Windows 10 开发工具包。

   c. 确保您有活动的有线以太网连接。

构建并运行 FreeRTOS 演示项目

> **Important**
>
> Microchip ATECC6 08A 设备具有一次性初始化，该初始化会在项目首次运行时（在调用期间）锁定在设备上。`C_InitToken`但是，FreeRTOS 演示项目和测试项目的配置不同。如果设备在演示项目配置期间被锁定，则测试项目中的所有测试都不可能成功。

使用 Visual Studio IDE 构建并运行 FreeRTOS 演示项目

1. 在 Visual Studio 中加载项目。

   从 File (文件) 菜单上，选择 Open (打开)。选择 File/Solution (文件/解决方案)，导航到 *freertos*`\projects\microchip\ecc608a_plus_winsim\visual_studio\aws_demos\aws_demos.sln` 文件，然后选择 Open (打开)。

2. 重新定位演示项目。

   此演示项目取决于 Windows 开发工具包，但未指定 Windows 开发工具包版本。默认情况下，IDE 可能会尝试使用计算机上不存在的开发工具包版本构建演示。要设置 Windows 开发工具包版本，请右键单击 aws_demos，然后选择 Retarget Projects (重新定位项目)。这将打开 Review Solution Actions (审核解决方案操作) 窗口。选择计算机上现有的一个 Windows 开发工具包版本（使用下拉列表中的初始值），然后选择 OK (确定)。

3. 构建并运行项目。

   从构建菜单中选择构建解决方案，确保解决方案已构建且没有错误。选择 Debug (调试)、Start Debugging (开始调试) 以运行项目。在第一次运行时，您需要配置您的设备接口并重新编译。有关更多信息，请参阅 配置网络接口。

4. 配置 Microchip ATECC6 08A。

   Microchip 提供了多种脚本工具来帮助设置 ATECC6 08A 部件。导航到 *freertos*`\vendors\microchip\secure_elements\app\example_trust_chain_tool`，然后打开 README.md 文件。

   请按照 `README.md` 文件中的说明预置您的设备。相应步骤包括：

   1. 创建并注册证书颁发机构 AWS。
   2. 在 Microchip ATECC6 08A 上生成密钥并导出公钥和设备序列号。

3. 为设备生成证书并向注册该证书 AWS。
4. 将 CA 证书和设备证书加载到设备上。

5. 构建并运行 FreeRTOS 示例。

   再次重新运行演示项目。这次应该能成功连接！

故障排除

有关一般故障排除信息，请参阅问题排查入门。

开始使用 Espressif ESP32-C 和 DevKit ESP-WROVER-KIT

> **Important**
>
> 该参考集成托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

> **Note**
>
> 要探索如何将 FreeRTOS 模块化库和演示集成到您自己的 Espressif IDF 项目中，请参阅我们精选的-C3 平台参考集成。 ESP32

按照本教程开始使用配备-WROOM- ESP32 32、-SOLO-1 或 ESP-WROVER 模块的 Espressif ESP32 ESP32-DevKit C 以及 ESP-WROVER-KIT-VB 要在合作伙伴设备目录中向我们的 AWS 合作伙伴购买一台，请使用以下链接：

- ESP32-WROOM-32 C DevKit
- ESP32-SOLO-1
- ESP32-WROVER-KIT

这些版本的开发主板在 FreeRTOS 上均受支持。

有关这些主板最新版本的更多信息，请参阅 Espressif 网站上的 ESP32-DevKit C ESP-WROVER-KIT V4 或 v4.1。

> **Note**
>
> 目前，-WROVER-KIT 和 DevKit ESP C 的 FreeRTOS 端口不支持 ESP32对称多处理 (SMP) 功能。

概览

该教程将指导您完成以下步骤：

1. 将主板连接到主机。
2. 在主机上安装软件来开发和调试微控制器主板的嵌入式应用程序。
3. 将 FreeRTOS 演示应用程序交叉编译为二进制映像。
4. 将应用程序二进制映像加载到您的主板上，然后运行该应用程序。
5. 跨串行连接与主板上运行的应用程序进行交互，以便进行监视和调试。

先决条件

在开始在乐鑫看板上使用 FreeRTOS 之前，您必须设置账户和权限。 AWS

注册获取 AWS 账户

如果您没有 AWS 账户，请完成以下步骤来创建一个。

要注册 AWS 账户

1. 打开https://portal.aws.amazon.com/billing/注册。
2. 按照屏幕上的说明操作。

   注册过程的一部分涉及接听电话或短信，并在电话键盘上输入验证码。

   当您注册时 AWS 账户，就会创建AWS 账户根用户一个。根用户有权访问该账户中的所有 AWS 服务 和资源。作为最佳安全实践，请为用户分配管理访问权限，并且只使用根用户来执行需要根用户访问权限的任务。

AWS 注册过程完成后会向您发送一封确认电子邮件。您可以随时前往 https://aws.amazon.com/并选择“我的账户”，查看您当前的账户活动并管理您的账户。

创建具有管理访问权限的用户

注册后，请保护您的安全 AWS 账户 AWS 账户根用户 AWS IAM Identity Center，启用并创建管理用户，这样您就不会使用 root 用户执行日常任务。

保护你的 AWS 账户根用户

1. 选择 Root 用户并输入您的 AWS 账户 电子邮件地址，以账户所有者的身份登录。AWS Management Console在下一页上，输入您的密码。

   要获取使用根用户登录方面的帮助，请参阅《AWS 登录 用户指南》中的 Signing in as the root user。
2. 为您的根用户启用多重身份验证（MFA）。

   有关说明，请参阅 I A M 用户指南中的为 AWS 账户 根用户（控制台）启用虚拟 MFA 设备。

创建具有管理访问权限的用户

1. 启用 IAM Identity Center。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的 Enabling AWS IAM Identity Center。
2. 在 IAM Identity Center 中，为用户授予管理访问权限。

   有关使用 IAM Identity Center 目录 作为身份源的教程，请参阅《用户指南》 IAM Identity Center 目录中的使用默认设置配置AWS IAM Identity Center 用户访问权限。

以具有管理访问权限的用户身份登录

- 要使用您的 IAM Identity Center 用户身份登录，请使用您在创建 IAM Identity Center 用户时发送到您的电子邮件地址的登录网址。

  有关使用 IAM Identity Center 用户登录的帮助，请参阅AWS 登录 用户指南中的登录 AWS 访问门户。

将访问权限分配给其他用户

1. 在 IAM Identity Center 中，创建一个权限集，该权限集遵循应用最低权限的最佳做法。

有关说明，请参阅《AWS IAM Identity Center 用户指南》中的 Create a permission set。

2. 将用户分配到一个组，然后为该组分配单点登录访问权限。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的 Add groups。

要提供访问权限，请为您的用户、组或角色添加权限：

- 中的用户和群组 AWS IAM Identity Center：

  创建权限集合。按照《AWS IAM Identity Center 用户指南》中创建权限集的说明进行操作。

- 通过身份提供商在 IAM 中托管的用户：

  创建适用于身份联合验证的角色。按照《IAM 用户指南》中针对第三方身份提供商创建角色（联合身份验证）的说明进行操作。

- IAM 用户：
  - 创建您的用户可以担任的角色。按照《IAM 用户指南》中为 IAM 用户创建角色的说明进行操作。
  - （不推荐使用）将策略直接附加到用户或将用户添加到用户组。按照《IAM 用户指南》中向用户添加权限（控制台）中的说明进行操作。

开始使用

> **Note**
>
> 本教程中的 Linux 命令要求您使用 Bash Shell。

1. 设置 Espressif 硬件。

   - 有关设置 ESP32-DevKit C 开发板硬件的信息，请参阅 ESP32-DevKit C V4 入门指南。
   - 有关设置 ESP-WROVER-KIT开发板硬件的信息，请参阅 ESP-WROVER-KITV4.1 入门指南。

   > **Important**
   >
   > 当您阅读到 Espressif 手册的入门 部分时，请暂停并返回到此页面上的说明部分。

2. 从以下网址下载亚马逊 FreeRTOS。GitHub（有关说明，请参阅 README.md 文件。）

3. 设置开发环境。

   要与您的主板通信，必须安装工具链。Espressif 提供了 ESP-IDF 来为主板开发软件。由于 ESP-IDF 将其 FreeRTOS 内核版本集成为组件，因此，Amazon FreeRTOS 包含删除了 FreeRTOS 内核的 ESP-IDF v4.2 自定义版本。这修复了编译时重复文件的问题。要使用 Amazon FreeRTOS 附带的 ESP-IDF v4.2 的自定义版本，请按照以下主机操作系统的说明进行操作。

   Windows

   1. 下载 ESP-IDF 的 Windows 版通用在线安装程序。
   2. 运行通用在线安装程序。
   3. 在下载或使用 ESP-IDF步骤，请选择使用现有 ESP-IDF 目录并将选择现有 ESP-IDF 目录设置为 *freertos*/vendors/espressif/esp-idf。
   4. 完成安装。

   macOS

   1. 按照适用于 macOS 的工具链标准设置先决条件 (ESP-IDF v4.2) 中的说明进行操作。

      > **Important**
      >
      > 在后续步骤下的“获取 ESP-IDF”说明部分，请暂停并返回到此页面上的说明。

   2. 打开一个命令行窗口。
   3. 转到 FreeRTOS 下载目录，然后运行以下脚本来下载并安装适用于您平台的 espressif 工具链。

      ```
      vendors/espressif/esp-idf/install.sh
      ```

   4. 使用以下命令以将 ESP-IDF 工具链工具添加到终端路径中。

      ```
      source vendors/espressif/esp-idf/export.sh
      ```

   Linux

   1. 按照适用于 Linux 的工具链标准设置先决条件 (ESP-IDF v4.2) 中的说明进行操作。

> **Important**
>
> 在后续步骤下的“获取 ESP-IDF”说明部分，请暂停并返回到此页面上的说明。

2. 打开一个命令行窗口。
3. 转到 FreeRTOS 下载目录，然后运行以下脚本来下载并安装适用于您平台的 Espressif 工具链。

```
vendors/espressif/esp-idf/install.sh
```

4. 使用以下命令以将 ESP-IDF 工具链工具添加到终端路径中。

```
source vendors/espressif/esp-idf/export.sh
```

4. 建立串行连接。

   a. 要在主机和 ESP32-DevKit C 之间建立串行连接，必须安装 CP21 0x USB 到 UART Bridge VCP 驱动程序。您可以从 Silicon Labs 下载这些驱动程序。

      要在主机与 ESP32-WROVER-KIT 之间建立串行连接，必须安装 FTDI 虚拟 COM 端口驱动程序。您可以从 FTDI 下载该驱动程序。

   b. 按照步骤与建立串行连接 ESP32。

   c. 建立串行连接后，记下主板连接的串行端口。您需要它来刷写演示。

配置 FreeRTOS 演示应用程序

在本教程中，FreeRTOS 配置文件位于以下文件中：*freertos*/vendors/espressif/boards/*board-name*/aws_demos/config_files/FreeRTOSConfig.h。（例如，如果选择 AFR_BOARD espressif.esp32_devkitc，则配置文件位于以下文件中：*freertos*/vendors/espressif/boards/esp32/aws_demos/config_files/FreeRTOSConfig.h。）

1. 如果您运行的是 macOS 或 Linux，请打开终端提示符。如果您运行的是 Windows，请打开“ESP-IDF 4.x CMD”应用程序（如果您在安装 ESP-IDF 工具链时包含了此选项），否则打开“命令提示符”应用程序。
2. 要验证您是否已安装 Python3，请运行

```
python --version
```

此时显示已安装的版本。如果您未安装 Python 3.0.1 或更高版本，可以从 Python 网站进行安装。

3. 您需要 AWS 命令行界面 (CLI) 才能运行 AWS IoT 命令。如果你运行的是 Windows，请使用该`easy_install awscli` AWS 命令在 “命令” 或 “ESP-IDF 4.x CMD” 应用程序中安装 CLI。

   如果你运行的是 macOS 或 Linux，请参阅安装 CLI AWS。

4. 运行

   ```
   aws configure
   ```

   并使用您的 AWS 访问密钥 ID、私有访问密钥和默认 AWS 区域配置 AWS CLI。有关更多信息，请参阅配置 AWS CLI。

5. 使用以下命令安装适用于 Python 的 AWS 开发工具包 (boto3)：

   - 在 Windows 上，在“命令”或“ESP-IDF 4.x CMD”应用程序中，运行

     ```
     pip install boto3 --user
     ```

     > **Note**
     >
     > 有关详细信息，请参阅 Boto3 文档。

   - 在 macOS 或 Linux 上，运行

     ```
     pip install tornado nose --user
     ```

     然后运行

     ```
     pip install boto3 --user
     ```

FreeRTOS 包含 `SetupAWS.py` 脚本，可以更轻松地设置您的 Espressif 主板以连接到 AWS IoT。要配置此脚本，请打开 *`freertos`*`/tools/aws_config_quick_start/configure.json` 并设置以下属性：

`afr_source_dir`

计算机上的 *`freertos`* 目录的完整路径。确保您使用正斜杠来指定此路径。

thing_name

您要为代表您的看板 AWS IoT 的事物分配的名称。

wifi_ssid

Wi-Fi 网络的 SSID。

wifi_password

Wi-Fi 网络的密码。

wifi_security

Wi-Fi 网络的安全类型。

下面是有效的安全类型：

- eWiFiSecurityOpen（开放，不安全）
- eWiFiSecurityWEP（WEP 安全性）
- eWiFiSecurityWPA（WPA 安全性）
- eWiFiSecurityWPA2（WPA2 安全）

6. 运行配置脚本。

a. 如果您运行的是 macOS 或 Linux，请打开终端提示符。如果您运行的是 Windows，请打开“ESP-IDF 4.x CMD”或“命令”应用程序。

b. 导航到 *freertos*/tools/aws_config_quick_start 目录运行

```
python SetupAWS.py setup
```

脚本执行以下操作：

- 创建一个 IoT 事物、证书和策略
- 将 IoT 策略附加到证书，将证书附加到 AWS IoT 事物。
- 使用您的 AWS IoT 终端节点、Wi-Fi SSID 和凭证填充 aws_clientcredential.h 文件
- 设置您的证书和私有密钥格式，然后将其写入 aws_clientcredential_keys.h 标头文件

> **Note**
>
> 出于演示目的，对该证书进行了硬编码。生产级应用程序应将这些文件存储在安全位置。

有关 SetupAWS.py 的更多信息，请参阅 *freertos*/tools/aws_config_quick_start 目录中的 README.md 文件。

在云上监控 MQTT 消息

在运行 FreeRTOS 演示项目之前，您可以在控制台中 AWS IoT 设置 MQTT 客户端，以监控您的设备发送到云端的消息。 AWS

使用 MQTT 客户端订阅 M AWS IoT QTT 主题

1. 导航至 AWS IoT 控制台。
2. 在导航窗格中，选择测试，然后选择 MQTT 测试客户端。
3. 在 Subscription topic (订阅主题)中，输入 *your-thing-name*/example/topic，然后选择 Subscribe to topic (订阅主题)。

当演示项目在您的设备上成功运行时，您会多次看到“Hello World！” 发送到您订阅的主题。

使用 idf.py 脚本构建、刷写和运行 FreeRTOS 演示项目

您可以使用 Espressif 的 IDF 实用工具 (idf.py) 来构建项目，并将二进制文件刷写到设备上。

> **Note**
>
> 某些设置可能需要您使用在 idf.py 中使用端口选项 "-p port-name" 来指定正确的端口，如以下示例所示。
>
> ```
> idf.py -p /dev/cu.usbserial-00101301B flash
> ```

在 Windows、Linux 和 macOS 上构建和刷写 FreeRTOS (ESP-IDF v4.2)

1. 转到 FreeRTOS 下载目录的根目录。
2. 在命令行窗口中输入以下命令以将 ESP-IDF 工具添加到终端 PATH 中。

   Windows（“命令”应用程序）

   ```
   vendors\espressif\esp-idf\export.bat
   ```

   Windows（“ESP-IDF 4.x CMD”应用程序）

   （打开应用程序时就已经完成此操作。）

   Linux / macOS

   ```
   source vendors/espressif/esp-idf/export.sh
   ```

3. 在 build 目录中配置 cmake 并使用以下命令构建固件映像。

   ```
   idf.py -DVENDOR=espressif -DBOARD=esp32_wrover_kit -DCOMPILER=xtensa-esp32 build
   ```

   将会看到类似下面的输出。

   ```
   Running cmake in directory /path/to/hello_world/build
   Executing "cmake -G Ninja --warn-uninitialized /path/to/hello_world"...
   Warn about uninitialized values.
   -- Found Git: /usr/bin/git (found version "2.17.0")
   -- Building empty aws_iot component due to configuration
   -- Component names: ...
   -- Component paths: ...

   ... (more lines of build system output)

   [527/527] Generating hello-world.bin
   esptool.py v2.3.1

   Project build complete. To flash, run this command:
   ../../../components/esptool_py/esptool/esptool.py -p (PORT) -b 921600
  write_flash --flash_mode dio --flash_size detect --flash_freq 40m 0x10000 build/
hello-world.bin  build 0x1000 build/bootloader/bootloader.bin 0x8000 build/
partition_table/partition-table.bin
   ```

```
   or run 'idf.py -p PORT flash'
```

如果没有错误，构建会生成固件二进制 .bin 文件。

4. 使用以下命令擦除开发主板的闪存。

```
idf.py erase_flash
```

5. 使用 `idf.py` 脚本将应用程序二进制文件刷写到主板。

```
idf.py flash
```

6. 使用以下命令监控主板串行端口的输出。

```
idf.py monitor
```

> **Note**
>
> 您可以合并这些命令，如以下示例所示。
>
> ```
> idf.py erase_flash flash monitor
> ```

对于某些主机设置，您必须在刷写主板时指定端口，如以下示例所示。

```
idf.py erase_flash flash monitor -p /dev/ttyUSB1
```

使用构建和刷新 FreeRTOS CMake

除了 IDF SDK 提供的用于构建和运行代码的`idf.py`脚本外，您还可以使用构建项目 CMake。目前，它支持 Unix Makefile 和 Ninja 构建系统。

构建和刷写项目

1. 在命令行窗口中，转到 FreeRTOS 下载目录的根目录。
2. 运行以下脚本，将 ESP-IDF 工具添加到 Shell 的 PATH 中。

Windows

```
vendors\espressif\esp-idf\export.bat
```

Linux / macOS

```
source vendors/espressif/esp-idf/export.sh
```

3. 使用以下命令来生成构建文件。

对于 Unix Makefiles

```
cmake -DVENDOR=espressif -DBOARD=esp32_wrover_kit -DCOMPILER=xtensa-esp32 -S . -
B ./YOUR_BUILD_DIRECTORY -DAFR_ENABLE_ALL_MODULES=1 -DAFR_ENABLE_TESTS=0
```

对于 Ninja

```
cmake -DVENDOR=espressif -DBOARD=esp32_wrover_kit -DCOMPILER=xtensa-esp32 -S . -
B ./YOUR_BUILD_DIRECTORY -DAFR_ENABLE_ALL_MODULES=1 -DAFR_ENABLE_TESTS=0 -GNinja
```

4. 构建 项目。

对于 Unix Makefiles

```
make -C ./YOUR_BUILD_DIRECTORY -j8
```

对于 Ninja

```
ninja -C ./YOUR_BUILD_DIRECTORY -j8
```

5. 擦除闪存，然后刷写主板。

对于 Unix Makefiles

```
make -C ./YOUR_BUILD_DIRECTORY erase_flash
```

```
make -C ./YOUR_BUILD_DIRECTORY flash
```

对于 Ninja

```
ninja -C ./YOUR_BUILD_DIRECTORY erase_flash
```

```
ninja -C ./YOUR_BUILD_DIRECTORY flash
```

运行低功耗蓝牙演示

FreeRTOS 支持低功耗蓝牙库连接。

要跨低功耗蓝牙运行 FreeRTOS 演示项目，您必须在 iOS 或 Android 移动设备上运行 FreeRTOS 低功耗蓝牙移动开发工具包演示应用程序。

设置 FreeRTOS 低功耗蓝牙移动开发工具包演示应用程序

1. 按照 SDKs 适用于 FreeRTOS 蓝牙设备的移动设备 中的说明，在您的主机上下载并安装适用于移动平台的开发工具包。
2. 按照 FreeRTOS 低功耗蓝牙移动开发工具包演示应用程序 中的说明，在您的移动设备上设置演示移动应用程序。

有关如何在主板上运行低功耗蓝牙 MQTT 演示的说明，请参阅低功耗蓝牙 MQTT。

有关如何在主板上运行 Wi-Fi 预置演示的说明，请参阅 Wi-Fi 预置。

在你自己的项目中使用 FreeRTOS CMake ESP32

如果您想在 CMake 自己的项目中使用 FreeRTOS，可以将其设置为子目录，然后将其与您的应用程序一起构建。首先，从那里获取 FreeRTOS 的副本。GitHub您也可以使用以下命令将它设置为 Git 子模块，以便将来更轻松地更新。

```
git submodule add -b release https://github.com/aws/amazon-freertos.git freertos
```

如果已发布更新的版本，则可使用这些命令更新本地副本。

```
# Pull the latest changes from the remote tracking branch.
git submodule update --remote -- freertos
```

```
# Commit the submodule change because it is pointing to a different revision now.
```

```
git add freertos
```

```
git commit -m "Update FreeRTOS to a new release"
```

如果您的项目具有以下目录结构：

```
- freertos (the copy that you obtained from GitHub or the AWS IoT console)
- src
  - main.c (your application code)
- CMakeLists.txt
```

下面是在 FreeRTOS 中可用于构建应用程序的顶级 CMakeLists.txt 文件的示例。

```
cmake_minimum_required(VERSION 3.13)

project(freertos_examples)

# Tell IDF build to link against this target.
set(IDF_EXECUTABLE_SRCS "<complete_path>/src/main.c")
set(IDF_PROJECT_EXECUTABLE my_app)

# Add FreeRTOS as a subdirectory. AFR_BOARD tells which board to target.
set(AFR_BOARD espressif.esp32_devkitc CACHE INTERNAL "")
add_subdirectory(freertos)

# Link against the mqtt library so that we can use it. Dependencies are transitively
# linked.
target_link_libraries(my_app PRIVATE AFR::core_mqtt)
```

要生成项目，请运行以下 CMake 命令。确保 ESP32 编译器位于 PATH 环境变量中。

```
cmake -S . -B build-directory -DCMAKE_TOOLCHAIN_FILE=freertos/tools/cmake/toolchains/
xtensa-esp32.cmake -GNinja
```

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

要将应用程序刷写到主板，请运行以下命令。

```
cmake --build build-directory --target flash
```

使用 FreeRTOS 中的组件

运行后 CMake，您可以在摘要输出中找到所有可用的组件。它应该类似于以下示例。

```
====================Configuration for FreeRTOS====================
  Version:                  202107.00
  Git version:              202107.00-g79ad6defb

Target microcontroller:
  vendor:                   Espressif
  board:                    ESP32-DevKitC
  description:              Development board produced by Espressif that comes in two
                            variants either with ESP-WROOM-32 or ESP32-WROVER module
  family:                   ESP32
  data ram size:            520KB
  program memory size:      4MB

Host platform:
  OS:                       Linux-4.15.0-66-generic
  Toolchain:                xtensa-esp32
  Toolchain path:           /opt/xtensa-esp32-elf
  CMake generator:          Ninja

FreeRTOS modules:
  Modules to build:         backoff_algorithm, common, common_io, core_http,
                            core_http_demo_dependencies, core_json, core_mqtt,
                            core_mqtt_agent, core_mqtt_agent_demo_dependencies,
                            core_mqtt_demo_dependencies, crypto, defender, dev_mode_key_
                            provisioning, device_defender, device_defender_demo_
                            dependencies, device_shadow,
 device_shadow_demo_dependencies,
                            freertos_cli_plus_uart, freertos_plus_cli, greengrass,
                            http_demo_helpers, https, jobs, jobs_demo_dependencies,
                            kernel, logging, mqtt, mqtt_agent_interface, mqtt_demo_
                            helpers, mqtt_subscription_manager, ota, ota_demo_
                            dependencies, ota_demo_version, pkcs11, pkcs11_helpers,
                            pkcs11_implementation, pkcs11_utils, platform,
 secure_sockets,
                            serializer, shadow, tls, transport_interface_secure_sockets,
                            wifi
  Enabled by user:          common_io, core_http_demo_dependencies, core_json,
                            core_mqtt_agent_demo_dependencies, core_mqtt_demo_
                            dependencies, defender, device_defender,
 device_defender_demo_
```

```
                            dependencies, device_shadow,
 device_shadow_demo_dependencies,
                            freertos_cli_plus_uart, freertos_plus_cli, greengrass,
 https,
                            jobs, jobs_demo_dependencies, logging,
 ota_demo_dependencies,
                            pkcs11, pkcs11_helpers, pkcs11_implementation, pkcs11_utils,
                            platform, secure_sockets, shadow, wifi
  Enabled by dependency:    backoff_algorithm, common, core_http, core_mqtt,
                            core_mqtt_agent, crypto, demo_base,
 dev_mode_key_provisioning,
                            freertos, http_demo_helpers, kernel, mqtt, mqtt_agent_
                            interface, mqtt_demo_helpers, mqtt_subscription_manager,
 ota,
                            ota_demo_version, pkcs11_mbedtls, serializer, tls,
                            transport_interface_secure_sockets, utils
  3rdparty dependencies:    jsmn, mbedtls, pkcs11, tinycbor
  Available demos:          demo_cli_uart, demo_core_http, demo_core_mqtt,
 demo_core_mqtt_
                            agent, demo_device_defender, demo_device_shadow,
                            demo_greengrass_connectivity, demo_jobs, demo_ota_core_http,
                            demo_ota_core_mqtt, demo_tcp
  Available tests:
================================================================================
```

您可以引用Modules to build列表中的任何组件。要将它们链接到您的应用程序，请将 AFR:: 命名空间置于名称的前面，例如 AFR::core_mqtt、AFR::ota 等。

使用 ESP-IDF 添加自定义组件

使用 ESP-IDF 时，您可以添加更多组件。例如，假定您想添加一个名为 example_component 的组件，并且您的项目如下所示：

```
- freertos
- components
  - example_component
    - include
      - example_component.h
    - src
      - example_component.c
    - CMakeLists.txt
- src
  - main.c
```

```
- CMakeLists.txt
```

下面是您的组件的 CMakeLists.txt 文件示例。

```
add_library(example_component src/example_component.c)
target_include_directories(example_component PUBLIC include)
```

然后，在顶层 CMakeLists.txt 文件中，通过在 add_subdirectory(freertos) 后面插入以下行来添加组件。

```
add_subdirectory(component/example_component)
```

然后，修改 target_link_libraries 以包含您的组件。

```
target_link_libraries(my_app PRIVATE AFR::core_mqtt PRIVATE example_component)
```

默认情况下，此组件现在将自动链接到您的应用程序代码。现在，您可以包含其标头文件并调用它定义的函数。

覆盖 FreeRTOS 的配置

目前，未提供明确定义的方法来重新定义 FreeRTOS 源树外部的配置。默认情况下，CMake 将查找*freertos*/vendors/espressif/boards/esp32/aws_demos/config_files/和*freertos*/demos/include/目录。但是，可以使用解决方法来告知编译器首先搜索其他目录。例如，您可以为 FreeRTOS 配置添加其他文件夹：

```
- freertos
- freertos-configs
  - aws_clientcredential.h
  - aws_clientcredential_keys.h
  - iot_mqtt_agent_config.h
  - iot_config.h
- components
- src
- CMakeLists.txt
```

freertos-configs 下的文件是从 *freertos*/vendors/espressif/boards/esp32/aws_demos/config_files/ 和 *freertos*/demos/include/ 目录复制的。然后，在顶层

CMakeLists.txt 文件中，在 add_subdirectory(freertos) 的前面添加此行，以便编译器首先搜索此目录。

```
include_directories(BEFORE freertos-configs)
```

为 ESP-IDF 提供您自己的 sdkconfig

如果你想提供自己的变量sdkconfig.default，可以从命令行设置 CMake 变量IDF_SDKCONFIG_DEFAULTS：

```
cmake -S . -B build-directory -DIDF_SDKCONFIG_DEFAULTS=path_to_your_sdkconfig_defaults
 -DCMAKE_TOOLCHAIN_FILE=freertos/tools/cmake/toolchains/xtensa-esp32.cmake -GNinja
```

如果您不为自己的 sdkconfig.default 文件指定位置，则 FreeRTOS 将使用位于 *freertos*/vendors/espressif/boards/esp32/aws_demos/sdkconfig.defaults 的默认文件。

有关更多信息，请参阅 Espressif API 参考中的项目配置；如果您在成功编译后遇到问题，请参阅该页面上关于已弃用选项及其替换的部分。

摘要

如果您的项目具有一个名为 example_component 的组件，并且您要覆盖某些配置，以下是顶级 CMakeLists.txt 文件的完整示例。

```
cmake_minimum_required(VERSION 3.13)

project(freertos_examples)

set(IDF_PROJECT_EXECUTABLE my_app)
set(IDF_EXECUTABLE_SRCS "src/main.c")

# Tell IDF build to link against this target.
set(IDF_PROJECT_EXECUTABLE my_app)

# Add some extra components. IDF_EXTRA_COMPONENT_DIRS is a variable used by ESP-IDF
# to collect extra components.
get_filename_component(
    EXTRA_COMPONENT_DIRS
    "components/example_component" ABSOLUTE
)
list(APPEND IDF_EXTRA_COMPONENT_DIRS ${EXTRA_COMPONENT_DIRS})
```

```
# Override the configurations for FreeRTOS.
include_directories(BEFORE freertos-configs)

# Add FreeRTOS as a subdirectory. AFR_BOARD tells which board to target.
set(AFR_BOARD espressif.esp32_devkitc CACHE INTERNAL "")
add_subdirectory(freertos)

# Link against the mqtt library so that we can use it. Dependencies are transitively
# linked.
target_link_libraries(my_app PRIVATE AFR::core_mqtt)
```

故障排除

- 如果你运行的是 macOS，但操作系统无法识别你的 ESP-WROVER-KIT，请确保你没有安装 D2XX 驱动程序。要卸载它们，请按照适用于 macOS X 的 FTDI 驱动程序安装指南中的说明操作。
- ESP-IDF 提供的监控实用工具（使用 make 监控调用）可帮助您解码地址。因此，在应用程序停止工作时，它可以帮助您获取一些有意义的回溯跟踪信息。有关更多信息，请参阅 Espressif 网站上的自动解码地址。
- 也可以在不需要任何特殊的 JTAG 硬件的情况下启用 GDBstub 与 gdb 的通信。有关更多信息，请参阅乐鑫 GDBStub网站上的 “使用启动 GDB”。
- 有关设置基于 OpenOCD 的环境的信息（如果需要基于 JTAG 硬件的调试），请参阅 Espressif 网站上的 JTAG 调试。
- 如果无法使用 `pip` 在 macOS 上安装 `pyserial`，请从 pyserial 网站下载。
- 如果主板连续重置，请尝试通过在终端上输入以下命令来擦除闪存：

  ```
  make erase_flash
  ```

- 如果您在运行 `idf_monitor.py` 时看到错误，请使用 Python 2.7。
- ESP-IDF 中必需的库包括在 FreeRTOS 中，因此无需从外部下载。如果已设置 `IDF_PATH` 环境变量，我们建议您在构建 FreeRTOS 之前将其清除。
- 在 Windows 上，生成项目可能需要 3-4 分钟。您可在 make 命令上使用 `-j4` 开关来缩短构建时间。

  ```
  make flash monitor -j4
  ```

- 如果您的设备在连接时遇到问题 AWS IoT，请打开该`aws_clientcredential.h`文件，并验证文件中是否正确定义了配置变量。 `clientcredentialMQTT_BROKER_ENDPOINT[]`应该看起来像`1234567890123-ats.iot.us-east-1.amazonaws.com`。

- 如果您遵循在你自己的项目中使用 FreeRTOS CMake ESP32中的步骤，并且您看到来自链接器的未定义的引用错误，则通常是由于缺少关联的库或演示所致。要添加它们，请使用标准 CMake 函数更新CMakeLists.txt文件（位于根目录下）target_link_libraries。
- ESP-IDF v4.2 支持使用 xtensa\-esp32\-elf\-gcc 8\.2\.0\. 工具链。如果您使用的是较早版本的 Xtensa 工具链，请下载所需的版本。
- 如果您看到如下错误日志，表明不满足 ESP-IDF v4.2 的 python 依赖项要求：

```
The following Python requirements are not satisfied:
  click>=5.0
  pyserial>=3.0
  future>=0.15.2
  pyparsing>=2.0.3,<2.4.0
  pyelftools>=0.22
  gdbgui==0.13.2.0
  pygdbmi<=0.9.0.2
  reedsolo>=1.5.3,<=1.5.4
  bitstring>=3.1.6
  ecdsa>=0.16.0
  Please follow the instructions found in the "Set up the tools" section of ESP-IDF
 Getting Started Guide
```

请使用以下 Python 命令在您的平台上安装 python 依赖项：

```
root/vendors/espressif/esp-idf/requirements.txt
```

有关问故障排除的更多信息，请参阅问题排查入门。

调试

在 Espressif ESP32-DevKit C 和 ESP-WROVER-KIT (ESP-IDF v4.2) 上调试代码

本节介绍如何使用 ESP-IDF v4.2 调试 Espressif 硬件。您需要 JTAG 到 USB 电缆。我们使用 USB 到 MPSSE 电缆（例如，FTDI C232HM-DDHSL-0）。

ESP-DevKit C JTAG 设置

对于 FTDI C232HM-DDHSL-0 电缆，这些是与 Dev ESP32 Kitc 的连接。

```
| C232HM-DDHSL-0 Wire Color | ESP32 GPIO Pin | JTAG Signal Name |
```

```
| ------------------------ | -------------- | ---------------- |
|  Brown (pin 5)           |  IO14          |  TMS             |
|  Yellow (pin 3)          |  IO12          |  TDI             |
|  Black (pin 10)          |  GND           |  GND             |
|  Orange (pin 2)          |  IO13          |  TCK             |
|  Green (pin 4)           |  IO15          |  TDO             |
```

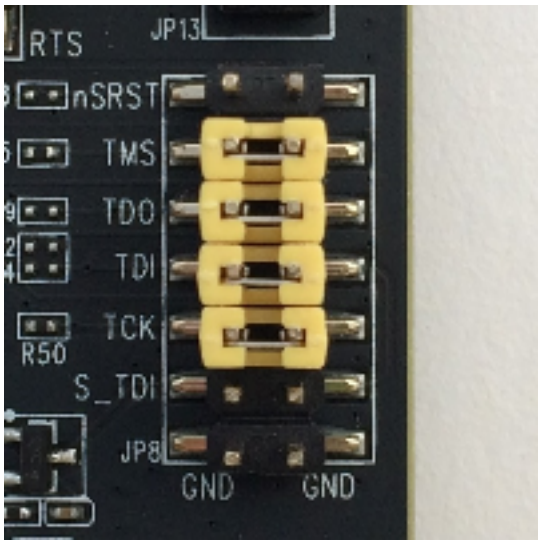
ESP-WROVER-KIT JTAG 安装程序

对于 FTDI C232HM-DDHSL-0 电缆，这些是与-WROVER ESP32-KIT 的连接。

```
| C232HM-DDHSL-0 Wire Color | ESP32 GPIO Pin | JTAG Signal Name |
| ------------------------ | -------------- | ---------------- |
|  Brown (pin 5)           |  IO14          |  TMS             |
|  Yellow (pin 3)          |  IO12          |  TDI             |
|  Orange (pin 2)          |  IO13          |  TCK             |
|  Green (pin 4)           |  IO15          |  TDO             |
```

这些表源自 FTDI C232HM-DDHSL-0 数据表。有关更多信息，请参阅数据表中的“C232HM MPSSE 电缆连接和机械详细信息”。

要在上启用 JTAG，请将跳线放在 TMS ESP-WROVER-KIT、TDO、TDI、TCK 和 S_TDI 引脚上，如下所示。



在 Windows 上进行调试 (ESP-IDF v4.2)

在 Windows 上设置调试

1. 将 FTDI C232HM-DDHSL-0 的 USB 一端连接到您的计算机，另一端的操作如在 Espressif ESP32-DevKit C 和 ESP-WROVER-KIT (ESP-IDF v4.2) 上调试代码中所述。FTDI C232HM-DDHSL-0 设备应显示在设备管理器下的通用串行总线控制器中。
2. 在通用串行总线设备列表下，右键单击 C232HM-DDHSL-0 设备，然后选择属性。

   > **Note**
   >
   > 该设备可能被列为 USB Serial Port (USB 串行端口)。

   要查看设备的属性，请在属性窗口中选择详细信息选项卡。如果未列出设备，请安装适用于 FTDI C232HM-DDHSL-0 的 Windows 驱动程序。
3. 在 “详细信息” 选项卡上，选择 “属性”，然后选择 “硬件” IDs。您将在值字段中看到与以下类似的内容。

   ```
   FTDIBUS\COMPORT&VID_0403&PID_6014
   ```

   在此示例中，供应商 ID 为 0403，产品 ID 为 6014。

   验证它们是否 IDs 匹配`projects/espressif/esp32/make/aws_demos/esp32_devkitj_v1.cfg`。 IDs 在以 IDs 开头的行中指定，`ftdi_vid_pid`后面是供应商 ID 和产品 ID。

   ```
   ftdi_vid_pid 0x0403 0x6014
   ```
4. 下载 OpenOCD for Windows。
5. 将文件解压缩到 `C:\` 并将 `C:\openocd-esp32\bin` 添加到系统路径。
6. OpenOCD 需要 libusb，默认情况下未在 Windows 上安装。安装 libusb

   a. 下载 zadig.exe。

   b. 运行 `zadig.exe`。在 Options (选项) 菜单中，选择 List All Devices (列出所有设备)。

   c. 从下拉菜单中，选择 C232HM-DDHSL-0。

   d. 在绿色箭头右侧的目标驱动程序字段中，选择 WinUSB。

e. 从目标驱动程序字段下的列表中，选择箭头，然后选择安装驱动程序。选择 Replace Driver (替换驱动程序)。

7. 打开命令提示符，转到 FreeRTOS 下载目录的根目录，然后运行以下命令。

```
idf.py openocd
```

保持此命令提示符窗口处于打开状态。

8. 打开新的命令提示符，转到 FreeRTOS 下载目录的根目录，然后运行

```
idf.py flash monitor
```

9. 打开另一个命令提示符，转到 FreeRTOS 下载目录的根目录，然后等待演示开始在您的主板上运行。开始演示时，请运行

```
idf.py gdb
```

程序应在 `main` 函数中停止。

> **Note**
>
> 最多 ESP32 支持两个断点。

在 macOS 上进行调试 (ESP-IDF v4.2)

1. 下载适用于 macOS 的 FTDI 驱动程序。
2. 下载 OpenOCD。
3. 提取下载的 .tar 文件，并将 `.bash_profile` 中的路径设置为 `OCD_INSTALL_DIR/openocd-esp32/bin`。
4. 使用以下命令在 macOS 上安装 `libusb`。

```
brew install libusb
```

5. 使用以下命令卸载串行端口驱动程序。

```
sudo kextunload -b com.FTDI.driver.FTDIUSBSerialDriver
```

6. 使用以下命令卸载串行端口驱动程序。

```
sudo kextunload -b com.FTDI.driver.FTDIUSBSerialDriver
```

7. 如果您在高于 10.9 的 macOS 版本上运行，请使用以下命令卸载 Apple 的 FTDI 驱动程序。

```
sudo kextunload -b com.apple.driver.AppleUSBFTDI
```

8. 使用以下命令获取 FTDI 电缆的产品 ID 和供应商 ID。它会列出连接的 USB 设备。

```
system_profiler SPUSBDataType
```

`system_profiler` 的输出应与以下内容类似。

```
   DEVICE:

   Product ID: product-ID
   Vendor ID: vendor-ID (Future Technology Devices International Limited)
```

9. 打开 `projects/espressif/esp32/make/aws_demos/esp32_devkitj_v1.cfg`文件。设备的供应商 ID 和产品 ID 在以 `ftdi_vid_pid` 开头的行中指定。更改 IDs 以匹配上一步 IDs 中`system_profiler`输出的内容。
10. 打开终端窗口，转到 FreeRTOS 下载目录的根目录，然后使用以下命令运行 OpenOCD。

```
idf.py openocd
```

让该终端窗口保持打开状态。

11. 打开一个新的终端，使用以下命令来加载 FTDI 串行端口驱动程序。

```
sudo kextload -b com.FTDI.driver.FTDIUSBSerialDriver
```

12. 转到 FreeRTOS 下载目录的根目录，然后运行

```
idf.py flash monitor
```

13. 打开另一个新的终端，转到 FreeRTOS 下载目录的根目录，然后运行

```
idf.py gdb
```

程序应在 main 停止。

在 Linux 上进行调试 (ESP-IDF v4.2)

1. 下载 OpenOCD。提取 tarball 并按照自述文件中的安装说明操作。
2. 使用以下命令在 Linux 上安装 libusb。

```
sudo apt-get install libusb-1.0
```

3. 打开终端并输入 ls -l /dev/ttyUSB* 来列出连接到您计算机的所有 USB 设备。这可以帮助您检查操作系统是否识别主板上的 USB 端口。将会看到类似下面的输出。

```
   $ls -l /dev/ttyUSB*
   crw-rw----    1    root    dialout    188,    0    Jul    10    19:04    /
dev/ttyUSB0
   crw-rw----    1    root    dialout    188,    1    Jul    10    19:04    /
dev/ttyUSB1
```

4. 注销，然后登录并重新对主板加电，使更改生效。在终端提示符下，列出 USB 设备。确保组所有者已从 dialout 更改为 plugdev。

```
   $ls -l /dev/ttyUSB*
   crw-rw----    1    root    plugdev    188,    0    Jul    10    19:04    /
dev/ttyUSB0
   crw-rw----    1    root    plugdev    188,    1    Jul    10    19:04    /
dev/ttyUSB1
```

   较小编号的 /dev/ttyUSBn 接口用于 JTAG 通信。另一个接口被路由到 ESP32的串行端口 (UART)，用于将代码上传到 ESP32的闪存。

5. 在终端窗口中，转到 FreeRTOS 下载目录的根目录，然后使用以下命令运行 OpenOCD。

```
idf.py openocd
```

6. 打开另一个终端，转到 FreeRTOS 下载目录的根目录，然后运行以下命令。

```
idf.py flash monitor
```

7. 打开另一个终端，转到 FreeRTOS 下载目录的根目录，然后运行以下命令：

```
idf.py gdb
```

程序应在 `main()` 中停止。

乐鑫-WROOM-32SE ESP32 入门

> **Important**
>
> 该参考集成托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

> **Note**
>
> - 要探索如何将 FreeRTOS 模块化库和演示集成到您自己的 Espressif IDF 项目中，请参阅我们精选的-C3 平台参考集成。 ESP32
> - 目前，-WROOM-32SE 的 FreeRTOS 端口不支持 ESP32对称多处理 (SMP) 功能。

本教程向您展示了如何开始使用乐鑫-WROOM-32SE ESP32。要在合作伙伴设备目录中向我们的合作伙伴购买一台，请参阅 ESP32-WROOM-32SE。 AWS

概览

该教程将指导您完成以下步骤：

1. 将主板连接到主机。
2. 在您的主机上安装软件，以开发和调试微控制器主板的嵌入式应用程序。
3. 将 FreeRTOS 演示应用程序交叉编译为二进制映像。
4. 将应用程序二进制映像加载到您的主板中，然后运行该应用程序。
5. 使用串行连接监控和调试正在运行的应用程序。

先决条件

在开始在乐鑫看板上使用 FreeRTOS 之前，您必须设置账户和权限。 AWS

注册获取 AWS 账户

如果您没有 AWS 账户，请完成以下步骤来创建一个。

报名参加 AWS 账户

1. 打开https://portal.aws.amazon.com/billing/注册。
2. 按照屏幕上的说明操作。

   注册过程的一部分涉及接听电话或短信，并在电话键盘上输入验证码。

   当您注册时 AWS 账户，就会创建AWS 账户根用户一个。根用户有权访问该账户中的所有 AWS 服务 和资源。作为最佳安全实践，请为用户分配管理访问权限，并且只使用根用户来执行需要根用户访问权限的任务。

AWS 注册过程完成后会向您发送一封确认电子邮件。您可以随时前往 https://aws.amazon.com/并选择“我的账户”，查看您当前的账户活动并管理您的账户。

创建具有管理访问权限的用户

注册后，请保护您的安全 AWS 账户 AWS 账户根用户 AWS IAM Identity Center，启用并创建管理用户，这样您就不会使用 root 用户执行日常任务。

保护你的 AWS 账户根用户

1. 选择 Root 用户并输入您的 AWS 账户 电子邮件地址，以账户所有者的身份登录。AWS Management Console在下一页上，输入您的密码。

   要获取使用根用户登录方面的帮助，请参阅《AWS 登录 用户指南》中的 Signing in as the root user。
2. 为您的根用户启用多重身份验证（MFA）。

   有关说明，请参阅 I A M 用户指南中的为 AWS 账户 根用户启用虚拟 MFA 设备（控制台）。

创建具有管理访问权限的用户

1. 启用 IAM Identity Center。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的 Enabling AWS IAM Identity Center。

2. 在 IAM Identity Center 中，为用户授予管理访问权限。

   有关使用 IAM Identity Center 目录 作为身份源的教程，请参阅《用户指南》 IAM Identity Center 目录中的使用默认设置配置AWS IAM Identity Center 用户访问权限。

以具有管理访问权限的用户身份登录

- 要使用您的 IAM Identity Center 用户身份登录，请使用您在创建 IAM Identity Center 用户时发送到您的电子邮件地址的登录网址。

  有关使用 IAM Identity Center 用户登录的帮助，请参阅AWS 登录 用户指南中的登录 AWS 访问门户。

将访问权限分配给其他用户

1. 在 IAM Identity Center 中，创建一个权限集，该权限集遵循应用最低权限的最佳做法。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的 Create a permission set。
2. 将用户分配到一个组，然后为该组分配单点登录访问权限。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的 Add groups。

要提供访问权限，请为您的用户、组或角色添加权限：

- 中的用户和群组 AWS IAM Identity Center：

  创建权限集合。按照《AWS IAM Identity Center 用户指南》中创建权限集的说明进行操作。
- 通过身份提供商在 IAM 中托管的用户：

  创建适用于身份联合验证的角色。按照《IAM 用户指南》中针对第三方身份提供商创建角色（联合身份验证）的说明进行操作。
- IAM 用户：
  - 创建您的用户可以担任的角色。按照《IAM 用户指南》中为 IAM 用户创建角色的说明进行操作。
  - （不推荐使用）将策略直接附加到用户或将用户添加到用户组。按照《IAM 用户指南》中向用户添加权限（控制台）中的说明进行操作。

开始使用

> **Note**
>
> 本教程中的 Linux 命令要求您使用 Bash Shell。

1. 设置 Espressif 硬件。

   有关设置 ESP32-WROOM-32SE 开发板硬件的信息，请参阅 ESP32-DevKit C V4 入门指南。

   > **Important**
   >
   > 在该指南的分步安装部分，请按照步骤进行操作，直到完成步骤 4（设置环境变量）。完成步骤 4 后，请暂停，然后按照此处的其余步骤进行操作。

2. 从以下网址下载亚马逊 FreeRTOS。GitHub（有关说明，请参阅 README.md 文件。）
3. 设置开发环境。

   要与您的主板通信，必须安装工具链。Espressif 提供了 ESP-IDF 来为主板开发软件。由于 ESP-IDF 将其 FreeRTOS 内核版本集成为组件，因此，Amazon FreeRTOS 包含删除了 FreeRTOS 内核的 ESP-IDF v4.2 自定义版本。这修复了编译时重复文件的问题。要使用 Amazon FreeRTOS 附带的 ESP-IDF v4.2 的自定义版本，请按照以下主机操作系统的说明进行操作。

   Windows

   1. 下载 ESP-IDF 的 Windows 版通用在线安装程序。
   2. 运行通用在线安装程序。
   3. 在下载或使用 ESP-IDF步骤，请选择使用现有 ESP-IDF 目录并将选择现有 ESP-IDF 目录设置为 *freertos*/vendors/espressif/esp-idf。
   4. 完成安装。

   macOS

   1. 按照适用于 macOS 的工具链标准设置先决条件 (ESP-IDF v4.2) 中的说明进行操作。

> **Important**
>
> 在后续步骤下的“获取 ESP-IDF”说明部分，请暂停并返回到此页面上的说明。

2. 打开一个命令行窗口。
3. 转到 FreeRTOS 下载目录，然后运行以下脚本来下载并安装适用于您平台的 espressif 工具链。

```
vendors/espressif/esp-idf/install.sh
```

4. 使用以下命令以将 ESP-IDF 工具链工具添加到终端路径中。

```
source vendors/espressif/esp-idf/export.sh
```

Linux

1. 按照适用于 Linux 的工具链标准设置先决条件 (ESP-IDF v4.2) 中的说明进行操作。

> **Important**
>
> 在后续步骤下的“获取 ESP-IDF”说明部分，请暂停并返回到此页面上的说明。

2. 打开一个命令行窗口。
3. 转到 FreeRTOS 下载目录，然后运行以下脚本来下载并安装适用于您平台的 Espressif 工具链。

```
vendors/espressif/esp-idf/install.sh
```

4. 使用以下命令以将 ESP-IDF 工具链工具添加到终端路径中。

```
source vendors/espressif/esp-idf/export.sh
```

4. 建立串行连接。
   a. 要在主机和 ESP32-WROOM-32SE 之间建立串行连接，请安装 CP21 0x USB 到 UART Bridge VCP 驱动程序。您可以从 Silicon Labs 下载这些驱动程序。
   b. 按照步骤与建立串行连接 ESP32。

c. 建立串行连接后，记下主板连接的串行端口。您需要它来刷写演示。

配置 FreeRTOS 演示应用程序

在本教程中，FreeRTOS 配置文件位于以下文件中：*freertos*/vendors/espressif/boards/*board-name*/aws_demos/config_files/FreeRTOSConfig.h。（例如，如果选择 AFR_BOARD espressif.esp32_devkitc，则配置文件位于以下文件中：*freertos*/vendors/espressif/boards/esp32/aws_demos/config_files/FreeRTOSConfig.h。）

> **Important**
>
> ATECC608A 设备具有一次性初始化，该初始化会在首次运行项目时（在调用期间C_InitToken）锁定在设备上。但是，FreeRTOS 演示项目和测试项目的配置不同。如果设备在演示项目配置期间被锁定，则并非测试项目中的所有测试都会成功。

1. 按照配置 FreeRTOS 演示中的步骤配置 FreeRTOS 演示项目。进入最后一步后，要格式化 AWS IoT 凭据，请停止并执行以下步骤。
2. Microchip 提供了多种脚本工具来帮助设置 ATECC6 08A 部件。导航到 *freertos*/vendors/microchip/example_trust_chain_tool 目录并打开 README.md 文件。
3. 请按照 README.md 文件中的说明预置您的设备。相应步骤包括：
   1. 创建并注册证书颁发机构 AWS。
   2. 在 ATECC6 08A 上生成密钥并导出公钥和设备序列号。
   3. 为设备生成证书并向注册该证书 AWS。
4. 按照 开发人员模式密钥预置 的说明，将 CA 证书和设备证书加载到设备上。

监控云端上的 MQTT 消息 AWS

在运行 FreeRTOS 演示项目之前，您可以在控制台中 AWS IoT 设置 MQTT 客户端，以监控您的设备发送到云端的消息。 AWS

使用 MQTT 客户端订阅 M AWS IoT QTT 主题

1. 登录 AWS IoT 控制台。
2. 在导航窗格中，选择测试，然后选择 MQTT 测试客户端。
3. 在订阅主题中，输入 *your-thing-name*/example/topic，然后选择订阅主题。

使用 idf.py 脚本构建、刷写和运行 FreeRTOS 演示项目

您可以使用 Espressif 的 IDF (`idf.py`) 生成构建文件，构建应用程序二进制文件，将二进制文件刷写到设备。

> **Note**
>
> 某些设置可能需要您使用在 `idf.py` 中使用端口选项“`-p port-name`”来指定正确的端口，如以下示例所示。
>
> ```
> idf.py -p /dev/cu.usbserial-00101301B flash
> ```

在 Windows、Linux 和 macOS 上构建和刷写 FreeRTOS (ESP-IDF v4.2)

1. 转到 FreeRTOS 下载目录的根目录。
2. 在命令行窗口中输入以下命令以将 ESP-IDF 工具添加到终端 PATH 中：

   Windows（“命令”应用程序）

   ```
   vendors\espressif\esp-idf\export.bat
   ```

   Windows（“ESP-IDF 4.x CMD”应用程序）

   （打开应用程序时就已经完成此操作。）

   Linux / macOS

   ```
   source vendors/espressif/esp-idf/export.sh
   ```
3. 在 `build` 目录中配置 cmake 并使用以下命令构建固件映像。

   ```
   idf.py -DVENDOR=espressif -DBOARD=esp32_ecc608a_devkitc -DCOMPILER=xtensa-esp32
    build
   ```

   您应该会看到以下示例的输出。

   ```
   Running cmake in directory /path/to/hello_world/build
   Executing "cmake -G Ninja --warn-uninitialized /path/to/hello_world"...
   Warn about uninitialized values.
   ```

```
   -- Found Git: /usr/bin/git (found version "2.17.0")
   -- Building empty aws_iot component due to configuration
   -- Component names: ...
   -- Component paths: ...

   ... (more lines of build system output)

   [527/527] Generating hello-world.bin
   esptool.py v2.3.1

   Project build complete. To flash, run this command:
   ../../../components/esptool_py/esptool/esptool.py -p (PORT) -b 921600
 write_flash --flash_mode dio --flash_size detect --flash_freq 40m 0x10000 build/
hello-world.bin  build 0x1000 build/bootloader/bootloader.bin 0x8000 build/
partition_table/partition-table.bin
   or run 'idf.py -p PORT flash'
```

如果没有错误，构建会生成固件二进制 .bin 文件。

4. 使用以下命令擦除开发主板的闪存。

```
idf.py erase_flash
```

5. 使用 `idf.py` 脚本将应用程序二进制文件刷写到主板。

```
idf.py flash
```

6. 使用以下命令监控主板串行端口的输出。

```
idf.py monitor
```

> **Note**
>
> - 您可以合并这些命令，如以下示例所示。
>
>   ```
>   idf.py erase_flash flash monitor
>   ```
>
> - 对于某些主机设置，您必须在刷写主板时指定端口，如以下示例所示。
>
>   ```
>   idf.py erase_flash flash monitor -p /dev/ttyUSB1
>   ```

使用构建和刷新 FreeRTOS CMake

除了使用 IDF SDK 提供的`idf.py`脚本来构建和运行代码外，您还可以使用构建项目 CMake。目前，它支持 Unix Makefile 和 Ninja 构建系统。

构建和刷写项目

1. 在命令行窗口中，转到 FreeRTOS 下载目录的根目录。
2. 运行以下脚本，将 ESP-IDF 工具添加到 Shell 的 PATH 中。

   Windows

   ```
   vendors\espressif\esp-idf\export.bat
   ```

   Linux / macOS

   ```
   source vendors/espressif/esp-idf/export.sh
   ```

3. 使用以下命令来生成构建文件。

   对于 Unix Makefiles

   ```
   cmake -DVENDOR=espressif -DBOARD=esp32_plus_ecc608a_devkitc -DCOMPILER=xtensa-
   esp32 -S . -B ./YOUR_BUILD_DIRECTORY -DAFR_ENABLE_ALL_MODULES=1 -
   DAFR_ENABLE_TESTS=0
   ```

   对于 Ninja

   ```
   cmake -DVENDOR=espressif -DBOARD=esp32_plus_ecc608a_devkitc -DCOMPILER=xtensa-
   esp32 -S . -B ./YOUR_BUILD_DIRECTORY -DAFR_ENABLE_ALL_MODULES=1 -
   DAFR_ENABLE_TESTS=0 -GNinja
   ```

4. 擦除闪存，然后刷写主板。

   对于 Unix Makefiles

   ```
   make -C ./YOUR_BUILD_DIRECTORY erase_flash
   ```

   ```
   make -C ./YOUR_BUILD_DIRECTORY flash
   ```

对于 Ninja

```
ninja -C ./YOUR_BUILD_DIRECTORY erase_flash
```

```
ninja -C ./YOUR_BUILD_DIRECTORY flash
```

其他信息

有关使用 Espressif ESP32 主板和对其进行故障排除的更多信息，请参阅以下主题：

- 在你自己的项目中使用 FreeRTOS CMake ESP32
- 故障排除
- 调试

乐鑫-S ESP32 2 入门

> **Important**
>
> 该参考集成托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

> **Note**
>
> 要探索如何将 FreeRTOS 模块化库和演示集成到您自己的 Espressif IDF 项目中，请参阅我们精选的-C3 平台参考集成。 ESP32

本教程向您展示如何开始使用乐鑫 ESP32-S2 SoC 和-S2-Saola-1 开发板。ESP32

概览

该教程将指导您完成以下步骤：

1. 将主板连接到主机。
2. 在您的主机上安装软件，以开发和调试微控制器主板的嵌入式应用程序。

3. 将 FreeRTOS 演示应用程序交叉编译为二进制映像。
4. 将应用程序二进制映像加载到您的主板中，然后运行该应用程序。
5. 使用串行连接监控和调试正在运行的应用程序。

先决条件

在开始在乐鑫看板上使用 FreeRTOS 之前，您必须设置账户和权限。 AWS

注册获取 AWS 账户

如果您没有 AWS 账户，请完成以下步骤来创建一个。

要注册 AWS 账户

1. 打开https://portal.aws.amazon.com/billing/注册。
2. 按照屏幕上的说明操作。

   注册过程的一部分涉及接听电话或短信，并在电话键盘上输入验证码。

   当您注册时 AWS 账户，就会创建AWS 账户根用户一个。根用户有权访问该账户中的所有 AWS 服务 和资源。作为最佳安全实践，请为用户分配管理访问权限，并且只使用根用户来执行需要根用户访问权限的任务。

AWS 注册过程完成后会向您发送一封确认电子邮件。您可以随时前往 https://aws.amazon.com/并选择“我的账户”，查看您当前的账户活动并管理您的账户。

创建具有管理访问权限的用户

注册后，请保护您的安全 AWS 账户 AWS 账户根用户 AWS IAM Identity Center，启用并创建管理用户，这样您就不会使用 root 用户执行日常任务。

保护你的 AWS 账户根用户

1. 选择 Root 用户并输入您的 AWS 账户 电子邮件地址，以账户所有者的身份登录。AWS Management Console在下一页上，输入您的密码。

   要获取使用根用户登录方面的帮助，请参阅《AWS 登录 用户指南》中的 Signing in as the root user。
2. 为您的根用户启用多重身份验证（MFA）。

有关说明，请参阅 I A M 用户指南中的为 AWS 账户 根用户启用虚拟 MFA 设备（控制台）。

创建具有管理访问权限的用户

1. 启用 IAM Identity Center。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的 Enabling AWS IAM Identity Center。
2. 在 IAM Identity Center 中，为用户授予管理访问权限。

   有关使用 IAM Identity Center 目录 作为身份源的教程，请参阅《用户指南》 IAM Identity Center 目录中的使用默认设置配置AWS IAM Identity Center 用户访问权限。

以具有管理访问权限的用户身份登录

- 要使用您的 IAM Identity Center 用户身份登录，请使用您在创建 IAM Identity Center 用户时发送到您的电子邮件地址的登录网址。

  有关使用 IAM Identity Center 用户登录的帮助，请参阅AWS 登录 用户指南中的登录 AWS 访问门户。

将访问权限分配给其他用户

1. 在 IAM Identity Center 中，创建一个权限集，该权限集遵循应用最低权限的最佳做法。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的 Create a permission set。
2. 将用户分配到一个组，然后为该组分配单点登录访问权限。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的 Add groups。

要提供访问权限，请为您的用户、组或角色添加权限：

- 中的用户和群组 AWS IAM Identity Center：

  创建权限集合。按照《AWS IAM Identity Center 用户指南》中创建权限集的说明进行操作。
- 通过身份提供商在 IAM 中托管的用户：

创建适用于身份联合验证的角色。按照《IAM 用户指南》中针对第三方身份提供商创建角色（联合身份验证）的说明进行操作。

- IAM 用户：
  - 创建您的用户可以担任的角色。按照《IAM 用户指南》中为 IAM 用户创建角色的说明进行操作。
  - （不推荐使用）将策略直接附加到用户或将用户添加到用户组。按照《IAM 用户指南》中向用户添加权限（控制台）中的说明进行操作。

开始使用

> **Note**
>
> 本教程中的 Linux 命令要求您使用 Bash Shell。

1. 设置 Espressif 硬件。

   有关设置 ESP32-S2 开发板硬件的信息，请参阅-S ESP322-Saola- 1 入门指南。

   > **Important**
   >
   > 当您阅读到 Espressif 手册的入门 部分时，请暂停并返回到此页面上的说明部分。

2. 从以下网址下载亚马逊 FreeRTOS。GitHub（有关说明，请参阅 README.md 文件。）
3. 设置开发环境。

   要与您的主板通信，必须安装工具链。Espressif 提供了 ESP-IDF 来为主板开发软件。由于 ESP-IDF 将其 FreeRTOS 内核版本集成为组件，因此，Amazon FreeRTOS 包含删除了 FreeRTOS 内核的 ESP-IDF v4.2 自定义版本。这修复了编译时重复文件的问题。要使用 Amazon FreeRTOS 附带的 ESP-IDF v4.2 的自定义版本，请按照以下主机操作系统的说明进行操作。

   Windows

   1. 下载 ESP-IDF 的 Windows 版通用在线安装程序。
   2. 运行通用在线安装程序。
   3. 在下载或使用 ESP-IDF步骤，请选择使用现有 ESP-IDF 目录并将选择现有 ESP-IDF 目录设置为 *freertos*/vendors/espressif/esp-idf。

4. 完成安装。

macOS

1. 按照适用于 macOS 的工具链标准设置先决条件 (ESP-IDF v4.2) 中的说明进行操作。

> **Important**
> 在后续步骤下的“获取 ESP-IDF”说明部分，请暂停并返回到此页面上的说明。

2. 打开一个命令行窗口。
3. 转到 FreeRTOS 下载目录，然后运行以下脚本来下载并安装适用于您平台的 espressif 工具链。

```
vendors/espressif/esp-idf/install.sh
```

4. 使用以下命令以将 ESP-IDF 工具链工具添加到终端路径中。

```
source vendors/espressif/esp-idf/export.sh
```

Linux

1. 按照适用于 Linux 的工具链标准设置先决条件 (ESP-IDF v4.2) 中的说明进行操作。

> **Important**
> 在后续步骤下的“获取 ESP-IDF”说明部分，请暂停并返回到此页面上的说明。

2. 打开一个命令行窗口。
3. 转到 FreeRTOS 下载目录，然后运行以下脚本来下载并安装适用于您平台的 Espressif 工具链。

```
vendors/espressif/esp-idf/install.sh
```

4. 使用以下命令以将 ESP-IDF 工具链工具添加到终端路径中。

```
source vendors/espressif/esp-idf/export.sh
```

4. 建立串行连接。

   a. 要在主机和 ESP32-DevKit C 之间建立串行连接，请安装 CP21 0x USB 到 UART Bridge VCP 驱动程序。您可以从 Silicon Labs 下载这些驱动程序。

   b. 按照步骤与建立串行连接 ESP32。

   c. 建立串行连接后，记下主板连接的串行端口。您需要它来刷写演示。

配置 FreeRTOS 演示应用程序

在本教程中，FreeRTOS 配置文件位于以下文件中：*freertos*/vendors/espressif/boards/*board-name*/aws_demos/config_files/FreeRTOSConfig.h。（例如，如果选择 AFR_BOARD espressif.esp32_devkitc，则配置文件位于以下文件中：*freertos*/vendors/espressif/boards/esp32/aws_demos/config_files/FreeRTOSConfig.h。）

1. 如果您运行的是 macOS 或 Linux，请打开终端提示符。如果您运行的是 Windows，请打开“ESP-IDF 4.x CMD”应用程序（如果您在安装 ESP-IDF 工具链时包含了此选项），否则打开“命令提示符”应用程序。
2. 要验证您是否已安装 Python3，请运行以下命令：

   ```
   python --version
   ```

   此时显示已安装的版本。如果您未安装 Python 3.0.1 或更高版本，可以从 Python 网站进行安装。
3. 您需要 AWS 命令行界面 (CLI) 才能运行 AWS IoT 命令。如果你运行的是 Windows，请使用该easy_install awscli AWS 命令在 “命令” 或 “ESP-IDF 4.x CMD” 应用程序中安装 CLI。

   如果你运行的是 macOS 或 Linux，请参阅安装 CLI AWS。
4. 运行

   ```
   aws configure
   ```

   并使用您的 AWS 访问密钥 ID、私有访问密钥和默认 AWS 区域配置 AWS CLI。有关更多信息，请参阅配置 AWS CLI。
5. 使用以下命令安装适用于 Python 的 AWS 开发工具包 (boto3)：

   - 在 Windows 上，在“命令”或“ESP-IDF 4.x CMD”应用程序中，运行

```
easy_install boto3
```

- 在 macOS 或 Linux 上，运行

```
pip install tornado nose --user
```

然后运行

```
pip install boto3 --user
```

FreeRTOS 包含 SetupAWS.py 脚本，可以更轻松地设置您的 Espressif 主板以连接到 AWS IoT。

运行配置脚本

1. 要配置此脚本，请打开 *freertos*/tools/aws_config_quick_start/configure.json 并设置以下属性：

   afr_source_dir

   计算机上的 *freertos* 目录的完整路径。确保您使用正斜杠来指定此路径。

   thing_name

   您要为代表您的看板 AWS IoT 的事物分配的名称。

   wifi_ssid

   Wi-Fi 网络的 SSID。

   wifi_password

   Wi-Fi 网络的密码。

   wifi_security

   Wi-Fi 网络的安全类型。下面是有效的安全类型：

   - eWiFiSecurityOpen（开放，不安全）
   - eWiFiSecurityWEP（WEP 安全性）
   - eWiFiSecurityWPA（WPA 安全性）
   - eWiFiSecurityWPA2（WPA2 安全）

2. 如果您运行的是 macOS 或 Linux，请打开终端提示符。如果您运行的是 Windows，请打开“ESP-IDF 4.x CMD”或“命令”应用程序。
3. 导航到 *freertos*/tools/aws_config_quick_start 目录运行

```
python SetupAWS.py setup
```

脚本执行以下操作：

- 创建 AWS IoT 事物、证书和策略。
- 将 AWS IoT 策略附加到证书，将证书附加到 AWS IoT 事物。
- 使用您的 AWS IoT 终端节点、Wi-Fi SSID 和凭证填充 aws_clientcredential.h 文件
- 设置您的证书和私有密钥格式，然后将其写入 aws_clientcredential_keys.h 标头文件

> **Note**
>
> 出于演示目的，对该证书进行了硬编码。生产级应用程序应将这些文件存储在安全位置。

有关 SetupAWS.py 的更多信息，请参阅 *freertos*/tools/aws_config_quick_start 目录中的 README.md 文件。

监控云端上的 MQTT 消息 AWS

在运行 FreeRTOS 演示项目之前，您可以在控制台中 AWS IoT 设置 MQTT 客户端，以监控您的设备发送到云端的消息。 AWS

使用 MQTT 客户端订阅 M AWS IoT QTT 主题

1. 登录 AWS IoT 控制台。
2. 在导航窗格中，选择测试，然后选择 MQTT 测试客户端。
3. 在 Subscription topic (订阅主题)中，输入 *your-thing-name*/example/topic，然后选择 Subscribe to topic (订阅主题)。

当演示项目在您的设备上成功运行时，您会多次看到“Hello World！” 发送到您订阅的主题。

使用 idf.py 脚本构建、刷写和运行 FreeRTOS 演示项目

您可以使用 Espressif 的 IDF 实用工具生成构建文件，构建应用程序二进制文件和刷写主板。

在 Windows、Linux 和 macOS 上构建和刷写 FreeRTOS (ESP-IDF v4.2)

使用 `idf.py` 脚本构建项目并将二进制文件刷写到设备上。

> **Note**
>
> 某些设置可能需要您使用在 `idf.py` 中使用端口选项 `-p port-name` 来指定正确的端口，如以下示例所示。
>
> ```
> idf.py -p /dev/cu.usbserial-00101301B flash
> ```

构建和刷写项目

1. 转到 FreeRTOS 下载目录的根目录。
2. 在命令行窗口中输入以下命令以将 ESP-IDF 工具添加到终端 PATH 中：

   Windows（“命令”应用程序）

   ```
   vendors\espressif\esp-idf\export.bat
   ```

   Windows（“ESP-IDF 4.x CMD”应用程序）

   （打开应用程序时就已经完成此操作。）

   Linux / macOS

   ```
   source vendors/espressif/esp-idf/export.sh
   ```
3. 在 `build` 目录中配置 cmake 并使用以下命令构建固件映像。

   ```
   idf.py -DVENDOR=espressif -DBOARD=esp32s2_saola_1 -DCOMPILER=xtensa-esp32s2 build
   ```

   您应该会看到以下示例的输出。

   ```
   Executing action: all (aliases: build)
   ```

```
   Running cmake in directory /path/to/hello_world/build
   Executing "cmake -G Ninja -DPYTHON_DEPS_CHECKED=1 -DESP_PLATFORM=1
 -DVENDOR=espressif -DBOARD=esp32s2_saola_1 -DCOMPILER=xtensa-esp32s2 -
DCCACHE_ENABLE=0 /path/to/hello_world"...
   -- The C compiler identification is GNU 8.4.0
   -- The CXX compiler identification is GNU 8.4.0
   -- The ASM compiler identification is GNU

   ... (more lines of build system output)

   [1628/1628] Generating binary image from built executable
   esptool.py v3.0
   Generated /path/to/hello_world/build/aws_demos.bin

   Project build complete. To flash, run this command:
   esptool.py -p (PORT) -b 460800 --before default_reset --after hard_reset --chip
 esp32s2  write_flash --flash_mode dio --flash_size detect --flash_freq 80m 0x1000
 build/bootloader/bootloader.bin 0x8000 build/partition_table/partition-table.bin
 0x16000 build/ota_data_initial.bin 0x20000 build/aws_demos.bin
   or run 'idf.py -p (PORT) flash'
```

如果没有错误，构建会生成固件二进制 .bin 文件。

4. 使用以下命令擦除开发主板的闪存。

```
idf.py erase_flash
```

5. 使用 `idf.py` 脚本将应用程序二进制文件刷写到主板。

```
idf.py flash
```

6. 使用以下命令监控主板串行端口的输出。

```
idf.py monitor
```

> **Note**
>
> - 您可以合并这些命令，如以下示例所示。
>
>   ```
>   idf.py erase_flash flash monitor
>   ```
>
> - 对于某些主机设置，您必须在刷写主板时指定端口，如以下示例所示。

```
idf.py erase_flash flash monitor -p /dev/ttyUSB1
```

使用构建和刷新 FreeRTOS CMake

除了使用 IDF SDK 提供的`idf.py`脚本来构建和运行代码外，您还可以使用构建项目 CMake。目前，它支持 Unix Makefile 和 Ninja 构建系统。

构建和刷写项目

1. 在命令行窗口中，转到 FreeRTOS 下载目录的根目录。
2. 运行以下脚本，将 ESP-IDF 工具添加到 Shell 的 PATH 中。
   - Windows

     ```
     vendors\espressif\esp-idf\export.bat
     ```

   - Linux / macOS

     ```
     source vendors/espressif/esp-idf/export.sh
     ```

3. 使用以下命令来生成构建文件。
   - 对于 Unix Makefiles

     ```
     cmake -DVENDOR=espressif -DBOARD=esp32s2_saola_1 -DCOMPILER=xtensa-esp32s2 -S . -
     B ./YOUR_BUILD_DIRECTORY -DAFR_ENABLE_ALL_MODULES=1 -DAFR_ENABLE_TESTS=0
     ```

   - 对于 Ninja

     ```
     cmake -DVENDOR=espressif -DBOARD=esp32s2_saola_1 -DCOMPILER=xtensa-esp32s2 -S . -
     B ./YOUR_BUILD_DIRECTORY -DAFR_ENABLE_ALL_MODULES=1 -DAFR_ENABLE_TESTS=0 -GNinja
     ```

4. 构建 项目。
   - 对于 Unix Makefiles

     ```
     make -C ./YOUR_BUILD_DIRECTORY -j8
     ```

   - 对于 Ninja

```
ninja -C ./YOUR_BUILD_DIRECTORY -j8
```

5. 擦除闪存，然后刷写主板。

   - 对于 Unix Makefiles

     ```
     make -C ./YOUR_BUILD_DIRECTORY erase_flash
     ```

     ```
     make -C ./YOUR_BUILD_DIRECTORY flash
     ```

   - 对于 Ninja

     ```
     ninja -C ./YOUR_BUILD_DIRECTORY erase_flash
     ```

     ```
     ninja -C ./YOUR_BUILD_DIRECTORY flash
     ```

其他信息

有关使用 Espressif ESP32 主板和对其进行故障排除的更多信息，请参阅以下主题：

- 在你自己的项目中使用 FreeRTOS CMake ESP32
- 故障排除
- 调试

英飞凌 00 XMC48 物联网连接套件入门

> **Important**
>
> 该参考集成托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

本教程提供了英飞凌 00 XMC48 物联网连接套件的入门说明。如果您没有 Infineon 00 XMC48 IoT 连接套件，请访问 AWS 合作伙伴设备目录，从我们的合作伙伴处购买。

如果要打开与主板的串行连接以查看日志记录和调试信息，则需要一个 3.3V USB/Serial converter, in addition to the XMC4800 IoT Connectivity Kit. The CP2104 is a common USB/Serial 转换器，该转换器广泛应用于 Adafruit 的 04 Friend 等主板。CP21

在开始之前，您必须进行配置 AWS IoT 并下载 FreeRTOS 才能将您的设备连接到云端。 AWS 有关说明，请参阅初始步骤：在本教程中，FreeRTOS 下载目录的路径称为 *freertos*。

概览

本教程包含有关以下入门步骤的说明：

1. 在主机上安装软件来开发和调试微控制器主板的嵌入式应用程序。
2. 将 FreeRTOS 演示应用程序交叉编译为二进制映像。
3. 将应用程序二进制映像加载到您的主板上，然后运行该应用程序。
4. 跨串行连接与主板上运行的应用程序进行交互，以便进行监视和调试。

设置开发环境

FreeRTOS 使用英飞凌的 DAVE 开发环境对 00 进行编程。 XMC48在开始之前，您需要下载并安装 DAVE 和一些 J-Link 驱动程序，以便与板载调试器通信。

安装 DAVE

1. 转到 Infineon 的 DAVE 软件下载页面。
2. 根据您的操作系统选择 DAVE 程序包，然后提交您的注册信息。注册到 Infineon 之后，您应收到确认电子邮件，其中包括下载 .zip 文件的链接。
3. 下载 DAVE 程序包 .zip 文件 (`DAVE_`*`version_os_date`*`.zip`)，然后将其解压缩到您要安装 DAVE 的位置（例如，`C:\DAVE4`）。

   > **Note**
   >
   > 一些 Windows 用户报告在使用 Windows 资源管理器解压缩文件时遇到问题。我们建议您使用第三方程序，如 7-Zip。

4. 要启动 DAVE，运行在解压缩的 `DAVE_`*`version_os_date`*`.zip` 文件夹中找到的可执行文件。

有关更多信息，请参阅 DAVE 快速入门指南。

安装 Segger J-Link 驱动程序

要与 XMC48 00 Relax EtherCAT 主板的板载调试探器进行通信，你需要 J-Link 软件和文档包中包含的驱动程序。您可以从 Segger 的 J-Link 软件下载页面下载 J-Link Software and Documentation Pack。

建立串行连接

设置串行连接是可选的，但建议设置。通过串行连接，您的主板使用可在开发计算机上查看的格式，发送日志记录和调试信息。

XMC4800 演示应用程序在引脚 P0.0 和 P0.1 上使用 UART 串行连接，这些引脚标在 00 R XMC48 elax EtherCAT 主板的丝网印刷上。要设置串行连接，请执行以下操作：

1. 将标记为“RX<P0.0”的针脚连接到您的 USB/串口转换器的“TX”针脚。
2. 将标记为“TX>P0.1”的针脚连接到您的 USB/串口转换器的“RX”针脚。
3. 将您的串口转换器的接地针脚连接到主板上标记为“GND”的其中一个针脚。设备必须共享一个公用接地。

电源从 USB 调试端口提供，因此无需将串口适配器的正电压针脚连接到主板。

> **Note**
>
> 一些串行电缆使用 5V 信号电平。 XMC4800 主板和 Wi-Fi Click 模块需要 3.3V 电压。请勿使用主板的 IOREF 跳线将主板的信号更改为 5V。

通过电缆连接，您可以打开与终端仿真器（例如 GNU 屏幕）的串行连接。默认情况下，波特率设置为 115200，8 个数据位，无奇偶校验，1 个停止位。

在云上监控 MQTT 消息

在运行 FreeRTOS 演示之前，您可以在控制台中 AWS IoT 设置 MQTT 客户端，以监控您的设备发送到云端的消息。 AWS

使用 MQTT 客户端订阅 M AWS IoT QTT 主题

1. 登录 AWS IoT 控制台。
2. 在导航窗格中选择测试，然后选择 MQTT 测试客户端，以便打开 MQTT 客户端。
3. 在 Subscription topic (订阅主题)中，输入 ***your-thing-name*****/example/topic**，然后选择 Subscribe to topic (订阅主题)。

当演示项目在您的设备上成功运行时，您会多次看到“Hello World！” 发送到您订阅的主题。

构建并运行 FreeRTOS 演示项目

将 FreeRTOS 演示导入 DAVE

1. 启动 DAVE。
2. 在 DAVE 中，依次选择 File (文件) 和 Import (导入)。在 Import (导入) 窗口中，展开 Infineon 文件夹，选择 DAVE Project (DAVE 项目)，然后选择 Next (下一步)。
3. 在 “导入 DAVE 项目” 窗口中，选择 “选择根目录”，选择 “浏览”，然后选择 XMC48 00 演示项目。

   在您解压缩 FreeRTOS 下载文件的目录中，演示项目位于 `projects/infineon/xmc4800_iotkit/dave4/aws_demos` 中。

   确保 Copy Projects Into Workspace (将项目复制到工作区) 处于未选中状态。
4. 选择完成。

   `aws_demos` 项目应导入到工作区中并且激活。
5. 从 Project (项目) 菜单，选择 Build Active Project (生成活动项目)。

   确保项目生成，没有错误。

运行 FreeRTOS 演示项目

1. 使用 USB 线将你的 XMC48 00 IoT 连接套件连接到电脑。主板有两个 microUSB 连接器。使用标记为“X101”的一个，旁边的主板上印刷字样显示“Debug”。
2. 从 Project (项目) 菜单，选择 Rebuild Active Project (重新生成活动项目) 以重新生成 `aws_demos` 并确保接受您的配置更改。
3. 在 Project Explorer (项目资源管理器) 中，右键单击 `aws_demos`，选择 Debug As (调试方式)，然后选择 DAVE C/C++ Application (DAVE C/C++ 应用程序)。
4. 双击 GDB SEGGER J-Link Debugging (GDB SEGGER J-Link 调试) 以创建调试确认。选择 Debug (调试)。
5. 当调试器在 `main()` 中的断点停止时，在 Run (运行) 菜单中选择 Resume (恢复)。

在 AWS IoT 控制台中，步骤 4-5 中的 MQTT 客户端应显示您的设备发送的 MQTT 消息。如果您使用串行连接，则您会看到如下所示的 UART 输出：

```
0 0 [Tmr Svc] Starting key provisioning...
1 1 [Tmr Svc] Write root certificate...
2 4 [Tmr Svc] Write device private key...
3 82 [Tmr Svc] Write device certificate...
4 86 [Tmr Svc] Key provisioning done...
5 291 [Tmr Svc] Wi-Fi module initialized. Connecting to AP...
.6 8046 [Tmr Svc] Wi-Fi Connected to AP. Creating tasks which use network...
7 8058 [Tmr Svc] IP Address acquired [IP Address]
8 8058 [Tmr Svc] Creating MQTT Echo Task...
9 8059 [MQTTEcho] MQTT echo attempting to connect to [MQTT Broker].
...10 23010 [MQTTEcho] MQTT echo connected.
11 23010 [MQTTEcho] MQTT echo test echoing task created.
.12 26011 [MQTTEcho] MQTT Echo demo subscribed to iotdemo/#
13 29012 [MQTTEcho] Echo successfully published 'Hello World 0'
.14 32096 [Echoing] Message returned with ACK: 'Hello World 0 ACK'
.15 37013 [MQTTEcho] Echo successfully published 'Hello World 1'
16 40080 [Echoing] Message returned with ACK: 'Hello World 1 ACK'
.17 45014 [MQTTEcho] Echo successfully published 'Hello World 2'
.18 48091 [Echoing] Message returned with ACK: 'Hello World 2 ACK'
.19 53015 [MQTTEcho] Echo successfully published 'Hello World 3'
.20 56087 [Echoing] Message returned with ACK: 'Hello World 3 ACK'
.21 61016 [MQTTEcho] Echo successfully published 'Hello World 4'
22 64083 [Echoing] Message returned with ACK: 'Hello World 4 ACK'
.23 69017 [MQTTEcho] Echo successfully published 'Hello World 5'
.24 72091 [Echoing] Message returned with ACK: 'Hello World 5 ACK'
.25 77018 [MQTTEcho] Echo successfully published 'Hello World 6'
26 80085 [Echoing] Message returned with ACK: 'Hello World 6 ACK'
.27 85019 [MQTTEcho] Echo successfully published 'Hello World 7'
.28 88086 [Echoing] Message returned with ACK: 'Hello World 7 ACK'
.29 93020 [MQTTEcho] Echo successfully published 'Hello World 8'
.30 96088 [Echoing] Message returned with ACK: 'Hello World 8 ACK'
.31 101021 [MQTTEcho] Echo successfully published 'Hello World 9'
32 104102 [Echoing] Message returned with ACK: 'Hello World 9 ACK'
.33 109022 [MQTTEcho] Echo successfully published 'Hello World 10'
.34 112047 [Echoing] Message returned with ACK: 'Hello World 10 ACK'
.35 117023 [MQTTEcho] Echo successfully published 'Hello World 11'
36 120089 [Echoing] Message returned with ACK: 'Hello World 11 ACK'
.37 122068 [MQTTEcho] MQTT echo demo finished.
38 122068 [MQTTEcho] ----Demo finished----
```

使用以下方法构建 FreeRTOS 演示 CMake

如果您不想使用 IDE 进行 FreeRTOS 开发，也可以 CMake 使用来生成和运行使用第三方代码编辑器和调试工具开发的演示应用程序或应用程序。

> **Note**
>
> 本节介绍如何 CMake 在 Windows 上使用 minGW 作为原生构建系统。有关与其他操作系统和选项 CMake 一起使用的更多信息，请参阅 CMake 与 FreeRTOS 一起使用。（MinGW 是一个适用于本机 Microsoft Windows 应用程序的简约开发环境。）

要使用构建 FreeRTOS 演示 CMake

1. 设置 GNU Arm 嵌入式工具链。

   a. 从 Arm 嵌入式工具链下载页面下载工具链的 Windows 版本。

   > **Note**
   >
   > 我们建议您下载“8-2018-q4-major”之外的版本，因为在该版本中使用“objcopy”实用程序方面存在错误。

   b. 打开下载的工具链安装程序，按照安装向导中的说明安装工具链。

   > **Important**
   >
   > 在安装向导的最后一个页面上，选择 Add path to environment variable (将路径添加到环境变量) 以将工具链路径添加到系统路径环境变量。

2. 安装 CMake 并使用 minW。

   有关说明，请参阅CMake 先决条件。

3. 创建一个包含生成的构建文件的文件夹 (*build-folder*)。
4. 将目录更改为您的 FreeRTOS 下载目录 (*freertos*)，并使用以下命令来生成构建文件：

   ```
   cmake -DVENDOR=infineon -DBOARD=xmc4800_iotkit -DCOMPILER=arm-gcc -S . -B build-folder -G "MinGW Makefiles" -DAFR_ENABLE_TESTS=0
   ```

5. 将目录更改为 build 目录 (*build-folder*)，然后使用以下命令生成二进制文件：

```
cmake --build . --parallel 8
```

此命令会在构建目录中构建输出二进制文件 aws_demos.hex。

6. 使用 JLINK 刷写和运行映像。

   a. 在生成目录 (*build-folder*) 中，使用以下命令创建 Flash 脚本：

   ```
   echo loadfile aws_demos.hex > flash.jlink
   ```

   ```
   echo r >> flash.jlink
   ```

   ```
   echo g >> flash.jlink
   ```

   ```
   echo q >> flash.jlink
   ```

   b. 使用 JLNIK 可执行文件刷写映像。

   ```
   JLINK_PATH\JLink.exe  -device XMC4800-2048 -if SWD -speed auto -CommanderScript
    flash.jlink
   ```

   通过您与主板建立的串行连接应该可以看到应用程序日志。

故障排除

如果您还没有，请务必进行配置 AWS IoT 并下载 FreeRTOS，以便将您的设备连接到云端。 AWS 有关说明，请参阅初始步骤：

有关 FreeRTOS 入门的常规故障排除信息，请参阅问题排查入门。

开始使用英飞凌 OPTIGA Trust X 和 00 XMC48 物联网连接套件

> **Important**
>
> 该参考集成托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

本教程提供了英飞凌 OPTIGA Trust X Secure Element 和 00 XMC48 物联网连接套件的入门说明。与英飞凌 00 XMC48 物联网连接套件入门 教程相比，本指南将介绍如何使用 Infineon OPTIGA Trust X 安全元件提供安全凭证。

您需要以下硬件：

1. 托管 MCU-英飞凌 00 XMC48 物联网连接套件，请访问 AWS 合作伙伴设备目录，从我们的合作伙伴处购买。
2. 安全扩展包：
   - 安全元件 - Infineon OPTIGA Trust X。

     访问 AWS 合作伙伴设备目录，从我们的合作伙伴处购买。
   - 个性化主板 – Infineon OPTIGA 个性化主板。
   - 适配器板-英飞凌 MyIo T 型适配器。

要按照此处的步骤操作，必须打开与主板的串行连接以查看日志记录和调试信息。（其中一个步骤要求您从开发板的串行调试输出中复制公钥并将其粘贴到文件中。）为此，除了 00 XMC48 IoT 连接套件之外，您还需要一个 3.3V 的 USB/串行转换器。众所周知，JBtek EL-PN-47310126 USB/串行转换器适用于此演示。您还需要三根 male-to-male跳线（用于接收 (RX)、传输 (TX) 和接地 (GND)），才能将串行电缆连接到英飞凌 MyIo T 适配器板。

在开始之前，您必须进行配置 AWS IoT 并下载 FreeRTOS 才能将您的设备连接到云端。 AWS 有关说明，请参阅 选项 2：板载私有密钥生成。在本教程中，FreeRTOS 下载目录的路径称为 *freertos*。

概览

本教程包含以下步骤：

1. 在主机上安装软件，以开发和调试微控制器主板的嵌入式应用程序。
2. 将 FreeRTOS 演示应用程序交叉编译为二进制映像。
3. 将应用程序二进制映像加载到您的主板中，然后运行该应用程序。
4. 要进行监控和调试，请跨串行连接与主板上运行的应用程序进行交互。

设置开发环境

FreeRTOS 使用英飞凌的 DAVE 开发环境对 00 进行编程。 XMC48在开始之前，请下载并安装 DAVE 和一些 J-Link 驱动程序，以便与板载调试器通信。

安装 DAVE

1. 转到 Infineon 的 DAVE 软件下载页面。
2. 根据您的操作系统选择 DAVE 程序包，然后提交您的注册信息。注册后，您应收到确认电子邮件，其中包括下载 .zip 文件的链接。
3. 下载 DAVE 程序包 .zip 文件 (DAVE_*version_os_date*.zip)，然后将其解压缩到您要安装 DAVE 的位置（例如，C:\DAVE4）。

   > **Note**
   >
   > 一些 Windows 用户报告在使用 Windows 资源管理器解压缩文件时遇到问题。我们建议您使用第三方程序，如 7-Zip。

4. 要启动 DAVE，运行在解压缩的 DAVE_*version_os_date*.zip 文件夹中找到的可执行文件。

有关更多信息，请参阅 DAVE 快速入门指南。

安装 Segger J-Link 驱动程序

要与 XMC48 00 IoT Connectivity 套件的板载调试器进行通信，你需要 J-Link 软件和文档包中包含的驱动程序。您可以从 Segger 的 J-Link 软件下载页面下载 J-Link Software and Documentation Pack。

建立串行连接

将 USB/串口转换器电缆连接至 Infineon Shield2Go 适配器。此操作可使您的主板使用可在开发计算机上查看的格式发送日志记录和调试信息。要设置串行连接，请执行以下操作：

1. 将 RX 针脚连接到您的 USB/串口转换器的 TX 针脚。
2. 将 TX 引脚连接到 USB/串口转换器的 RX 引脚。
3. 将您的串口转换器的接地针脚连接到主板上的 GND 针脚之一。设备必须共享一个公用接地。

电源从 USB 调试端口提供，因此无需将串口适配器的正电压针脚连接到主板。

> **Note**
>
> 一些串行电缆使用 5V 信号电平。 XMC4800 主板和 Wi-Fi Click 模块需要 3.3V 电压。请勿使用主板的 IOREF 跳线将主板的信号更改为 5V。

通过电缆连接，您可以打开与终端仿真器（例如 GNU 屏幕）的串行连接。默认情况下，波特率设置为 115200，8 个数据位，无奇偶校验，1 个停止位。

在云上监控 MQTT 消息

在运行 FreeRTOS 演示项目之前，您可以在控制台中 AWS IoT 设置 MQTT 客户端，以监控您的设备发送到云端的消息。 AWS

使用 MQTT 客户端订阅 M AWS IoT QTT 主题

1. 登录 AWS IoT 控制台。
2. 在导航窗格中选择测试，然后选择 MQTT 测试客户端，以便打开 MQTT 客户端。
3. 在 Subscription topic (订阅主题)中，输入 ***your-thing-name*/example/topic**，然后选择 Subscribe to topic (订阅主题)。

当演示项目在您的设备上成功运行时，您会多次看到“Hello World！” 发送到您订阅的主题。

构建并运行 FreeRTOS 演示项目

将 FreeRTOS 演示导入 DAVE

1. 启动 DAVE。
2. 从 DAVE 中，选择 File (文件)，然后选择 Import (导入)。展开 Infineon 文件夹，选择 DAVE Project (DAVE 项目)，然后选择 Next (下一步)。
3. 在 “导入 DAVE 项目” 窗口中，选择 “选择根目录”，选择 “浏览”，然后选择 XMC48 00 演示项目。

   在您解压缩 FreeRTOS 下载文件的目录中，演示项目位于 `projects/infineon/xmc4800_plus_optiga_trust_x/dave4/aws_demos/dave4` 中。

   确保清除了 Copy Projects Into Workspace (将项目复制到工作区)。
4. 选择完成。

   `aws_demos` 项目应导入到工作区中并且激活。
5. 从 Project (项目) 菜单，选择 Build Active Project (生成活动项目)。

   确保项目生成，没有错误。

运行 FreeRTOS 演示项目

1. 从 Project (项目) 菜单，选择 Rebuild Active Project (重新生成活动项目) 以重新生成 aws_demos 并确认接受您的配置更改。
2. 在 Project Explorer (项目资源管理器) 中，右键单击 aws_demos，选择 Debug As (调试方式)，然后选择 DAVE C/C++ Application (DAVE C/C++ 应用程序)。
3. 双击 GDB SEGGER J-Link Debugging (GDB SEGGER J-Link 调试) 以创建调试确认。选择 Debug (调试)。
4. 当调试器在 main() 中的断点停止时，在 Run (运行) 菜单中选择 Resume (恢复)。

此时，请继续执行选项 2：板载私有密钥生成中的公有密钥提取步骤。所有步骤完成后，转到 AWS IoT 控制台。您之前设置的 MQTT 客户端应显示您的设备发送的 MQTT 消息。通过设备的串行连接，您应该在 UART 输出上看到类似以下内容的信息：

```
0 0 [Tmr Svc] Starting key provisioning...
1 1 [Tmr Svc] Write root certificate...
2 4 [Tmr Svc] Write device private key...
3 82 [Tmr Svc] Write device certificate...
4 86 [Tmr Svc] Key provisioning done...
5 291 [Tmr Svc] Wi-Fi module initialized. Connecting to AP...
.6 8046 [Tmr Svc] Wi-Fi Connected to AP. Creating tasks which use network...
7 8058 [Tmr Svc] IP Address acquired [IP Address]
8 8058 [Tmr Svc] Creating MQTT Echo Task...
9 8059 [MQTTEcho] MQTT echo attempting to connect to [MQTT Broker].
...10 23010 [MQTTEcho] MQTT echo connected.
11 23010 [MQTTEcho] MQTT echo test echoing task created.
.12 26011 [MQTTEcho] MQTT Echo demo subscribed to iotdemo/#
13 29012 [MQTTEcho] Echo successfully published 'Hello World 0'
.14 32096 [Echoing] Message returned with ACK: 'Hello World 0 ACK'
.15 37013 [MQTTEcho] Echo successfully published 'Hello World 1'
16 40080 [Echoing] Message returned with ACK: 'Hello World 1 ACK'
.17 45014 [MQTTEcho] Echo successfully published 'Hello World 2'
.18 48091 [Echoing] Message returned with ACK: 'Hello World 2 ACK'
.19 53015 [MQTTEcho] Echo successfully published 'Hello World 3'
.20 56087 [Echoing] Message returned with ACK: 'Hello World 3 ACK'
.21 61016 [MQTTEcho] Echo successfully published 'Hello World 4'
22 64083 [Echoing] Message returned with ACK: 'Hello World 4 ACK'
.23 69017 [MQTTEcho] Echo successfully published 'Hello World 5'
.24 72091 [Echoing] Message returned with ACK: 'Hello World 5 ACK'
.25 77018 [MQTTEcho] Echo successfully published 'Hello World 6'
```

```
26 80085 [Echoing] Message returned with ACK: 'Hello World 6 ACK'
.27 85019 [MQTTEcho] Echo successfully published 'Hello World 7'
.28 88086 [Echoing] Message returned with ACK: 'Hello World 7 ACK'
.29 93020 [MQTTEcho] Echo successfully published 'Hello World 8'
.30 96088 [Echoing] Message returned with ACK: 'Hello World 8 ACK'
.31 101021 [MQTTEcho] Echo successfully published 'Hello World 9'
32 104102 [Echoing] Message returned with ACK: 'Hello World 9 ACK'
.33 109022 [MQTTEcho] Echo successfully published 'Hello World 10'
.34 112047 [Echoing] Message returned with ACK: 'Hello World 10 ACK'
.35 117023 [MQTTEcho] Echo successfully published 'Hello World 11'
36 120089 [Echoing] Message returned with ACK: 'Hello World 11 ACK'
.37 122068 [MQTTEcho] MQTT echo demo finished.
38 122068 [MQTTEcho] ----Demo finished----
```

使用以下方法构建 FreeRTOS 演示 CMake

本节介绍如何 CMake 在 Windows 上使用 minGW 作为原生构建系统。有关与其他操作系统和选项 CMake 一起使用的更多信息，请参阅 CMake 与 FreeRTOS 一起使用。（ MinGW 是一个适用于本机 Microsoft Windows 应用程序的简约开发环境。 ）

如果您不想使用 IDE 进行 FreeRTOS 开发，则可以使用来生成和运行 CMake 使用第三方代码编辑器和调试工具开发的演示应用程序或应用程序。

要使用构建 FreeRTOS 演示 CMake

1. 设置 GNU Arm 嵌入式工具链。

   a. 从 Arm 嵌入式工具链下载页面下载工具链的 Windows 版本。

   > **Note**
   >
   > 由于 objcopy 实用程序中报告了一个错误，我们建议您下载“8-2018-q4-major”以外的版本。

   b. 打开下载的工具链安装程序，然后按照向导中的说明操作。

   c. 在安装向导的最后一个页面上，选择 Add path to environment variable (将路径添加到环境变量) 以将工具链路径添加到系统路径环境变量。

2. 安装 CMake 并使用 minW。

   有关说明，请参阅CMake 先决条件。

3. 创建一个包含生成的构建文件的文件夹 (*build-folder*)。

4. 将目录更改为您的 FreeRTOS 下载目录 (*freertos*)，并使用以下命令来生成构建文件：

```
cmake -DVENDOR=infineon -DBOARD=xmc4800_plus_optiga_trust_x -DCOMPILER=arm-gcc -S .
 -B build-folder -G "MinGW Makefiles" -DAFR_ENABLE_TESTS=0
```

5. 将目录更改为生成目录 (*build-folder*)，然后使用以下命令生成二进制文件：

```
cmake --build . --parallel 8
```

此命令会在构建目录中构建输出二进制文件 aws_demos.hex。

6. 使用 JLINK 刷写和运行映像。

   a. 在生成目录 (*build-folder*) 中，使用以下命令创建 Flash 脚本：

```
echo loadfile aws_demos.hex > flash.jlink
echo r >> flash.jlink
echo g >> flash.jlink
echo q >> flash.jlink
```

   b. 使用 JLNIK 可执行文件刷写映像。

```
JLINK_PATH\JLink.exe  -device XMC4800-2048 -if SWD -speed auto -CommanderScript
 flash.jlink
```

通过您与主板建立的串行连接应该可以看到应用程序日志。继续执行 选项 2：板载私有密钥生成 中的公有密钥提取步骤。完成所有步骤后，进入 AWS IoT 控制台。您之前设置的 MQTT 客户端应显示您的设备发送的 MQTT 消息。

故障排除

有关一般故障排除信息，请参阅问题排查入门。

入门套件 MW32x AWS IoT 入门

> **Important**
>
> 该参考集成托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

AWS IoT 入门套件是基于恩智浦最新集成Cortex M4微控制器 MW32 88 0/88的开发套件MW322，它在单个微控制器芯片上集成了802.11b/g/n Wi-Fi。此开发工具包已通过 FCC 认证。如需更多信息，请参阅AWS 合作伙伴设备目录来从我们的合作伙伴购买一个。88 MW32 0/88 MW322 模块还获得了 FCC 认证，可进行定制和批量销售。

本入门指南介绍如何在主机上交叉编译应用程序和开发工具包，然后使用开发工具包提供的工具将生成的二进制文件加载到主板上。当应用程序开始在主板上运行时，您可以从主机的串行控制台对其进行调试或与其交互。

Ubuntu 16.04 是支持开发和调试的主机平台。您也可以使用其他平台，但这些平台不受官方支持。您必须具有在主机平台上安装软件的权限。构建开发工具包需要以下外部工具：

- Ubuntu 16.04 主机平台
- ARM 工具链版本 4_9_2015q3
- Eclipse 4.9.0 IDE

交叉编译应用程序和开发工具包需要使用 ARM 工具链。SDK 利用最新版本的工具链来优化映像占用空间，在更小的空间中容纳更多功能。本指南假设您使用的是 4_9_2015q3 版本的工具链。建议不要使用旧版本工具链。开发工具包已预先刷写无线微控制器演示项目固件。

主题

- 设置硬件
- 设置开发环境
- 构建并运行 FreeRTOS 演示项目
- 调试
- 故障排除

设置硬件

使用 mini-USB 转 USB 电缆将 MW32x 主板连接到笔记本电脑。将 mini-USB 线缆连接到主板上唯一的 mini-USB 连接器。您不需要更改跳线。

如果将主板连接到笔记本电脑或台式计算机，则不需要外部电源。

该 USB 连接提供以下功能：

- 通过控制台访问主板。在开发主机上注册了一个虚拟 tty/com 端口，该端口可用于访问控制台。

- 通过 JTAG 访问开发主板。这可用于将固件映像加载或卸载到主板的 RAM 或闪存中，或者用于调试目的。

设置开发环境

出于开发目的，最低要求是 ARM 工具链和与开发工具包捆绑在一起的工具。以下各节提供了有关 ARM 工具链设置的详细信息。

GNU 工具链

开发工具包正式支持 GCC 编译器工具链。GNU ARM 的交叉编译器工具链已在 GNU Arm 嵌入式工具链 4.9-2015-q3-update 中发布。

默认情况下，构建系统配置为使用 GNU 工具链。Makefile 假设 GNU 编译器工具链二进制文件在用户的 PATH 上可用，并且可以从 Makefile 中调用。Makefiles 还假设 GNU 工具链二进制文件的文件名带有 `arm-none-eabi-` 前缀。

GCC 工具链可以与 GDB 一起使用，通过 OpenOCD（与开发工具包捆绑在一起）进行调试。其中提供了与 JTAG 接口的软件。

我们推荐使用工具链的 4_9_2015q3 版本。 gcc-arm-embedded

Linux 工具链设置过程

按照以下步骤操作，以便在 Linux 中设置 GCC 工具链。

1. 在 GNU Arm 嵌入式工具链 4.9-2015-q3-update 中下载工具链 tarball。文件是 `gcc-arm-none-eabi-4_9-2015q3-20150921-linux.tar.bz2`。
2. 将文件复制到所选目录。确保目录名称不包含空格。
3. 使用以下命令解压缩文件。

   ```
   tar -vxf filename
   ```

4. 将安装的工具链的路径添加到系统 PATH 中。例如，在位于 /home/*user-name* 目录中的 `.profile` 文件末尾添加以下行。

   ```
   PATH=$PATH:path to gcc-arm-none-eabit-4_9_2015_q3/bin
   ```

> **Note**
>
> Ubuntu 的一些新分发包含 Debian 版本的 GCC 交叉编译器。如果是这样，则必须移除本机交叉编译器，并按照上述设置过程进行操作。

使用 Linux 开发主机

您可以使用任何现代 Linux 桌面发行版，例如 Ubuntu 或 Fedora。但是，我们建议您升级到最新版本。以下步骤已经过验证，可以在Ubuntu 16.04 上运行，并假设您使用的是该版本。

安装程序包

开发工具包有一个脚本，用于在新设置的 Linux 计算机上快速设置开发环境。该脚本会尝试自动检测计算机类型并安装相应的软件，包括 C 库、USB 库、FTDI 库、ncurses、python 和 latex。在本节中，通用目录名称*amzsdk_bundle-x.y.z*表示 AWS SDK 的根目录。实际目录名称可能会有所不同。您必须拥有根权限。

- 导航到 *amzsdk_bundle-x.y.z/* 目录并运行此命令。

```
./lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/bin/installpkgs.sh
```

避免使用 sudo

在本指南中，`flashprog` 操作使用 `flashprog.py` 脚本刷写主板的 NAND，如下所述。同样，`ramload` 操作使用 `ramload.py` 脚本将固件映像从主机直接复制到微控制器的 RAM，无需刷写 NAND。

您可以将 Linux 开发主机配置为执行 `flashprog` 和 `ramload` 操作，而无需每次都使用 `sudo` 命令。为此，请运行以下命令。

```
./lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/bin/perm_fix.sh
```

> **Note**
>
> 您必须这样配置 Linux 开发主机权限才能确保流畅的 Eclipse IDE 体验。

设置串行控制台

将 USB 线缆插入 Linux 主机 USB 插槽中。这会触发对设备的检测。在 /var/log/messages 文件中，或执行 dmesg 命令后，您应该会看到类似以下的消息。

```
Jan 6 20:00:51 localhost kernel: usb 4-2: new full speed USB device using uhci_hcd and
 address 127
Jan 6 20:00:51 localhost kernel: usb 4-2: configuration #1 chosen from 1 choice
Jan 6 20:00:51 localhost kernel: ftdi_sio 4-2:1.0: FTDI USB Serial Device converter
 detected
Jan 6 20:00:51 localhost kernel: ftdi_sio: Detected FT2232C
Jan 6 20:00:51 localhost kernel: usb 4-2: FTDI USB Serial Device converter now attached
 to ttyUSB0
Jan 6 20:00:51 localhost kernel: ftdi_sio 4-2:1.1: FTDI USB Serial Device converter
 detected
Jan 6 20:00:51 localhost kernel: ftdi_sio: Detected FT2232C
Jan 6 20:00:51 localhost kernel: usb 4-2: FTDI USB Serial Device converter now attached
 to ttyUSB1
```

确认已创建两个 ttyUSB 设备。第二个 ttyUSB 是串行控制台。在上面的示例中，它被命名为 “ttyUSB1”。

在本指南中，我们使用 minicom 来查看串行控制台的输出。您也可以使用其他串行程序，例如 putty。运行以下命令以在设置模式下执行 minicom。

```
minicom -s
```

在 minicom 中，转到串行端口设置并获取以下设置。

```
| A - Serial Device : /dev/ttyUSB1
| B - Lockfile Location : /var/lock
| C - Callin Program :
| D - Callout Program :
| E - Bps/Par/Bits : 115200 8N1
| F - Hardware Flow Control : No
| G - Software Flow Control : No
```

您可以将这些设置保存在 minicom 中以供将来使用。minicom 窗口现在显示来自串行控制台的消息。

选择串行控制台窗口，然后按 Enter 键。这会在屏幕上显示一个哈希值 (#)。

> **Note**
>
> 开发主板包含一个 FTDI 硅芯片设备。FTDI 设备公开了主机的两个 USB 接口。第一个接口与 MCU 的 JTAG 功能相关联，第二个接口与 MCU 的物理 UARTx 端口相关联。

安装 OpenOCD

OpenOCD 是一款为嵌入式目标设备提供调试、系统内编程和边界扫描测试的软件。

需要 OpenOCD 版本 0.9。Eclipse 功能也需要该软件。如果您的 Linux 主机上安装了早期版本（例如版本 0.7），请使用适用于当前用途的 Linux 发行版的相应命令删除该存储库。

运行标准 Linux 命令来安装 OpenOCD，

```
apt-get install openocd
```

如果上述命令未安装 0.9 或更高版本，请使用以下过程下载和编译 openocd 源代码。

安装 OpenOCD

1. 运行以下命令以安装 libusb-1.0。

   ```
   sudo apt-get install libusb-1.0
   ```

2. 从 http://openocd.org/ 下载 openocd 0.9.0 源代码。
3. 提取 openocd 并导航到解压缩目录。
4. 使用以下命令配置 openocd。

   ```
   ./configure --enable-ftdi --enable-jlink
   ```

5. 运行 make 实用工具来编译 opencd。

   ```
   make install
   ```

设置 Eclipse

> **Note**
>
> 本节假设您已完成避免使用 sudo中的步骤。

Eclipse 是适用于应用程序开发和调试的首选 IDE。它提供了一个丰富且对用户友好的 IDE，具有集成的调试支持，还包括线程感知调试功能。本节介绍所有支持的开发主机的常见 Eclipse 设置。

设置 Eclipse

1. 下载并安装 Java 运行时环境 (JRE)。

   Eclipse 要求安装 JRE。尽管可以在安装 Eclipse 之后进行安装，但我们建议您先安装 JRE。JRE 版本（32/64 位）必须与 Eclipse 版本（32/64 位）匹配。您可以从 Oracle 网站上的 Java SE 运行时环境 8 下载中下载 JRE。
2. 从 http://www.eclipse.org 下载并安装“适用于 C/C++ 开发人员的 Eclipse IDE”。支持 Eclipse 版本 4.9.0 及更高版本。只需解压缩下载的存档即可安装。您可以运行特定于平台的 Eclipse 可执行文件来启动应用程序。

构建并运行 FreeRTOS 演示项目

运行 FreeRTOS 演示项目的方法有两种：

- 使用命令行。
- 使用 Eclipse IDE。

本主题涵盖了这两个选项。

预置

- 根据您使用的是测试应用程序还是演示应用程序，请在以下文件之一中设置预配数据：
  - `./tests/common/include/aws_clientcredential.h`
  - `./demos/common/include/aws_clientcredential.h`

  例如：

```
#define clientcredentialWIFI_SSID "Wi-Fi SSID"
#define clientcredentialWIFI_PASSWORD "Wi-Fi password"
#define clientcredentialWIFI_SECURITY "Wi-Fi security"
```

> **Note**
>
> 您可以输入以下 Wi-Fi 安全值：
>
> - eWiFiSecurityOpen
> - eWiFiSecurityWEP
> - eWiFiSecurityWPA
> - eWiFiSecurityWPA2
>
> SSID 和密码应包含在双引号内。

使用命令行构建并运行 FreeRTOS 演示

1. 使用以下命令开始构建演示应用程序。

```
cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S .  -Bbuild -
DAFR_ENABLE_TESTS=0
```

确保您获得相同的输出，如以下示例所示。

```
marvell@pe-lt586:amzsdk$
marvell@pe-lt586:amzsdk$ cmake -DVENDOR=marvell -DBOARD=mw300_rd -DCOMPILER=arm-gcc -S . -
Bbuild -DAFR_ENABLE_TESTS=0

====================Configuration for Amazon FreeRTOS====================
  Version:                 v1.2.4
  Git version:             AMZSDK_V1.2.r6.p1-12-gdd17d10

Target microcontroller:
  vendor:                  Marvell
  board:                   mw300_rd
  description:             Marvell Board for AmazonFreeRTOS
  family:                  Wireless Microcontroller
  data ram size:           512KB
  program memory size:     2MB

Host platform:
  OS:                      Linux-4.15.0-47-generic
  Toolchain:               arm-gcc
  Toolchain path:          /home/marvell/Software/gcc-arm-none-eabi-4_9-2015q3
  CMake generator:         Unix Makefiles

Amazon FreeRTOS modules:
  Modules to build:        kernel, freertos_plus_tcp, bufferpool, crypto, greengrass, mqtt
, ota, pkcs11, secure_sockets, shadow, tls, wifi
  Disabled by user:
  Disabled by dependency:  posix

  Available demos:         demo_key_provisioning, demo_logging, demo_mqtt_hello_world, dem
o_mqtt_pubsub, demo_tcp, demo_shadow, demo_greengrass, demo_ota
  Available tests:         test_crypto, test_greengrass, test_mqtt, test_ota, test_pkcs11,
 test_secure_sockets, test_tls, test_shadow, test_wifi, test_memory
=========================================================================

-- Configuring done
-- Generating done
-- Build files have been written to: /home/marvell/gerrit/amzsdk/build
marvell@pe-lt586:amzsdk$
```

2. 导航到构建目录。

```
cd build
```

3. 运行 make 实用工具来构建应用程序。

```
make all -j4
```

确保您获得相同的输出，如下图所示：

```
[ 92%] Building C object CMakeFiles/afr_ota.dir/lib/third_party/tinycbor/cborencoder.c.obj
[ 93%] Building C object CMakeFiles/afr_ota.dir/lib/third_party/tinycbor/cborencoder_close
_container_checked.c.obj
[ 93%] Building C object CMakeFiles/afr_ota.dir/lib/third_party/tinycbor/cborerrorstrings.
c.obj
[ 93%] Building C object CMakeFiles/afr_ota.dir/lib/third_party/tinycbor/cborparser.c.obj
[ 94%] Building C object CMakeFiles/afr_ota.dir/lib/third_party/tinycbor/cborparser_dup_st
ring.c.obj
[ 94%] Building C object CMakeFiles/afr_ota.dir/lib/third_party/tinycbor/cborpretty.c.obj
[ 94%] Building C object CMakeFiles/afr_ota.dir/lib/third_party/jsmn/jsmn.c.obj
[ 95%] Building C object CMakeFiles/afr_ota.dir/demos/common/logging/aws_logging_task_dyna
mic_buffers.c.obj
[ 95%] Building C object CMakeFiles/afr_ota.dir/demos/common/demo_runner/aws_demo_runner.c
.obj
[ 95%] Linking C static library afr_ota.a
[ 95%] Built target afr_ota
[ 96%] Linking C executable aws_demos.axf
[100%] Built target aws_demos
marvell@pe-lt586:build$
```

4. 使用以下命令来构建测试应用程序。

```
cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S .  -Bbuild -
DAFR_ENABLE_TESTS=1
cd build
make all -j4
```

每次在 aws_demos project 和 aws_tests project 之间切换时都运行 cmake 命令。

5. 将固件映像写入开发主板的闪存。固件将在开发主板重置后执行。将映像刷写到微控制器之前，必须构建开发工具包。

   a. 在刷写固件映像之前，请使用常用组件 Layout 和 Boot2 来准备开发主板的闪存。使用以下命令。

   ```
   cd amzsdk_bundle-x.y.z
   ./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py -l ./vendors/
   marvell/WMSDK/mw320/sdk/tools/OpenOCD/mw300/layout.txt --boot2 ./vendors/
   marvell/WMSDK/mw320/boot2/bin/boot2.bin
   ```

   flashprog 命令可启动以下功能：

   - 布局 – 首先指示 flashprog 实用工具将布局写入闪存。布局类似于闪存的分区信息。默认布局位于以下文件中：/lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/OpenOCD/mw300/layout.txt。

- Boot2 – 这是 WMSDK 使用的引导加载程序。`flashprog` 命令还会将引导加载程序写入闪存。这是引导加载程序在完成刷写后加载微控制器固件映像的作业。确保您获得相同的输出，如下图所示。

```
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00007f14 msp: 0x20001000
29088 bytes written at address 0x00100000
downloaded 29088 bytes in 0.245458s (115.728 KiB/s)
verified 29088 bytes in 0.350004s (81.160 KiB/s)
semihosting is enabled

Flashprog version: 2.1.0
Erasing primary flash...done
Writing new flash layout...done
Writing "boot2" @0x0 (primary)...done
semihosting: *** application exited ***
Flashprog Complete
shutdown command invoked


target state: halted
target halted due to breakpoint, current mode: Thread
xPSR: 0x21000000 pc: 0x00100658 msp: 0x0015ffe4, semihosting
```

b. 该固件使用 Wi-Fi 芯片组来实现功能，Wi-Fi 芯片组有自己的固件，该固件也必须存在于闪存中。使用 `flashprog.py` 实用工具刷写 Wi-Fi 固件的方式与刷写 Boot2 引导加载程序和 MCU 固件的方式相同。使用以下命令刷写 Wi-Fi 固件。

```
cd amzsdk_bundle-x.y.z
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py --wififw ./
vendors/marvell/WMSDK/mw320/wifi-firmware/mw30x/mw30x_uapsta_W14.88.36.p135.bin
```

确保命令的输出与下图类似。

```
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00007f14 msp: 0x20001000
29088 bytes written at address 0x00100000
downloaded 29088 bytes in 0.245498s (115.709 KiB/s)
verified 29088 bytes in 0.350229s (81.108 KiB/s)
semihosting is enabled

Flashprog version: 2.1.0
Writing "wififw" @0x12a000 (primary).....done
semihosting: *** application exited ***
Flashprog Complete
shutdown command invoked


target state: halted
target halted due to breakpoint, current mode: Thread
xPSR: 0x21000000 pc: 0x00100658 msp: 0x0015ffe4, semihosting
```

c. 使用以下命令刷写 MCU 固件。

```
cd amzsdk_bundle-x.y.z
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py --mcufw build/
cmake/vendors/marvell/mw300_rd/aws_demos.bin -r
```

d. 重置主板。您应该会看到演示应用程序的日志。

e. 要运行测试应用程序，请刷写位于同一目录下的 aws_tests.bin 二进制文件。

```
cd amzsdk_bundle-x.y.z
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py --mcufw build/
cmake/vendors/marvell/mw300_rd/aws_tests.bin -r
```

您的命令应类似于下图所示。

```
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00007f14 msp: 0x20001000
29088 bytes written at address 0x00100000
downloaded 29088 bytes in 0.245499s (115.708 KiB/s)
verified 29088 bytes in 0.350231s (81.107 KiB/s)
semihosting is enabled

Flashprog version: 2.1.0
Writing "mcufw" @0x6a000 (primary)...done
semihosting: *** application exited ***
Flashprog Complete
shutdown command invoked


target state: halted
target halted due to breakpoint, current mode: Thread
xPSR: 0x21000000 pc: 0x00100658 msp: 0x0015ffe4, semihosting
Resetting board...
Using OpenOCD interface file ftdi.cfg
Open On-Chip Debugger 0.9.0 (2015-07-15-15:28)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
adapter speed: 3000 kHz
adapter_nsrst_delay: 100
Info : auto-selecting first available session transport "jtag". To override use 'transport
 select <transport>'.
jtag_ntrst_delay: 100
cortex_m reset_config sysresetreq
sh_load
Info : clock speed 3000 kHz
Info : JTAG tap: wmcore.cpu tap/device found: 0x4ba00477 (mfg: 0x23b, part: 0xba00, ver: 0
x4)
Info : wmcore.cpu: hardware has 6 breakpoints, 4 watchpoints
Info : JTAG tap: wmcore.cpu tap/device found: 0x4ba00477 (mfg: 0x23b, part: 0xba00, ver: 0
x4)
shutdown command invoked
Resetting board done...
```

6. 刷写固件并重置主板后，演示应用程序应按下图所示启动。

```
Network connection successful.

Wi-Fi Connected to AP. Creating tasks which use network...
2 6293 [Startup Hook] Write certificate...
3 6296 [Startup Hook] Write device private key...
4 6362 [Startup Hook] Creating MQTT Echo Task...
6 11668 [MQTTEcho] MQTT echo connected.to connect to a2wtm15blvjjs8-ats.iot.us-east-2.amazonaws.com
7 11668 [MQTTEcho] MQTT echo test echoing task created.
8 11961 [MQTTEcho] MQTT Echo demo subscribed to freertos/demos/echo
9 12248 [MQTTEcho] Echo successfully published 'Hello World 0'
10 12591 [Echoing] Message returned with ACK: 'Hello World 0 ACK'
11 17633 [MQTTEcho] Echo successfully published 'Hello World 1'
12 17927 [Echoing] Message returned with ACK: 'Hello World 1 ACK'
13 22953 [MQTTEcho] Echo successfully published 'Hello World 2'
14 23276 [Echoing] Message returned with ACK: 'Hello World 2 ACK'
15 28245 [MQTTEcho] Echo successfully published 'Hello World 3'
16 28575 [Echoing] Message returned with ACK: 'Hello World 3 ACK'
17 33542 [MQTTEcho] Echo successfully published 'Hello World 4'
18 33980 [Echoing] Message returned with ACK: 'Hello World 4 ACK'
19 38823 [MQTTEcho] Echo successfully published 'Hello World 5'
20 39279 [Echoing] Message returned with ACK: 'Hello World 5 ACK'
21 44139 [MQTTEcho] Echo successfully published 'Hello World 6'
22 44501 [Echoing] Message returned with ACK: 'Hello World 6 ACK'
23 49516 [MQTTEcho] Echo successfully published 'Hello World 7'
24 50270 [Echoing] Message returned with ACK: 'Hello World 7 ACK'
25 54796 [MQTTEcho] Echo successfully published 'Hello World 8'
26 55129 [Echoing] Message returned with ACK: 'Hello World 8 ACK'
27 60080 [MQTTEcho] Echo successfully published 'Hello World 9'
28 60389 [Echoing] Message returned with ACK: 'Hello World 9 ACK'
29 65378 [MQTTEcho] Echo successfully published 'Hello World 10'
30 65998 [Echoing] Message returned with ACK: 'Hello World 10 ACK'
31 70658 [MQTTEcho] Echo successfully published 'Hello World 11'
32 70964 [Echoing] Message returned with ACK: 'Hello World 11 ACK'
33 75958 [MQTTEcho] MQTT echo demo finished.
34 75958 [MQTTEcho] ----Demo finished----
```

7. （可选）作为测试映像的替代方法，您可以使用 flashprog 实用工具将微控制器映像从主机直接复制到微控制器 RAM 中。该映像不会在闪存中复制，因此，在重新启动微控制器后，它就会丢失。

   将固件映像加载到 SRAM 的操作速度更快，因为它会立即启动执行文件。此方法主要用于迭代开发。

   使用以下命令将固件加载到 SRAM 中。

   ```
   cd amzsdk_bundle-x.y.z
   ./lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/OpenOCD/ramload.py
    build/cmake/vendors/marvell/mw300_rd/aws_demos.axf
   ```

   命令输出如下图所示。

```
Using OpenOCD interface file ftdi.cfg
Open On-Chip Debugger 0.9.0 (2015-07-15-15:28)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
adapter speed: 3000 kHz
adapter_nsrst_delay: 100
Info : auto-selecting first available session transport "jtag". To override use 'transport
 select <transport>'.
jtag_ntrst_delay: 100
cortex_m reset_config sysresetreq
sh_load
Info : clock speed 3000 kHz
Info : JTAG tap: wmcore.cpu tap/device found: 0x4ba00477 (mfg: 0x23b, part: 0xba00, ver: 0
x4)
Info : wmcore.cpu: hardware has 6 breakpoints, 4 watchpoints
Info : JTAG tap: wmcore.cpu tap/device found: 0x4ba00477 (mfg: 0x23b, part: 0xba00, ver: 0
x4)
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00007f14 msp: 0x20001000
75072 bytes written at address 0x00100000
8 bytes written at address 0x00112540
468 bytes written at address 0x20000040
downloaded 75548 bytes in 0.636127s (115.979 KiB/s)
verified 75548 bytes in 0.959023s (76.930 KiB/s)
shutdown command invoked
```

命令执行完成后，您应该会看到演示应用程序的日志。

使用 Eclipse IDE 构建并运行 FreeRTOS 演示

1. 在设置 Eclipse 工作区之前，您必须运行 cmake 命令。

   运行以下命令以使用 aws_demos Eclipse 项目。

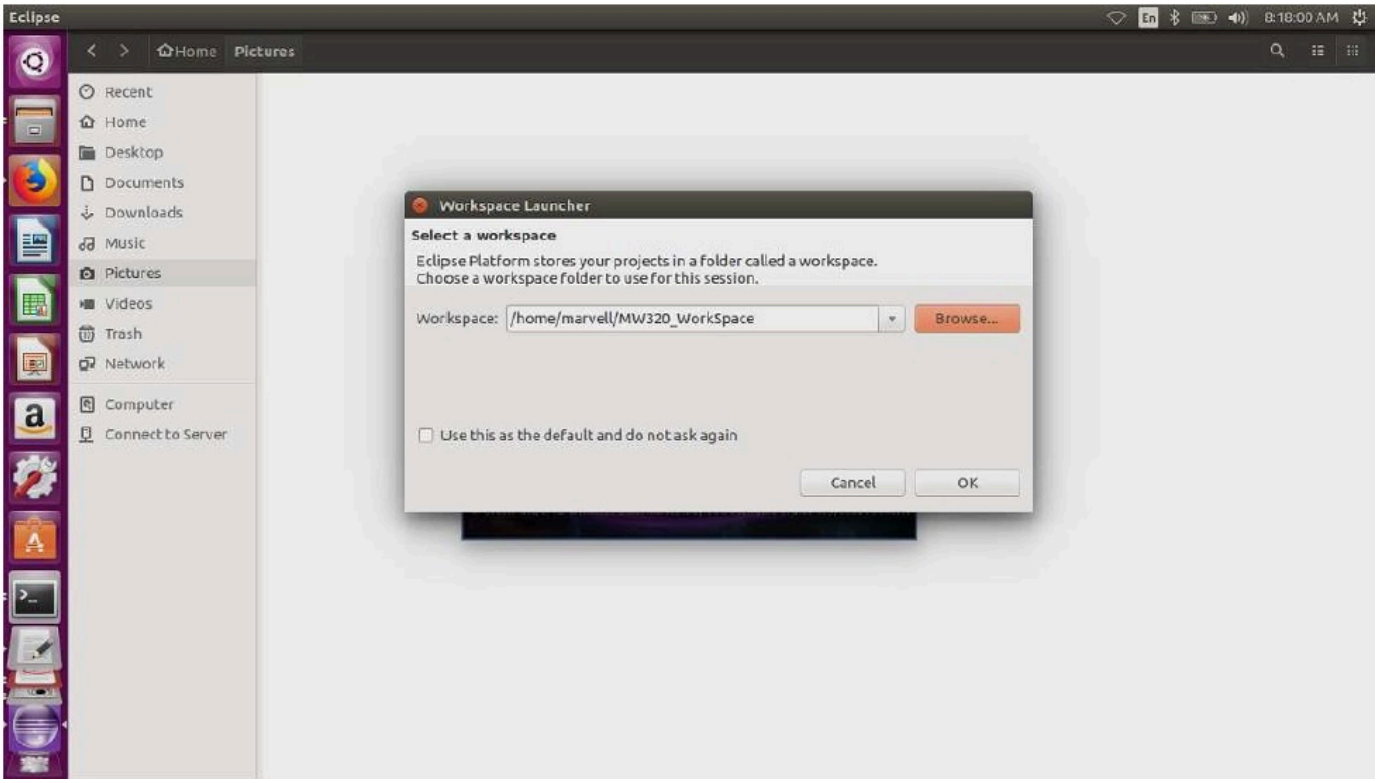
   ```
   cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S .  -Bbuild -
   DAFR_ENABLE_TESTS=0
   ```

   运行以下命令以使用 aws_tests Eclipse 项目。

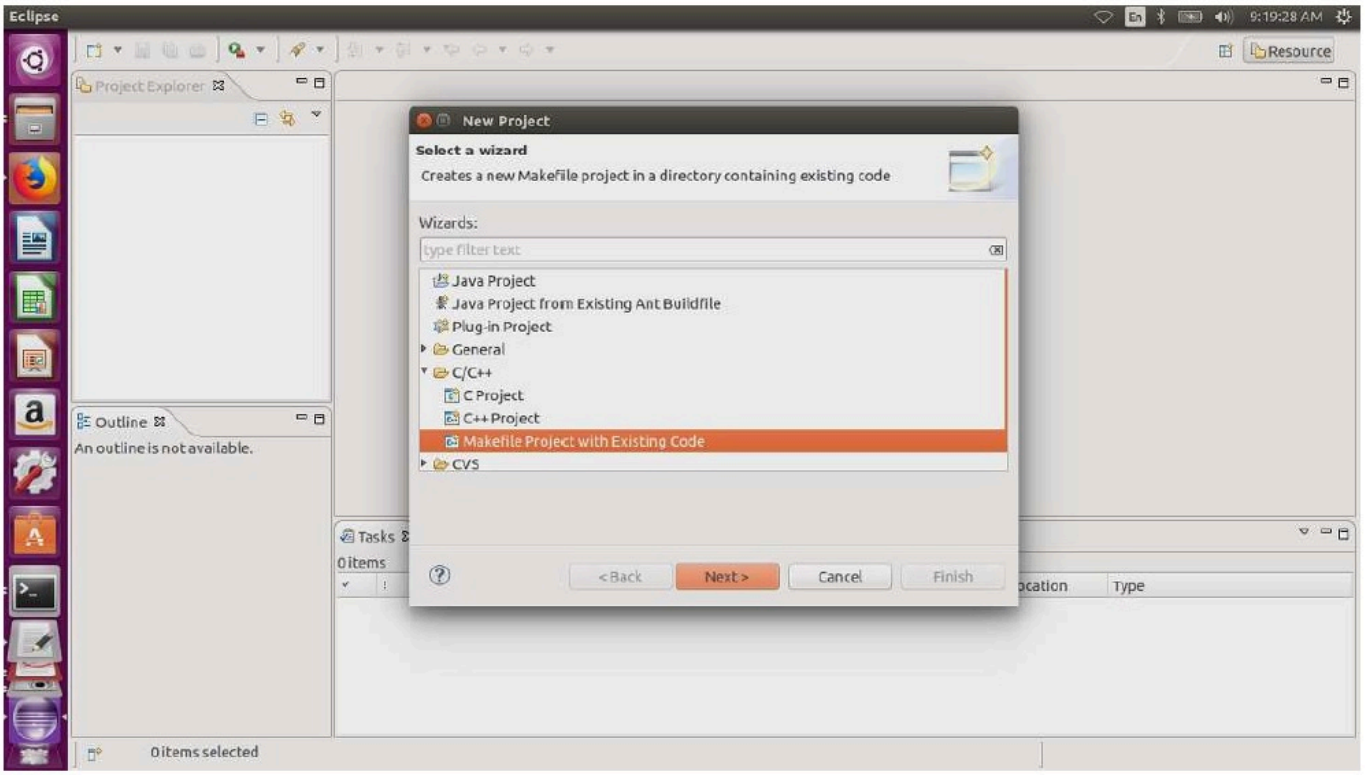
   ```
   cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S .  -Bbuild -
   DAFR_ENABLE_TESTS=1
   ```

> **Tip**
>
> 每次在 aws_demos 项目和 aws_tests 项目之间切换时都运行 cmake 命令。

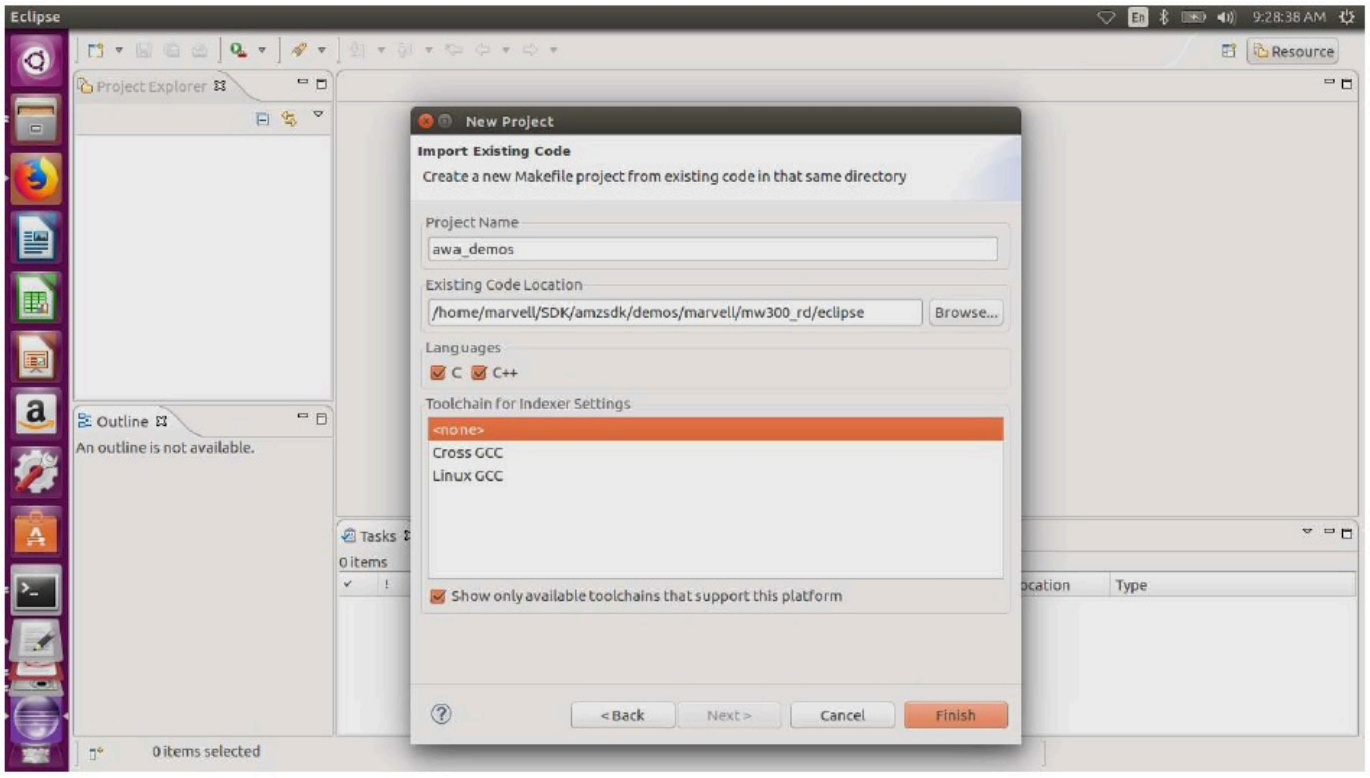
2. 打开 Eclipse，然后在出现提示时选择您的 Eclipse 工作区，如下图所示。



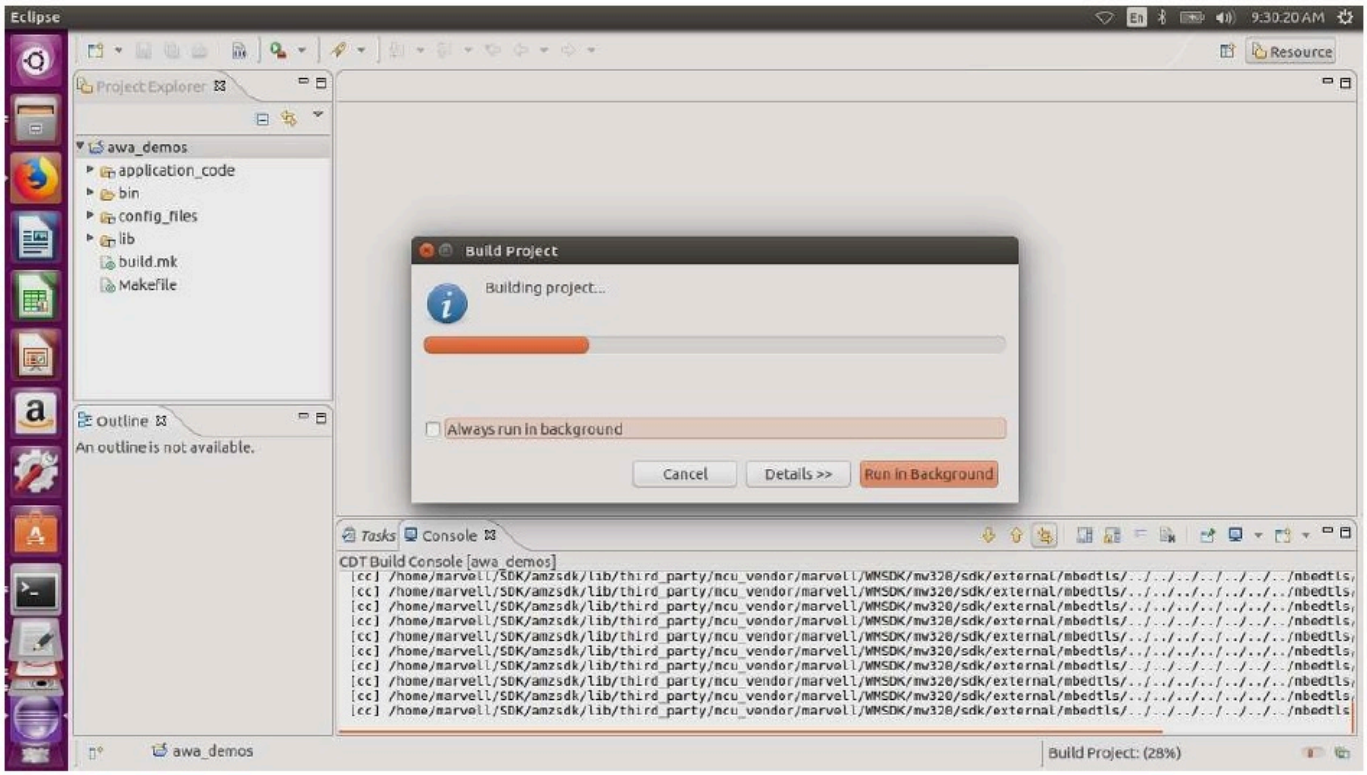
3. 选择创建 Makefile 项目：使用现有代码的选项，如下图所示。



4. 选择浏览，指定现有代码的目录，然后选择完成。



5. 在导航窗格中，在项目资源管理器中选择 aws_demos。右键单击 aws_demos 以打开菜单，然后选择构建。



如果构建成功，则会生成 build/cmake/vendors/marvell/mw300_rd/aws_demos.bin 文件。

6. 使用命令行工具刷写布局文件 (layout.txt)、Boot2 二进制文件 (boot2.bin)、MCU 固件二进制文件 (aws_demos.bin) 和 Wi-Fi 固件。

   a. 在刷写固件映像之前，请使用常用组件 Layout 和 Boot2 来准备开发主板的闪存。使用以下命令。

   ```
   cd amzsdk_bundle-x.y.z
   ./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py -l ./vendors/
   marvell/WMSDK/mw320/sdk/tools/OpenOCD/mw300/layout.txt --boot2 ./vendors/
   marvell/WMSDK/mw320/boot2/bin/boot2.bin
   ```

   flashprog 命令可启动以下功能：

   - 布局 – 首先指示 flashprog 实用工具将布局写入闪存。布局类似于闪存的分区信息。默认布局位于以下文件中：/lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/OpenOCD/mw300/layout.txt。

- Boot2 – 这是 WMSDK 使用的引导加载程序。flashprog 命令还会将引导加载程序写入闪存。这是引导加载程序在完成刷写后加载微控制器固件映像的作业。确保您获得相同的输出，如下图所示。

```
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00007f14 msp: 0x20001000
29088 bytes written at address 0x00100000
downloaded 29088 bytes in 0.245458s (115.728 KiB/s)
verified 29088 bytes in 0.350004s (81.160 KiB/s)
semihosting is enabled

Flashprog version: 2.1.0
Erasing primary flash...done
Writing new flash layout...done
Writing "boot2" @0x0 (primary)...done
semihosting: *** application exited ***
Flashprog Complete
shutdown command invoked


target state: halted
target halted due to breakpoint, current mode: Thread
xPSR: 0x21000000 pc: 0x00100658 msp: 0x0015ffe4, semihosting
```

b. 该固件使用 Wi-Fi 芯片组来实现功能，Wi-Fi 芯片组有自己的固件，该固件也必须存在于闪存中。使用 `flashprog.py` 实用工具刷写 Wi-Fi 固件的方式与刷写 boot2 引导加载程序和 MCU 固件的方式相同。使用以下命令刷写 Wi-Fi 固件。

```
cd amzsdk_bundle-x.y.z
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py --wififw ./
vendors/marvell/WMSDK/mw320/wifi-firmware/mw30x/mw30x_uapsta_W14.88.36.p135.bin
```

确保命令的输出与下图类似。

```
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00007f14 msp: 0x20001000
29088 bytes written at address 0x00100000
downloaded 29088 bytes in 0.245498s (115.709 KiB/s)
verified 29088 bytes in 0.350229s (81.108 KiB/s)
semihosting is enabled

Flashprog version: 2.1.0
Writing "wififw" @0x12a000 (primary).....done
semihosting: *** application exited ***
Flashprog Complete
shutdown command invoked


target state: halted
target halted due to breakpoint, current mode: Thread
xPSR: 0x21000000 pc: 0x00100658 msp: 0x0015ffe4, semihosting
```

c. 使用以下命令刷写 MCU 固件。

```
cd amzsdk_bundle-x.y.z
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py --mcufw build/
cmake/vendors/marvell/mw300_rd/aws_demos.bin -r
```

d. 重置主板。您应该会看到演示应用程序的日志。

e. 要运行测试应用程序，请刷写位于同一目录下的 `aws_tests.bin` 二进制文件。

```
cd amzsdk_bundle-x.y.z
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py --mcufw build/
cmake/vendors/marvell/mw300_rd/aws_tests.bin -r
```

您的命令应类似于下图所示。

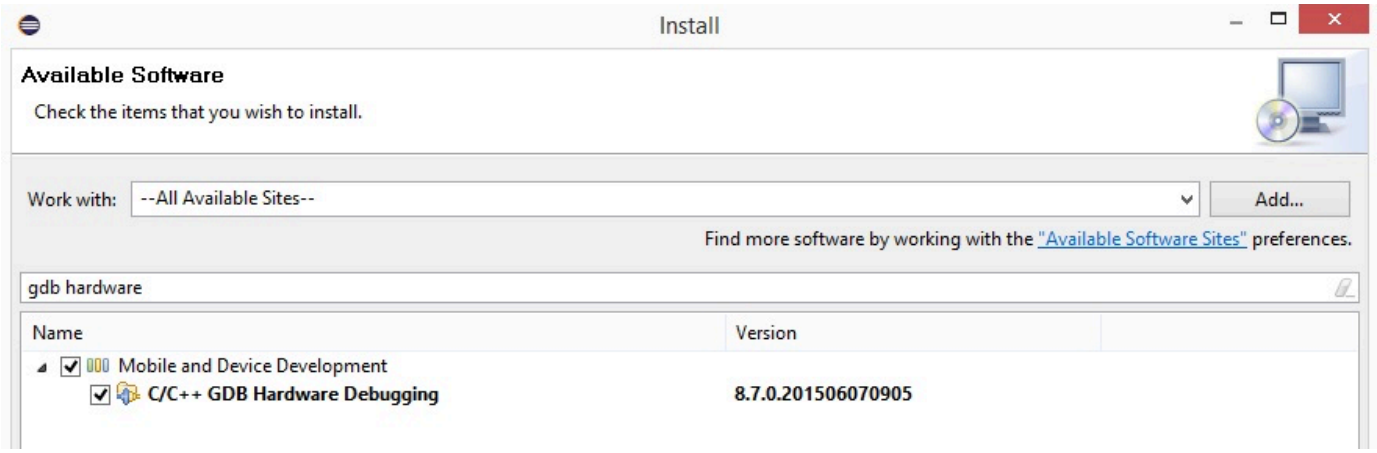
```
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00007f14 msp: 0x20001000
29088 bytes written at address 0x00100000
downloaded 29088 bytes in 0.245499s (115.708 KiB/s)
verified 29088 bytes in 0.350231s (81.107 KiB/s)
semihosting is enabled

Flashprog version: 2.1.0
Writing "mcufw" @0x6a000 (primary)...done
semihosting: *** application exited ***
Flashprog Complete
shutdown command invoked


target state: halted
target halted due to breakpoint, current mode: Thread
xPSR: 0x21000000 pc: 0x00100658 msp: 0x0015ffe4, semihosting
Resetting board...
Using OpenOCD interface file ftdi.cfg
Open On-Chip Debugger 0.9.0 (2015-07-15-15:28)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
adapter speed: 3000 kHz
adapter_nsrst_delay: 100
Info : auto-selecting first available session transport "jtag". To override use 'transport
 select <transport>'.
jtag_ntrst_delay: 100
cortex_m reset_config sysresetreq
sh_load
Info : clock speed 3000 kHz
Info : JTAG tap: wmcore.cpu tap/device found: 0x4ba00477 (mfg: 0x23b, part: 0xba00, ver: 0
x4)
Info : wmcore.cpu: hardware has 6 breakpoints, 4 watchpoints
Info : JTAG tap: wmcore.cpu tap/device found: 0x4ba00477 (mfg: 0x23b, part: 0xba00, ver: 0
x4)
shutdown command invoked
Resetting board done...
```

调试

- 启动 Eclipse 并选择帮助，然后选择安装新软件。在使用菜单中，选择所有可用站点。输入筛选文本 GDB Hardware。选择 C/C++ GDB 硬件调试选项并安装插件。



故障排除

网络问题

检查您的网络凭证。请参阅构建并运行 FreeRTOS 演示项目中的“配置”。

启用其他日志

- 启用特定于主板的日志。

  在 main.c 文件的函数 prvMiscInitialization 中启用对 wmstdio_init(UART0_ID, 0) 的调用。

- 启用 Wi-Fi 日志

  启用 *freertos*/vendors/marvell/WMSDK/mw320/sdk/src/incl/autoconf.h 文件中的宏 CONFIG_WLCMGR_DEBUG。

使用 GDB

我们建议您使用与开发工具包打包在一起的 arm-none-eabi-gdb 和 gdb 命令文件。导航到 目录。

```
cd freertos/lib/third_party/mcu_vendor/marvell/WMSDK/mw320
```

运行以下命令（只有一行）以连接到 GDB。

```
arm-none-eabi-gdb -x ./sdk/tools/OpenOCD/gdbinit ../../../../../../build/cmake/
vendors/marvell/mw300 _rd/aws_demos.axf
```

MediaTek MT7697Hx 开发套件入门

> **Important**
>
> 该参考集成托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

本教程提供了 MediaTek MT7697 Hx 开发套件的入门说明。如果您没有 MediaTek MT7697 Hx 开发套件，请访问 AWS 合作伙伴设备目录，从我们的合作伙伴处购买。

在开始之前，必须进行配置 AWS IoT 并下载 FreeRTOS 才能将设备连接到云端。 AWS 有关说明，请参阅初始步骤：在本教程中，FreeRTOS 下载目录的路径称为 *freertos*。

概览

本教程包含有关以下入门步骤的说明：

1. 在主机上安装软件来开发和调试微控制器主板的嵌入式应用程序。
2. 将 FreeRTOS 演示应用程序交叉编译为二进制映像。
3. 将应用程序二进制映像加载到您的主板上，然后运行该应用程序。
4. 跨串行连接与主板上运行的应用程序进行交互，以便进行监视和调试。

设置开发环境

在设置环境之前，请将计算机连接到 MediaTek MT7697 Hx 开发套件上的 USB 端口。

下载并安装 Keil MDK。

您可以使用基于 GUI 的 Keil 微控制器开发工具包 (MDK)，在主板上配置、构建并运行 FreeRTOS 项目。Keil MDK 包括 µVision IDE 和 µVision Debugger。

> **Note**
>
> 只有 Windows 7、Windows 8 和 Windows 10 64 位计算机上支持 Keil MDK。

下载并安装 Keil MDK

1. 转到 Keil MDK Getting Started (Keil MDK 入门) 页面，然后选择 Download MDK-Core (下载 MDK-Core)。
2. 输入并提交信息以注册到 Keil。
3. 右键单击 MDK 可执行文件并将 Keil MDK 安装程序保存到您的计算机。
4. 打开 Keil MDK 安装程序并按照步骤完成操作。请务必安装 MediaTek 设备包（MT76x7 系列）。

建立串行连接

使用 USB 线缆将主板连接到主机。Windows 设备管理器中显示的 COM 端口。要进行调试，您可以使用终端实用工具（如 HyperTerminal 或）打开与端口的会话 TeraTerm。

在云上监控 MQTT 消息

在运行 FreeRTOS 演示项目之前，您可以在控制台中 AWS IoT 设置 MQTT 客户端，以监控您的设备发送到云端的消息。 AWS

使用 MQTT 客户端订阅 M AWS IoT QTT 主题

1. 登录 AWS IoT 控制台。
2. 在导航窗格中选择测试，然后选择 MQTT 测试客户端，以便打开 MQTT 客户端。
3. 在 Subscription topic (订阅主题)中，输入 ***your-thing-name*****example/topic**，然后选择 Subscribe to topic (订阅主题)。

当演示项目在您的设备上成功运行时，您会多次看到“Hello World！” 发送到您订阅的主题。

使用 Keil MDK 构建并运行 FreeRTOS 演示项目

在 Keil Vision 中构建 FreeRTOS 演示项目

1. 从开始菜单，打开 Keil µVision 5。
2. 打开 `projects/mediatek/mt7697hx-dev-kit/uvision/aws_demos/aws_demos.uvprojx` 项目文件。
3. 从菜单中，选择 Project (项目)，然后选择 Build target (生成目标)。

   生成代码之后，您可以看到位于 `projects/mediatek/mt7697hx-dev-kit/uvision/aws_demos/out/Objects/aws_demo.axf` 的演示可执行文件。

运行 FreeRTOS 演示项目

1. 将 MediaTek MT7697 Hx 开发套件设置为编程模式。

   要将工具包设置为 PROGRAM 模式，请按住 PROG 按钮。按住 PROG 按钮的同时，按下并释放 RESET 按钮，然后释放 PROG 按钮。
2. 从菜单中，选择 Flash (闪存)，然后选择 Configure Flash Tools (配置闪存工具)。
3. 在 Options for Target '**aws_demo**' (目标“aws_demo”的选项) 中，选择 Debug (调试) 选项卡。选择 Use (使用)，将调试器设置为 CMSIS-DAP Debugger (CMSIS-DAP 调试器)，然后选择 OK (确定)。
4. 从菜单中，选择 Flash (闪存)，然后选择 Download (下载)。

   µVision 在下载完成后通知您。
5. 使用终端实用程序打开串行控制台窗口。将串行端口设置为 115200 bps、无奇偶校验、8 位和 1 个停止位。
6. 在 MediaTek MT7697 Hx 开发套件上选择 R ES ET 按钮。

故障排除

在 Keil µVision 中调试 FreeRTOS 项目

目前，必须先编辑 Keil µVision 中包含的 MediaTek 软件包，然后才能使用 Keil µVision 调试 FreeRTOS 演示项目。 MediaTek

编辑 MediaTek 软件包以调试 FreeRTOS 项目

1. 查找并打开位于您 Keil MDK 安装程序文件夹中的 `Keil_v5\ARM\PACK\.Web \MediaTek.MTx.pdsc` 文件。
2. 将出现的所有 `flag = Read32(0x20000000);` 替换为 `flag = Read32(0x0010FBFC);`。
3. 将出现的所有 `Write32(0x20000000, 0x76877697);` 替换为 `Write32(0x0010FBFC, 0x76877697);`。

开始调试项目

1. 从菜单中，选择 Flash (闪存)，然后选择 Configure Flash Tools (配置闪存工具)。
2. 选择 Target (目标) 选项卡，然后选择 Read/Write Memory Areas (读/写内存区域)。确认 IRAM1 并 IRAM2均已选中。

3. 选择 Debug (调试) 选项卡，然后选择 CMSIS-DAP Debugger (调试器)。
4. 打开 `vendors/mediatek/boards/mt7697hx-dev-kit/aws_demos/application_code/main.c`，并将宏 `MTK_DEBUGGER` 设置为 `1`。
5. 在 µVision 中重新构建演示项目。
6. 将 MediaTek MT7697 Hx 开发套件设置为编程模式。

   要将工具包设置为 PROGRAM 模式，请按住 PROG 按钮。按住 PROG 按钮的同时，按下并释放 RESET 按钮，然后释放 PROG 按钮。
7. 从菜单中，选择 Flash (闪存)，然后选择 Download (下载)。

   µVision 在下载完成后通知您。
8. 按下 MediaTek MT7697 Hx 开发套件上的 R ES ET 按钮。
9. 从 µVision 菜单，选择调试，然后选择启动/停止调试会话。在您启动调试会话时将打开 Call Stack + Locals (调用堆栈 + 本地) 窗口。
10. 从菜单中，选择 Debug (调试)，然后选择 Stop (停止) 以暂停代码执行。程序计数器在以下行停止：

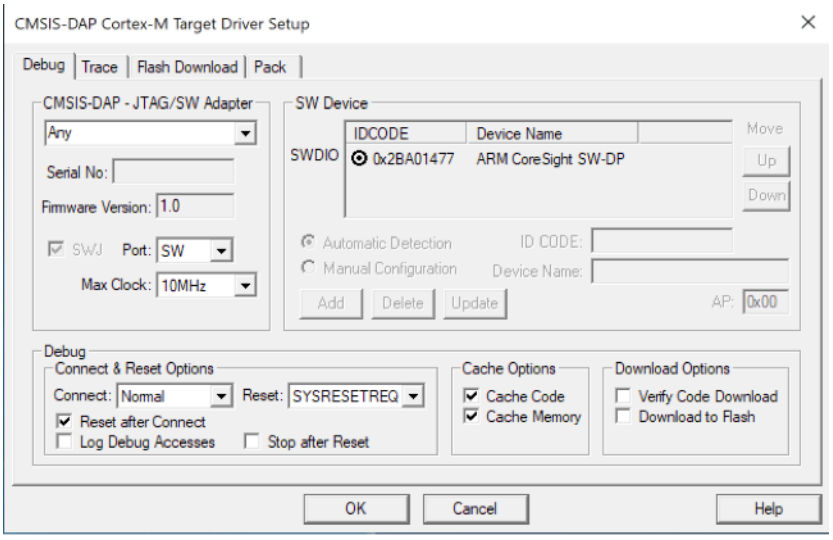
    ```
    { volatile int wait_ice = 1 ; while ( wait_ice ) ; }
    ```
11. 在 Call Stack + Locals (调用堆栈 + 局部变量) 窗口上，将 `wait_ice` 的值更改为 `0`。
12. 在项目源代码中设置断点，然后运行代码。

解决 IDE 调试器设置的问题

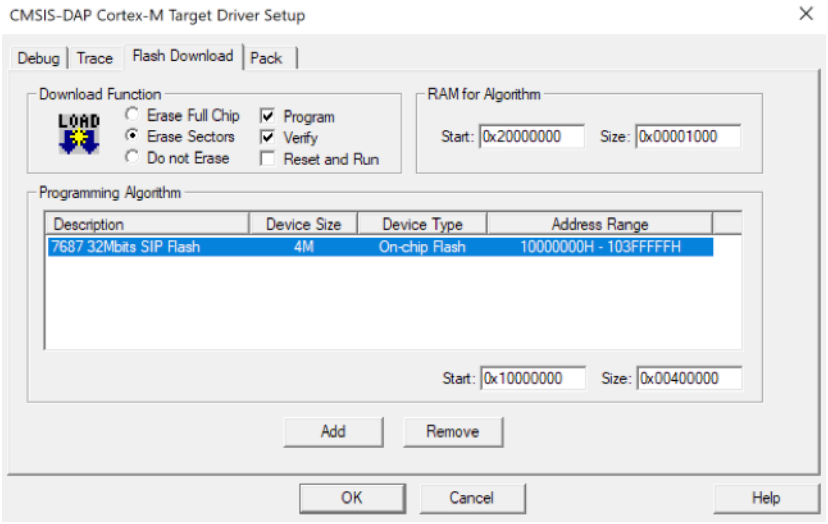
如果您在调试应用程序时遇到问题，则调试器设置可能不正确。

验证您的调试器设置是否正确

1. 打开 Keil µVision。
2. 右键单击 `aws_demos` 项目，选择选项，然后在实用工具选项卡下，选择--使用调试驱动程序--旁边的设置。
3. 验证 Debug (调试) 选项卡下的设置是否如下所示：



4. 验证 Flash Download (闪存下载) 选项卡下的设置是否如下所示：



有关 FreeRTOS 入门的常规故障排除信息，请参阅问题排查入门。

Microchip Curiosity PIC32 MZ EF 入门

> **Important**
>
> 该参考集成托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

> **Note**
>
> 根据Microchip的协议，我们将从FreeRTOS参考集成存储库主分支中删除好奇号 PIC32 MZEF（DM320104），并且将在新版本中不再使用它。Microchip已发布官方通知，不再建议将 PIC32 MZEF（DM320104）用于新设计。 PIC32MZEF 项目和源代码仍然可以通过之前的版本标签进行访问。Microchip 建议客户使用好奇号 PIC32MZ-EF-2.0 开发板 (DM320209) 进行新设计。该 PIC32MZv1 平台仍然可以在 FreeRTOS 参考集成存储库的 v202012. 00 中找到。但是，FreeRTOS 参考的 v202107.00 不再支持该平台。

本教程提供了 Microchip Curiosity PIC32 MZ EF 入门的说明。如果您没有 Microchip Curios PIC32 ity MZ EF 捆绑包，请访问 AWS 合作伙伴设备目录从我们的合作伙伴处购买。

该包包含以下项目：

- 好奇号 PIC32 MZ EF 开发委员会
- MikroElectronika USB UART 点击板
- MikroElectronika WiFi 7 点击 Board
- PIC32 LAN8720 PHY 子板

您还需要以下项目以进行调试：

- MPLAB Snap 电路内置调试器
- （可选）PICkit 3 个编程电缆套件

在开始之前，您必须进行配置 AWS IoT 并下载 FreeRTOS 才能将您的设备连接到云端。 AWS 有关说明，请参阅初始步骤：

> **Important**
>
> - 在本主题中，FreeRTOS 下载目录的路径称为 *`freertos`*。
> - *`freertos`* 路径中的空格字符可能会导致构建失败。克隆或复制存储库时，请确保您创建的路径不包含空格字符。
> - Microsoft Windows 上的文件路径最大长度为 260 个字符。FreeRTOS 下载目录路径过长可能会导致构建操作失败。
> - 由于源代码可能包含符号链接，因此，如果您使用 Windows 提取存档，则可能必须：

- 启用开发者模式，或者，
- 使用已提升为管理员的控制台。

这样，Windows 就可以在提取存档文件时正确创建符号链接。否则，符号链接将作为普通文件写入，该文件会将符号链接的路径作为文本，或者是空文件。有关更多信息，请参阅博客文章 Windows 10 中的符号链接。

如果您在 Windows 下使用 Git，则必须启用开发者模式，或者必须：

- 使用以下命令将 `core.symlinks` 设置为 true：

```
git config --global core.symlinks true
```

- 每当您使用写入系统的 git 命令（例如，git pull、git clone 和 git submodule update --init --recursive）时，请使用具有管理员权限的控制台。

概览

本教程包含有关以下入门步骤的说明：

1. 将主板连接到主机。
2. 在主机上安装软件来开发和调试微控制器主板的嵌入式应用程序。
3. 将 FreeRTOS 演示应用程序交叉编译为二进制映像。
4. 将应用程序二进制映像加载到您的主板上，然后运行该应用程序。
5. 跨串行连接与主板上运行的应用程序进行交互，以便进行监视和调试。

设置 Microchip Curiosity PIC32 MZ EF 硬件

1. 将 MikroElectronika USB UART click Board 连接到 Microchip Curiosity MZ EF 上的 micro PIC32 Bus 1 连接器。
2. 将 PIC32 LAN872 0 PHY 子板连接到 Microchip Curiosity PIC32 MZ EF 上的 J18 接头。
3. 使用 USB A MikroElectronika 转 USB mini-B 电缆将 USB UART click Board 连接到电脑。
4. 要将主板连接到 Internet，请使用以下选项之一：
   - 要使用 Wi-Fi，请将 Wi- MikroElectronika Fi 7 click Board 连接到 Microchip Curiosity MZ EF 上的 MicroBus 2 接口。 PIC32请参阅 配置 FreeRTOS 演示。

- 要使用以太网将 Microchip Curiosity PIC32 MZ EF 板连接到互联网，请将 PIC32 LAN872 0 PHY 子板连接到 Microchip Curiosity MZ EF 上的 J18 接头。 PIC32将以太网电缆的一端连接到 LAN872 0 PHY 子板。将另一端连接到路由器或其他 Internet 端口。还必须定义预处理器宏 `PIC32_USE_ETHERNET`。

5. 如果还没有完成，请将角度连接器焊接到 Microchip Curiosity PIC32 MZ EF 上的 ICSP 接头上。
6. 将 ICSP 电缆的一端从 PICkit 3 编程电缆套件连接到 Microchip Curiosity PIC32 MZ EF。

   如果你没有 PICkit 3 个编程电缆套件，你可以改用 M-F Dupont 跳线来接线。请注意，白色圆圈表示针脚 1 的位置。
7. 将 ICSP 线缆的另一端（或跳线）连接到 MPLAB Snap Debugger。8 针 SIL 编程连接器的针脚 1 由板右下部上的黑色三角形标记。

   确保 Microchip Curiosity PIC32 MZ EF 上引脚 1 的所有电缆（以白色圆圈表示）都与 MPLAB Snap Debugger 上的引脚 1 对齐。

   有关 MPLAB Snap Debugger 的更多信息，请参阅 MPLAB Snap 电路内置调试器信息表。

使用 PICkit On Board (PKOB) 设置 PIC32 Microchip Curiosity MZ EF 硬件

我们建议您按照上一节中的安装过程进行操作。但是，您可以按照以下步骤使用 PICkit 集成的板载 (PKOB) 编程器/调试器来评估和运行带有基本调试功能的 FreeRTOS 演示。

1. 将 MikroElectronika USB UART click Board 连接到 Microchip Curiosity MZ EF 上的 micro PIC32 Bus 1 连接器。
2. 要将主板连接到 Internet，请执行以下操作之一：

   - 要使用 Wi-Fi，请将 Wi- MikroElectronika Fi 7 click Board 连接到 Microchip Curiosity MZ EF 上的 MicroBus 2 接口。 PIC32（按照 配置 FreeRTOS 演示 中的步骤“配置您的 Wi-Fi”操作。
   - 要使用以太网将 Microchip Curiosity PIC32 MZ EF 板连接到互联网，请将 PIC32 LAN872 0 PHY 子板连接到 Microchip Curiosity MZ EF 上的 J18 接头。 PIC32将以太网电缆的一端连接到 LAN872 0 PHY 子板。将另一端连接到路由器或其他 Internet 端口。还必须定义预处理器宏 `PIC32_USE_ETHERNET`。
3. 使用 USB A 型转 USB micro-B 电缆，将 Microchip Curiosity PIC32 MZ EF 板上名为 “USB DEBUG” 的 USB micro-B 端口连接到电脑。
4. 使用 USB A MikroElectronika 转 USB mini-B 电缆将 USB UART click Board 连接到电脑。

设置开发环境

> **Note**
>
> 此设备的 FreeRTOS 项目基于 MPLAB Harmony v2。要构建项目，你需要使用与 Harmony v2 兼容的 MPLAB 工具版本，例如 MPLAB XC32 编译器的 v2.10 和 MPLAB Harmony Configurator (MHC) 的 2.X.X 版本。

1. 安装 Python 版本 3.x 或更高版本。
2. 安装 MPLAB X IDE：

   > **Note**
   >
   > FreeRTO AWS S 参考集成 v202007.00 目前仅在 .35 上支持。 MPLabv5.40 支持以前版本的 F AWS reeRTOS 参考集成。 MPLabv5

   MPLabv5.35 次下载

   - MPLAB X Integrated Development Environment for Windows
   - MPLAB X Integrated Development Environment for macOS
   - MPLAB X Integrated Development Environment for Linux

   最新 MPLab 下载次数 (MPLabv5.40)

   - MPLAB X Integrated Development Environment for Windows
   - MPLAB X Integrated Development Environment for macOS
   - MPLAB X Integrated Development Environment for Linux
3. 安装 MPLAB XC32 编译器：

   - 适用于 Windows 的 MPLAB XC32 /32++ 编译器
   - 适用于 macOS 的 MPLAB XC32 /32++ 编译器
   - 适用于 Linux 的 MPLAB XC32 /32++ 编译器
4. 启动一个 UART 终端仿真器，使用以下设置建立连接：

   - 波特率：115200

- 数据：8 位
- 奇偶校验：无
- 停止位：1
- 流控制：无

在云上监控 MQTT 消息

在运行 FreeRTOS 演示项目之前，您可以在控制台中 AWS IoT 设置 MQTT 客户端，以监控您的设备发送到云端的消息。 AWS

使用 MQTT 客户端订阅 M AWS IoT QTT 主题

1. 登录 AWS IoT 控制台。
2. 在导航窗格中选择测试，然后选择 MQTT 测试客户端，以便打开 MQTT 客户端。
3. 在 Subscription topic (订阅主题)中，输入 ***your-thing-name*/example/topic**，然后选择 Subscribe to topic (订阅主题)。

当演示项目在您的设备上成功运行时，您会多次看到“Hello World！” 发送到您订阅的主题。

构建并运行 FreeRTOS 演示项目

在 MPLAB IDE 中打开 FreeRTOS 演示

1. 打开 MPLAB IDE。如果您要安装多个版本的编译器，则需要选择要在 IDE 中使用的编译器。
2. 从 File (文件) 菜单中，选择 New Project (新建项目)。
3. 浏览到并打开 `projects/microchip/curiosity_pic32mzef/mplab/aws_demos`。
4. 选择 Open project (打开项目)。

> **Note**
>
> 当您首次打开项目时，可能会收到一条有关编译器的错误消息。在 IDE 中，导航到 Tools (工具)、Options (选项)、Embedded (嵌入式)，然后选择您要用于项目的编译器。

要使用以太网进行连接，必须定义预处理器宏 `PIC32_USE_ETHERNET`。

通过 MPLAB IDE 使用以太网进行连接

1. 在 MPLAB IDE 中右键单击项目，然后选择属性。
2. 在 “项目属性” 对话框中，选择 ***compiler-name***（全局选项）将其展开，然后选择 ***compiler-name***-gcc。
3. 对于选项类别，选择预处理和消息，然后将 PIC32_USE_ETHERNET 字符串添加到预处理器宏中。

运行 FreeRTOS 演示项目

1. 重新生成您的项目。
2. 在项目选项卡上，右键单击 aws_demos 顶级文件夹，然后选择调试。
3. 当调试器在 main() 中的断点停止时，在 Run (运行) 菜单中选择 Resume (恢复)。

使用以下方法构建 FreeRTOS 演示 CMake

如果您不想使用 IDE 进行 FreeRTOS 开发，也可以 CMake 使用来生成和运行使用第三方代码编辑器和调试工具开发的演示应用程序或应用程序。

要使用构建 FreeRTOS 演示 CMake

1. 创建一个目录来包含生成的生成文件，例如*build-directory*。
2. 使用以下命令从源代码生成构建文件。

```
cmake -DVENDOR=microchip -DBOARD=curiosity_pic32mzef -DCOMPILER=xc32 -
DMCHP_HEXMATE_PATH=path/microchip/mplabx/version/mplab_platform/bin  -
DAFR_TOOLCHAIN_PATH=path/microchip/xc32/version/bin -S freertos -B build-folder
```

> **Note**
>
> 您必须指定 Hexmate 和工具链二进制文件的正确路径，例如 C:\Program Files (x86)\Microchip\MPLABX\v5.35\mplab_platform\bin 和 C:\Program Files \Microchip\xc32\v2.40\bin 路径。

3. 将目录更改为生成目录 (*build-directory*)，然后make从该目录运行。

有关更多信息，请参阅 CMake 与 FreeRTOS 一起使用。

要使用以太网进行连接，必须定义预处理器宏 PIC32_USE_ETHERNET。

故障排除

有关问题排查信息，请参阅问题排查入门。

Nordic n RF5284 0-DK 入门

> **Important**
>
> 该参考集成托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

本教程提供了 Nordic n RF5284 0-DK 入门的说明。如果您没有 Nordic n RF5284 0-DK，请访问 AWS 合作伙伴设备目录，从我们的合作伙伴处购买。

在开始之前，您需要为 FreeRTOS 低功耗蓝牙设置 AWS IoT 和亚马逊 Cognito。

要运行 FreeRTOS 低功耗蓝牙演示，您还需要具有蓝牙和 Wi-Fi 功能的 iOS 或 Android 移动设备。

> **Note**
>
> 如果您使用的是 iOS 设备，则需要 Xcode 来构建演示移动应用程序。如果您使用的是 Android 设备，则可使用 Android Studio 来构建演示移动应用程序。

概览

本教程包含有关以下入门步骤的说明：

1. 将主板连接到主机。
2. 在主机上安装软件来开发和调试微控制器主板的嵌入式应用程序。
3. 将 FreeRTOS 演示应用程序交叉编译为二进制映像。
4. 将应用程序二进制映像加载到您的主板上，然后运行该应用程序。
5. 跨串行连接与主板上运行的应用程序进行交互，以便进行监视和调试。

设置 Nordic 硬件

将您的主机连接到标有 J2 的 USB 端口，该端口位于 Nordic n RF5284 0 主板上的纽扣电池座正上方。

有关设置 Nordic n RF5284 0-DK 的更多信息，请参阅 n RF5284 0 开发套件用户指南。

设置开发环境

下载并安装 Segger Embedded Studio

FreeRTOS 支持 Segger Embedded Studio 作为 Nordic n 0-DK 的开发环境。RF5284

要设置您的环境，您需要在主机上下载并安装 Segger Embedded Studio。

下载并安装 Segger Embedded Studio

1. 转至 Segger Embedded Studio 下载页面，并选择适合您的操作系统的 Embedded Studio for ARM 选项。
2. 运行安装程序，然后按照提示完成操作。

设置 FreeRTOS 低功耗蓝牙移动开发工具包演示应用程序

要跨低功耗蓝牙运行 FreeRTOS 演示项目，您需要在移动设备上运行 FreeRTOS 低功耗蓝牙移动开发工具包演示应用程序。

设置 FreeRTOS 低功耗蓝牙移动开发工具包演示应用程序

1. 按照 SDKs 适用于 FreeRTOS 蓝牙设备的移动设备 中的说明，在您的主机上下载并安装适用于移动平台的开发工具包。
2. 按照 FreeRTOS 低功耗蓝牙移动开发工具包演示应用程序 中的说明，在您的移动设备上设置演示移动应用程序。

建立串行连接

Segger Embedded Studio 包括一个终端仿真器，可用于通过与主板的串行连接来接收日志消息。

建立与 Segger Embedded Studio 的串行连接

1. 打开 Segger Embedded Studio。
2. 从顶部菜单中，依次选择 Target (目标)、Connect J-Link (连接 J-Link)。

3. 从顶部菜单中，依次选择 Tools (工具)、Terminal Emulator (终端仿真器) 和 Properties (属性)，然后按照安装终端仿真器中所述设置属性。
4. 从顶部菜单中选择 “工具”、“终端仿真器”、“Connect” ***port*** (115200、N ,8,1)。

> **Note**
>
> Segger Embedded Studio 终端模拟器不支持输入功能。为此，请使用像 Pu TTy、Tera Term 或 GNU Screen 这样的终端仿真器。将终端配置为通过串行连接来连接到主板，如安装终端仿真器中所述。

下载并配置 FreeRTOS

设置硬件和环境后，您可以下载 FreeRTOS。

下载 FreeRTOS

要下载适用于 RF5284 Nordic n 0-DK 的 FreeRTOS，请前往 FreeRTOS 页面并克隆存储库 GitHub 。有关说明，请参阅 README.md 文件。

> **Important**
>
> - 在本主题中，FreeRTOS 下载目录的路径称为 *freertos*。
> - *freertos* 路径中的空格字符可能会导致构建失败。克隆或复制存储库时，请确保您创建的路径不包含空格字符。
> - Microsoft Windows 上的文件路径最大长度为 260 个字符。FreeRTOS 下载目录路径过长可能会导致构建操作失败。
> - 由于源代码可能包含符号链接，因此，如果您使用 Windows 提取存档，则可能必须：
>   - 启用开发者模式，或者，
>   - 使用已提升为管理员的控制台。
>
>   这样，Windows 就可以在提取存档文件时正确创建符号链接。否则，符号链接将作为普通文件写入，该文件会将符号链接的路径作为文本，或者是空文件。有关更多信息，请参阅博客文章 Windows 10 中的符号链接。
>
>   如果您在 Windows 下使用 Git，则必须启用开发者模式，或者必须：
>   - 使用以下命令将 `core.symlinks` 设置为 true：

```
git config --global core.symlinks true
```

- 每当您使用写入系统的 git 命令（例如，git pull、git clone 和 git submodule update --init --recursive）时，请使用具有管理员权限的控制台。

配置项目

要启用演示，您需要配置要使用的项目 AWS IoT。要配置要使用的项目 AWS IoT，必须将您的设备注册为 AWS IoT 事物。您在 为 FreeRTOS 低功耗蓝牙设置 AWS IoT 和亚马逊 Cognito 时，应该已对设备进行了注册。

配置您的 AWS IoT 终端节点

1. 登录 AWS IoT 控制台。
2. 在导航窗格中，选择 Settings（设置）。

   您的 AWS IoT 终端节点显示在设备数据端点文本框中。它应该类似于 *1234567890123*-ats.iot.*us-east-1*.amazonaws.com。记下此终端节点。
3. 在导航窗格中，选择管理，然后选择事物。记下设备 AWS IoT 的事物名称。
4. 准备好 AWS IoT 端点和 AWS IoT 事物名称后，*freertos*/demos/include/aws_clientcredential.h在 IDE 中打开，然后为以下#define常量指定值：

   - clientcredentialMQTT_BROKER_ENDPOINT *Your AWS IoT endpoint*
   - clientcredentialIOT_THING_NAME *Your board's AWS IoT thing name*

启用演示

1. 确保已启用了低功耗蓝牙 GATT 演示。在 vendors/nordic/boards/nrf52840-dk/aws_demos/config_files/iot_ble_config.h 中，添加 #define IOT_BLE_ADD_CUSTOM_SERVICES (1) 到列表以定义语句。
2. 打开 vendors/nordic/boards/nrf52840-dk/aws_demos/config_files/aws_demo_config.h，然后定义 CONFIG_OTA_MQTT_BLE_TRANSPORT_DEMO_ENABLED 或 CONFIG_OTA_HTTP_BLE_TRANSPORT_DEMO_ENABLED，如本例中所示。

```
/* To run a particular demo you need to define one of these.
 * Only one demo can be configured at a time
```

```
 *
 *          CONFIG_BLE_GATT_SERVER_DEMO_ENABLED
 *          CONFIG_MQTT_BLE_TRANSPORT_DEMO_ENABLED
 *          CONFIG_SHADOW_BLE_TRANSPORT_DEMO_ENABLED
 *          CONFIG_OTA_MQTT_BLE_TRANSPORT_DEMO_ENABLED
 *          CONFIG_OTA_HTTP_BLE_TRANSPORT_DEMO_ENABLED
 *          CONFIG_POSIX_DEMO_ENABLED
 *
 *  These defines are used in iot_demo_runner.h for demo selection */

#define CONFIG_OTA_MQTT_BLE_TRANSPORT_DEMO_ENABLED
```

3. 由于 Nordic 芯片只带有很少的 RAM (250 KB)，因此可能需要更改 BLE 配置，以允许比每个属性更大的 GATT 表条目。通过这种方式，您可以调整应用程序获取的内存量。为此，请覆盖文件 *freertos*/vendors/nordic/boards/nrf52840-dk/aws_demos/config_files/sdk_config.h 中以下属性的定义：

   - NRF_SDH_BLE_VS_UUID_COUNT

     供应商特 UUIDs定的数量。添加特定于供应商的新 UUID 时，将此计数增加 1。

   - NRF_SDH_BLE_GATTS_ATTR_TAB_SIZE

     属性表大小（以字节为单位）。大小必须是 4 的倍数。该值表示专为属性表设置的内存量（包括特征大小），因此这会因项目而异。如果超过属性表的大小，则会出现 NRF_ERROR_NO_MEM 错误。如果您修改 NRF_SDH_BLE_GATTS_ATTS_ATTS_TAB_SIZE，通常还必须重新配置 RAM 设置。

   （对于测试，文件的位置为 *freertos*/vendors/nordic/boards/nrf52840-dk/aws_tests/config_files/sdk_config.h。）

构建并运行 FreeRTOS 演示项目

在下载 FreeRTOS 并配置演示项目后，可以在主板上构建和运行演示项目。

> **Important**
>
> 如果这是您首次在此主板上运行演示，则需要先将启动加载程序刷写到主板，然后才能运行演示。
> 要构建和刷写启动加载程序，请按照以下步骤操作，但不要使用 projects/nordic/nrf52840-dk/ses/aws_demos/aws_demos.emProject 项目文件，

而是使用 `projects/nordic/nrf52840-dk/ses/aws_demos/bootloader/bootloader.emProject`。

从 Segger Embedded Studio 构建并运行 FreeRTOS 低功耗蓝牙演示

1. 打开 Segger Embedded Studio。从顶部菜单中，选择 File (文件)，再选择 Open Solution (打开解决方案)，然后导航到项目文件 `projects/nordic/nrf52840-dk/ses/aws_demos/aws_demos.emProject`
2. 如果您使用的是 Segger Embedded Studio 终端仿真器，请从顶部菜单中选择 Tools (工具)，然后依次选择 Terminal Emulator (终端仿真器)、Terminal Emulator (终端仿真器) 以显示来自您串行连接的信息。

   如果使用的是其他终端工具，您可以从串行连接监控该工具的输出。
3. 在项目资源管理器中，右键单击 `aws_demos` 演示项目，然后选择构建。

   > **Note**
   >
   > 如果这是您首次使用 Segger Embedded Studio，您可能会看到警告“No license for commercial use (无商业使用许可证)”。可免费将 Segger Embedded Studio 用于 Nordic 半导体设备。申请免费许可证，在设置过程中选择激活您的免费许可证，然后按照说明进行操作。
4. 选择 Debug (调试)，然后选择 Go (开始)。

   演示开始时，它等待通过低功耗蓝牙与移动设备配对。
5. 按照低功耗蓝牙 MQTT 演示应用程序中的说明操作，将 FreeRTOS 低功耗蓝牙移动开发工具包演示应用程序作为移动 MQTT 代理完成演示。

故障排除

有关 FreeRTOS 入门的常规故障排除信息，请参阅问题排查入门。

新 NuMaker-IoT-M腾 487 入门

> **Important**
>
> 该参考集成托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

本教程提供了新唐 NuMaker-IoT-M 487开发板的入门说明。该系列微控制器，包括内置 RJ45 以太网和Wi-Fi模块。如果您没有 Nuvoton NuMaker-IoT-M 487，请访问AWS 合作伙伴设备目录，从我们的合作伙伴处购买。

在开始之前，您必须配置 AWS IoT 和您的 FreeRTOS 软件以将您的开发板连接到云端。 AWS 有关说明，请参阅 初始步骤。在本教程中，FreeRTOS 下载目录的路径称为 *freertos*。

概览

该教程将指导您完成以下步骤：

1. 在您的主机上安装软件，以开发和调试微控制器主板的嵌入式应用程序。
2. 将 FreeRTOS 演示应用程序交叉编译为二进制映像。
3. 将应用程序二进制映像加载到您的主板中，然后运行该应用程序。

设置开发环境

Keil MDK Nuvoton 版本是专为 Nuvoton M487 主板的应用程序开发和调试而设计的。Keil MDK v5 Essential、Plus 或 Pro 版本也适用于 Nuvoton M487（Cortex-M4 核心）MCU。你可以以折扣价下载新腾 Cortex-M4 系列的 Keil MDK Nuvoton 版本。 MCUs仅在 Windows 上支持 Keil MDK。

安装 NuMaker-IoT-M 487 的开发工具

1. 从 Keil MDK 网站中下载 Keil MDK Nuvoton 版本。
2. 使用您的许可证在主机上安装 Keil MDK。Keil MDK 包括 Keil µVision IDE、C/C++ 编译工具链和 µVision 调试器。

   如果您在安装过程中遇到问题，请与 Nuvoton 联系以寻求帮助。
3. 安装 Nu-Link_Keil_Driver_V3.06.7215r（或最新版本），它位于 Nuvoton 开发工具页面上。

构建并运行 FreeRTOS 演示项目

构建 FreeRTOS 演示项目

1. 打开 Keil µVision IDE。
2. 在 File (文件) 菜单上，选择 Open (打开)。在 Open file (打开文件) 对话框中，确保将文件类型选择器设置为 Project Files (项目文件)。
3. 选择要构建的 Wi-Fi 或以太网演示项目。

   - 要打开 Wi-Fi 演示项目，请在 *freertos*\projects\nuvoton\numaker_iot_m487_wifi\uvision\aws_demos 目录中选择目标项目 aws_demos.uvproj。
   - 要打开以太网演示项目，请在 *freertos*\projects\nuvoton\numaker_iot_m487_wifi\uvision\aws_demos_eth 目录中选择目标项目 aws_demos_eth.uvproj。
4. 要确保具有正确的设置以刷写主板，请在 IDE 中右键单击 aws_demo 项目，然后选择 Options (选项)。（有关更多详细信息，请参阅故障排除。）
5. 在 Utilities (实用程序) 选项卡上，确认已选择 Use Target Driver for Flash Programming (使用目标驱动程序进行刷写编程)，并将 Nuvoton Nu-Link Debugger (Nuvoton Nu-Link 调试器) 设置为目标驱动程序。
6. 在 Debug (调试) 选项卡上，选择 Nuvoton Nu-Link Debugger (Nuvoton Nu-Link 调试器) 旁边的 Settings (设置)。
7. 确认 Chip Type (芯片类型) 设置为 M480。
8. 在 Keil µVision IDE Project (项目) 导航窗格中，选择 aws_demos 项目。在 Project (项目) 菜单上，选择 Build Target (构建目标)。

您可以在 AWS IoT 控制台中使用 MQTT 客户端来监控您的设备发送到 AWS 云端的消息。

使用 MQTT 客户端订阅 M AWS IoT QTT 主题

1. 登录 AWS IoT 控制台。
2. 在导航窗格中选择测试，然后选择 MQTT 测试客户端，以便打开 MQTT 客户端。
3. 在 Subscription topic (订阅主题)中，输入 ***your-thing-name*/example/topic**，然后选择 Subscribe to topic (订阅主题)。

运行 FreeRTOS 演示项目

1. 将 Numaker-IoT-M 487 主板连接到主机（计算机）。

2. 重新构建项目。
3. 在 Keil µVision IDE 中，在 Flash (刷写) 菜单上选择 Download (下载)。
4. 在 Debug (调试) 菜单上，选择 Start/Stop Debug Session (启动/停止调试会话)。
5. 当调试器在 `main()` 中的断点处停止时，打开 Run (运行) 菜单，然后选择 Run (F5) (运行 (F5))。

   您应该在控制台的 MQTT 客户端中看到您的设备发送的 MQTT 消息。 AWS IoT

CMake 与 FreeRTOS 一起使用

您还可以使用 CMake 构建和运行 FreeRTOS 演示应用程序或使用第三方代码编辑器和调试工具开发的应用程序。

确保你已经安装了 CMake 编译系统。按照 CMake 与 FreeRTOS 一起使用中的说明进行操作，然后执行本节中的步骤。

> **Note**
>
> 确保编译器 (Keil) 位置的路径位于 Path 系统变量中，例如，`C:\Keil_v5\ARM\ARMCC\bin`。

您还可以在 AWS IoT 控制台中使用 MQTT 客户端来监控您的设备发送到 AWS 云端的消息。

使用 MQTT 客户端订阅 M AWS IoT QTT 主题

1. 登录 AWS IoT 控制台。
2. 在导航窗格中选择测试，然后选择 MQTT 测试客户端，以便打开 MQTT 客户端。
3. 在 Subscription topic (订阅主题)中，输入 ***your-thing-name*/example/topic**，然后选择 Subscribe to topic (订阅主题)。

从源文件中生成构建文件并运行演示项目

1. 在您的主机上，打开命令提示符并导航到 *freertos* 文件夹。
2. 创建一个文件夹以包含生成的构建文件。我们将此文件夹称为*BUILD_FOLDER*。
3. 为 Wi-Fi 或以太网演示生成构建文件。

   - 对于 Wi-Fi：

导航到包含 FreeRTOS 演示项目的源文件的目录。然后，运行以下命令来生成构建文件。

```
cmake -DVENDOR=nuvoton -DBOARD=numaker_iot_m487_wifi -DCOMPILER=arm-keil -S . -
B BUILD_FOLDER -G Ninja
```

- 对于以太网：

  导航到包含 FreeRTOS 演示项目的源文件的目录。然后，运行以下命令来生成构建文件。

```
cmake -DVENDOR=nuvoton -DBOARD=numaker_iot_m487_wifi -DCOMPILER=arm-keil -
DAFR_ENABLE_ETH=1 -S . -B BUILD_FOLDER -G Ninja
```

4. 运行以下命令以生成二进制文件，以便刷写到 M487 上。

```
cmake --build BUILD_FOLDER
```

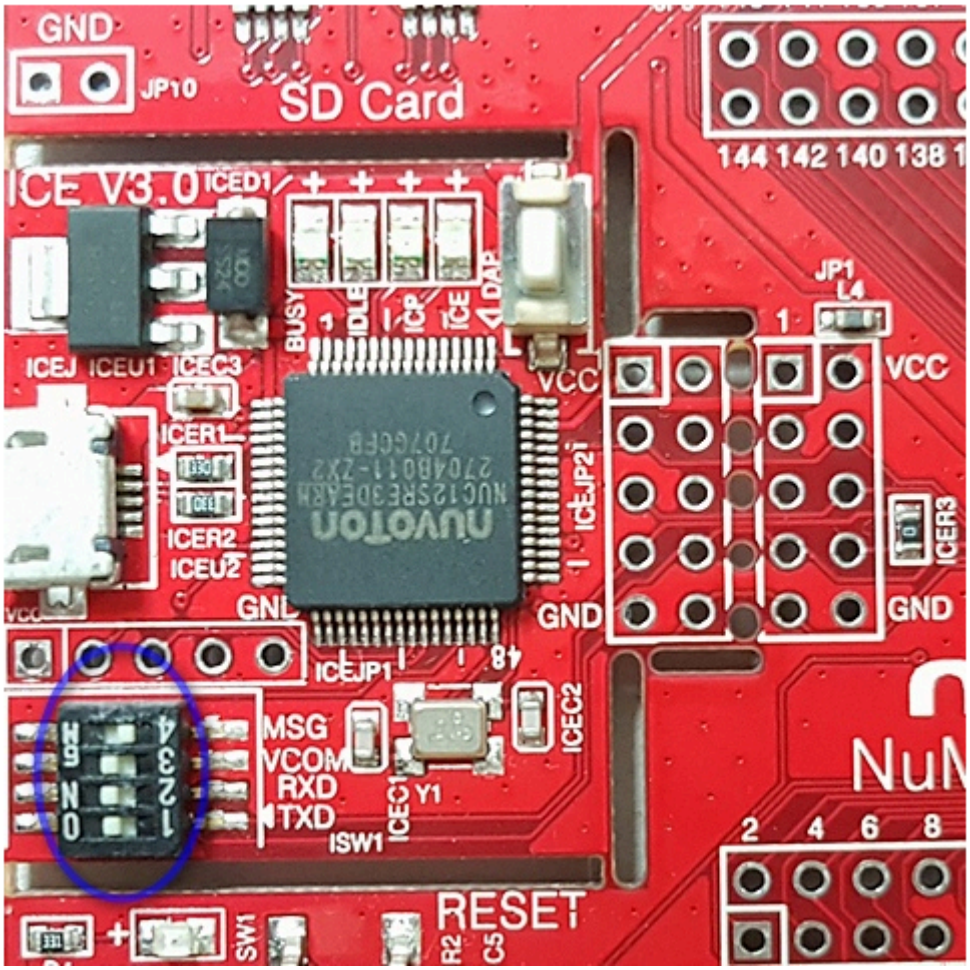
   此时，二进制文件 aws_demos.bin 应位于 *BUILD_FOLDER*/vendors/Nuvoton/boards/numaker_iot_m487_wifi 文件夹中。

5. 要将主板配置为闪烁模式，请确保 MSG 开关（ICE ISW1 上的 4 号）已开启。在插入主板时，将分配一个窗口（和驱动器）。（请参见 故障排除。）
6. 打开终端仿真器以通过 UART 查看消息。按照安装终端仿真器中的说明进行操作。
7. 将生成的二进制文件复制到设备上以运行演示项目。

   如果您通过 MQTT 客户端订阅了 MQT AWS IoT T 主题，则应该会在控制台中看到您的设备发送的 MQTT 消息。 AWS IoT

故障排除

- 如果你的 Windows 无法识别设备VCOM，请通过链接 Nu-Link USB Driver v1.6 安装 NuMaker Windows 串行端口驱动程序。
- 如果你通过 Nu-Link 将设备连接到 Keil MDK (IDE)，请确保 MSG 开关（ICE ISW1 上的 4 号）已关闭，如图所示。



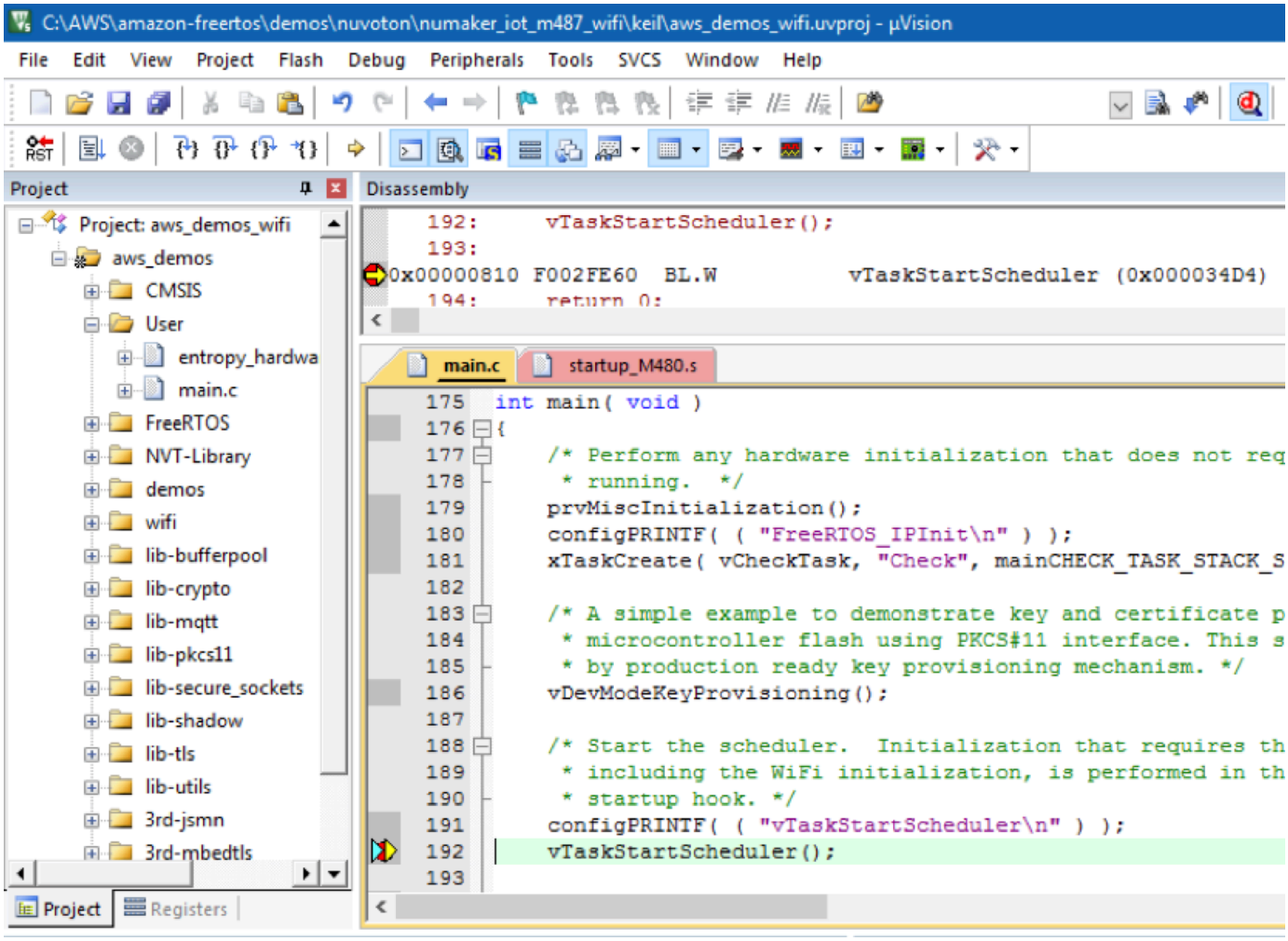
如果您在设置开发环境或连接到主板时遇到问题，请联系 Nuvoton。

在 Keil µVision 中调试 FreeRTOS 项目

在 Keil µVision 中启动调试会话

1. 打开 Keil µVision。
2. 按照构建并运行 FreeRTOS 演示项目中的步骤构建 FreeRTOS 演示项目。
3. 在 Debug (调试) 菜单上，选择 Start/Stop Debug Session (启动/停止调试会话)。

   在启动调试会话时，将显示 Call Stack + Locals (调用堆栈 + 局部变量) 窗口。µVision 将演示刷写到主板上，运行演示，然后在 `main()` 函数开始处停止。
4. 在项目的源代码中设置断点，然后运行代码。项目应如下所示。

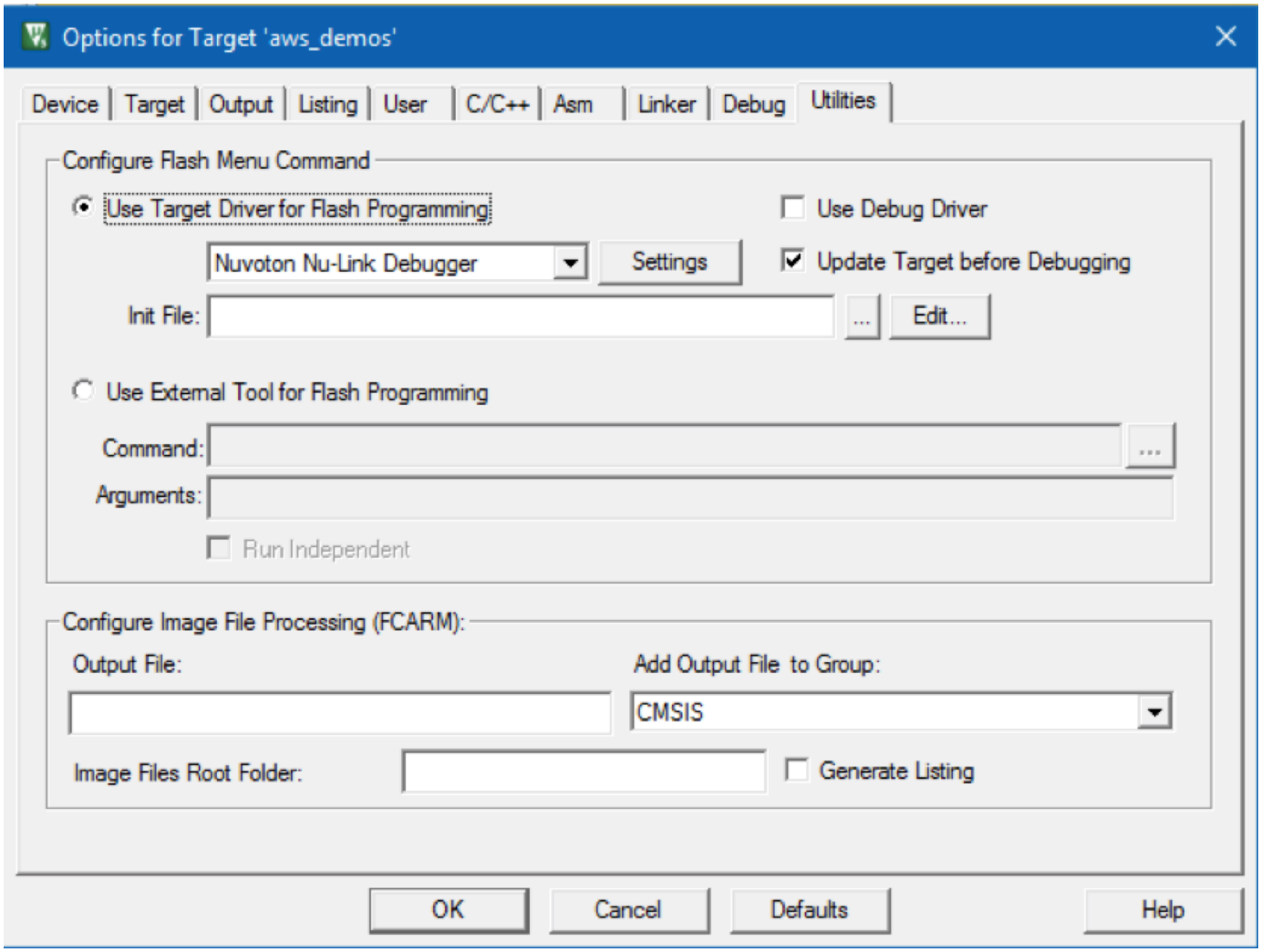


µVision 调试设置故障排除

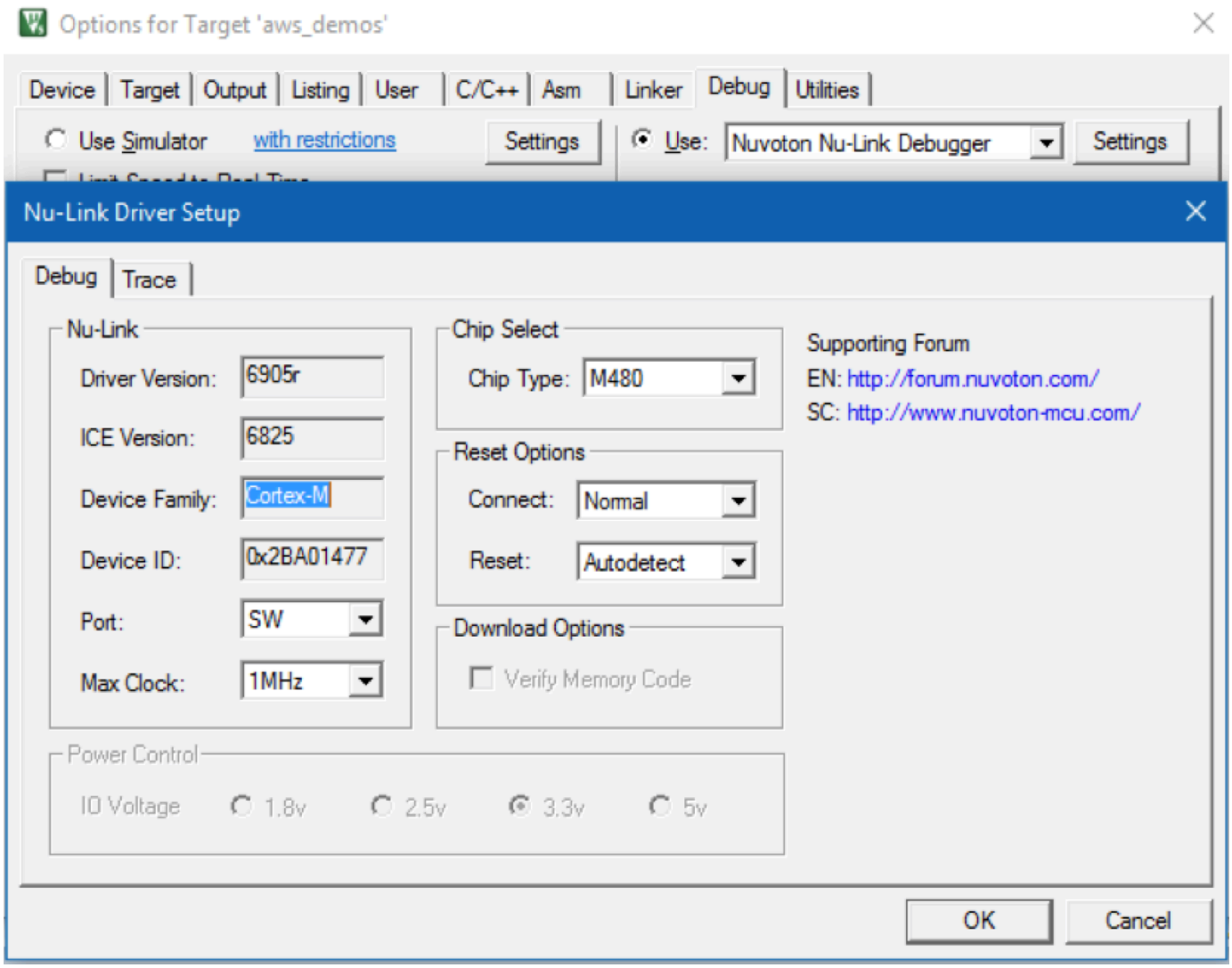
如果在调试应用程序时遇到问题，请检查是否在 Keil µVision 中正确设置了调试设置。

验证 µVision 调试设置是否正确

1. 打开 Keil µVision。
2. 在 IDE 中右键单击 aws_demo 项目，然后选择 Options (选项)。
3. 在 Utilities (实用程序) 选项卡上，确认已选择 Use Target Driver for Flash Programming (使用目标驱动程序进行刷写编程)，并将 Nuvoton Nu-Link Debugger (Nuvoton Nu-Link 调试器) 设置为目标驱动程序。



4. 在 Debug (调试) 选项卡上，选择 Nuvoton Nu-Link Debugger (Nuvoton Nu-Link 调试器) 旁边的 Settings (设置)。



5. 确认 Chip Type (芯片类型) 设置为 M480。

开始使用恩智浦 LPC54 018物联网模块

> **Important**
>
> 该参考集成托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

本教程提供了开始使用恩智浦 LPC54 018物联网模块的说明。如果您没有恩智浦 LPC54 018物联网模块，请访问 AWS 合作伙伴设备目录，从我们的合作伙伴处购买一个。使用USB线将恩智浦 LPC54 018物联网模块连接到电脑。

在开始之前，必须进行配置 AWS IoT 并下载 FreeRTOS 才能将设备连接到云端。 AWS 有关说明，请参阅初始步骤：在本教程中，FreeRTOS 下载目录的路径称为 *freertos*。

概览

本教程包含有关以下入门步骤的说明：

1. 将主板连接到主机。
2. 在主机上安装软件来开发和调试微控制器主板的嵌入式应用程序。
3. 将 FreeRTOS 演示应用程序交叉编译为二进制映像。
4. 将应用程序二进制映像加载到您的主板上，然后运行该应用程序。

设置 NXP 硬件

设置恩智浦018 LPC54

- 将您的电脑连接到恩智浦 LPC54 018的USB端口。

设置 JTAG Debugger

你需要一个JTAG调试器来启动和调试在恩智浦 LPC54 018主板上运行的代码。FreeRTOS 已使用 OM4 0006 物联网模块进行了测试。有关支持的调试器的更多信息，请参阅 0007 OM4018 LPC54 物联网模块产品页面上的恩智浦018物联网模块用户手册。 LPC54

1. 如果您使用的是 OM4 0006物联网模块调试器，请使用转换器电缆将调试器中的20针连接器连接到恩智浦物联网模块上的10针连接器。
2. 使用mini-USB转USB电缆将恩智浦 LPC54 018和000 OM4 6物联网模块调试器连接到计算机上的USB端口。

设置开发环境

FreeRTOS支持 IDEs 两个用于恩智浦 LPC54 018物联网模块：IAR嵌入式工作台和嵌入式工作台。MCUXpresso

在开始之前，请安装其中一个 IDEs。

安装 IAR Embedded Workbench for ARM

1. 浏览适用于 ARM 的 IAR Embedded Workbench并下载该软件。

   > **Note**
   >
   > IAR Embedded Workbench for ARM 需要 Microsoft Windows。

2. 运行安装程序，然后按照提示完成操作。
3. 在 License Wizard (许可证向导) 中，选择 Register with IAR Systems to get an evaluation license (注册 IAR 系统以获取评估许可证）。
4. 在尝试运行任何演示之前，将引导加载程序置于设备上。

MCUXpresso 从恩智浦安装

1. 从恩智浦下载并运行 MCUXpresso 安装程序。

   > **Note**
   >
   > 支持版本 10.3.x 及更高版本。

2. 浏览到 MCUXpresso SDK，然后选择 “构建你的 SDK”。

   > **Note**
   >
   > 支持版本 2.5 及更高版本。

3. 选择 Select Development Board (选择开发主板)。
4. 在 Select Development Board (选择开发主板) 的 Search by Name (按名称搜索) 中，输入 **`LPC54018-IoT-Module`**。
5. 在 “主板” 下，选择 LPC54018- IoT 模块。
6. 验证硬件详细信息，然后选择 B uild MCUXepresso SDK。
7. 使用 MCUXpresso IDE 的适用于 Windows 的 SDK 已经构建完毕。选择 Download SDK。如果您在使用其他操作系统，在 Host OS (主机操作系统) 下，选择您的操作系统，然后选择 Download SDK (下载开发工具包)。
8. 启动 MCUXpresso IDE，然后选择 “已安装 SDKs” 选项卡。
9. 将下载的 SDK 存档文件拖放到 SDKs “已安装” 窗口中。

如果您在安装期间遇到问题，请参阅 NXP 技术支持或 NXP 开发人员资源。

在云上监控 MQTT 消息

在运行 FreeRTOS 演示项目之前，您可以在控制台中 AWS IoT 设置 MQTT 客户端，以监控您的设备发送到云端的消息。 AWS

使用 MQTT 客户端订阅 M AWS IoT QTT 主题

1. 登录 AWS IoT 控制台。
2. 在导航窗格中选择测试，然后选择 MQTT 测试客户端，以便打开 MQTT 客户端。
3. 在 Subscription topic (订阅主题)中，输入 ***your-thing-name*/example/topic**，然后选择 Subscribe to topic (订阅主题)。

当演示项目在您的设备上成功运行时，您会多次看到“Hello World！” 发送到您订阅的主题。

构建并运行 FreeRTOS 演示项目

将 FreeRTOS 演示导入 IDE

将 FreeRTOS 示例代码导入 IAR Embedded Workbench IDE

1. 打开 IAR Embedded Workbench，从 File (文件) 菜单中选择 Open Workspace (打开工作区)。
2. 在 search-directory (搜索目录) 文本框中，输入 `projects/nxp/lpc54018iotmodule/iar/aws_demos`，然后选择 aws_demos.eww。
3. 从 Project (项目) 菜单，选择 Rebuild All (全部重新生成)。

将 FreeRTOS 示例代码导入 IDE MCUXpresso

1. 打开 MCUXpresso，然后从 “文件” 菜单中选择 “从文件系统打开项目”。
2. 在 Directory (目录) 文本框中，输入 `projects/nxp/lpc54018iotmodule/mcuxpresso/aws_demos`，然后选择 Finish (完成)
3. 从 Project (项目) 菜单，选择 Build All (全部生成)。

运行 FreeRTOS 演示项目

使用 IAR Embedded Workbench IDE 运行 FreeRTOS 演示项目

1. 在您的 IDE 中，从 Project (项目) 菜单，选择 Build (生成)。
2. 从 Project (项目) 菜单，选择 Download and Debug (下载并调试)。
3. 从 Debug (调试) 菜单，选择 Start Debugging (启动调试)。
4. 当调试器在 `main` 中的断点停止时，从 Debug (调试) 菜单中选择 Go (执行)。

> **Note**
>
> 如果打开了 J-Link Device Selection (J-Link 设备选择) 对话框，请选择 OK (确定) 以继续。在 Target Device Settings (目标设备设置) 对话框中，依次选择 Unspecified (未指定)、Cortex-M4 和 OK (确定)。这些操作只需要执行一次。

使用 IDE 运行 FreeRTOS 演示项目 MCUxpresso

1. 在您的 IDE 中，从 Project (项目) 菜单，选择 Build (生成)。
2. 如果这是您首次调试，请选择 `aws_demos` 项目，然后从 Debug (调试) 工具栏中，选择蓝色的调试按钮。
3. 此时将显示任何检测到的调试探测器。选择您要使用的探测器，然后选择 OK (确定) 启动调试。

> **Note**
>
> 当调试器在 `main()` 中的断点停止时，按一次调试重启按钮
>
> 
>
> 可重置调试会话。（这是因为 NXP54 018-IOT 模块的 MCUXpresso 调试器存在错误，因此需要这样做）。

4. 当调试器在 `main()` 中的断点停止时，从 Debug (调试) 菜单中选择 Go (执行)。

故障排除

有关 FreeRTOS 入门的常规故障排除信息，请参阅问题排查入门。

开始使用适用于 N-2MB 的瑞萨电子入门套件+ RX65

> **Important**
>
> 该参考集成托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

本教程提供了如何开始使用适用于 N-2MB 的瑞萨电子入门套件+ 的说明。 RX65如果您没有适用于 RX65 N-2MB 的瑞萨 RSK+，请访问 AWS 合作伙伴设备目录，然后从我们的合作伙伴处购买。

在开始之前，必须进行配置 AWS IoT 并下载 FreeRTOS 才能将设备连接到云端。 AWS 有关说明，请参阅初始步骤：在本教程中，FreeRTOS 下载目录的路径称为 *freertos*。

概览

本教程包含有关以下入门步骤的说明：

1. 将主板连接到主机。
2. 在主机上安装软件来开发和调试微控制器主板的嵌入式应用程序。
3. 将 FreeRTOS 演示应用程序交叉编译为二进制映像。
4. 将应用程序二进制映像加载到您的主板上，然后运行该应用程序。

设置 Renesas 硬件

为 N-2MB 设置 RSK+ RX65

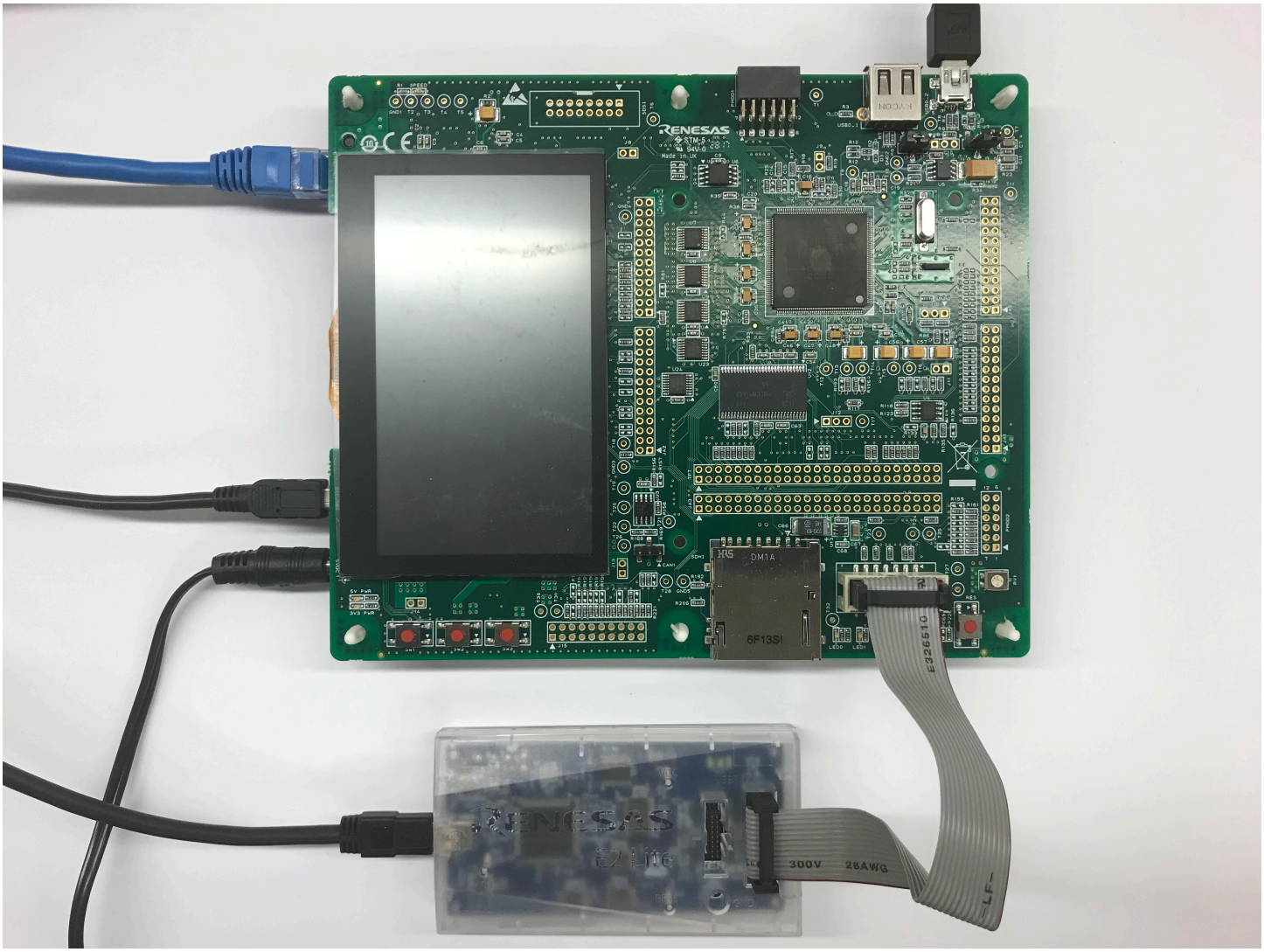
1. 将 +5V 正极电源适配器连接到 N-2MB 的 RSK+ 上的 PWR 连接器。 RX65
2. 将您的计算机连接到 N-2MB 版 RSK+ 上的 USB2 .0 FS 端口。 RX65
3. 将您的计算机连接到适用 RX65于 N-2MB 的 RSK+ 上的 USB-to-serial端口。
4. 将路由器或连接互联网的以太网端口连接到 N-2MB 的 RSK+ 上的以太网端口。 RX65

设置 E2 Lite 调试器模块

1. 使用 14 针带状电缆将 E2 Lite 调试器模块连接到 N-2MB 的 RSK+ 上的 “E1/E2 Lite” 端口。 RX65
2. 使用 USB 电缆将 E2 Lite 调试器模块连接到主机。当 E2 Lite 调试器连接到主板和计算机时，调试器上绿色的“ACT”LED 会闪烁。

3. 在调试器连接到您的主机和 RX65 N-2MB 的 RSK+ 之后，E2 Lite 调试器驱动程序开始安装。

   请注意，安装驱动程序需要管理员权限。



设置开发环境

要为 N-2MB 的 RSK+ 设置 FreeR RX65 TOS 配置，请使用瑞萨 e 2 studio IDE 和 CC-RX 编译器。

> **Note**
>
> 仅 Windows 7、8 和 10 操作系统支持 Renesas e^{2}studio IDE 和 CC-RX 编译器。

下载并安装 e²studio

1. 转到 Renesas e²studio 安装程序下载页面，并下载脱机安装程序。
2. 您将会转到 Renesas 登录页面。

   如果您拥有 Renesas 账户，请输入您的登录凭证，然后选择登录。

   如果您没有账户，请选择 Register now (立即注册)，并按照第一次注册步骤操作。您应该收到一封电子邮件，其中包含用于激活您的 Renesas 账户的链接。按照此链接完成 Renesas 注册，然后登录到 Renesas。
3. 登录后，将 e²studio 安装程序下载到您的计算机。
4. 打开安装程序并按照步骤完成操作。

有关更多信息，请参阅 Renesas 网站上的 e²studio。

下载并安装 RX 系列 C/C++ 编译器包

1. 转到 RX 系列 C/C++ 编译器包下载页面，并下载 V3.00.00 包。
2. 打开可执行文件并安装编译器。

有关更多信息，请参阅 Renesas 网站上的 RX 系列 C/C++ 编译器包。

> **Note**
>
> 编译器仅评估版本可免费使用，有效期为 60 天。在第 61 天，您需要获取许可证密钥。有关更多信息，请参阅评估软件工具。

构建并运行 FreeRTOS 示例。

现在您已配置演示项目，可以在主板上构建和运行项目。

在 e²studio 中构建 FreeRTOS 演示

在 e²studio 中导入和构建演示

1. 从“开始”菜单启动 e²studio。
2. 在 Select a directory as a workspace (选择一个目录作为工作区) 窗口中，浏览到要在其中工作的文件夹，然后选择 Launch (启动)。

3. 当您首次打开 e^{2}studio 时，Toolchain Registry (工具链注册表) 窗口会打开。选择 Renesas Toolchains (Renesas 工具链)，并确认已选择 **CC-RX v3.00.00**。选择 Register (注册)，然后选择 OK (确定)。
4. 如果您是首次打开 e^{2}studio，将显示 Code Generator Registration (代码生成器注册) 窗口。选择确定。
5. 将显示 Code Generator COM component register (代码生成器 COM 组件注册) 窗口。在请重启 e^{2}studio 以使用代码生成器下，选择确定。
6. 此时将显示重启 e^{2}studio窗口。选择确定。
7. e^{2}studio 将重启。在 Select a directory as a workspace (选择一个目录作为工作区) 窗口中，选择 Launch (启动)。
8. 在 e^{2}studio 欢迎屏幕上，选择 Go to the e^{2}studio workbench (转到 e2studio 工作台) 箭头图标。
9. 右键单击 Project Explorer (项目资源管理器) 窗口，然后选择 Import (导入)。
10. 在导入向导中，选择 General (常规)、Existing Projects into Workspace (现有项目到工作区)，然后选择 Next (下一步)。
11. 选择 Browse (浏览)，找到目录 `projects/renesas/rx65n-rsk/e2studio/aws_demos`，然后选择 Finish (完成)。
12. 从项目菜单中，选择项目、全部生成。

    生成控制台将发出未安装许可证管理器的警告消息。您可以忽略此消息，除非您有 CC-RX 编译器的许可证密钥。要安装许可证管理器，请参阅许可证管理器下载页面。

在云上监控 MQTT 消息

在运行 FreeRTOS 演示项目之前，您可以在控制台中 AWS IoT 设置 MQTT 客户端，以监控您的设备发送到云端的消息。 AWS

使用 MQTT 客户端订阅 M AWS IoT QTT 主题

1. 登录 AWS IoT 控制台。
2. 在导航窗格中选择测试，然后选择 MQTT 测试客户端，以便打开 MQTT 客户端。
3. 在 Subscription topic (订阅主题)中，输入 ***your-thing-name*/example/topic**，然后选择 Subscribe to topic (订阅主题)。

当演示项目在您的设备上成功运行时，您会多次看到“Hello World！” 发送到您订阅的主题。

运行 FreeRTOS 项目

在 e²studio 中运行项目

1. 确认你已将 E2 Lite 调试器模块连接到适用于 N-2MB 的 RSK+ RX65
2. 从顶部菜单中，选择 Run (运行)、Debug Configuration (调试配置)。
3. 展开瑞萨电子 GDB 硬件调试，然后选择 aws_demos。 HardwareDebug
4. 选择 Debugger (调试器) 选项卡，然后选择 Connection Settings (连接设置) 选项卡。确认您的连接设置正确。
5. 选择 Debug (调试) 来将代码下载到主板并开始调试。

   系统可能会通过 `e2-server-gdb.exe` 的防火墙警告提示您。选中 Private networks, such as my home or work network (私有网络，如我的家庭或工作网络)，然后选择 Allow access (允许访问)。
6. e²studio 可能会要求更改为 Renesas Debug Perspective (Renesas 调试模式)。请选择是。

   E2 Lite 调试器上绿色的“ACT”LED 会亮起。
7. 在代码下载到主板后，选择 Resume (恢复) 以使代码运行到主函数的第一行。再次选择 Resume (恢复) 以运行其余代码。

有关瑞萨电子发布的最新项目，请参阅上`amazon-freertos`存储库的`renesas-rx`分支。GitHub

故障排除

有关 FreeRTOS 入门的常规故障排除信息，请参阅问题排查入门。

STMicroelectronics STM32L4 探索套件物联网节点入门

> **Important**
>
> 该参考集成托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

本教程提供了 STMicroelectronics STM32 L4 探索套件 IoT 节点入门的说明。如果您还没有 STMicroelectronics STM32 L4 Discovery Kit 物联网节点，请访问 AWS 合作伙伴设备目录从我们的合作伙伴处购买。

确保您已安装了最新的 Wi-Fi 固件。要下载最新的 Wi-Fi 固件，请参阅 STM32L4 Discovery 套件物联网节点、低功耗无线、低功耗蓝牙、NFC、Sub GHz、 Wi-Fi。在 Binary Resources (二进制文件资源) 中，选择 Inventek ISM 43362 Wi-Fi module firmware update (read the readme file for instructions) (Inventek ISM 43362 Wi-Fi 模块固件更新 (有关说明请阅读自述文件))。

在开始之前，您必须配置 AWS IoT FreeRTOS 下载和 Wi-Fi，以便将您的设备连接到云端。 AWS 有关说明，请参阅初始步骤：在本教程中，FreeRTOS 下载目录的路径称为 *freertos*。

概览

本教程包含有关以下入门步骤的说明：

1. 在主机上安装软件来开发和调试微控制器主板的嵌入式应用程序。
2. 将 FreeRTOS 演示应用程序交叉编译为二进制映像。
3. 将应用程序二进制映像加载到您的主板上，然后运行该应用程序。

设置开发环境

安装系统工作台 STM32

1. 浏览到 O STM32pen .org。
2. 在 Open STM32 网页上注册。您需要登录以下载 System Workbench。
3. 浏览到系统工作台以获取 STM32 安装程序，下载并安装 System Workbench。

如果您在安装过程中遇到问题，请参阅 Sy stem FAQs Workbench 网站上的。

构建并运行 FreeRTOS 演示项目

将 FreeRTOS 演示导入系统工作台 STM32

1. 打开 STM32 系统工作台并输入新工作区的名称。
2. 从文件菜单中，选择导入。展开 General (常规)，选择 Existing Projects into Workspace (现有项目到工作区)，然后选择 Next (下一步)。
3. 在 Select Root Directory (选择根目录) 中，输入 `projects/st/stm32l475_discovery/ac6/aws_demos`。
4. 默认情况下应选中项目 `aws_demos`。
5. 选择 “完成” 将项目导入 STM32系统工作台。

6. 从 Project (项目) 菜单，选择 Build All (全部生成)。确认项目成功编译，没有任何错误。

在云上监控 MQTT 消息

在运行 FreeRTOS 演示项目之前，您可以在控制台中 AWS IoT 设置 MQTT 客户端，以监控您的设备发送到云端的消息。 AWS

使用 MQTT 客户端订阅 M AWS IoT QTT 主题

1. 登录 AWS IoT 控制台。
2. 在导航窗格中选择测试，然后选择 MQTT 测试客户端，以便打开 MQTT 客户端。
3. 在 Subscription topic (订阅主题)中，输入 ***your-thing-name*****/example/topic**，然后选择 Subscribe to topic (订阅主题)。

当演示项目在您的设备上成功运行时，您会多次看到“Hello World！” 发送到您订阅的主题。

运行 FreeRTOS 演示项目

1. 使用 USB 电缆将 STMicroelectronics STM32 L4 Discovery Kit 物联网节点连接到计算机。（请查看主板附带的制造商文档，了解要使用的正确 USB 端口。）
2. 在 “项目资源管理器” 中，右键单击`aws_demos`，选择 “调试为”，然后选择 Ac6 STM32 C/C++ 应用程序。

   如果在首次启动调试会话时出现调试错误，请执行以下步骤：

   1. 在 STM32 System Workbench 中，从 “运行” 菜单中选择 “调试配置”。
   2. 选择 aws_demos Debug (aws_demos 调试)。（您可能需要展开 A c6 STM32 调试。）
   3. 选择 Debugger (调试程序) 选项卡。
   4. 在 Configuration Script (配置脚本) 中，选择 Show Generator Options (显示生成器选项)。
   5. 在 Mode Setup (模式设置) 中，将 Reset Mode (重置模式) 设置为 Software System Reset (软件系统重置)。选择 Apply，然后选择 Debug。
3. 当调试器在 `main()` 中的断点停止时，在 Run (运行) 菜单中选择 Resume (恢复)。

CMake 与 FreeRTOS 一起使用

如果您不想使用 IDE 进行 FreeRTOS 开发，也可以 CMake 使用来生成和运行使用第三方代码编辑器和调试工具开发的演示应用程序或应用程序。

首先创建一个包含生成的构建文件的文件夹 (*build-folder*)。

使用以下命令可生成构建文件：

```
cmake -DVENDOR=st -DBOARD=stm32l475_discovery -DCOMPILER=arm-gcc -S freertos -B build-folder
```

如果arm-none-eabi-gcc不在你的 shell 路径中，你还需要设置AFR_TOOLCHAIN_PATH CMake 变量。例如：

```
-D AFR_TOOLCHAIN_PATH=/home/user/opt/gcc-arm-none-eabi/bin
```

有关 CMake 与 FreeRTOS 配合使用的更多信息，请参阅。CMake 与 FreeRTOS 一起使用

故障排除

如果您在演示应用程序的 UART 输出中看到以下内容，您需要更新 Wi-Fi 模块的固件：

```
[Tmr Svc] WiFi firmware version is: xxxxxxxxxxxx
[Tmr Svc] [WARN] WiFi firmware needs to be updated.
```

要下载最新的 Wi-Fi 固件，请参阅 STM32L4 Discovery 套件物联网节点、低功耗无线、低功耗蓝牙、NFC、Sub GHz、 Wi-Fi。在 Binary Resources (二进制文件资源) 下，选择 Inventek ISM 43362 Wi-Fi module firmware update (Inventek ISM 43362 Wi-Fi 模块固件更新) 的下载链接。

有关 FreeRTOS 入门的常规故障排除信息，请参阅问题排查入门。

德州仪器 CC322 0SF-LAUNCHXL 入门

> **Important**
>
> 该参考集成托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

本教程提供了德州仪器 CC322 0SF-LAUNCHXL 入门指南。如果您没有德州仪器 (TI) CC322 0SF-LAUNCHXL 开发套件，请访问 AWS 合作伙伴设备目录从我们的合作伙伴处购买。

在开始之前，您必须进行配置 AWS IoT 并下载 FreeRTOS 才能将您的设备连接到云端。 AWS 有关说明，请参阅初始步骤：在本教程中，FreeRTOS 下载目录的路径称为 *freertos*。

概览

本教程包含有关以下入门步骤的说明：

1. 在主机上安装软件来开发和调试微控制器主板的嵌入式应用程序。
2. 将 FreeRTOS 演示应用程序交叉编译为二进制映像。
3. 将应用程序二进制映像加载到您的主板上，然后运行该应用程序。

设置开发环境

按照以下步骤设置您的开发环境，以便开始使用 FreeRTOS。

请注意，FreeRTOS 支持 IDEs 两个 T CC322 l 0SF-LAUNCHXL 开发套件：Code Composer Studio 和 IAR Embedded Workbench 版本 8.32。您可以使用任一 IDE 来开始。

安装 Code Composer Studio

1. 浏览到 TI Code Composer Studio。
2. 下载适用于您主机平台（Windows、macOS 或 Linux 64 位）的脱机安装程序。
3. 解压缩并运行脱机安装程序。按照提示操作。
4. 要安装产品系列，请选择 SimpleLink Wi-Fi CC32xx 无线 MCUs。
5. 在下一页上，接受调试探测器的默认设置，然后选择 Finish (完成)。

如果您在安装 Code Composer Studio 时遇到问题，请参阅 TI 开发工具支持、C ode Composer Studio FAQs 和 CCS 疑难解答。

安装 IAR Embedded Workbench

1. 下载并运行适用于 IAR Embedded Workbench for ARM 版本 8.32 的 Windows 安装程序。在 Debug probe drivers (调试探测器驱动程序) 中，确保选中了 TI XDS。
2. 完成安装并启动程序。在 License Wizard (许可证向导) 页面上，选择 Register with IAR Systems to get an evaluation license (注册 IAR 系统以获取评估许可证)，或者使用您自己的 IAR 许可证。

安装 SimpleLink CC322 0 软件开发工具包

安装 SimpleLink CC3220 软件开发工具包。 SimpleLink Wi-Fi CC322 0 SDK 包含 CC322 0SF 可编程 MCU 的驱动程序、40 多个示例应用程序以及使用示例所需的文档。

安装 Uniflash

安装 Uniflash。CCS Uniflash 是一款用于在 TI 上对片上闪存进行编程的独立工具。 MCUsUniflash 具有一个 GUI、命令行和脚本编写界面。

安装最新的 Service Pack

1. 在 TI CC322 0SF-LAUNCHXL 上，将 SOP 跳线放在中间的一组引脚（位置 = 1）上，然后重置电路板。
2. 启动 Uniflash。如果您的 CC322 0SF 看 LaunchPad 板出现在 “检测到的设备” 下，请选择 “开始”。如果未检测到您的主板，请从 “新配置” 下的看板列表中选择 CC3220SF-LAUNCHXL，然后选择 “启动图像创建器”。
3. 选择新项目。
4. 在 Start new project (启动新项目) 页面上，输入项目名称。对于 “设备类型”，选择 CC3220SF。对于 Device Mode (设备模式)，选择 Develop (开发)，然后选择 Create Project (创建项目)。
5. 在 Uniflash 应用程序窗口的右侧，选择 Connect (连接)。
6. 从左栏中，选择 Advanced (高级)、Files (文件)，然后选择 Service Pack。
7. 选择 “浏览”，然后导航到您安装了 CC322 0 SimpleLink SF SDK 的位置。Service Pack 位于 `ti/simplelink_cc32xx_sdk_VERSION/tools/cc32xx_tools/servicepack-cc3x20/sp_VERSION.bin`。
8. 选择 Burn (烧入)

   ()

   按钮，然后选择 Program Image (Create & Program) (编程映像(创建并编程)) 来安装 Service Pack。请记住将 SOP 跳线切换回位置 0 并重置主板。

配置 Wi-Fi 预置

要为您的主板配置 Wi-Fi 设置，请执行以下操作之一：

- 按配置 FreeRTOS 演示中所述配置 FreeRTOS 演示应用程序。
- 使用德SmartConfig州仪器。

构建并运行 FreeRTOS 演示项目

在 TI Code Composer 中构建并运行 FreeRTOS 演示项目

将 FreeRTOS 演示导入 TI Code Composer

1. 打开 TI Code Composer，然后选择 OK (确定) 以接受默认工作区名称。
2. 在 Getting Started (入门) 页面上，选择 Import Project (导入项目)。
3. 在 Select search-directory (选择搜索目录) 中，输入 `projects/ti/cc3220_launchpad/ccs/aws_demos`。默认情况下应选中项目 `aws_demos`。要将项目导入 TI Code Composer，请选择 Finish (完成)。
4. 在 Project Explorer (项目资源管理器) 中，双击 aws_demos 使项目处于活动状态。
5. 从项目中，选择生成项目以确保项目成功编译，没有错误或警报。

在 TI Code Composer 中运行 FreeRTOS 演示

1. 确保德州仪器 CC322 0SF-LAUNCHXL 上的 Sense On Power (SOP) 跳线位于位置 0。有关更多信息，请参阅《CC3x3x 网络处理器用户指南》中的 SimpleLink Wi-Fi CC3x2 0。
2. 使用 USB 电缆将德州仪器 CC322 0SF-LAUNCHXL 连接到计算机。
3. 在项目资源管理器中，确保已选择 `CC3220SF.ccxml` 作为活动的目标配置。要使其成为活动状态，请右键单击该文件并选择 Set as active target configuration (设置为活动的目标配置)。
4. 在 TI Code Composer 的 Run (运行) 中，选择 Debug (调试)。
5. 当调试器在 `main()` 中的断点停止时，转到 Run (运行) 菜单，然后选择 Resume (恢复)。

在云上监控 MQTT 消息

在运行 FreeRTOS 演示项目之前，您可以在控制台中 AWS IoT 设置 MQTT 客户端，以监控您的设备发送到云端的消息。 AWS

使用 MQTT 客户端订阅 M AWS IoT QTT 主题

1. 登录 AWS IoT 控制台。
2. 在导航窗格中选择测试，然后选择 MQTT 测试客户端，以便打开 MQTT 客户端。
3. 在 Subscription topic (订阅主题)中，输入 ***your-thing-name*/example/topic**，然后选择 Subscribe to topic (订阅主题)。

当演示项目在您的设备上成功运行时，您会多次看到“Hello World！” 发送到您订阅的主题。

在 IAR Embedded Workbench 中构建并运行 FreeRTOS 演示项目

将 FreeRTOS 演示导入 IAR Embedded Workbench

1. 打开 IAR Embedded Workbench，选择 File (文件)，然后选择 Open Workspace (打开工作区)。
2. 导航到 `projects/ti/cc3220_launchpad/iar/aws_demos`，选择 aws_demos.eww，然后选择 OK (确定)。
3. 右键单击项目名称 (`aws_demos`)，然后选择 Make (生成)。

在 IAR Embedded Workbench 中运行 FreeRTOS 演示

1. 确保德州仪器 CC322 0SF-LAUNCHXL 上的 Sense On Power (SOP) 跳线位于位置 0。有关更多信息，请参阅《CC3x3x 网络处理器用户指南》中的 SimpleLink Wi-Fi CC3x2 0。
2. 使用 USB 电缆将德州仪器 CC322 0SF-LAUNCHXL 连接到计算机。
3. 重新生成您的项目。

   要重新生成项目，请从 Project (项目) 菜单，选择 Make (生成)。
4. 从 Project (项目) 菜单，选择 Download and Debug (下载并调试)。如果显示 “警告：初始化失败” EnergyTrace，则可以将其忽略。有关的更多信息 EnergyTrace，请参阅 MSP EnergyTrace 技术。
5. 当调试器在 `main()` 中的断点停止时，转到 Debug (调试) 菜单，然后选择 Go (执行)。

CMake 与 FreeRTOS 一起使用

如果您不想使用 IDE 进行 FreeRTOS 开发，也可以 CMake 使用来生成和运行使用第三方代码编辑器和调试工具开发的演示应用程序或应用程序。

要使用构建 FreeRTOS 演示 CMake

1. 创建一个包含生成的构建文件的文件夹 (*`build-folder`*)。
2. 确保您的搜索路径（$PATH 环境变量）包含 TI CGT 编译器二进制文件所在的文件夹（例如 `C:\ti\ccs910\ccs\tools\compiler\ti-cgt-arm_18.12.2.LTS\bin`）。

   如果您在 TI 主板上使用 TI ARM 编译器，请使用以下命令从源代码生成构建文件：

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

有关更多信息，请参阅 CMake 与 FreeRTOS 一起使用。

故障排除

如果您在 AWS IoT 控制台的 MQTT 客户端中看不到消息，则可能需要为主板配置调试设置。

配置 TI 主板的调试设置

1. 在 Code Composer 中的 Project Explorer (项目资源管理器) 中，选择 aws_demos。
2. 在 Run (运行) 菜单上，选择 Debug Configurations (调试配置)。
3. 在导航窗格中，选择 aws_demos。
4. 在 Target (目标) 选项卡上，选择 Connection Options (连接选项)，然后选择 Reset the target on a connect (在连接时重置目标)。
5. 选择 Apply，然后选择 Close。

如果这些步骤不起作用，请在串行终端中查看程序的输出。您应看到一些文本，指示问题的根源。

有关 FreeRTOS 入门的常规故障排除信息，请参阅问题排查入门。

Windows 设备模拟器入门

本教程提供有关 FreeRTOS Windows 设备模拟器入门的说明。

在开始之前，您必须进行配置 AWS IoT 并下载 FreeRTOS 才能将您的设备连接到云端。 AWS 有关说明，请参阅初始步骤：在本教程中，FreeRTOS 下载目录的路径称为 *freertos*。

FreeRTOS 以 zip 文件格式发布，包含您所指定平台的 FreeRTOS 库和示例应用程序。要在 Windows 计算机上运行此示例，请下载移植到 Windows 上运行的库和示例。这组文件称为适用于 Windows 的 FreeRTOS 仿真器。

> **Note**
>
> 本教程无法在亚马逊 EC2 Windows 实例上成功运行。

设置开发环境

1. 安装最新版本的 Npcap。在安装过程中选择 “WinPcap API 兼容模式”。
2. 安装 Microsoft Visual Studio。

   Visual Studio 2017 和 2019 版已知可用。支持 Visual Studio 的所有版本（社区版、专业版或企业版）。

   在 IDE 之外，请安装 Desktop development with C++ (C++ 桌面开发)组件。

   安装最新的 Windows 10 开发工具包。您可以在使用 C++ 的桌面开发组件的可选部分下选择此选项。
3. 确保您有活动的有线以太网连接。
4. （可选）如果您想使用 CMake基于的编译系统来构建 FreeRTOS 项目，请安装最新版本的。CMakeFreeRTOS CMake 需要版本 3.13 或更高版本。

在云上监控 MQTT 消息

在运行 FreeRTOS 演示项目之前，您可以在控制台中 AWS IoT 设置 MQTT 客户端，以监控您的设备发送到云端的消息。 AWS

使用 MQTT 客户端订阅 M AWS IoT QTT 主题

1. 登录 AWS IoT 控制台。
2. 在导航窗格中选择测试，然后选择 MQTT 测试客户端，以便打开 MQTT 客户端。
3. 在 Subscription topic (订阅主题)中，输入 ***your-thing-name*****/example/topic**，然后选择 Subscribe to topic (订阅主题)。

当演示项目在您的设备上成功运行时，您会多次看到“Hello World！” 发送到您订阅的主题。

构建并运行 FreeRTOS 演示项目

你可以使用 Visual Studio 或者 CMake 来构建 FreeRTOS 项目。

使用 Visual Studio IDE 构建并运行 FreeRTOS 演示项目

1. 在 Visual Studio 中加载项目。

在 Visual Studio 中，从 File (文件) 菜单，选择 Open (打开)。选择 File/Solution (文件/解决方案)，导航到 `projects/pc/windows/visual_studio/aws_demos/aws_demos.sln` 文件，然后选择 Open (打开)。

2. 重新定位演示项目。

   提供的演示项目取决于 Windows 开发工具包，但未指定 Windows 开发工具包版本。默认情况下，IDE 可能会尝试使用计算机上不存在的开发工具包版本构建演示。要设置 Windows 开发工具包版本，请右键单击 `aws_demos`，然后选择 Retarget Projects (重新定位项目)。这将打开 Review Solution Actions (审核解决方案操作) 窗口。选择计算机上现有的一个 Windows 开发工具包版本（下拉列表中的初始值即可），然后选择确定。

3. 构建并运行项目。

   从生成菜单，选择生成解决方案，确保解决方案已生成且没有错误或警告。选择 Debug (调试)、Start Debugging (开始调试) 以运行项目。在首次运行时，您必须选择一个网络接口。

使用以下命令构建和运行 FreeRTOS 演示项目 CMake

我们建议您使用 CMake GUI 而不是 CMake 命令行工具来构建 Windows 模拟器的演示项目。

安装完成后 CMake，打开 CMake GUI。在 Windows 上，你可以从 “开始” 菜单下CMake的 “开始” 菜单中找到它 CMake （cmake- gui）。

1. 设置 FreeRTOS 源代码目录。

   在 GUI 中，将源代码位于什么位置设置为 FreeRTOS 源代码目录 (*`freertos`*)。

   将在何处构建二进制文件设置为 *`freertos`*`/build`。

2. 配置 CMake 项目。

   在 CMake GUI 中，选择 “添加条目”，然后在 “添加缓存条目” 窗口中设置以下值：

   名称

   AFR_BOARD

   类型

   string

值

pc.windows

描述

（可选）

3. 选择 配置。如果 CMake 提示您创建生成目录，请选择 “是”，然后在 “指定此项目的生成器” 下选择生成器。我们建议使用 Visual Studio 作为生成器，不过也支持 Ninja。（请注意，当使用 Visual Studio 2019 时，平台应设置为 Win32，而不是其默认设置。）保留其他生成器选项不变，然后选择完成。
4. 生成并打开 CMake 项目。

   配置完项目后， CMake GUI 会显示生成的项目的所有可用选项。对于本教程中的使用，您可以保留选项的默认值。

   选择 Generate (生成) 以创建 Visual Studio 解决方案，然后选择 Open Project (打开项目) 以在 Visual Studio 中打开项目。

   在 Visual Studio 中，右键单击该aws_demos项目，然后选择 “设置为 StartUp 项目”。这使您能够构建并运行项目。在首次运行时，您必须选择一个网络接口。

有关 CMake 与 FreeRTOS 配合使用的更多信息，请参阅。 CMake 与 FreeRTOS 一起使用

配置网络接口

首次运行演示项目时，您必须选择要使用的网络接口。程序会计算您的网络接口数。找到您的有线连接以太网接口号。输出应该如下所示：

```
0 0 [None] FreeRTOS_IPInit
1 0 [None] vTaskStartScheduler
1. rpcap://\Device\NPF_{AD01B877-A0C1-4F33-8256-EE1F4480B70D}
(Network adapter 'Intel(R) Ethernet Connection (4) I219-LM' on local host)

2. rpcap://\Device\NPF_{337F7AF9-2520-4667-8EFF-2B575A98B580}
(Network adapter 'Microsoft' on local host)

The interface that will be opened is set by "configNETWORK_INTERFACE_TO_USE", which
should be defined in FreeRTOSConfig.h

ERROR:  configNETWORK_INTERFACE_TO_USE is set to 0, which is an invalid value.
```

```
Please set configNETWORK_INTERFACE_TO_USE to one of the interface numbers listed above,
then re-compile and re-start the application.  Only Ethernet (as opposed to Wi-Fi)
interfaces are supported.
```

确定了有线连接的以太网接口的编号之后，关闭应用程序窗口。在前述示例中，使用的编号是 1。

打开 `FreeRTOSConfig.h` 并将 `configNETWORK_INTERFACE_TO_USE` 设置为与有线连接网络接口对应的编号。

> **Important**
>
> 仅支持以太网接口。不支持 Wi-Fi。

故障排除

Windows 常见问题故障排查

尝试使用 Visual Studio 构建演示项目时，您可能会遇到以下错误：

```
Error "The Windows SDK version X.Y was not found" when building the provided Visual
 Studio solution.
```

项目必须定位到计算机上存在的 Windows 开发工具包版本。

有关 FreeRTOS 入门的常规故障排除信息，请参阅问题排查入门。

Xilinx Avnet MicroZed 工业物联网套件入门

> **Important**
>
> 该参考集成托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

本教程提供了 Xilinx Avnet MicroZed 工业物联网套件的入门说明。如果您没有 Xilinx Avnet MicroZed 工业物联网套件，请访问 AWS 合作伙伴设备目录，从我们的合作伙伴处购买。

在开始之前，必须进行配置 AWS IoT 并下载 FreeRTOS 才能将设备连接到云端。 AWS 有关说明，请参阅初始步骤：在本教程中，FreeRTOS 下载目录的路径称为 *`freertos`*。

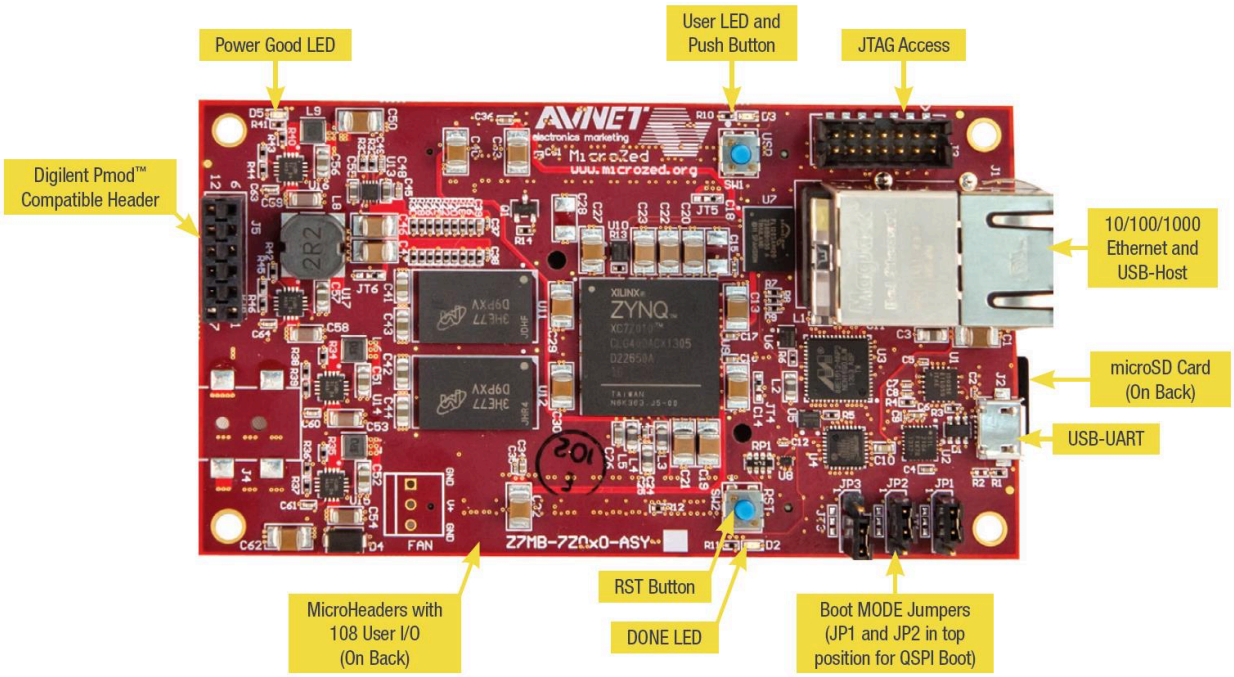
概览

本教程包含有关以下入门步骤的说明：

1. 将主板连接到主机。
2. 在主机上安装软件来开发和调试微控制器主板的嵌入式应用程序。
3. 将 FreeRTOS 演示应用程序交叉编译为二进制映像。
4. 将应用程序二进制映像加载到您的主板上，然后运行该应用程序。

设置 MicroZed 硬件

在设置 MicroZed 硬件时，下图可能会有所帮助：



设置看 MicroZed 板

1. 将电脑连接到主板上的 USB-UART 端口。 MicroZed
2. 将电脑连接到主 MicroZed 板上的 JTAG 接入端口。
3. 将路由器或联网的以太网端口连接到主板上的以太网和 USB 主机端口。 MicroZed

设置开发环境

要为套件设置 FreeRTOS 配置，必须使用 Xilinx 软件开发套件 (XSDK)。 MicroZed XSDK 在 Windows 和 Linux 上受支持。

下载并安装 XSDK

要安装 Xilinx 软件，您需要一个免费的 Xilinx 账户。

下载 XSDK

1. 转到软件开发套件独立 WebInstall 客户端下载页面。
2. 选择适合您的操作系统的选项。
3. 您将定向到 Xilinx 登录页。

   如果您拥有 Xilinx 账户，请输入您的登录凭证，然后选择登录。

   如果您没有账户，请选择 Create your account (创建账户)。注册后，您将收到一封电子邮件，其中包含用于激活您的 Xilinx 账户的链接。
4. 在 Name and Address Verification (名称和地址验证) 页面上，输入您的信息，然后选择 Next (下一步)。下载应可以开始了。
5. 保存 `Xilinx_SDK_version_os` 文件。

安装 XSDK

1. 打开 `Xilinx_SDK_version_os`文件。
2. 在 Select Edition to Install (选择要安装的版本) 中，选择 Xilinx Software Development Kit (XSDK) (Xilinx 开发工具包 (XSDK))，然后选择 Next (下一步)。
3. 在安装向导的以下页面上，在 Installation Options (安装选项) 下，选择 Install Cable Drivers (安装电缆驱动程序)，然后选择 Next (下一步)。

如果您的计算机未检测到 USB-UART 连接，请手动安装 CP21 0x B USB-to-UART ridge VCP 驱动程序。 MicroZed有关说明，请参阅 Silicon Labs CP21 0x USB-to-UART 安装指南。

有关 XSDK 的更多信息，请参阅 Xilink 网站上的 Xilinx 开发工具包入门。

在云上监控 MQTT 消息

在运行 FreeRTOS 演示项目之前，您可以在控制台中 AWS IoT 设置 MQTT 客户端，以监控您的设备发送到云端的消息。 AWS

使用 MQTT 客户端订阅 M AWS IoT QTT 主题

1. 登录 AWS IoT 控制台。

2. 在导航窗格中选择测试，然后选择 MQTT 测试客户端，以便打开 MQTT 客户端。
3. 在 Subscription topic (订阅主题)中，输入 ***your-thing-name*****/example/topic**，然后选择 Subscribe to topic (订阅主题)。

构建并运行 FreeRTOS 演示项目

在 XSDK IDE 中打开 FreeRTOS 演示

1. 启动 XSDK IDE 并将工作区目录设置为 *freertos*/projects/xilinx/microzed/xsdk。
2. 关闭欢迎页面。从菜单中，选择 Project (项目)，然后清除 Build Automatically (自动构建)。
3. 从菜单中，选择 File (文件)，然后选择 Import (导入)。
4. 在 Select (选择) 页面上，展开 General (常规)，选择 Existing Projects into Workspace (现有项目到工作区)，然后选择 Next (下一步)。
5. 在 Import Projects (导入项目) 页面上，选择 Select root directory (选择根目录)，然后输入演示项目的根目录：*freertos*/projects/xilinx/microzed/xsdk/aws_demos要浏览目录，请选择 Browse (浏览)。

   在指定一个根目录后，该目录中的项目将显示在 Import Projects (导入项目) 页面上。默认情况下，将选择所有可用项目。

   > **Note**
   >
   > 如果 Import Projects (导入项目) 页面顶部显示一条警告（“Some projects cannot be imported because they already exist in the workspace (由于一些项目已在工作区中存在，因此无法导入这些项目)。”），您可以忽略它。

6. 选定所有项目后，选择 Finish (完成)。
7. 如果在项目窗格中看不到 aws_bsp、fsbl 和 MicroZed_hw_platform_0 项目，请从 #3 开始重复上述步骤（但根目录设置为 *freertos*/vendors/xilinx），然后导入 aws_bsp、fsbl 和 MicroZed_hw_platform_0。
8. 从文件菜单中，选择 Window (窗口)，然后选择 Preferences (首选项)。
9. 在导航窗格中，展开 Run/Debug (运行/调试)，选择 String Substitution (字符串替换项)，然后选择 New (新建)。
10. 在 New String Substitution Variable (新字符串替换变量) 中，对于 Name (名称)，输入 **AFR_ROOT**。对于 Value (值)，输入 *freertos*/projects/xilinx/microzed/xsdk/

aws_demos 的根路径。选择 OK (确定)，然后选择 OK (确定) 以保存变量并关闭 Preferences (首选项)。

构建 FreeRTOS 演示项目

1. 在 XSDK IDE 中，从菜单中选择 Project (项目)，然后选择 Clean (清理)。
2. 在 Clean (清理) 中，将选项保留其默认值，然后选择 OK (确定)。XSDK 将清理和构建所有项目，然后生成 `.elf` 文件。

> **Note**
>
> 要构建所有项目而不进行清理，请选择 Project (项目)，然后选择 Build All (全部构建)。要构建单个项目，请选择要构建的项目，选择 Project (项目)，然后选择 Build Project (构建项目)。

为 FreeRTOS 演示项目生成启动映像

1. 在 XSDK IDE 中，右键单击 aws_demos，然后选择 Create Boot Image (创建启动映像)。
2. 在 Create Boot Image (创建启动映像) 中，选择 Create new BIF file (创建新的 BIF 文件)。
3. 在 Output BIF file path (输出 BIF 文件路径) 的旁边，选择 Browse (浏览)，然后选择 `aws_demos.bif`（位于 *<freertos>*`/vendors/xilinx/microzed/aws_demos/aws_demos.bif` 中）。
4. 选择 添加。
5. 在 Add new boot image partition (添加新的启动映像分区) 上，在 File path (文件路径) 的旁边，选择 Browse (浏览)，然后选择 `fsbl.elf`（位于 `vendors/xilinx/fsbl/Debug/fsbl.elf` 中）。
6. 对于 Partition type (分区类型)，选择 bootloader，然后选择 OK (确定)。
7. 在 Create Boot Image (创建启动映像) 上，选择 Create Image (创建映像)。在 Override Files (覆盖文件) 上，选择 OK (确定) 以覆盖现有 `aws_demos.bif` 并生成 `BOOT.bin` 文件（位于 `projects/xilinx/microzed/xsdk/aws_demos/BOOT.bin` 中）。

JTAG 调试

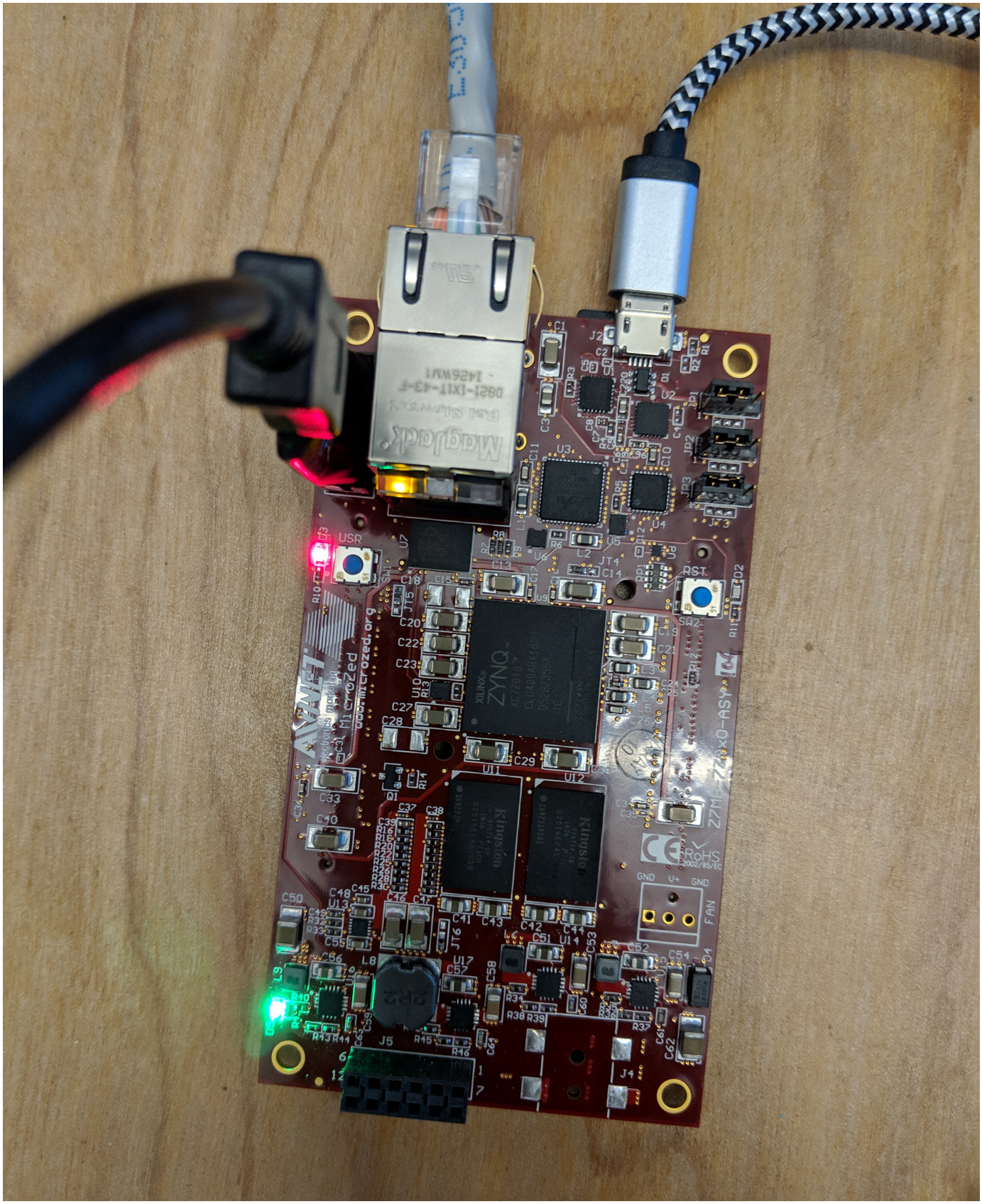
1. 将 MicroZed 主板的启动模式跳线设置为 JTAG 启动模式。



2. 将您的 MicroSD 卡插入位于 USB-UART 端口正下方的 MicroSD 卡槽。

> **Note**
>
> 在调试之前，请确保已对 MicroSD 卡上的所有内容进行备份。

您的主板应与下图类似：


USR
RST
ZYNQ
Kingston
Kingston
RoHS
GND V+ GND
FAN

3. 在 XSDK IDE 中，右键单击 aws_demos，选择 Debug As (调试方式)，然后选择 1 Launch on System Hardware (System Debugger) (1 在系统硬件上启动 (系统调试程序))。
4. 当调试程序在 `main()` 中的断点停止时，从菜单中选择 Run (运行)，然后选择 Resume (恢复)。

> **Note**
>
> 第一次运行应用程序时，将在非易失性存储器中导入新的证书密钥对。对于后续运行，您可以先在 `main.c` 文件中注释 `vDevModeKeyProvisioning()`，然后再重新构建映像和 `BOOT.bin` 文件。这将阻止每次运行时将证书和密钥复制到存储中。

你可以选择从 microSD 卡或 QSPI 闪存启动 MicroZed 主板来运行 FreeRTOS 演示项目。有关说明，请参阅为 FreeRTOS 演示项目生成启动映像和运行 FreeRTOS 演示项目。

运行 FreeRTOS 演示项目

要运行 FreeRTOS 演示项目，你可以从 microSD 卡 MicroZed 或 QSPI 闪存启动主板。

在设置 MicroZed 主板以运行 FreeRTOS 演示项目时，请参阅中的示意图。设置 MicroZed 硬件确保已将 MicroZed 主板连接到计算机。

从 MicroSD 卡启动 FreeRTOS 项目

格式化 Xilinx 工业 MicroZed物联网套件附带的 microSD 卡。

1. 将 `BOOT.bin` 文件复制到 MicroSD 卡中。
2. 将卡插入位于 USB-UART 端口正下方的 MicroSD 卡槽。
3. 将 MicroZed 启动模式跳线设置为 SD 启动模式。



4. 按 RST 按钮以重置设备并开始启动应用程序。您也可以从 USB-UART 端口拔出 USB-UART 电缆，然后重新插入该电缆。

从 QSPI 闪存启动 FreeRTOS 演示项目

1. 将 MicroZed 主板的启动模式跳线设置为 JTAG 启动模式。



2. 确认您的计算机已连接到 USB-UART 和 JTAG Access 端口。绿色电源正常 LED 指示灯应亮起。
3. 在 XSDK IDE 中，从菜单中选择 Xilinx，然后选择 Program Flash (对闪存编程)。
4. 在 Program Flash Memory (对闪存编程) 中，应自动填充硬件平台。对于 Connecti on，请选择您的 MicroZed 硬件服务器以将主板与主机连接起来。

> **Note**
>
> 如果您使用的是 Xilinx Smart Lync JTAG 电缆，则必须在 XSDK IDE 中创建硬件服务器。选择 New (新建)，然后定义您的服务器。

5. 在 Image File (映像文件) 中，输入 `BOOT.bin` 映像文件的目录路径。选择 Browse (浏览) 以改为浏览文件。
6. 在 Offset (偏移量) 中，输入 **`0x0`**。
7. 在 FSBL File (FSBL 文件) 中，输入 `fsbl.elf` 文件的目录路径。选择 Browse (浏览) 以改为浏览文件。
8. 选择 Program (编程) 以对主板编程。
9. 在 QSPI 编程完成后，拔掉 USB-UART 电缆以使主板断电。
10. 将 MicroZed 主板的启动模式跳线设置为 QSPI 启动模式。
11. 将卡插入位于 USB-UART 端口正下方的 MicroSD 卡槽。

> **Note**
>
> 确保已对 MicroSD 卡上的所有内容进行备份。

12. 按 RST 按钮以重置设备并开始启动应用程序。您也可以从 USB-UART 端口拔出 USB-UART 电缆，然后重新插入该电缆。

故障排除

如果您遇到与错误路径相关的构建错误，请尝试清理并重新构建项目，如构建 FreeRTOS 演示项目中所述。

如果您使用的是 Windows，请确保在 Windows XSDK IDE 中设置字符串替代变量时使用正斜杠。

有关 FreeRTOS 入门的常规故障排除信息，请参阅问题排查入门。

FreeRTOS 的后续步骤

> **Important**
>
> 此页面指向已弃用的 Amazon FreeRTOS 存储库。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

为主板构建、刷写和运行 FreeRTOS 演示项目后，您可以访问 FreeRTOS.org 网站，详细了解如何创建新的 FreeRTOS 项目。还有许多 FreeRTOS 库的演示，展示如何执行重要任务、 AWS IoT 与服务交互以及编程板特定的功能（例如蜂窝调制解调器）。有关更多信息，请参阅 FreeRTOS 库类别页面。

FreeRTOS.org 网站还提供有关 FreeRTOS 内核的深入信息，以及实时操作系统的基本概念。有关更多信息，请参阅 FreeRTOS 内核开发者文档和 FreeRTOS 辅助内核文档页面。

FreeRTOS 更新 Over-the-Air

> **Note**
>
> 有关 Over-the-air执行 AWS IoT Over-the-Air (OTA) 更新的最新信息，请参阅 FreeRTOS 网站上的 (OTA)。

Over-the-air (OTA) 更新允许您将固件更新部署到队列中的一台或多台设备。尽管 OTA 更新旨在更新设备固件，但可以用来将任意文件发送到已注册到 AWS IoT的一个或多个设备。在以无线方式发送更新时，建议您对更新进行数字签名，以便接收文件的设备能够验证更新在传输途中未经篡改。

可以使用 Code Signing for AWS IoT 来签署文件，也可以使用自己的代码签名工具来签署文件。

在创建 OTA 更新时，OTA Update Manager 服务将创建一个 AWS IoT 作业，通知设备有可用的更新。OTA 演示应用程序在您的设备上运行，并创建一个 FreeRTOS 任务，该任务 AWS IoT 用于订阅任务的通知主题并监听更新消息。当有可用的更新时，OTA 代理会将请求发布到 AWS IoT 并使用 HTTP 或 MQTT 协议接收更新，具体取决于您选择的设置。OTA 代理 将检查所下载文件的数字签名，

如果文件有效，则安装固件更新。如果不使用 FreeRTOS OTA 更新演示应用程序，则必须将 AWS IoT 空中 (OTA) 图书馆 集成到您自己的应用程序中，以获取固件更新功能。

Fre over-the-air eRTOS 更新使您可以：

- 在部署前，对固件进行数字签名。
- 将固件映像部署到单个设备、一组设备或整个机群。
- 在将设备添加到组，或重置或重新预配置设备时，将固件部署到设备。
- 在新固件部署到设备之后，验证其真实性和完整性。
- 监控部署进度。
- 调试失败的部署。

标记 OTA 资源

为了帮助您管理您的 OTA 资源，您可以选择以标签格式向更新和流分配您自己的元数据。标签使您能够以不同的方式对 AWS IoT 资源进行分类（例如，按用途、所有者或环境）。这在您有许多相同类型的资源时会非常有用。可以根据您分配给资源的标签来快速识别资源。

有关更多信息，请参阅标记 AWS IoT 资源。

OTA 更新先决条件

要使用 over-the-air (OTA) 更新，请执行以下操作：

- 检查 使用 HTTP 的 OTA 更新的先决条件 或 使用 MQTT 的 OTA 更新的先决条件。
- 创建 Amazon S3 存储桶以存储更新.
- 创建 OTA 更新服务角色.
- 创建 OTA 用户策略.
- 创建代码签名证书.
- 如果您将代码签名用于 AWS IoT，授予对代码签名的访问权限 AWS IoT.
- 下载 FreeRTOS 及 OTA 库.

创建 Amazon S3 存储桶以存储更新

OTA 更新文件存储在 Amazon S3 存储桶中。

如果您使用的是代码签名 AWS IoT，则用于创建代码签名任务的命令将使用源存储桶（未签名的固件映像所在的位置）和目标存储桶（写入已签名的固件映像的地方）。可以为源和目标指定相同的存储桶。文件名已更改为， GUIDs 因此原始文件不会被覆盖。

创建 Amazon S3 存储桶

1. 登录 Amazon S3 控制台，网址为https://console.aws.amazon.com/s3/。
2. 选择 创建存储桶 。
3. 输入存储桶名称。
4. 在阻止公有访问的存储桶设置下，保持选中阻止所有公有访问以接受默认权限。
5. 在存储桶版本控制下，请选择启用以在同一存储桶中保留所有版本。
6. 选择 创建存储桶 。

有关 Amazon S3 的更多信息，请参阅《Amazon Simple Storage Service 用户指南》。

创建 OTA 更新服务角色

OTA 更新服务假定此角色代表您创建和管理 OTA 更新作业。

创建 OTA 服务角色

1. 登录到 https://console.aws.amazon.com/iam/。
2. 从导航窗格中，选择 Roles。
3. 选择创建角色。
4. 在选择受信任实体的类型下，选择 AWS 服务。
5. 从 AWS 服务列表中选择 IoT。
6. 在选择您的使用案例下面，选择 IoT。
7. 选择下一步: 权限。
8. 选择下一步: 标签。
9. 选择下一步：审核。
10. 输入角色名称和描述，然后选择 Create role (创建角色)。

有关 IAM 角色的更多信息，请参阅 IAM 角色。

> **Important**
>
> 要解决令人困惑的代理安全问题，您必须按照 AWS IoT Core 指南中的说明进行操作。

将 OTA 更新权限添加到 OTA 服务角色

1. 在 IAM 控制台页面上的搜索框中，输入角色的名称，然后从列表中选择该角色。
2. 选择附加策略。
3. 在搜索框中，输入 AmazonFree RTOSOTAUpdate “”，AmazonFreeRTOSOTAUpdate从筛选的策略列表中进行选择，然后选择附加策略以将策略附加到您的服务角色。

将所需 IAM 权限添加到 OTA 服务角色

1. 在 IAM 控制台页面上的搜索框中，输入角色的名称，然后从列表中选择该角色。
2. 选择添加内联策略。
3. 选择 JSON 选项卡。
4. 将以下策略文档复制并粘贴到文本框中：

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "iam:PassRole"
            ],
            "Resource": "arn:aws:iam::your_account_id:role/your_role_name"
      }
    ]
}
```

   请务必*your_account_id*使用您的 AWS 账户 ID 和 OTA 服务角色的名称*your_role_name*进行替换。

5. 选择查看策略。
6. 为策略输入名称，然后选择 Create policy (创建策略)。

> **Note**
>
> 如果您的 Amazon S3 存储桶名称以“afr-ota”开头，则无需执行以下过程。如果是，则 AWS 托管策略`AmazonFreeRTOSOTAUpdate`已包含所需的权限。

将所需 Amazon S3 权限添加到 OTA 服务角色

1. 在 IAM 控制台页面上的搜索框中，输入角色的名称，然后从列表中选择该角色。
2. 选择添加内联策略。
3. 选择 JSON 选项卡。
4. 将以下策略文档复制并粘贴到框中。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObjectVersion",
                "s3:GetObject",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::example-bucket/*"
            ]
        }
    ]
}
```

   此策略授予您 OTA 服务角色权限以读取 Amazon S3 对象。请务必*example-bucket*替换为存储桶的名称。

5. 选择查看策略。
6. 为策略输入名称，然后选择 Create policy (创建策略)。

创建 OTA 用户策略

您必须向您的用户授予执行 over-the-air更新的权限。 用户必须具备以下权限：

- 访问存储固件更新的 S3 存储桶。
- 访问存储在中的证书 AWS Certificate Manager。
- 访问 AWS IoT 基于 MQTT 的文件传输功能。
- 访问 FreeRTOS OTA 更新。
- 访问 AWS IoT 职位。
- 访问 IAM。
- 的访问代码签名 AWS IoT。请参阅授予对代码签名的访问权限 AWS IoT。
- 列出 FreeRTOS 硬件平台。
- 标记和取消标记 AWS IoT 资源。

要为您的用户授予所需的权限，请参阅 IAM policy。另请参阅授权用户和云服务使用作 AWS IoT 业。

要提供访问权限，请为您的用户、组或角色添加权限：

- 中的用户和群组 AWS IAM Identity Center：

  创建权限集合。按照《AWS IAM Identity Center 用户指南》中创建权限集的说明进行操作。
- 通过身份提供商在 IAM 中托管的用户：

  创建适用于身份联合验证的角色。按照《IAM 用户指南》中针对第三方身份提供商创建角色（联合身份验证）的说明进行操作。
- IAM 用户：
  - 创建您的用户可以担任的角色。按照《IAM 用户指南》中为 IAM 用户创建角色的说明进行操作。
  - （不推荐使用）将策略直接附加到用户或将用户添加到用户组。按照《IAM 用户指南》中向用户添加权限（控制台）中的说明进行操作。

创建代码签名证书

要对固件映像进行数字签名，需要代码签名证书和私有密钥。出于测试目的，您可以创建自签名证书和私有密钥。对于生产环境，请通过众所周知的证书颁发机构 (CA) 购买证书。

不同平台需要不同类型的代码签名证书。以下部分介绍如何为符合 FreeRTOS 条件的不同平台创建代码签名证书。

主题

- 为德州仪器 0SF-LAUNCHXL 创建代码签名证书 CC322
- 为 Espressif 创建代码签名证书 ESP32
- 为 Nordic nrf52840-dk 创建代码签名证书
- 为 FreeRTOS Windows 模拟器创建代码签名证书
- 为自定义硬件创建代码签名证书

为德州仪器 0SF-LAUNCHXL 创建代码签名证书 CC322

> **Important**
>
> 该参考集成托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

SimpleLink Wi-Fi CC322 0SF 无线微控制器 Launchpad 开发套件支持两个用于固件代码签名的证书链：

- 生产 (certificate-catalog)

  要使用生产证书链，必须购买商用代码签名证书，并使用 TI Uniflash 工具将主板设置为生产模式。
- 测试和开发 (certificate-playground)

  Playground 证书链允许您使用自签名代码签名证书试用 OTA 更新。

使用 AWS Command Line Interface 将您的代码签名证书、私钥和证书链导入。 AWS Certificate Manager有关更多信息，请参阅AWS Command Line Interface 《用户指南》中的安装 AWS CLI。

下载并安装最新版本的 SimpleLink CC3220 SDK。默认情况下，所需的文件位于以下位置：

`C:\ti\simplelink_cc32xx_sdk_version\tools\cc32xx_tools\certificate-playground` (Windows)

`/Applications/Ti/simplelink_cc32xx_version/tools/cc32xx_tools/certificate-playground` (macOS)

SimpleLink CC3220 SDK 中的证书采用 DER 格式。要创建自签名代码签名证书，必须将其转换为 PEM 格式。

按照以下步骤创建代码签名证书，该证书链接到德州仪器 playground 证书层次结构， AWS Certificate Manager 并且符合标准的 AWS IoT 代码签名。

> **Note**
>
> 要创建代码签名证书，请在计算机上安装 OpenSSL。安装 OpenSSL 后，请确保将 `openssl` 分配给命令提示符或终端环境中的 OpenSSL 可执行文件。

创建自签名代码签名证书

1. 使用管理员权限打开命令提示符或终端。
2. 在工作目录中，使用以下文本创建名为 `cert_config.txt` 的文件。*test_signer@amazon.com*用您的电子邮件地址替换。

```
[ req ]
prompt             = no
distinguished_name = my dn

[ my dn ]
commonName = test_signer@amazon.com

[ my_exts ]
keyUsage         = digitalSignature
extendedKeyUsage = codeSigning
```

3. 创建私有密钥和证书签名请求 (CSR)：

```
openssl req -config cert_config.txt -extensions my_exts -nodes -days 365 -newkey
 rsa:2048 -keyout tisigner.key -out tisigner.csr
```

4. 将 Texas Instruments 操场根 CA 私有密钥从 DER 格式转换为 PEM 格式。

   TI 操场根 CA 私有密钥位于以下位置：

   `C:\ti\simplelink_cc32xx_sdk_`*version*`\tools\cc32xx_tools\certificate-playground\dummy-root-ca-cert-key` (Windows)

   `/Applications/Ti/simplelink_cc32xx_sdk_`*version*`/tools/cc32xx_tools/certificate-playground/dummy-root-ca-cert-key` (macOS)

```
openssl rsa -inform DER -in dummy-root-ca-cert-key -out dummy-root-ca-cert-key.pem
```

5. 将 Texas Instruments 操场根 CA 证书从 DER 格式转换为 PEM 格式。

   TI 操场根证书位于以下位置：

   `C:\ti\simplelink_cc32xx_sdk_`*`version`*`\tools\cc32xx_tools\certificate-playground/dummy-root-ca-cert` (Windows)

   `/Applications/Ti/simplelink_cc32xx_sdk_`*`version`*`/tools/cc32xx_tools/certificate-playground/dummy-root-ca-cert` (macOS)

   ```
   openssl x509 -inform DER -in dummy-root-ca-cert -out dummy-root-ca-cert.pem
   ```

6. 使用 Texas Instruments 根 CA 对 CSR 进行签名：

   ```
   openssl x509 -extfile cert_config.txt -extensions my_exts  -req -days 365 -in
    tisigner.csr -CA dummy-root-ca-cert.pem -CAkey dummy-root-ca-cert-key.pem -
   set_serial 01 -out tisigner.crt.pem -sha1
   ```

7. 将代码签名证书 (`tisigner.crt.pem`) 转换为 DER 格式：

   ```
   openssl x509 -in tisigner.crt.pem -out tisigner.crt.der -outform DER
   ```

   > **Note**
   >
   > 稍后可将 `tisigner.crt.der` 证书写入到 TI 开发主板上。

8. 将代码签名证书、私钥和证书链导入： AWS Certificate Manager

   ```
   aws acm import-certificate --certificate fileb://tisigner.crt.pem --private-key
    fileb://tisigner.key --certificate-chain fileb://dummy-root-ca-cert.pem
   ```

   此命令显示了证书的 ARN。在创建 OTA 更新作业时需要此 ARN。

> **Note**
>
> 编写此步骤时假设您将使用代码签名对固件映像 AWS IoT 进行签名。尽管建议使用代码签名， AWS IoT 但您可以手动对固件映像进行签名。

为 Espressif 创建代码签名证书 ESP32

> **Important**
>
> 该参考集成托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

乐鑫 ESP32 主板支持带有 ECDSA 代码签名证书的自签名 SHA-256。

> **Note**
>
> 要创建代码签名证书，请在计算机上安装 OpenSSL。安装 OpenSSL 后，请确保将 `openssl` 分配给命令提示符或终端环境中的 OpenSSL 可执行文件。
> 使用 AWS Command Line Interface 将您的代码签名证书、私钥和证书链导入。 AWS Certificate Manager有关安装的信息 AWS CLI，请参阅安装 AWS CLI。

1. 在工作目录中，使用以下文本创建名为 `cert_config.txt` 的文件。*`test_signer@amazon.com`*用您的电子邮件地址替换：

```
[ req ]
prompt             = no
distinguished_name = my_dn

[ my_dn ]
commonName = test_signer@amazon.com

[ my_exts ]
keyUsage         = digitalSignature
extendedKeyUsage = codeSigning
```

2. 创建 ECDSA 代码签名私有密钥：

```
openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -pkeyopt
 ec_param_enc:named_curve -outform PEM -out ecdsasigner.key
```

3. 创建 ECDSA 代码签名证书：

```
openssl req -new -x509 -config cert_config.txt -extensions my_exts -nodes -days 365
 -key ecdsasigner.key -out ecdsasigner.crt
```

4. 将代码签名证书、私钥和证书链导入： AWS Certificate Manager

```
aws acm import-certificate --certificate fileb://ecdsasigner.crt --private-key
 fileb://ecdsasigner.key
```

此命令显示了证书的 ARN。在创建 OTA 更新作业时需要此 ARN。

> **Note**
>
> 编写此步骤时假设您将使用代码签名对固件映像 AWS IoT 进行签名。尽管建议使用代码签名， AWS IoT 但您可以手动对固件映像进行签名。

为 Nordic nrf52840-dk 创建代码签名证书

> **Important**
>
> 该参考集成托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

Nordic nrf52840-dk 支持使用 ECDSA 代码签名证书的自签名 SHA256 。

> **Note**
>
> 要创建代码签名证书，请在计算机上安装 OpenSSL。安装 OpenSSL 后，请确保将 `openssl` 分配给命令提示符或终端环境中的 OpenSSL 可执行文件。

使用 AWS Command Line Interface 将您的代码签名证书、私钥和证书链导入。 AWS Certificate Manager有关安装的信息 AWS CLI，请参阅安装 AWS CLI。

1. 在工作目录中，使用以下文本创建名为 cert_config.txt 的文件。*test_signer@amazon.com*用您的电子邮件地址替换：

```
[ req ]
prompt             = no
distinguished_name = my_dn

[ my_dn ]
commonName = test_signer@amazon.com

[ my_exts ]
keyUsage         = digitalSignature
extendedKeyUsage = codeSigning
```

2. 创建 ECDSA 代码签名私有密钥：

```
openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -pkeyopt
 ec_param_enc:named_curve -outform PEM -out ecdsasigner.key
```

3. 创建 ECDSA 代码签名证书：

```
openssl req -new -x509 -config cert_config.txt -extensions my_exts -nodes -days 365
 -key ecdsasigner.key -out ecdsasigner.crt
```

4. 将代码签名证书、私钥和证书链导入： AWS Certificate Manager

```
aws acm import-certificate --certificate fileb://ecdsasigner.crt --private-key
 fileb://ecdsasigner.key
```

此命令显示了证书的 ARN。在创建 OTA 更新作业时需要此 ARN。

> **Note**
>
> 编写此步骤时假设您将使用代码签名对固件映像 AWS IoT 进行签名。尽管建议使用代码签名， AWS IoT 但您可以手动对固件映像进行签名。

为 FreeRTOS Windows 模拟器创建代码签名证书

FreeRTOS Windows 模拟器需要带 ECDSA P-256 密钥和 SHA-256 哈希的代码签名证书，才能执行 OTA 更新。如果没有代码签名证书，请执行以下步骤来创建一个。

> **Note**
>
> 要创建代码签名证书，请在计算机上安装 OpenSSL。安装 OpenSSL 后，请确保将 `openssl` 分配给命令提示符或终端环境中的 OpenSSL 可执行文件。
> 使用 AWS Command Line Interface 将您的代码签名证书、私钥和证书链导入。 AWS Certificate Manager有关安装的信息 AWS CLI，请参阅安装 AWS CLI。

1. 在工作目录中，使用以下文本创建名为 `cert_config.txt` 的文件。*test_signer@amazon.com*用您的电子邮件地址替换：

```
[ req ]
prompt             = no
distinguished_name = my_dn

[ my_dn ]
commonName = test_signer@amazon.com

[ my_exts ]
keyUsage         = digitalSignature
extendedKeyUsage = codeSigning
```

2. 创建 ECDSA 代码签名私有密钥：

```
openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -pkeyopt
 ec_param_enc:named_curve -outform PEM -out ecdsasigner.key
```

3. 创建 ECDSA 代码签名证书：

```
openssl req -new -x509 -config cert_config.txt -extensions my_exts -nodes -days 365
 -key ecdsasigner.key -out ecdsasigner.crt
```

4. 将代码签名证书、私钥和证书链导入： AWS Certificate Manager

```
aws acm import-certificate --certificate fileb://ecdsasigner.crt --private-key
 fileb://ecdsasigner.key
```

此命令显示了证书的 ARN。在创建 OTA 更新作业时需要此 ARN。

> **Note**
>
> 编写此步骤时假设您将使用代码签名对固件映像 AWS IoT 进行签名。尽管建议使用代码签名， AWS IoT 但您可以手动对固件映像进行签名。

为自定义硬件创建代码签名证书

借助适当的工具集，可以为您的硬件创建自签名的证书和私有密钥。

使用 AWS Command Line Interface 将您的代码签名证书、私钥和证书链导入。 AWS Certificate Manager有关安装的信息 AWS CLI，请参阅安装 AWS CLI。

创建代码签名证书后，您可以使用将其导 AWS CLI 入 ACM：

```
aws acm import-certificate --certificate fileb://code-sign.crt --private-key fileb://
code-sign.key
```

此命令的输出显示了证书的 ARN。在创建 OTA 更新作业时需要此 ARN。

ACM 要求证书使用特定的算法和密钥大小。有关更多信息，请参阅导入证书的先决条件。有关 ACM 的更多信息，请参阅将证书导入 AWS Certificate Manager。

必须将代码签名证书的内容复制并粘贴到 `vendors/vendor/boards/board/aws_demos/config_files/ota_demo_config.h` 文件中并进行格式化，该文件属于您稍后下载的 FreeRTOS 代码的一部分。

授予对代码签名的访问权限 AWS IoT

要提供访问权限，请为您的用户、组或角色添加权限：

- 中的用户和群组 AWS IAM Identity Center：

  创建权限集合。按照《AWS IAM Identity Center 用户指南》中创建权限集的说明进行操作。
- 通过身份提供商在 IAM 中托管的用户：

  创建适用于身份联合验证的角色。按照《IAM 用户指南》中针对第三方身份提供商创建角色（联合身份验证）的说明进行操作。

- IAM 用户：
  - 创建您的用户可以担任的角色。按照《IAM 用户指南》中为 IAM 用户创建角色的说明进行操作。
  - （不推荐使用）将策略直接附加到用户或将用户添加到用户组。按照《IAM 用户指南》中向用户添加权限（控制台）中的说明进行操作。

下载 FreeRTOS 及 OTA 库

你可以从中克隆或下载 FreeRTOS。GitHub有关说明，请参阅 README.md 文件。

有关设置和运行 OTA 演示应用程序的信息，请参阅Over-the-air 更新演示应用程序。

Important

- 在本主题中，FreeRTOS 下载目录的路径称为 *freertos*。
- *freertos* 路径中的空格字符可能会导致构建失败。克隆或复制存储库时，请确保您创建的路径不包含空格字符。
- Microsoft Windows 上的文件路径最大长度为 260 个字符。FreeRTOS 下载目录路径过长可能会导致构建操作失败。
- 由于源代码可能包含符号链接，因此，如果您使用 Windows 提取存档，则可能必须：
  - 启用开发者模式，或者，
  - 使用已提升为管理员的控制台。

  这样，Windows 就可以在提取存档文件时正确创建符号链接。否则，符号链接将作为普通文件写入，该文件会将符号链接的路径作为文本，或者是空文件。有关更多信息，请参阅博客文章 Windows 10 中的符号链接。

  如果您在 Windows 下使用 Git，则必须启用开发者模式，或者必须：
  - 使用以下命令将 `core.symlinks` 设置为 true：

    ```
    git config --global core.symlinks true
    ```
  - 每当您使用写入系统的 git 命令（例如，git pull、git clone 和 git submodule update --init --recursive）时，请使用具有管理员权限的控制台。

使用 MQTT 的 OTA 更新的先决条件

本节介绍使用 MQTT 执行任务的一般要求 over-the-air（OTA 更新）。

最低要求

- 设备固件必须包含必要的 FreeRTOS 库（coreMQTT 代理、OTA 更新及其依赖项）。
- 要求安装 FreeRTOS 1.4.0 版或更高版本。但是，我们建议您尽可能使用最新的版本。

配置

从 201912.00 版本开始，FreeRTOS OTA 可以使用 HTTP 或 MQTT 协议将固件更新映像从设备传输到设备。 AWS IoT 如果在 FreeRTOS 中创建 OTA 更新时指定两个协议，则每个设备都将确定用于传输映像的协议。请参阅使用 HTTP 的 OTA 更新的先决条件了解更多信息。

默认情况下，`ota_config.h` 中的 OTA 协议配置将使用 MQTT 协议。

设备特定的配置

无。

内存使用量

在将 MQTT 用于数据传输时，MQTT 连接不需要额外的内存，因为它将在控制操作和数据操作之间共享。

设备策略

每台使用 MQTT 接收 OTA 更新的设备都必须注册为事物， AWS IoT 并且该设备必须附有类似于此处列出的策略。您可以在 AWS IoT 核心策略操作和 AWS IoT 核心操作资源中找到有关 `"Action"` 和 `"Resource"` 对象中项目的更多信息。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:Connect",
            "Resource": "arn:partition:iot:region:account:client/
${iot:Connection.Thing.ThingName}"
        },
```

```
        {
            "Effect": "Allow",
            "Action": "iot:Subscribe",
            "Resource": [
                "arn:partition:iot:region:account:topicfilter/$aws/things/
${iot:Connection.Thing.ThingName}/streams/*",
                "arn:partition:iot:region:account:topicfilter/$aws/things/
${iot:Connection.Thing.ThingName}/jobs/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish",
                "iot:Receive"
            ],
            "Resource": [
                "arn:partition:iot:region:account:topic/$aws/things/
${iot:Connection.Thing.ThingName}/streams/*",
                "arn:partition:iot:region:account:topic/$aws/things/
${iot:Connection.Thing.ThingName}/jobs/*"
            ]
        }
    ]
}
```

备注

- 这些`iot:Connect`权限允许您的设备通过 MQTT AWS IoT 进行连接。
- AWS IoT 任务主题的`iot:Subscribe`和`iot:Publish`权限 (`.../jobs/*`) 允许连接的设备接收作业通知和作业文档，并发布任务执行的完成状态。
- AWS IoT OTA 直播主题的`iot:Subscribe`和`iot:Publish`权限 (`.../streams/*`) 允许连接的设备从中获取 OTA 更新数据 AWS IoT。在通过 MQTT 执行固件更新时，需要这些权限。
- 这些`iot:Receive`权限 AWS IoT Core 允许将有关这些主题的消息发布到连接的设备。每次传输 MQTT 消息时，都将检查此权限。您可以使用此权限，撤消对当前订阅主题的客户端的访问权限。

使用 HTTP 的 OTA 更新的先决条件

本节介绍使用 HTTP 执行 over-the-air (OTA) 更新的一般要求。从 201912.00 版本开始，FreeRTOS OTA 可以使用 HTTP 或 MQTT 协议将固件更新映像从设备传输到设备。 AWS IoT

> **Note**
>
> - 尽管可以使用 HTTP 协议来传输固件映像，但仍然需要 CoreMQTT 代理库，因为与之的其他交互都 AWS IoT Core 使用 CoreMQTT 代理库，包括发送或接收任务执行通知、作业文档和执行状态更新。
> - 在为 OTA 更新作业同时指定 MQTT 和 HTTP 协议时，每个设备上的 OTA 代理软件的设置将确定用于传输固件映像的协议。要将 OTA 代理从默认 MQTT 协议方法更改为 HTTP 协议，可以修改用于编译设备的 FreeRTOS 源代码的标头文件。

最低要求

- 设备固件必须包含必要的 FreeRTOS 库（coreMQTT 代理、HTTP、OTA 代理及其依赖项）。
- 要求安装 FreeRTOS 版本 201912.00 或更高版本来更改 OTA 协议的配置，以便启用通过 HTTP 传输 OTA 数据。

配置

请参阅 \vendors\boards*board*\aws_demos\config_files\ota_config.h 文件中的以下 OTA 协议配置。

启用通过 HTTP 进行的 OTA 数据传输

1. 将 `configENABLED_DATA_PROTOCOLS` 更改为 `OTA_DATA_OVER_HTTP`。
2. 在进行 OTA 更新时，您可以指定两项协议以便能够使用 MQTT 协议或 HTTP 协议。您可以通过将 `configOTA_PRIMARY_DATA_PROTOCOL` 更改为 `OTA_DATA_OVER_HTTP` 来将设备所使用的主协议设置为 HTTP。

> **Note**
>
> OTA 数据操作仅支持 HTTP。对于控制操作，您必须使用 MQTT。

设备特定的配置

ESP32

由于 RAM 量是有限的，因此，在将 HTTP 作为 OTA 数据协议启用时，必须关闭 BLE。在 vendors/espressif/boards/esp32/aws_demos/config_files/aws_iot_network_config.h 文件中，仅将 configENABLED_NETWORKS 更改为 AWSIOT_NETWORK_TYPE_WIFI。

```
/**
     * @brief Configuration flag which is used to enable one or more network
 interfaces for a board.
     *
     * The configuration can be changed any time to keep one or more network enabled
 or disabled.
     * More than one network interfaces can be enabled by using 'OR' operation with
 flags for
     * each network types supported. Flags for all supported network types can be
 found
     * in "aws_iot_network.h"
     *
     */
    #define configENABLED_NETWORKS      ( AWSIOT_NETWORK_TYPE_WIFI )
```

内存使用量

在将 MQTT 用于数据传输时，MQTT 连接不需要额外的堆内存，因为它将在控制操作和数据操作之间共享。不过，通过 HTTP 启用数据需要额外的堆内存。以下是使用 FreeRTOS xPortGetFreeHeapSize API 计算的所有受支持平台的堆内存使用情况数据。您必须确保有足够的 RAM 才能使用 OTA 库。

德州仪器 CC322 0SF-LAUNCHXL

控制操作 (MQTT)：12 KB

数据操作 (HTTP)：10 KB

> **Note**
>
> TI 使用更少的 RAM，因为它在硬件上使用 SSL，所以它不使用 mbedtls 库。

微芯片好奇号 PIC32 MZEF

控制操作 (MQTT)：65 KB

数据操作 (HTTP)：43 KB

Espressif ESP32

控制操作 (MQTT)：65 KB

数据操作 (HTTP)：45 KB

> **Note**
>
> BLE ESP32 开启需要大约 87 KB 的内存。RAM 不足，无法启用所有这些操作，上面的特定于设备的配置中已说明这一点。

Windows 模拟器

控制操作 (MQTT)：82 KB

数据操作 (HTTP)：63 KB

Nordic nrf52840-dk

不支持 HTTP。

设备策略

通过此策略，您能够使用 MQTT 或 HTTP 进行 OTA 更新。

每个使用 HTTP 接收 OTA 更新的设备都必须在 AWS IoT 中注册为事物，并且该设备必须具有类似此处所列策略的附加策略。您可以在 AWS IoT 核心策略操作和 AWS IoT 核心操作资源中找到有关 `"Action"` 和 `"Resource"` 对象中项目的更多信息。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:Connect",
```

```
            "Resource": "arn:partition:iot:region:account:client/
${iot:Connection.Thing.ThingName}"
        },
        {
            "Effect": "Allow",
            "Action": "iot:Subscribe",
            "Resource": [
                "arn:partition:iot:region:account:topicfilter/$aws/things/
${iot:Connection.Thing.ThingName}/jobs/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish",
                "iot:Receive"
            ],
            "Resource": [
                "arn:partition:iot:region:account:topic/$aws/things/
${iot:Connection.Thing.ThingName}/jobs/*"
            ]
        }
    ]
}
```

备注

- 利用 `iot:Connect` 权限，您的设备可以通过 MQTT 连接到 AWS IoT 。
- AWS IoT 任务主题的`iot:Subscribe`和`iot:Publish`权限 (`.../jobs/*`) 允许连接的设备接收作业通知和作业文档，并发布任务执行的完成状态。
- 这些`iot:Receive`权限 AWS IoT Core 允许将有关这些主题的消息发布到当前连接的设备。每次传输 MQTT 消息时，都将检查此权限。您可以使用此权限，撤消对当前订阅主题的客户端的访问权限。

OTA 教程

本部分包含了如何使用 OTA 更新在运行 FreeRTOS 的设备上更新固件的教程。除了固件映像之外，您还可以使用 OTA 更新将任何类型的文件发送到已连接 AWS IoT的设备。

您可以使用 AWS IoT 控制台或创建 AWS CLI OTA 更新。控制台是开始使用 OTA 的最简便方式，因为它为您执行了大量工作。 AWS CLI 在自动执行 OTA 更新任务、使用大量设备或使用不符合

FreeRTOS 资格的设备时，这非常有用。有关获得 FreeRTOS 资格认证的设备的更多信息，请参阅 FreeRTOS 合作伙伴网站。

创建 OTA 更新

1. 将初始版本的固件部署到一个或多个设备。
2. 验证固件的运行是否正常。
3. 需要固件更新时，修改代码并构建新映像。
4. 如果手动签署固件，则对固件映像进行签名，之后将其上传到 Amazon S3 存储桶。如果您使用代码签名 AWS IoT，请将未签名的固件映像上传到 Amazon S3 存储桶。
5. 创建 OTA 更新。

在创建 OTA 更新时，可以指定映像传输协议 MQTT 或 HTTP，也可以指定两者来让设备进行选择。设备上的 FreeRTOS OTA 代理将接收更新后的固件映像，并验证数字签名、校验和以及新映像的版本号。如果固件更新通过验证，设备将重置，并根据应用程序定义的逻辑提交更新。如果设备未运行 FreeRTOS，则必须实现一个在设备上运行的 OTA 代理。

安装初始固件

要更新固件，必须安装初始版本的固件，该固件使用 OTA 代理库侦听 OTA 更新作业。如果未运行 FreeRTOS，请跳过此步骤。而是必须将 OTA 代理实现复制到设备上。

主题

- 在德州仪器 CC322 0SF-LAUNCHXL 上安装固件的初始版本
- 在 Espressif 上安装固件的初始版本 ESP32
- 在 Nordic n RF5284 0 DK 上安装固件的初始版本
- Windows 模拟器上的初始固件
- 在自定义主板上安装初始版本的固件

在德州仪器 CC322 0SF-LAUNCHXL 上安装固件的初始版本

> **Important**
>
> 该参考集成托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

编写以下步骤时，假定您已经按照在德州仪器 0SF-LAUNCHXL 上下载、构建、刷新和运行 FreeRTOS OTA 演示 CC322 中的说明，构建了 aws_demos 项目。

1. 在德州仪器 CC322 0SF-LAUNCHXL 上，将 SOP 跳线放在中间的一组引脚（位置 = 1）上，然后重置电路板。
2. 下载并安装 TI Uniflash 工具。
3. 启动 Uniflash。从配置列表中选择 CC3220SF-LAUNCHXL，然后选择 “启动图像创建器”。
4. 选择新项目。
5. 在 Start new project (启动新项目) 页面上，输入项目名称。对于设备类型，选择 CC3220SF。对于 Device Mode (设备模式)，选择 Develop (开发)。选择创建项目。
6. 断开终端模拟器。
7. 在 Uniflash 应用程序窗口的右侧，选择 Connect (连接)。
8. 在 Advanced (高级)、Files (文件) 下，选择 User Files (用户文件)。
9. 在 File (文件) 选择器窗格中，选择 Add File (添加文件) 图标

   。
10. 浏览至 /Applications/Ti/simplelink_cc32xx_sdk_*version*/tools/cc32xx_tools/certificate-playground 目录，选择 dummy-root-ca-cert，然后依次选择 Open (打开) 和 Write (写入)。
11. 在 File (文件) 选择器窗格中，选择 Add File (添加文件) 图标

    。
12. 浏览至创建代码签名证书和私有密钥的工作目录，选择 tisigner.crt.der，然后依次选择 Open (打开) 和 Write (写入)。
13. 从 Action (操作) 下拉列表中，选择 Select MCU Image (选择 MCU 映像)，然后选择 Browse (浏览) 以选择要写入设备的固件映像 (aws_demos.bin)。此文件位于 *freertos*/vendors/ti/boards/cc3220_launchpad/aws_demos/ccs/Debug 目录中。选择打开。
    a. 在“File (文件)”对话框中，确认文件名设置为 mcuflashimg.bin
    b. 选中 Vendor (供应商) 复选框。
    c. 在 File Token (文件令牌) 下，键入 **1952007250**。
    d. 在 Private Key File Name (私有密钥文件名称) 下，选择 Browse (浏览)，然后从创建代码签名证书和私有密钥的工作目录中选择 tisigner.key。
    e. 在 Certification File Name (认证文件名称) 下，选择 tisigner.crt.der。
    f. 选择写入。

14. 在左侧窗格的 Files (文件) 下，选择 Service Pack。
15. 在 Service Pack File Name (Service Pack 文件名称) 下，选择 Browse (浏览)，浏览至 `simplelink_cc32x_sdk_version/tools/cc32xx_tools/servicepack-cc3x20`，选择 `sp_3.7.0.1_2.0.0.0_2.2.0.6.bin`，然后选择 Open (打开)。
16. 在左侧窗格的 Files (文件) 下，选择 Trusted Root-Certificate Catalog (受信任的根证书目录)。
17. 清除 Use default Trusted Root-Certificate Catalog (使用默认的受信任的根证书目录) 复选框。
18. 在 “源文件” 下，选择 “浏览”，选择 s implelink_cc32xx_sdk_/20160911.ls ***version*** t，然后选择 “打开”。tools/cc32xx_tools/certificate-playground/certcatalogPlayGround
19. 在 “签名源文件” 下，选择 “浏览”，选择 s implelink_cc32xx_sdk_/20160911.lst.signed_3220.b ***version*** in，然后选择 “打开”。tools/cc32xx_tools/certificate-playground/certcatalogPlayGround
20. 选择

    按钮保存项目。
21. 选择

    按钮。
22. 选择 Program Image (Create and Program) (编程映像 (创建和编程))。
23. 编程过程完成之后，将 SOP 跳线放在第一组针脚（位置 = 0）上，重置主板，然后重新连接终端模拟器，以确保输出与使用 Code Composer Studio 调试演示代码时是相同的。记下终端输出中的应用程序版本号。稍后可以使用此版本号验证固件是否已通过 OTA 更新进行了更新。

    终端应显示类似于以下内容的输出。

```
0 0 [Tmr Svc] Simple Link task created

Device came up in Station mode

1 369 [Tmr Svc] Starting key provisioning...
2 369 [Tmr Svc] Write root certificate...
3 467 [Tmr Svc] Write device private key...
4 568 [Tmr Svc] Write device certificate...
SL Disconnect...

5 664 [Tmr Svc] Key provisioning done...
Device came up in Station mode
```

```
Device disconnected from the AP on an ERROR..!!

[WLAN EVENT] STA Connected to the AP: Guest , BSSID: 11:22:a1:b2:c3:d4

[NETAPP EVENT] IP acquired by the device


Device has connected to Guest

Device IP Address is 111.222.3.44


6 1716 [OTA] OTA demo version 0.9.0
7 1717 [OTA] Creating MQTT Client...
8 1717 [OTA] Connecting to broker...
9 1717 [OTA] Sending command to MQTT task.
10 1717 [MQTT] Received message 10000 from queue.
11 2193 [MQTT] MQTT Connect was accepted. Connection established.
12 2193 [MQTT] Notifying task.
13 2194 [OTA] Command sent to MQTT task passed.
14 2194 [OTA] Connected to broker.
15 2196 [OTA Task] Sending command to MQTT task.
16 2196 [MQTT] Received message 20000 from queue.
17 2697 [MQTT] MQTT Subscribe was accepted. Subscribed.
18 2697 [MQTT] Notifying task.
19 2698 [OTA Task] Command sent to MQTT task passed.
20 2698 [OTA Task] [OTA] Subscribed to topic: $aws/things/TI-LaunchPad/jobs/$next/
get/accepted

21 2699 [OTA Task] Sending command to MQTT task.
22 2699 [MQTT] Received message 30000 from queue.
23 2800 [MQTT] MQTT Subscribe was accepted. Subscribed.
24 2800 [MQTT] Notifying task.
25 2801 [OTA Task] Command sent to MQTT task passed.
26 2801 [OTA Task] [OTA] Subscribed to topic: $aws/things/TI-LaunchPad/jobs/notify-
next

27 2814 [OTA Task] [OTA] Check For Update #0
28 2814 [OTA Task] Sending command to MQTT task.
29 2814 [MQTT] Received message 40000 from queue.
30 2916 [MQTT] MQTT Publish was successful.
31 2916 [MQTT] Notifying task.
32 2917 [OTA Task] Command sent to MQTT task passed.
```

```
33 2917 [OTA Task] [OTA] Set job doc parameter [ clientToken: 0:TI-LaunchPad ]
34 2917 [OTA Task] [OTA] Missing job parameter: execution
35 2917 [OTA Task] [OTA] Missing job parameter: jobId
36 2918 [OTA Task] [OTA] Missing job parameter: jobDocument
37 2918 [OTA Task] [OTA] Missing job parameter: ts_ota
38 2918 [OTA Task] [OTA] Missing job parameter: files
39 2918 [OTA Task] [OTA] Missing job parameter: streamname
40 2918 [OTA Task] [OTA] Missing job parameter: certfile
41 2918 [OTA Task] [OTA] Missing job parameter: filepath
42 2918 [OTA Task] [OTA] Missing job parameter: filesize
43 2919 [OTA Task] [OTA] Missing job parameter: sig-sha1-rsa
44 2919 [OTA Task] [OTA] Missing job parameter: fileid
45 2919 [OTA Task] [OTA] Missing job parameter: attr
47 3919 [OTA] [OTA] Queued: 1   Processed: 1   Dropped: 0
48 4919 [OTA] [OTA] Queued: 1   Processed: 1   Dropped: 0
49 5919 [OTA] [OTA] Queued: 1   Processed: 1   Dropped: 0
```

在 Espressif 上安装固件的初始版本 ESP32

> **Important**
>
> 该参考集成托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

本指南假设您已经执行了 Espressif ESP32-DevKit C 入门中的步骤，以及更新先决 ESP-WROVER-KIT条件。Over-the-Air 在尝试 OTA 更新之前，可能需要运行 FreeRTOS 入门中所述的 MQTT 演示项目，以确保主板和工具链设置正确。

将初始出厂映像刷写到主板

1. 打开 *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h，注释掉 #define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED 并定义 CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED 或 CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED。

2. 将 OTA 更新先决条件 中生成的 SHA-256/ECDSA PEM 格式的代码签名证书复制到 vendors/*vendor*/boards/*board*/aws_demos/config_files/ota_demo_config.h。应按以下方式为其设置格式。

```
#define otapalconfigCODE_SIGNING_CERTIFICATE  \
"-----BEGIN CERTIFICATE-----\n" \
"...base64 data...\n" \
"-----END CERTIFICATE-----\n";
```

3. 选择 OTA Update 演示后，按照入门中概述的 ESP32相同步骤构建和刷新映像。如果之前已构建并刷写了项目，则可能需要先运行 make clean。在运行 make flash monitor 后，将会看到类似于以下内容的输出。某些消息的顺序可能会有所不同，因为演示应用程序同时运行多个任务。

```
I (28) boot: ESP-IDF v3.1-dev-322-gf307f41-dirty 2nd stage bootloader
I (28) boot: compile time 16:32:33
I (29) boot: Enabling RNG early entropy source...
I (34) boot: SPI Speed : 40MHz
I (38) boot: SPI Mode : DIO
I (42) boot: SPI Flash Size : 4MB
I (46) boot: Partition Table:
I (50) boot: ## Label Usage Type ST Offset Length
I (57) boot: 0 nvs WiFi data 01 02 00010000 00006000
I (64) boot: 1 otadata OTA data 01 00 00016000 00002000
I (72) boot: 2 phy_init RF data 01 01 00018000 00001000
I (79) boot: 3 ota_0 OTA app 00 10 00020000 00100000
I (87) boot: 4 ota_1 OTA app 00 11 00120000 00100000
I (94) boot: 5 storage Unknown data 01 82 00220000 00010000
I (102) boot: End of partition table
I (106) esp_image: segment 0: paddr=0x00020020 vaddr=0x3f400020 size=0x14784
 ( 83844) map
I (144) esp_image: segment 1: paddr=0x000347ac vaddr=0x3ffb0000 size=0x023ec
 ( 9196) load
I (148) esp_image: segment 2: paddr=0x00036ba0 vaddr=0x40080000 size=0x00400
 ( 1024) load
I (151) esp_image: segment 3: paddr=0x00036fa8 vaddr=0x40080400 size=0x09068
 ( 36968) load
I (175) esp_image: segment 4: paddr=0x00040018 vaddr=0x400d0018 size=0x719b8
 (465336) map
I (337) esp_image: segment 5: paddr=0x000b19d8 vaddr=0x40089468 size=0x04934
 ( 18740) load
I (345) esp_image: segment 6: paddr=0x000b6314 vaddr=0x400c0000 size=0x00000 ( 0)
 load
```

```
I (353) boot: Loaded app from partition at offset 0x20000
I (353) boot: ota rollback check done
I (354) boot: Disabling RNG early entropy source...
I (360) cpu_start: Pro cpu up.
I (363) cpu_start: Single core mode
I (368) heap_init: Initializing. RAM available for dynamic allocation:
I (375) heap_init: At 3FFAE6E0 len 00001920 (6 KiB): DRAM
I (381) heap_init: At 3FFC0748 len 0001F8B8 (126 KiB): DRAM
I (387) heap_init: At 3FFE0440 len 00003BC0 (14 KiB): D/IRAM
I (393) heap_init: At 3FFE4350 len 0001BCB0 (111 KiB): D/IRAM
I (400) heap_init: At 4008DD9C len 00012264 (72 KiB): IRAM
I (406) cpu_start: Pro cpu start user code
I (88) cpu_start: Starting scheduler on PRO CPU.
I (113) wifi: wifi firmware version: f79168c
I (113) wifi: config NVS flash: enabled
I (113) wifi: config nano formating: disabled
I (113) system_api: Base MAC address is not set, read default base MAC address from
 BLK0 of EFUSE
I (123) system_api: Base MAC address is not set, read default base MAC address from
 BLK0 of EFUSE
I (133) wifi: Init dynamic tx buffer num: 32
I (143) wifi: Init data frame dynamic rx buffer num: 32
I (143) wifi: Init management frame dynamic rx buffer num: 32
I (143) wifi: wifi driver task: 3ffc73ec, prio:23, stack:4096
I (153) wifi: Init static rx buffer num: 10
I (153) wifi: Init dynamic rx buffer num: 32
I (163) wifi: wifi power manager task: 0x3ffcc028 prio: 21 stack: 2560
0 6 [main] WiFi module initialized. Connecting to AP <Your_WiFi_SSID>...
I (233) phy: phy_version: 383.0, 79a622c, Jan 30 2018, 15:38:06, 0, 0
I (233) wifi: mode : sta (30:ae:a4:80:0a:04)
I (233) WIFI: SYSTEM_EVENT_STA_START
I (363) wifi: n:1 0, o:1 0, ap:255 255, sta:1 0, prof:1
I (1343) wifi: state: init -> auth (b0)
I (1343) wifi: state: auth -> assoc (0)
I (1353) wifi: state: assoc -> run (10)
I (1373) wifi: connected with <Your_WiFi_SSID>, channel 1
I (1373) WIFI: SYSTEM_EVENT_STA_CONNECTED
1 302 [IP-task] vDHCPProcess: offer c0a86c13ip
I (3123) event: sta ip: 192.168.108.19, mask: 255.255.224.0, gw: 192.168.96.1
I (3123) WIFI: SYSTEM_EVENT_STA_GOT_IP
2 302 [IP-task] vDHCPProcess: offer c0a86c13ip
3 303 [main] WiFi Connected to AP. Creating tasks which use network...
4 304 [OTA] OTA demo version 0.9.6
5 304 [OTA] Creating MQTT Client...
```

```
6 304 [OTA] Connecting to broker...
I (4353) wifi: pm start, type:0

I (8173) PKCS11: Initializing SPIFFS
I (8183) PKCS11: Partition size: total: 52961, used: 0
7 1277 [OTA] Connected to broker.
8 1280 [OTA Task] [prvSubscribeToJobNotificationTopics] OK: $aws/things/
<Your_Thing_Name>/jobs/$next/get/accepted
I (12963) ota_pal: prvPAL_GetPlatformImageState
I (12963) esp_ota_ops: [0] aflags/seq:0x2/0x1, pflags/seq:0xffffffff/0x0
9 1285 [OTA Task] [prvSubscribeToJobNotificationTopics] OK: $aws/things/
<Your_Thing_Name>/jobs/notify-next
10 1286 [OTA Task] [OTA_CheckForUpdate] Request #0
11 1289 [OTA Task] [prvParseJSONbyModel] Extracted parameter [ clientToken:
 0:<Your_Thing_Name> ]
12 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: execution
13 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: jobId
14 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: jobDocument
15 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: afr_ota
16 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: streamname
17 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: files
18 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: filepath
19 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: filesize
20 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: fileid
21 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: certfile
22 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: sig-sha256-ecdsa
23 1289 [OTA Task] [prvParseJobDoc] Ignoring job without ID.
24 1289 [OTA Task] [prvOTA_Close] Context->0x3ffbb4a8
25 1290 [OTA] [OTA_AgentInit] Ready.
26 1390 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
27 1490 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
28 1590 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
29 1690 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
[ ... ]
```

4. ESP32 董事会现在正在监听 OTA 更新。ESP-IDF 监视器通过 `make flash monitor` 命令启动。您可以按 Ctrl+] 以退出。您也可以使用最中意的 TTY 终端程序（例如，PuTTY、Tera Term 或 GNU Screen）来侦听主板的串行输出。请注意，连接到主板的串行端口可能会导致主板重新启动。

在 Nordic n RF5284 0 DK 上安装固件的初始版本

> **Important**
>
> 该参考集成托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

本指南假设您已经执行了Nordic n RF5284 0-DK 入门和Over-the-Air 更新先决条件中的步骤。在尝试 OTA 更新之前，可能需要运行 FreeRTOS 入门中所述的 MQTT 演示项目，以确保主板和工具链设置正确。

将初始出厂映像刷写到主板

1. 打开 *freertos*/vendors/nordic/boards/nrf52840-dk/aws_demos/config_files/aws_demo_config.h。
2. 将 `#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED` 替换为 `CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` 或 `CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`。
3. 使用所选的 OTA 更新演示，按照 Nordic n RF5284 0-DK 入门中概述的相同步骤，构建并刷写映像。

   您应该可以看到类似于如下所示的输出内容。

```
9 1285 [OTA Task] [prvSubscribeToJobNotificationTopics] OK: $aws/things/your-thing-name/jobs/notify-next
10 1286 [OTA Task] [OTA_CheckForUpdate] Request #0
11 1289 [OTA Task] [prvParseJSONbyModel] Extracted parameter [ clientToken: 0:your-thing-name ]
12 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: execution
13 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: jobId
14 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: jobDocument
15 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: afr_ota
16 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: streamname
17 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: files
18 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: filepath
19 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: filesize
20 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: fileid
21 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: certfile
```

```
22 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: sig-sha256-ecdsa
23 1289 [OTA Task] [prvParseJobDoc] Ignoring job without ID.
24 1289 [OTA Task] [prvOTA_Close] Context->0x3ffbb4a8
25 1290 [OTA] [OTA_AgentInit] Ready.
26 1390 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
27 1490 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
28 1590 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
29 1690 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
```

您的主板此时正在侦听 OTA 更新。

Windows 模拟器上的初始固件

在使用 Windows 模拟器时，无需刷写初始版本的固件。Windows 模拟器是 aws_demos 应用程序的一部分，其中也包含了固件。

在自定义主板上安装初始版本的固件

使用 IDE 构建 aws_demos 项目，确保包括了 OTA 库。有关 FreeRTOS 源代码结构的更多信息，请参阅 FreeRTOS 演示。

请确保 FreeRTOS 项目或设备中包括了代码签名证书、私有密钥和证书信任链。

使用适当的工具将应用程序刻录到主板上，并确保其正常运行。

更新固件版本

FreeRTOS 包含的 OTA 代理会检查任何更新版本，仅当该版本高于现有固件版本时才会进行安装。以下步骤展示了如何递增 OTA 演示应用程序的固件版本。

1. 在 IDE 中打开 aws_demos 项目。
2. 找到文件 /vendors/*vendor*/boards/*board*/aws_demos/config_files/ota_demo_config.h 并增加 APP_VERSION_BUILD 的值。
3. 要安排对文件类型不是 0（非固件文件）的 Renesas rx65n 平台进行更新，必须使用 Renesas Secure Flash Progradmer 工具对文件进行签名，否则设备上的签名检查将失败。该工具创建带有扩展名 .rsu 的签名文件包，该文件包是 Renesas 的专有文件类型。该工具可以在 Github 上找到。您可以使用以下示例命令生成映像：

```
"Renesas Secure Flash Programmer.exe" CUI Update "RX65N(ROM 2MB)/Secure
 Bootloader=256KB" "sig-sha256-ecdsa" 1 "file_name" "output_file_name.rsu"
```

4. 重新构建项目。

必须将固件更新复制到您创建的 Amazon S3 存储桶中，如创建 Amazon S3 存储桶以存储更新中所述。需要复制到 Amazon S3 的文件名称取决于所使用的硬件平台：

- 德州仪器 CC322 0SF-LAUNCHXL：`vendors/ti/boards/cc3220_launchpad/aws_demos/ccs/debug/aws_demos.bin`
- Espressif ESP32: `vendors/espressif/boards/esp32/aws_demos/make/build/aws_demos.bin`

创建 OTA 更新（AWS IoT 控制台）

1. 在 AWS IoT 控制台的导航窗格中，在 “管理” 下选择 “远程操作”，然后选择 “作业”。
2. 请选择创建任务。
3. 在作业类型下，选择创建 FreeRTOS OTA 更新作业，然后选择下一步。
4. 在作业属性中，输入作业名称和（可选）作业的描述，然后选择下一步。
5. 可以将一个 OTA 更新部署到单个设备或一组设备。在要更新的设备下，从下拉列表中选择一个或多个事物或事物组。
6. 在选择文件传输协议下，选择 HTTP 或 MQTT，或者同时选择两者以允许每台设备确定要使用的协议。
7. 在签名并选择文件下，选择为我签署一个新文件。
8. 在代码签名配置文件下，选择创建新配置文件。
9. 在 Create a code signing profile (创建代码签名配置文件) 中，为代码签名配置文件输入名称。
   a. 在 Device hardware platform (设备硬件平台) 下，选择硬件平台。

> **Note**
>
> 该列表中只显示了符合 FreeRTOS 条件的硬件平台。如果您正在测试不符合条件的平台，并且使用 ECDSA P-256 SHA-256 密码套件进行签名，则可以选择 Windows 模拟器代码签名配置文件以生成兼容签名。如果您使用的是不合格的平台，并且使用的是 ECDSA P-256 SHA-256 以外的密码套件进行签名，则可以使用代码签名 AWS IoT，也可以自己签署固件更新。有关更多信息，请参阅 对固件更新进行数字签名。

b. 在代码签名证书下，选择现有证书，然后选择以前导入的证书，或者选择导入新的代码签名证书，选择您的文件，然后选择导入，以便导入新证书。

c. 在 Pathname of code signing certificate on device (设备上代码签名证书的路径名) 下，输入代码签名证书在设备上的完全限定路径名称。对于大多数设备，您可以将此字段留空。对于 Windows 模拟器以及将证书放在特定文件位置的设备，请在此处输入路径名。

> **Important**
>
> 在德州仪器 CC322 0SF-LAUNCHXL 上，如果您的代码签名证书存在于文件系统的根目录中，则不要在文件名前面加上前导斜杠 (**/**)。否则，OTA 更新将在身份验证过程中失败，且产生 `file not found` 错误。

d. 选择创建。

10. 在文件下，选择现有文件，然后选择浏览 S3。此时将显示 Amazon S3 存储桶的列表。选择包含固件更新的存储桶，然后在存储桶中选择固件更新。

> **Note**
>
> Microchip Curi PIC32 osity MZEF 演示项目生成两张默认名称为和的`mplab.production.bin`二进制图像。`mplab.production.ota.bin`如果要上传映像用于 OTA 更新，请使用第二个文件。

11. 在设备上文件的路径名下，输入设备上 OTA 作业要将固件映像复制到的位置的完全限定路径名称。此位置因平台而异。

> **Important**
>
> 在德州仪器 CC322 0SF-LAUNCHXL 上，由于安全限制，固件镜像路径名必须为。`/sys/mcuflashimg.bin`

12. 在文件类型下，输入一个介于 0-255 范围内的整数值。您输入的文件类型将添加到传送到 MCU 的 Job 文档中。MCU 固件/软件开发人员对如何使用此值拥有完全所有权。可能的场景包括带有辅助处理器的 MCU，其固件可以独立于主处理器进行更新。当设备收到 OTA 更新任务时，它可以使用文件类型来识别更新适用于哪个处理器。
13. 在 IAM 角色下，根据创建 OTA 更新服务角色中的说明选择一个角色。
14. 选择下一步。
15. 为 OTA 更新作业输入 ID 和描述。

16. 在 Job type (作业类型) 下，选择 Your job will complete after deploying to the selected devices/groups (snapshot) (您的作业将在部署到所选设备/组后完成 (快照))。
17. 为作业选择任何适当的可选配置（作业执行推出、作业中止、作业执行超时和标签）。
18. 选择创建。

选择以前签署的固件映像

1. 在 Select and sign your firmware image (选择并签署固件映像) 下，选择 Select a previously signed firmware image (选择以前签署的固件映像)。
2. 在 Pathname of firmware image on device (设备上固件映像的路径名) 下，输入设备上 OTA 作业要将固件映像复制到的位置的完全限定路径名称。此位置因平台而异。
3. 在 Previous code signing job (原有的代码签名作业) 下，选择 Select (选择)，然后选择用来签署用于 OTA 更新的固件映像的上一个代码签名作业。

使用自定义的已签署固件映像

1. 在 Select and sign your firmware image (选择并签署固件映像) 下，选择 Use my custom signed firmware image (使用我的自定义已签署固件映像)。
2. 在 Pathname of code signing certificate on device (设备上代码签名证书的路径名) 下，输入代码签名证书在设备上的完全限定路径名称。对于大多数设备，您可以将此字段留空。对于 Windows 模拟器以及将证书放在特定文件位置的设备，请在此处输入路径名。
3. 在 Pathname of firmware image on device (设备上固件映像的路径名) 下，输入设备上 OTA 作业要将固件映像复制到的位置的完全限定路径名称。此位置因平台而异。
4. 在 Signature (签名) 下，粘贴 PEM 格式的签名。
5. 在 Original hash algorithm (原始哈希算法) 下，选择在创建文件签名时所使用的哈希算法。
6. 在 Original encryption algorithm (原始加密算法) 下，选择在创建文件签名时所使用的算法。
7. 在在 Amazon S3 中选择固件映像下，选择 Amazon S3 存储桶以及 Amazon S3 存储桶中已签署的固件映像。

指定代码签名信息后，为更新指定 OTA 更新作业类型、服务角色和 ID。

> **Note**
>
> 请勿在 OTA 更新的作业 ID 中使用任何个人身份信息。个人身份信息示例包括：

- 名称。
- IP 地址。
- 电子邮件地址。
- 位置。
- 银行详细信息。
- 医疗信息。

1. 在 Job type (作业类型) 下，选择 Your job will complete after deploying to the selected devices/groups (snapshot) (您的作业将在部署到所选设备/组后完成 (快照))。
2. 在 IAM role for OTA update job (OTA 更新作业的 IAM 角色) 下，选择 OTA 服务角色。
3. 为作业输入字母数字形式的 ID，然后选择 Create (创建)。

任务显示在 AWS IoT 控制台中，状态为 “进行中”。

Note

- AWS IoT 控制台不会自动更新作业的状态。刷新浏览器可以查看更新。

将串行 UART 终端连接到设备。您应当看到输出表明设备正在下载更新后的固件。

设备在下载完更新后的固件之后，将重新启动，然后安装固件。在 UART 终端中可以看到所发生的情况。

有关说明如何使用控制台创建 OTA 更新的教程，请参阅Over-the-air 更新演示应用程序。

使用创建 OTA 更新 AWS CLI

使用创建 OTA 更新时，您： AWS CLI

1. 对固件映像进行数字签名。
2. 创建经数字签名的固件映像的流。
3. 启动 OTA 更新作业。

对固件更新进行数字签名

当你使用执行 AWS CLI OTA 更新时，你可以使用代码签名 AWS IoT，也可以自己签署固件更新。有关代码签名支持的加密签名和哈希算法的列表 AWS IoT，请参阅。 SigningConfigurationOverrides如果您想使用代码签名不支持的加密算法，则必须先对固件二进制文件进行签名 AWS IoT，然后再将其上传到 Amazon S3。

使用代码签名对固件映像进行签名 AWS IoT

要使用代码签名对固件映像进行签名 AWS IoT，可以使用AWS SDKs 或命令行工具之一。有关代码签名的更多信息 AWS IoT，请参阅的代码签名 AWS IoT。

安装和配置代码签名工具后，将未签名的固件映像复制到 Amazon S3 存储桶，然后使用以下命令开始代码签名任务。 AWS CLI put-signing-profile 命令用于创建一个可重复使用的代码签名配置文件。start-signing-job 命令用于启动签名作业。

```
aws signer put-signing-profile \
    --profile-name your_profile_name \
    --signing-material certificateArn=arn:aws:acm::your-region:your-aws-account-
id:certificate/your-certificate-id \
    --platform your-hardware-platform \
    --signing-parameters certname=your_certificate_path_on_device
```

```
aws signer start-signing-job \
    --source
 's3={bucketName=your_s3_bucket,key=your_s3_object_key,version=your_s3_object_version_id}'
 \
    --destination 's3={bucketName=your_destination_bucket}' \
    --profile-name your_profile_name
```

> **Note**
>
> *your-source-bucket-name*并且*your-destination-bucket-name*可以是相同的 Amazon S3 存储桶。

以下是 put-signing-profile 和 start-signing-job 命令的参数：

`source`

指定未签名的固件在 S3 存储桶中的位置。

- `bucketName`：S3 存储桶的名称。
- `key`：固件在 S3 存储桶中的密钥（文件名）。
- `version`：固件在 S3 存储桶中的 S3 版本。该版本不同于固件版本。要找到此版本，请转至 Amazon S3 控制台，选择存储桶，然后在页面顶部的版本旁选择显示。

`destination`

S3 存储桶中已签名的固件将被复制到的设备上的目的地。该参数的格式与 `source` 参数是一样的。

`signing-material`

代码签名证书的 ARN。在将证书导入 ACM 中时，生成此 ARN。

`signing-parameters`

签名的键值对映射。其中可以包括签名过程中要使用的任何信息。

> **Note**
>
> 在创建代码签名配置文件以使用 Code Signing for AWS IoT对 OTA 更新进行签名时，需要使用该参数。

`platform`

要将 OTA 更新分配到的硬件平台的 `platformId`。

要返回可用的平台及其 `platformId` 值的列表，请使用 `aws signer list-signing-platforms` 命令。

签名作业启动，将已签名的固件映像写入目标 Amazon S3 存储桶。已签名的固件映像的文件名是 GUID。创建流时将需要此文件名。浏览至 Amazon S3 控制台并选择存储桶即可找到此文件名。如果没有看到带有 GUID 文件名的文件，可刷新浏览器。

此命令将显示作业 ARN 和作业 ID。稍后会需要这些值。有关代码签名的更多信息 AWS IoT，请参阅的代码签名 AWS IoT。

手动签署固件映像

对固件映像进行数字签名，并将已签名的固件映像上传到 Amazon S3 存储桶。

创建固件更新流

流是设备可以使用的数据的抽象接口。流可以隐藏访问存储在不同位置或各种基于云的服务中的数据的复杂性。利用 OTA Update Manager 服务，您可以使用存储在 Amazon S3 中的各个位置的多个数据段来执行 OTA 更新。

创建 O AWS IoT TA 更新时，您还可以创建包含已签名固件更新的流。创建一个 JSON 文件 (`stream.json`)，该文件标识已签名的固件映像。JSON 文件应包含以下内容。

```
[
  {
    "fileId":"your_file_id",
    "s3Location":{
      "bucket":"your_bucket_name",
      "key":"your_s3_object_key"
    }
  }
]
```

以下是 JSON 文件中的属性：

`fileId`

介于 0–255 之间的任意整数，用于标识固件映像。

`s3Location`

固件到流的存储桶和密钥。

`bucket`

存储未签名的固件映像的 Amazon S3 存储桶。

`key`

已签名的固件映像在 Amazon S3 存储桶中的文件名。可以在 Amazon S3 控制台中，通过查看存储桶的内容来找到该值。

如果您使用代码签名 AWS IoT，则文件名是代码签名为 AWS IoT生成的 GUID。

使用 create-stream AWS CLI 命令创建流。

```
aws iot create-stream \
```

```
    --stream-id your_stream_id \
    --description your_description \
    --files file://stream.json \
    --role-arn your_role_arn
```

以下是该create-stream AWS CLI 命令的参数：

stream-id

任意字符串，用于标识流。

description

流的描述（可选）。

files

JSON 文件的一个或多个引用，包含有关固件映像到流的数据。JSON 文件必须包含以下属性：

fileId

任意文件 ID。

s3Location

存储已签名的固件映像的存储桶名称，以及已签名的固件映像的密钥（文件名）。

bucket

存储已签名的固件映像的 Amazon S3 存储桶。

key

已签名的固件映像的密钥（文件名）。

当您使用代码签名时 AWS IoT，此密钥是一个 GUID。

下面是一个 `stream.json` 示例文件。

```
[
    {
        "fileId":123,
        "s3Location": {
            "bucket":"codesign-ota-bucket",
            "key":"48c67f3c-63bb-4f92-a98a-4ee0fbc2bef6"
        }
    }
```

```
]
```

role-arn

OTA 服务角色还授予对存储固件映像的 Amazon S3 存储桶的访问权限。

要查找已签名固件映像的 Amazon S3 对象密钥，请使用aws signer describe-signing-job --job-id ***my-job-id***命令显示的任务 ID 在`my-job-id`哪里。create-signing-job AWS CLI describe-signing-job 命令的输出中包含已签名的固件映像的密钥。

```
... text deleted for brevity ...
  "signedObject": {
    "s3": {
      "bucketName": "ota-bucket",
      "key": "7309da2c-9111-48ac-8ee4-5a4262af4429"
    }
  }
... text deleted for brevity ...
```

创建 OTA 更新

使用create-ota-update AWS CLI 命令创建 OTA 更新任务。

> **Note**
>
> 请勿在 OTA 更新作业 ID 中使用任何个人身份信息 (PII)。个人身份信息示例包括：
>
> - 名称。
> - IP 地址。
> - 电子邮件地址。
> - 位置。
> - 银行详细信息。
> - 医疗信息。

```
aws iot  create-ota-update \
    --ota-update-id value \
    [--description value] \
```

```
    --targets value \
    [--protocols value] \
    [--target-selection value] \
    [--aws-job-executions-rollout-config value] \
    [--aws-job-presigned-url-config value] \
    [--aws-job-abort-config value] \
    [--aws-job-timeout-config value] \
    --files value \
    --role-arn value \
    [--additional-parameters value] \
    [--tags value]  \
    [--cli-input-json value] \
    [--generate-cli-skeleton]
```

cli-input-json 格式

```
{
  "otaUpdateId": "string",
  "description": "string",
  "targets": [
    "string"
  ],
  "protocols": [
    "string"
  ],
  "targetSelection": "string",
  "awsJobExecutionsRolloutConfig": {
    "maximumPerMinute": "integer",
    "exponentialRate": {
      "baseRatePerMinute": "integer",
      "incrementFactor": "double",
      "rateIncreaseCriteria": {
        "numberOfNotifiedThings": "integer",
        "numberOfSucceededThings": "integer"
      }
    }
  },
  "awsJobPresignedUrlConfig": {
    "expiresInSec": "long"
  },
  "awsJobAbortConfig": {
    "abortCriteriaList": [
      {
```

```
        "failureType": "string",
        "action": "string",
        "thresholdPercentage": "double",
        "minNumberOfExecutedThings": "integer"
      }
    ]
  },
  "awsJobTimeoutConfig": {
    "inProgressTimeoutInMinutes": "long"
  },
  "files": [
    {
      "fileName": "string",
      "fileType": "integer",
      "fileVersion": "string",
      "fileLocation": {
        "stream": {
          "streamId": "string",
          "fileId": "integer"
        },
        "s3Location": {
          "bucket": "string",
          "key": "string",
          "version": "string"
        }
      },
      "codeSigning": {
        "awsSignerJobId": "string",
        "startSigningJobParameter": {
          "signingProfileParameter": {
            "certificateArn": "string",
            "platform": "string",
            "certificatePathOnDevice": "string"
          },
          "signingProfileName": "string",
          "destination": {
            "s3Destination": {
              "bucket": "string",
              "prefix": "string"
            }
          }
        },
        "customCodeSigning": {
          "signature": {
```

```
            "inlineDocument": "blob"
          },
          "certificateChain": {
            "certificateName": "string",
            "inlineDocument": "string"
          },
          "hashAlgorithm": "string",
          "signatureAlgorithm": "string"
        }
      },
      "attributes": {
        "string": "string"
      }
    }
  ],
  "roleArn": "string",
  "additionalParameters": {
    "string": "string"
  },
  "tags": [
    {
      "Key": "string",
      "Value": "string"
    }
  ]
}
```

cli-input-json 字段

名称	类型	描述
otaUpdateId	字符串 (max:128 min:1)	要创建的 OTA 更新的 ID。
description	字符串 (max:2028)	OTA 更新的描述。
targets	列表	接收 OTA 更新的目标设备。
protocols	列表	用于传输 OTA 更新映像的协议。有效值为 [HTTP]、[M

名称	类型	描述
		QTT]、[HTTP, MQTT]。当同时指定 HTTP 和 MQTT 时，目标设备可以选择协议。
targetSelection	字符串	指定更新将继续运行 (CONTINUOUS)，还是在指定作为目标的所有事物完成更新之后完成 (SNAPSHOT)。如果继续运行，则在检测到目标中出现更改时，更新也会在事物上运行。例如，当某个事物添加到目标组时会在该事物上运行更新，即使是组中原有的全部事物已经完成了更新。有效值：CONTINUOUS | SNAPSHOT。 枚举：CONTINUOUS | SNAPSHOT
awsJobExecutionsRolloutConfig		配置 OTA 更新的推广。
maximumPerMinute	整数 (max:1000 min:1)	每分钟启动的最大 OTA 更新任务执行次数。
exponentialRate		任务推出的增速。此参数允许您定义作业推出的指数增长速率。
baseRatePerMinute	整数 (max:1000 min:1)	在作业推出开始时，每分钟接收待处理作业通知的事物的最小数量。这是作业推出的初始速率。

名称	类型	描述
rateIncreaseCriteria		启动任务推出速率提高的条件。 AWS IoT 支持小数点后最多一位数字（例如，1.5，但不是 1.55）。
numberOfNotifiedThings	整数 (min:1)	当通知此事物数后，推出速率将提高。
numberOfSucceededThings	整数 (min:1)	当此事物数在作业执行中成功应用后，推出速率将提高。
awsJobPresignedUrlConfig		预签名的 URLs配置信息。
expiresInSec	长整数	预签名的有效时长（以秒 URLs 为单位）。有效值为 60 – 3600，默认值为 1800 秒。收到工作文档的请求时，会生成预签名 URLs 。
awsJobAbortConfig		确定作业停止发生时间和方式的条件。
abortCriteriaList	列表	确定何时以及如何停止作业的条件列表。
failureType	字符串	可以启动作业停止的作业执行失败类型。 枚举：FAILED | REJECTED | TIMED_OUT | ALL

名称	类型	描述
action	字符串	用于启动作业停止的作业操作类型。 枚举：CANCEL
minNumberOfExecutedThings	整数 (min:1)	作业可以停止之前必须接收作业执行通知的事物的最小数量。
awsJobTimeoutConfig		指定每个设备完成其任务执行所具有的时间。计时器在任务执行状态设置为 IN_PROGRESS 时启动。如果任务执行状态未在时间到期之前设置为其他最终状态，它会自动设置为 TIMED_OUT 。
inProgressTimeoutInMinutes	长整数	指定此设备完成该任务执行所具有的时间，以分钟为单位。超时间隔可以为 1 分钟到 7 天（1 到 10080 分钟）之间的任意长度。进行中计时器无法更新，将应用到该任务的全部任务执行。只要任务执行保持在 IN_PROGRESS 状态的时间长度超过了此间隔，任务执行将失败，并切换为最终 TIMED_OUT 状态。
files	列表	应由 OTA 更新流式处理的文件。
fileName	字符串	文件的名称。

名称	类型	描述
fileType	整数 范围 – 最大值：255，最小值：0	您可以在作业文档中包含一个整数值，以允许您的设备识别从云接收的文件类型。
fileVersion	字符串	文件版本。
fileLocation		更新后固件的位置。
stream		包含 OTA 的流。
streamId	字符串 (max:128 min:1)	流 ID。
fileId	整数 (max:255 min:0)	与流关联的文件的 ID。
s3Location		更新后固件在 S3 中的位置。
bucket	字符串 (min:1)	S3 存储桶。
key	字符串 (min:1)	S3 键。
version	字符串	S3 存储桶版本。
codeSigning		文件的代码签名方法。
awsSignerJobId	字符串	为签署文件而创建的 AWSSigner Job 的 ID。
startSigningJobParameter		描述代码签名任务。

名称	类型	描述
signingProfilePara meter		描述代码签名配置文件。
certificateArn	字符串	证书 ARN。
platform	字符串	您设备的硬件平台。
certificatePathOnD evice	字符串	代码签名证书在设备上的位置。
signingProfileName	字符串	代码签名配置文件名称。
destination		编写代码签名文件的位置。
s3Destination		描述更新后固件在 S3 中的位置。
bucket	字符串 (min:1)	包含更新后固件的 S3 存储桶。
prefix	字符串	S3 前缀。
customCodeSigning		用于对文件执行代码签名的自定义方法。
signature		文件的签名。
inlineDocument	blob	代码签署签名的 base64 编码二进制表示形式。
certificateChain		证书链。
certificateName	字符串	证书的名称。
inlineDocument	字符串	代码签名证书链的 base64 编码二进制表示形式。

名称	类型	描述
hashAlgorithm	字符串	用于对文件进行代码签名的哈希算法。
signatureAlgorithm	字符串	用于对文件进行代码签名的签名算法。
attributes	映射	名称/属性对的列表。
roleArn	字符串 (max:2048 min:20)	一个 IAM 角色，用于授予 AWS IoT 访问 Amazon S3、AWS IoT 任务和 AWS 代码签名资源的访问权限以创建 OTA 更新任务。
additionalParameters	映射	其他 OTA 更新参数 (名称/值对) 的列表。
tags	列表	可用于管理更新的元数据。
Key	字符串 (max:128 min:1)	标签的键。
Value	字符串 (max:256 min:1)	标签的值。

输出

```
{
  "otaUpdateId": "string",
  "awsIotJobId": "string",
  "otaUpdateArn": "string",
  "awsIotJobArn": "string",
  "otaUpdateStatus": "string"
}
```

AWS CLI 输出字段

名称	类型	描述
otaUpdateId	字符串 (max:128 min:1)	OTA 更新 ID。
awsIotJobId	字符串	与 OTA 更新关联的 AWS IoT 任务 ID。
otaUpdateArn	字符串	OTA 更新 ARN。
awsIotJobArn	字符串	与 OTA 更新关联的 AWS IoT 任务 ARN。
otaUpdateStatus	字符串	OTA 更新状态。 枚举：CREATE_PENDING | CREATE_IN_PROGRESS | CREATE_COMPLETE | CREATE_FAILED

以下是传递到使用代码签名的create-ota-update命令中的 JSON 文件的示例 AWS IoT。

```
[
  {
    "fileName": "firmware.bin",
    "fileType": 1,
    "fileLocation": {
      "stream": {
        "streamId": "004",
        "fileId":123
      }
    },
    "codeSigning": {
      "awsSignerJobId": "48c67f3c-63bb-4f92-a98a-4ee0fbc2bef6"
    }
  }
]
```

以下是传递到create-ota-update AWS CLI 命令中的 JSON 文件的示例，该文件使用内联文件提供自定义代码签名材料。

```
[
  {
    "fileName": "firmware.bin",
    "fileType": 1,
    "fileLocation": {
      "stream": {
        "streamId": "004",
        "fileId": 123
      }
    },
    "codeSigning": {
      "customCodeSigning":{
        "signature":{
          "inlineDocument":"your_signature"
        },
        "certificateChain": {
          "certificateName": "your_certificate_name",
          "inlineDocument":"your_certificate_chain"
        },
        "hashAlgorithm":"your_hash_algorithm",
        "signatureAlgorithm":"your_signature_algorithm"
      }
    }
  }
]
```

以下是传递到create-ota-update AWS CLI 命令中的 JSON 文件的示例，该文件允许 FreeRTOS OTA 启动代码签名任务并创建代码签名配置文件和直播。

```
[
  {
    "fileName": "your_firmware_path_on_device",
    "fileType": 1,
    "fileVersion": "1",
    "fileLocation": {
      "s3Location": {
        "bucket": "your_bucket_name",
        "key": "your_object_key",
        "version": "your_S3_object_version"
      }
```

```
    },
    "codeSigning":{
      "startSigningJobParameter":{
        "signingProfileName": "myTestProfile",
        "signingProfileParameter": {
          "certificateArn": "your_certificate_arn",
          "platform": "your_platform_id",
          "certificatePathOnDevice": "certificate_path"
        },
        "destination": {
          "s3Destination": {
            "bucket": "your_destination_bucket"
          }
        }
      }
    }
  }
]
```

以下是传递到create-ota-update AWS CLI 命令中的 JSON 文件的示例，该文件用于创建 OTA 更新，该更新使用现有配置文件启动代码签名任务并使用指定的流。

```
[
  {
    "fileName": "your_firmware_path_on_device",
    "fileType": 1,
    "fileVersion": "1",
    "fileLocation": {
      "s3Location": {
        "bucket": "your_s3_bucket_name",
        "key": "your_object_key",
        "version": "your_S3_object_version"
      }
    },
    "codeSigning":{
      "startSigningJobParameter":{
        "signingProfileName": "your_unique_profile_name",
        "destination": {
          "s3Destination": {
            "bucket": "your_destination_bucket"
          }
        }
      }
```

```
    }
  }
]
```

以下是传递到create-ota-update AWS CLI 命令中的 JSON 文件的示例，该文件允许 FreeRTOS OTA 使用现有代码签名任务 ID 创建流。

```
[
  {
    "fileName": "your_firmware_path_on_device",
    "fileType": 1,
    "fileVersion": "1",
    "codeSigning":{
      "awsSignerJobId": "your_signer_job_id"
    }
  }
]
```

以下是传递到创建 OTA 更新的create-ota-update AWS CLI 命令中的 JSON 文件的示例。该更新根据指定的 S3 对象创建一个流，并使用自定义代码签名。

```
[
  {
    "fileName": "your_firmware_path_on_device",
    "fileType": 1,
    "fileVersion": "1",
    "fileLocation": {
      "s3Location": {
        "bucket": "your_bucket_name",
        "key": "your_object_key",
        "version": "your_S3_object_version"
      }
    },
    "codeSigning":{
      "customCodeSigning": {
        "signature":{
          "inlineDocument":"your_signature"
        },
        "certificateChain": {
          "inlineDocument":"your_certificate_chain",
          "certificateName": "your_certificate_path_on_device"
        },
```

```
        "hashAlgorithm":"your_hash_algorithm",
        "signatureAlgorithm":"your_sig_algorithm"
      }
    }
  }
]
```

列出 OTA 更新

您可以使用list-ota-updates AWS CLI 命令获取所有 OTA 更新的列表。

```
aws iot list-ota-updates
```

list-ota-updates 命令输出如下所示。

```
{
  "otaUpdates": [

    {
      "otaUpdateId": "my_ota_update2",
      "otaUpdateArn": "arn:aws:iot:us-west-2:123456789012:otaupdate/my_ota_update2",
      "creationDate": 1522778769.042
    },
    {
      "otaUpdateId": "my_ota_update1",
      "otaUpdateArn": "arn:aws:iot:us-west-2:123456789012:otaupdate/my_ota_update1",
      "creationDate": 1522775938.956
    },
    {
      "otaUpdateId": "my_ota_update",
      "otaUpdateArn": "arn:aws:iot:us-west-2:123456789012:otaupdate/my_ota_update",
      "creationDate": 1522775151.031
    }
  ]
}
```

获取有关 OTA 更新的信息

您可以使用get-ota-update AWS CLI 命令获取 OTA 更新的创建或删除状态。

```
aws iot get-ota-update --ota-update-id your-ota-update-id
```

get-ota-update 命令的输出如下所示。

```
{
    "otaUpdateInfo": {
        "otaUpdateId": "ota-update-001",
        "otaUpdateArn": "arn:aws:iot:region:123456789012:otaupdate/ota-update-001",
        "creationDate": 1575414146.286,
        "lastModifiedDate": 1575414149.091,
        "targets": [
            "arn:aws:iot:region:123456789012:thing/myDevice"
        ],
        "protocols": [ "HTTP" ],
        "awsJobExecutionsRolloutConfig": {
            "maximumPerMinute": 0
        },
        "awsJobPresignedUrlConfig": {
            "expiresInSec": 1800
        },
        "targetSelection": "SNAPSHOT",
        "otaUpdateFiles": [
            {
                "fileName": "my_firmware.bin",
                "fileType": 1,
                "fileLocation": {
                    "s3Location": {
                        "bucket": "my-bucket",
                        "key": "my_firmware.bin",
                        "version": "AvP3bfJC9gyqnwoxPHuTqM5GWENt4iii"
                    }
                },
                "codeSigning": {
                    "awsSignerJobId": "b7a55a54-fae5-4d3a-b589-97ed103737c2",
                    "startSigningJobParameter": {
                        "signingProfileParameter": {},
                        "signingProfileName": "my-profile-name",
                        "destination": {
                            "s3Destination": {
                                "bucket": "some-ota-bucket",
                                "prefix": "SignedImages/"
                            }
                        }
                    },
                    "customCodeSigning": {}
                }
```

```
            }
        ],
        "otaUpdateStatus": "CREATE_COMPLETE",
        "awsIotJobId": "AFR_OTA-ota-update-001",
        "awsIotJobArn": "arn:aws:iot:region:123456789012:job/AFR_OTA-ota-update-001"
    }
}
```

otaUpdateStatus 返回的值包括：

CREATE_PENDING

OTA 更新的创建正在等待处理。

CREATE_IN_PROGRESS

正在创建 OTA 更新。

CREATE_COMPLETE

OTA 更新已创建。

CREATE_FAILED

OTA 更新的创建失败。

DELETE_IN_PROGRESS

正在删除 OTA 更新。

DELETE_FAILED

OTA 更新的删除失败。

> **Note**
>
> 要在创建 OTA 更新后获取其执行状态，您需要使用 describe-job-execution 命令。有关更多信息，请参阅描述作业执行。

删除与 OTA 相关的数据

目前，您无法使用 AWS IoT 控制台删除直播或 OTA 更新。您可以使用删除直播、OTA 更新以及在 OTA 更新期间创建的 AWS IoT 任务。 AWS CLI

删除 OTA 流

在创建使用 MQTT 的 OTA 更新时，您可以使用命令行或 AWS IoT 控制台创建流，将固件分成多个块，以便通过 MQTT 发送。您可以使用delete-stream AWS CLI 命令删除此流，如以下示例所示。

```
aws iot delete-stream --stream-id your_stream_id
```

删除 OTA 更新

在创建 OTA 更新时，将创建以下各项：

- OTA 更新作业数据库中的一个条目。
- 执行更新的 AWS IoT 任务。
- 每台正在更新的设备执行的 AWS IoT 任务。

delete-ota-update 命令仅删除 OTA 更新作业数据库中的条目。必须使用 delete-job 命令来删除 AWS IoT 作业。

可以使用 delete-ota-update 命令删除一个 OTA 更新。

```
aws iot delete-ota-update --ota-update-id your_ota_update_id
```

ota-update-id

要删除的 OTA 更新的 ID。

delete-stream

删除与 OTA 更新关联的流。

force-delete-aws-job

删除与 OTA 更新关联的 AWS IoT 任务。如果未设置此标记且任务处于 `In_Progress` 状态，则作业不会被删除。

删除为 OTA 更新创建的 IoT 作业

当你创建 OTA 更新时，FreeRTOS 会 AWS IoT 创建一个任务。也会为处理作业的每个设备创建作业执行。您可以使用delete-job AWS CLI 命令删除作业及其关联的任务执行。

```
aws iot delete-job --job-id your-job-id --no-force
```

no-force 参数指定只能删除处于结束状态（COMPLETED 或 CANCELLED）的作业。可以通过传递 force 参数来删除处于非结束状态的作业。有关更多信息，请参阅 DeleteJob API。

> **Note**
>
> 如果删除状态为 IN_PROGRESS 的作业，则设备上处于 IN_PROGRESS 状态的任何作业执行都将中断，并可能导致设备处在不确定的状态。请确保已删除的正在执行作业的每个设备都可以恢复到已知状态。

删除作业可能需要几分钟时间，具体取决于为作业创建的作业执行数量以及其他因素。在删除作业的过程中，其状态为 DELETION_IN_PROGRESS。试图删除或取消已处于 DELETION_IN_PROGRESS 状态的作业将导致错误。

可以使用 delete-job-execution 删除作业执行。如果设备数量太少无法处理作业，您可能希望删除作业执行。这会删除单个设备的作业执行，如以下示例中所示。

```
aws iot delete-job-execution --job-id your-job-id --thing-name
                    your-thing-name --execution-number your-job-execution-number --no-
force
```

与delete-job AWS CLI 命令一样，您可以将--force参数传递delete-job-execution给以强制删除任务执行。有关更多信息，请参阅 DeleteJobExecutionAPI。

> **Note**
>
> 如果删除状态为 IN_PROGRESS 的作业执行，则设备上处于 IN_PROGRESS 状态的任何作业执行都将中断，并可能导致设备处在不确定的状态。请确保已删除的正在执行作业的每个设备都可以恢复到已知状态。

有关使用 OTA 更新应用程序的更多信息，请参阅Over-the-air 更新演示应用程序。

OTA Update Manager 服务

over-the-air(OTA) 更新管理器服务提供了一种方法：

- 创建 OTA 更新及其使用的资源，包括 AWS IoT 任务、 AWS IoT 直播和代码签名。
- 获取有关 OTA 更新的信息。

- 列出与您的 AWS 账户关联的所有 OTA 更新。
- 删除 OTA 更新。

OTA 更新是由 OTA Update Manager 服务维护的一种数据结构。其中包含：

- OTA 更新 ID。
- OTA 更新描述（可选）。
- 要更新的设备列表（目标）。
- OTA 更新的类型：CONTINUOUS 或 SNAPSHOT。有关所需的更新类型的讨论，请参阅《AWS IoT 开发人员指南》的作业部分。
- 用于执行 OTA 更新的协议为：[MQTT]、[HTTP] 或 [MQTT 和 HTTP]。在指定 MQTT 和 HTTP 时，设备设置会确定所使用的协议。
- 要发送到目标设备的文件列表。
- 一个 IAM 角色，用于授予 AWS IoT 访问 Amazon S3、 AWS IoT 任务和 AWS 代码签名资源的访问权限以创建 OTA 更新任务。
- 用户定义的名称值对列表（可选）。

OTA 更新旨在更新设备固件，但您可以使用它们将想要的任何文件发送到一台或多台注册的设备 AWS IoT。在以无线方式发送固件更新时，建议您对更新进行数字签名，以便接收更新的设备能够验证更新在传输途中未经篡改。

可以使用 HTTP 或 MQTT 协议发送更新后的固件映像，具体取决于您选择的设置。您可以使用适用于 FreeRTOS 的代码签名来对固件更新进行签名，也可以使用自己的代码签名工具。

为了更好地控制流程，您可以在通过 MQTT 发送更新时使用 CreateStreamAPI 创建流。在某些情况下，您可以修改 FreeRTOS 代理代码来调整发送和接收的块的大小。

在创建 OTA 更新时，OTA Manager 服务会创建一个 AWS IoT 作业来通知设备有可用的更新。FreeRTOS OTA 代理将在设备上运行，并侦听更新消息。当有可用更新时，它会通过 HTTP 或 MQTT 请求固件更新映像，并本地存储这些文件。它将检查所下载文件的数字签名，如果签名有效，则安装固件更新。如果未使用 FreeRTOS，则必须实施自己的 OTA 代理，以侦听和下载更新并执行任何安装操作。

将 OTA 代理集成到应用程序中

over-the-air(OTA) 代理旨在简化为产品添加 OTA 更新功能而必须编写的代码量。集成负担主要包括 OTA 代理的初始化，以及创建自定义回调函数以响应 OTA 代理完成事件消息。在操作系统初始化期

间，MQTT、HTTP（如果文件下载使用 HTTP）和平台特定实现 (PAL) 接口将传递给 OTA 代理。也可以初始化缓冲区并将其传递给 OTA 代理。

> **Note**
>
> 尽管将 OTA 更新功能集成到应用程序中非常简单，但 OTA 更新系统需要了解的不仅仅是设备代码集成。要熟悉如何使用 AWS IoT 内容、凭证、代码签名证书、配置设备和 OTA 更新任务来配置 AWS 账户，请参阅 FreeRTOS 先决条件。

连接管理

OTA 代理使用 MQTT 协议进行所有涉及 AWS IoT 服务的控制通信操作，但它不管理 MQTT 连接。要确保 OTA 代理不会干扰应用程序的连接管理策略，必须由主用户应用程序处理 MQTT 连接（包括断开连接和任何重新连接功能）。可以通过 MQTT 或 HTTP 协议下载该文件。可以在创建 OTA 作业时选择协议。如果选择 MQTT，则 OTA 代理将使用相同的连接来控制操作和下载文件。

简单的 OTA 演示

下面是一个简单的 OTA 演示节选，说明了代理如何连接到 MQTT 代理并初始化 OTA 代理。在该示例中，我们对演示进行配置，使用默认的 OTA 应用程序回调，并每一秒返回一次某些统计数据。为简洁起见，我们省略了演示的某些细节。

OTA 演示还通过监控断开回调连接并重新建立连接来演示 MQTT 连接管理。当断开连接时，此演示会先暂停 OTA 代理操作，然后尝试重新建立 MQTT 连接。MQTT 重新连接尝试会延迟一段时间，该时间以指数方式增加到最大值，并且还会增加抖动。如果重新建立连接，OTA 代理将继续运行。

有关使用 AWS IoT MQTT 代理的工作示例，请参阅`demos/ota`目录中的 OTA 演示代码。

由于 OTA 代理是它自己的任务，该示例中刻意的一秒延迟只会影响本应用程序。对代理的性能不会有任何影响。

```
static BaseType_t prvRunOTADemo( void )
{
    /* Status indicating a successful demo or not. */
    BaseType_t xStatus = pdFAIL;

    /* OTA library return status. */
    OtaErr_t xOtaError = OtaErrUninitialized;

    /* OTA event message used for sending event to OTA Agent.*/
    OtaEventMsg_t xEventMsg = { 0 };
```

```
    /* OTA interface context required for library interface functions.*/
    OtaInterfaces_t xOtaInterfaces;

    /* OTA library packet statistics per job.*/
    OtaAgentStatistics_t xOtaStatistics = { 0 };

    /* OTA Agent state returned from calling OTA_GetState.*/
    OtaState_t xOtaState = OtaAgentStateStopped;

    /* Set OTA Library interfaces.*/
    prvSetOtaInterfaces( &xOtaInterfaces );

    /************************** Init OTA Library. **************************/

    if( ( xOtaError = OTA_Init( &xOtaBuffer,
                                &xOtaInterfaces,
                                ( const uint8_t * ) ( democonfigCLIENT_IDENTIFIER ),
                                prvOtaAppCallback ) ) != OtaErrNone )
    {
        LogError( ( "Failed to initialize OTA Agent, exiting = %u.",
                    xOtaError ) );
    }
    else
    {
        xStatus = pdPASS;
    }

    /************************ Create OTA Agent Task. ************************/

    if( xStatus == pdPASS )
    {
        xStatus = xTaskCreate( prvOTAAgentTask,
                               "OTA Agent Task",
                               otaexampleAGENT_TASK_STACK_SIZE,
                               NULL,
                               otaexampleAGENT_TASK_PRIORITY,
                               NULL );

        if( xStatus != pdPASS )
        {
            LogError( ( "Failed to create OTA agent task:" ) );
        }
    }
```

```
    /****************************** Start OTA ******************************/

    if( xStatus == pdPASS )
    {
        /* Send start event to OTA Agent.*/
        xEventMsg.eventId = OtaAgentEventStart;
        OTA_SignalEvent( &xEventMsg );
    }

    /******************* Loop and display OTA statistics ********************/

    if( xStatus == pdPASS )
    {
        while( ( xOtaState = OTA_GetState() ) != OtaAgentStateStopped )
        {
            /* Get OTA statistics for currently executing job. */
            if( xOtaState != OtaAgentStateSuspended )
            {
                OTA_GetStatistics( &xOtaStatistics );

                LogInfo( ( " Received: %u   Queued: %u   Processed: %u   Dropped: %u",
                           xOtaStatistics.otaPacketsReceived,
                           xOtaStatistics.otaPacketsQueued,
                           xOtaStatistics.otaPacketsProcessed,
                           xOtaStatistics.otaPacketsDropped ) );
            }

            vTaskDelay( pdMS_TO_TICKS( otaexampleEXAMPLE_TASK_DELAY_MS ) );
        }
    }

    return xStatus;
}
```

以下是该演示应用程序的主要流程：

- 创建 MQTT 代理上下文。
- Connect 连接到您的 AWS IoT 终端节点。
- 初始化 OTA 代理。
- 循环执行 OTA 更新作业并每一秒输出一次统计数据。
- 如果 MQTT 断开连接，则暂停 OTA 代理操作。

- 尝试使用指数延迟和抖动再次连接。
- 如果已重新连接，则恢复 OTA 代理操作。
- 如果代理停止，则延迟一秒钟，然后尝试重新连接。

对 OTA 代理事件使用应用程序回调

前面的示例使用 prvOtaAppCallback 作为 OTA 代理事件的回调处理程序。（请参阅 OTA_Init API 调用的第四个参数）。如果要实现对完成事件的自定义处理，则必须在 OTA 演示/应用程序中更改默认处理。在 OTA 过程中，OTA 代理可将以下事件枚举中的任意一个发送给回调处理程序。如何以及何时处理这些事件由应用程序开发人员决定。

```
/**
 * @ingroup ota_enum_types
 * @brief OTA Job callback events.
 *
 * After an OTA update image is received and authenticated, the agent calls the user
 * callback (set with the @ref OTA_Init API) with the value OtaJobEventActivate to
 * signal that the device must be rebooted to activate the new image. When the device
 * boots, if the OTA job status is in self test mode, the agent calls the user callback
 * with the value OtaJobEventStartTest, signaling that any additional self tests
 * should be performed.
 *
 * If the OTA receive fails for any reason, the agent calls the user callback with
 * the value OtaJobEventFail instead to allow the user to log the failure and take
 * any action deemed appropriate by the user code.
 *
 * See the OtaImageState_t type for more information.
 */
typedef enum OtaJobEvent
{
    OtaJobEventActivate = 0,       /*!< @brief OTA receive is authenticated and ready
 to activate. */
    OtaJobEventFail = 1,           /*!< @brief OTA receive failed. Unable to use this
 update. */
    OtaJobEventStartTest = 2,      /*!< @brief OTA job is now in self test, perform
 user tests. */
    OtaJobEventProcessed = 3,      /*!< @brief OTA event queued by OTA_SignalEvent is
 processed. */
    OtaJobEventSelfTestFailed = 4, /*!< @brief OTA self-test failed for current job. */
    OtaJobEventParseCustomJob = 5, /*!< @brief OTA event for parsing custom job
 document. */
```

```
    OtaJobEventReceivedJob = 6,    /*!< @brief OTA event when a new valid AFT-OTA job
 is received. */
    OtaJobEventUpdateComplete = 7, /*!< @brief OTA event when the update is completed.
 */
    OtaLastJobEvent = OtaJobEventStartTest
} OtaJobEvent_t;
```

OTA 代理可以在主应用程序的活动处理期间，在后台接收更新。交付这些事件的目的在于，允许应用程序决定是立即采取行动，还是应当推迟行动，直到其他某些特定于应用程序的处理过程完成。这可以防止设备在活动处理期间（例如，执行 vacuum 操作时），由于固件更新后的重置而导致意外中断。以下是回调处理程序接收的作业事件：

OtaJobEventActivate

如果回调处理程序收到此事件，则可以立即重置设备，或安排调用以稍后重置设备。您可以通过此方法来推迟设备重置和自检阶段（如有必要）。

OtaJobEventFail

如果回调处理程序收到此事件，则更新已失败。在这种情况下不需要执行任何操作。您可能希望输出日志消息或执行某些特定于应用程序的操作。

OtaJobEventStartTest

自检阶段旨在允许最近更新的固件执行并测试自身，然后再确定该固件可以正常使用，并提交为最新的永久应用程序映像。当收到已经过身份验证的新的更新，且设备已重置时，OTA 代理会将 OtaJobEventStartTest 事件发送给已准备好进行测试的回调函数。开发人员可以添加任何必需的测试，以确定设备固件在更新后是否能正常工作。如果通过自检认为设备固件是可靠的，则代码必须调用 OTA_SetImageState(OtaImageStateAccepted) 函数，将该固件提交为新的永久映像。

OtaJobEventProcessed

已处理 OTA_SignalEvent 排队的 OTA 事件，因此可以执行清理操作，例如释放 OTA 缓冲区。

OtaJobEventSelfTestFailed

当前作业的 OTA 自检失败。此事件的默认处理方式是关闭并重新启动 OTA 代理，以便设备回滚到之前的映像。

OtaJobEventUpdateComplete

OTA 任务更新完成的通知事件。

OTA 安全性

以下是 over-the-air (OTA) 安全的三个方面：

连接安全性

OTA Update Manager 服务依赖于 AWS IoT所使用的现有安全机制，例如传输层安全性 (TLS) 双向身份验证。OTA 更新流量通过 AWS IoT 设备网关并使用 AWS IoT 安全机制。每个经由设备网关的传入和传出 HTTP 或 MQTT 消息都要经受严格的身份验证和授权。

OTA 更新的真实性和完整性

在 OTA 更新之前可以对固件进行数字签名，以确保固件来自可靠的来源且未经篡改。

FreeRTOS OTA 更新管理器服务使用代码签名来自动签署 AWS IoT 您的固件。有关更多信息，请参阅 Code Signing for AWS IoT。

将固件安装到设备时，运行于设备上的 OTA 代理将对固件执行完整性检查。

操作人员安全性

通过控制面板 API 发出的每个 API 调用都要经受标准的 IAM 签名版本 4 身份验证和授权。要创建部署，您必须拥有调用`CreateDeployment`、`CreateJob`和的权限`CreateStream` APIs。此外，在您的 Amazon S3 存储桶策略或 ACL 中，您必须向 AWS IoT 服务主体授予读取权限，以便可以在流式传输期间访问存储在 Amazon S3 中的固件更新。

的代码签名 AWS IoT

AWS IoT 控制台使用代码签名 AWS IoT为支持的任何设备自动签署固件映像 AWS IoT。

的代码签名 AWS IoT 使用您导入 ACM 的证书和私钥。您可以使用自签名证书进行测试，但我们建议您从众所周知的商业证书颁发机构 (CA) 获取证书。

代码签名证书使用 X.509 版本 3 `Key Usage` 和 `Extended Key Usage` 扩展。`Key Usage` 扩展设置为 `Digital Signature`，`Extended Key Usage` 扩展设置为 `Code Signing`。有关对代码映像进行签名的更多信息，请参阅《AWS IoT 开发人员代码签名指南》和《适用于 AWS IoT API 的代码签名参考》。

> **Note**
>
> 您可以从 Amazon Web AWS IoT Services 工具中下载软件开发工具包的代码签名。

OTA 故障排查

以下部分中包含的信息可以帮助您排查与 OTA 更新相关的问题。

主题

- 设置 OTA 更新的 CloudWatch 日志
- 使用记录 AWS IoT OTA API 调用 AWS CloudTrail
- 使用获取创建OTAUpdate 失败详情 AWS CLI
- 使用获取 OTA 失败代码 AWS CLI
- 对多个设备的 OTA 更新进行故障排除
- 使用德州仪器 CC322 0SF Launchpad 对 OTA 更新进行故障排除

设置 OTA 更新的 CloudWatch 日志

OTA 更新服务支持使用 Amazon 进行登录 CloudWatch。您可以使用 AWS IoT 控制台为 OTA 更新启用和配置 Amazon CloudWatch 日志记录。有关更多信息，请参阅 Cloudwatch Logs。

要启用日志记录，必须创建 IAM 角色并配置 OTA 更新日志记录。

> **Note**
>
> 在启用 OTA 更新日志记录之前，请确保您了解 CloudWatch 日志访问权限。有权访问 CloudWatch 日志的用户可以看到您的调试信息。有关信息，请参阅 Amazon CloudWatch 日志的身份验证和访问控制。

创建日志记录角色并启用日志记录

可以使用 AWS IoT 控制台创建日志记录角色并启用日志记录。

1. 从导航窗格中，选择设置。
2. 在 Logs (日志) 下，选择 Edit (编辑)。
3. 在 Level of verbosity (详细程度级别) 下，选择 Debug (调试)。
4. 在设置角色下，选择新建，为日志记录创建 IAM 角色。
5. 在 Name (名称) 下，为角色输入唯一名称。将创建具备所有必需权限的角色。

6. 选择更新。

OTA 更新日志

当发生以下任一情况时，OTA 更新服务将日志发布到账户：

- 创建 OTA 更新。
- OTA 更新已完成。
- 创建代码签名作业。
- 代码签名作业已完成。
- AWS IoT 工作已创建。
- AWS IoT 作业已完成。
- 创建流。

可以在 CloudWatch 控制台中查看日志。

在 CloudWatch 日志中查看 OTA 更新

1. 从导航窗格中，选择 Logs（日志）。
2. 在日志组中，选择 AWSIoTLogsV2.

OTA 更新日志可以包含以下属性：

accountId

生成日志的 AWS 账户 ID。

actionType

生成日志的操作。可以设置为以下值之一：

- `CreateOTAUpdate`：已创建 OTA 更新。
- `DeleteOTAUpdate`：已删除 OTA 更新。
- `StartCodeSigning`：已启动代码签名作业。
- `CreateAWSJob`: AWS IoT 就业机会已创建。
- `CreateStream`：已创建流。
- `GetStream`：已向 AWS IoT 基于 MQTT 的文件传输功能发送直播请求。

- `DescribeStream`：已向基于 AWS IoT MQTT 的文件传输功能发送关于流的信息的请求。

awsJobId

生成日志的 AWS IoT 任务 ID。

clientId

发出生成日志请求的 MQTT 客户端 ID。

clientToken

与生成日志请求关联的客户端令牌。

details

有关生成日志的操作的更多信息。

logLevel

日志的日志记录级别。对于 OTA 更新日志，该属性始终设置为 DEBUG。

otaUpdateId

生成日志的 OTA 更新的 ID。

protocol

用于发出生成日志请求的协议。

状态

生成日志操作的状态。有效值为：

- 成功
- Failure

streamId

生成日志的 AWS IoT 直播 ID。

timestamp

日志生成的时间。

topicName

用于发出生成日志请求的 MQTT 主题。

日志示例

以下是启动代码签名作业时生成的日志示例：

```
{
    "timestamp": "2018-07-23 22:59:44.955",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "actionType": "StartCodeSigning",
    "otaUpdateId": "08957b03-eea3-448a-87fe-743e6891ca3a",
    "details": "Start code signing job. The request status is SUCCESS."
}
```

以下是创建 AWS IoT 作业时生成的示例日志：

```
{
    "timestamp": "2018-07-23 22:59:45.363",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "actionType": "CreateAWSJob",
    "otaUpdateId": "08957b03-eea3-448a-87fe-743e6891ca3a",
    "awsJobId": "08957b03-eea3-448a-87fe-743e6891ca3a",
    "details": "Create AWS Job The request status is SUCCESS."
}
```

以下是创建 OTA 更新时生成的日志示例：

```
{
    "timestamp": "2018-07-23 22:59:45.413",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "actionType": "CreateOTAUpdate",
    "otaUpdateId": "08957b03-eea3-448a-87fe-743e6891ca3a",
    "details": "OTAUpdate creation complete. The request status is SUCCESS."
}
```

以下是创建流时生成的日志示例：

```
{
```

```
    "timestamp": "2018-07-23 23:00:26.391",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "actionType": "CreateStream",
    "otaUpdateId": "3d3dc5f7-3d6d-47ac-9252-45821ac7cfb0",
    "streamId": "6be2303d-3637-48f0-ace9-0b87b1b9a824",
    "details": "Create stream. The request status is SUCCESS."
}
```

以下是删除 OTA 更新时生成的日志示例：

```
{
    "timestamp": "2018-07-23 23:03:09.505",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "actionType": "DeleteOTAUpdate",
    "otaUpdateId": "9bdd78fb-f113-4001-9675-1b595982292f",
    "details": "Delete OTA Update. The request status is SUCCESS."
}
```

下面是设备向基于 MQTT 的文件传输功能请求流时生成的示例日志：

```
{
    "timestamp": "2018-07-25 22:09:02.678",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "actionType": "GetStream",
    "protocol": "MQTT",
    "clientId": "b9d2e49c-94fe-4ed1-9b07-286afed7e4c8",
    "topicName": "$aws/things/b9d2e49c-94fe-4ed1-9b07-286afed7e4c8/
streams/1e51e9a8-9a4c-4c50-b005-d38452a956af/get/json",
    "streamId": "1e51e9a8-9a4c-4c50-b005-d38452a956af",
    "details": "The request status is SUCCESS."
}
```

以下是设备调用 `DescribeStream` API 时生成的日志示例：

```
{
    "timestamp": "2018-07-25 22:10:12.690",
```

```
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "actionType": "DescribeStream",
    "protocol": "MQTT",
    "clientId": "581075e0-4639-48ee-8b94-2cf304168e43",
    "topicName": "$aws/things/581075e0-4639-48ee-8b94-2cf304168e43/streams/71c101a8-
bcc5-4929-9fe2-af563af0c139/describe/json",
    "streamId": "71c101a8-bcc5-4929-9fe2-af563af0c139",
    "clientToken": "clientToken",
    "details": "The request status is SUCCESS."
}
```

使用记录 AWS IoT OTA API 调用 AWS CloudTrail

FreeRTOS CloudTrail 与一项服务集成，该服务可 AWS IoT 捕获 OTA API 调用并将日志文件传送到您指定的 Amazon S3 存储桶。 CloudTrail 捕获从你的代码到 AWS IoT OTA 的 API 调用 APIs。使用收集的信息 CloudTrail，您可以确定向 AWS IoT OTA 发出的请求、发出请求的源 IP 地址、谁发出了请求、何时发出请求等。

有关更多信息 CloudTrail，包括如何配置和启用它，请参阅《AWS CloudTrail 用户指南》。

FreeRTOS 信息在 CloudTrail

在您的 AWS 账户中启用 CloudTrail 日志记录后，对 AWS IoT OTA 操作的 API 调用将在 CloudTrail 日志文件中进行跟踪，这些调用与其他 AWS 服务记录一起写入日志文件。 CloudTrail 根据时间段和文件大小决定何时创建和写入新文件。

以下 AWS IoT OTA 控制平面操作由以下人员记录 CloudTrail：

- CreateStream
- DescribeStream
- ListStreams
- UpdateStream
- DeleteStream
- 创建OTAUpdate
- 获取 OTAUpdate
- 列表OTAUpdates
- 删除OTAUpdate

> **Note**
>
> AWS IoT OTA 数据平面操作（设备端）不会由记录 CloudTrail。 CloudWatch 用于监视这些。

每个日志条目都包含有关生成请求的人员的信息。日志条目中的用户身份信息可帮助您确定以下内容：

- 请求是使用根用户凭证还是 IAM 用户凭证发出的。
- 请求是使用角色还是联合用户的临时安全凭证发出的。
- 请求是否由其他 AWS 服务发出。

有关更多信息，请参阅CloudTrail 用户身份元素。 AWS IoT OTA 操作记录在 AWS IoT OTA API 参考中。

日志文件可以在 Amazon S3 存储桶中存储任意长时间，不过您也可以定义 Amazon S3 生命周期规则以自动存档或删除日志文件。默认情况下，系统将使用 Amazon S3 服务器端加密 (SSE) 对日志文件进行加密。

如果您希望在日志文件传送时收到通知，则可以配置 CloudTrail 为发布 Amazon SNS 通知。有关更多信息，请参阅为其配置 Amazon SNS 通知。 CloudTrail

您还可以将来自多个 AWS 区域和多个 AWS 账户的 AWS IoT OTA 日志文件聚合到单个 Amazon S3 存储桶中。

有关更多信息，请参阅从多个区域接收 CloudTrail 日志文件和从多个账户接收 CloudTrail 日志文件。

了解 FreeRTOS 日志文件条目

CloudTrail 日志文件可以包含一个或多个日志条目。每个条目列出了多个 JSON 格式的事件。一个日志条目表示来自任何源的一个请求，包括有关所请求的操作、操作的日期和时间、请求参数等方面的信息。日志条目不是公用 API 调用的有序堆栈跟踪，因此它们不会以任何特定顺序显示。

以下示例显示了一个 CloudTrail 日志条目，该条目演示了号召性用语中的日志。`CreateOTAUpdate`

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EXAMPLE",
        "arn": "arn:aws:iam::your_aws_account:user/your_user_id",
        "accountId": "your_aws_account",
```

```
        "accessKeyId": "your_access_key_id",
        "userName": "your_username",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2018-08-23T17:27:08Z"
            }
        },
        "invokedBy": "apigateway.amazonaws.com"
    },
    "eventTime": "2018-08-23T17:27:19Z",
    "eventSource": "iot.amazonaws.com",
    "eventName": "CreateOTAUpdate",
    "awsRegion": "your_aws_region",
    "sourceIPAddress": "apigateway.amazonaws.com",
    "userAgent": "apigateway.amazonaws.com",
    "requestParameters": {
        "targets": [
            "arn:aws:iot:your_aws_region:your_aws_account:thing/Thing_CMH"
        ],
        "roleArn": "arn:aws:iam::your_aws_account:role/Role_FreeRTOSJob",
        "files": [
            {
                "fileName": "/sys/mcuflashimg.bin",
                "fileSource": {
                    "fileId": 0,
                    "streamId": "your_stream_id"
                },
                "codeSigning": {
                    "awsSignerJobId": "your_signer_job_id"
                }
            }
        ],
        "targetSelection": "SNAPSHOT",
        "otaUpdateId": "FreeRTOSJob_CMH-23-1535045232806-92"
    },
    "responseElements": {
        "otaUpdateArn": "arn:aws:iot:your_aws_region:your_aws_account:otaupdate/
FreeRTOSJob_CMH-23-1535045232806-92",
        "otaUpdateStatus": "CREATE_PENDING",
        "otaUpdateId": "FreeRTOSJob_CMH-23-1535045232806-92"
    },
    "requestID": "c9649630-a6f9-11e8-8f9c-e1cf2d0c9d8e",
    "eventID": "ce9bf4d9-5770-4cee-acf4-0e5649b845c0",
```

```
    "eventType": "AwsApiCall",
    "recipientAccountId": "recipient_aws_account"
}
```

使用获取创建OTAUpdate 失败详情 AWS CLI

如果创建 OTA 更新作业的过程失败，则可以采取一些措施来解决问题。当您创建 OTA 更新任务时，OTA 管理器服务会创建一个 IoT 任务并将其调度到目标设备，此过程还会在您的账户中创建或使用其他类型的 AWS 资源（代码签名任务、 AWS IoT 流、Amazon S3 对象）。遇到的任何错误都可能导致该进程在不创建 AWS IoT 作业的情况下失败。在此故障排除部分，我们提供了有关如何检索失败详细信息的说明。

1. 安装和配置 AWS CLI。
2. 运行 `aws configure` 并输入以下信息。

   ```
   $ aws configure
   AWS Access Key ID [None]: AccessID
   AWS Secret Access Key [None]: AccessKey
   Default region name [None]: Region
   Default output format [None]: json
   ```

   有关更多信息，请参阅使用 aws configure 进行快速配置。
3. 运行：

   ```
   aws iot get-ota-update --ota-update-id ota_update_job_001
   ```

   您在创建 OTA 更新时提供的 ID 在哪里*ota_update_job_001*。
4. 输出如下所示：

   ```
   {
       "otaUpdateInfo": {
           "otaUpdateId": "ota_update_job_001",
           "otaUpdateArn":
    "arn:aws:iot:region:account_id:otaupdate/ota_update_job_001",
           "creationDate": 1584646864.534,
           "lastModifiedDate": 1584646865.913,
           "targets": [
               "arn:aws:iot:region:account_id:thing/thing_001"
           ],
           "protocols": [
   ```

```
            "MQTT"
        ],
        "awsJobExecutionsRolloutConfig": {},
        "awsJobPresignedUrlConfig": {},
        "targetSelection": "SNAPSHOT",
        "otaUpdateFiles": [
            {
               "fileName": "/12ds",
                "fileLocation": {
                    "s3Location": {
                        "bucket": "bucket_name",
                        "key": "demo.bin",
                        "version": "Z7X.TWSAS7JSi4rybc02nMdcE41W1tV3"
                    }
                },
                "codeSigning": {
                    "startSigningJobParameter": {
                        "signingProfileParameter": {},
                        "signingProfileName": "signing_profile_name",
                        "destination": {
                            "s3Destination": {
                                "bucket": "bucket_name",
                                "prefix": "SignedImages/"
                            }
                        }
                    },
                    "customCodeSigning": {}
                }
            }
        ],
        "otaUpdateStatus": "CREATE_FAILED",
        "errorInfo": {
            "code": "AccessDeniedException",
            "message": "S3 object demo.bin not accessible. Please check
 your permissions (Service: AWSSigner; Status Code: 403; Error Code:
 AccessDeniedException; Request ID: 01d8e7a1-8c7c-4d85-9fd7-dcde975fdd2d)"
        }
    }
}
```

如果创建失败，则命令输出中的 otaUpdateStatus 字段将包含 CREATE_FAILED，而 errorInfo 字段将包含失败的详细信息。

使用获取 OTA 失败代码 AWS CLI

1. 安装和配置 AWS CLI。
2. 运行 `aws configure` 并输入以下信息。

```
$ aws configure
AWS Access Key ID [None]: AccessID
AWS Secret Access Key [None]: AccessKey
Default region name [None]: Region
Default output format [None]: json
```

有关更多信息，请参阅使用 `aws configure` 进行快速配置。

3. 运行：

```
aws iot describe-job-execution --job-id JobID --thing-name ThingName
```

哪里*JobID*是我们要获取其状态的任务的完整任务 ID 字符串（它在创建 OTA 更新任务时与 OTA 更新任务相关联），也是设备注册为 AWS IoT 的事物名称 *ThingName* AWS IoT

4. 输出如下所示：

```
{
    "execution": {
        "jobId": "AFR_OTA-****************",
        "status": "FAILED",
        "statusDetails": {
            "detailsMap": {
                "reason": "0xEEEEEEEE: 0xffffffff"
            }
        },
        "thingArn": "arn:aws:iot:Region:AccountID:thing/ThingName",
        "queuedAt": 1569519049.9,
        "startedAt": 1569519052.226,
        "lastUpdatedAt": 1569519052.226,
        "executionNumber": 1,
        "versionNumber": 2
    }
}
```

在此示例输出中，“detailsmap”中的“reason”有两个字段：显示为“0xEEEEEEEE”的字段包含 OTA 代理的常规错误代码；显示为“0xffffffff”的字段包含子代码。 https://docs.aws.amazon.com/

freertos/latest/lib-ref/html1/aws__ota__agent_8h.html 中列出了通用错误代码。错误代码带有“kOTA_Err_”前缀。子代码可以是平台特定的代码，也可以提供有关常规错误的更多详细信息。

对多个设备的 OTA 更新进行故障排除

要使用相同的固件映像 OTAs 在多个设备（事物）上执行，请实现一个clientcredentialIOT_THING_NAME从非易失性存储器中检索的功能（例如getThingName()）。请确保此函数从未被 OTA 覆盖的非易失性内存的一部分中读取事物名称，并且在运行第一个作业之前设置事物名称。如果您使用 JITP 流，可以从设备证书的公用名中读取事物名称。

使用德州仪器 CC322 0SF Launchpad 对 OTA 更新进行故障排除

CC3220SF Launchpad 平台提供了一种软件篡改检测机制。它使用一个安全警报计数器，每当出现完整性违规时，该计数器的计数就会递增。当安全警报计数器达到预先确定的阈值（默认值为 15），且主机收到 SL_ERROR_DEVICE_LOCKED_SECURITY_ALERT 异步事件时，设备将锁定。锁定的设备随后只具备有限的可访问性。要恢复设备，可以对其进行重新编程，也可以使用该 restore-to-factory过程恢复到出厂映像。您应当通过在 network_if.c 中更新异步事件处理程序来计划所需的行为。

FreeRTOS 库

FreeRTOS 库为 FreeRTOS 内核及其内部库提供其他功能。通过使用 FreeRTOS 库，您可以在嵌入式应用程序中实现联网和安全。FreeRTOS 库还使您的应用程序能够与服务进行交互。 AWS IoT 利用 FreeRTOS 所包含的库，可以：

- 使用 MQTT 和设备影子将设备安全地连接到 AWS IoT 云端。
- 发现并连接到 AWS IoT Greengrass 内核。
- 管理 Wi-Fi 连接。
- 侦听和处理 FreeRTOS 更新 Over-the-Air。

libraries 目录包含 FreeRTOS 库的源代码。协助实施库功能的帮助程序函数。建议您不要更改这些帮助程序函数。

FreeRTOS 移植库

以下移植库包含在可从 FreeRTOS 控制台下载的 FreeRTOS 的配置中。这些库与平台相关。其内容因硬件平台而异。有关将这些库移植到设备的信息，请参阅《FreeRTOS 移植指南》。

FreeRTOS 移植库

图书馆	API 参考	描述
低功耗蓝牙	低功耗蓝牙 API 参考	使用 FreeRTOS 低功耗蓝牙库，您的微控制器可以通过网关设备与 MQTT 代理 AWS IoT 进行通信。有关更多信息，请参阅 低功耗蓝牙库。
Over-the-Air 更新	AWS IoT Over-the-air 更新 API 参考资料	FreeRTO AWS IoT Over-the-air S (OTA) 更新库允许您在 FreeRTOS 设备上管理更新通知、下载更新以及对固件更新进行加密验证。 有关更多信息，请参阅 AWS IoT 空中 (OTA) 图书馆。
FreeRTOS+POSIX	FreeRTOS+POSIX API 参考	您可以使用 FreeRTOS+POSIX 库将与 POSIX 兼容的应用程序移植到 FreeRTOS 生态系统。 有关更多信息，请参阅 FreeRTOS+POSIX。
安全套接字	安全套接字 API 参考	有关更多信息，请参阅 安全套接字库。
FreeRTOS+TCP	FreeRTOS+TCP API 参考	FreeRTOS+TCP 是一个适用于 FreeRTOS 的可扩展的、线程安全的开源 TCP/IP 堆栈。 有关更多信息，请参阅 FreeRTOS+TCP。
Wi-Fi	Wi-Fi API 参考	利用 FreeRTOS Wi-Fi 库，您可以与微控制器的低级别无线堆栈进行交互。 有关更多信息，请参阅 Wi-Fi 库。

图书馆	API 参考	描述
核心 PKCS11		核心PKCS11 库是公钥加密标准 #11 的参考实现，用于支持配置和 TLS 客户端身份验证。 有关更多信息，请参阅 核心PKCS11 库。
TLS		有关更多信息，请参阅 传输层安全。
通用 I/O	通用 I/O API 参考	有关更多信息，请参阅 通用 I/O。
蜂窝接口	蜂窝接口 API 参考	蜂窝接口库通过统一的 API 公开了一些流行的蜂窝调制解调器的功能。有关更多信息，请参阅 蜂窝接口库。

FreeRTOS 应用程序库

您可以选择在 FreeRTOS 配置中包含以下独立应用程序库，以便 AWS IoT 与云上的服务进行交互。

> **Note**
>
> 有些应用程序库与适用于嵌入式 C 的 AWS IoT 设备 SDK 中的库 APIs 相同。有关这些库，请参阅AWS IoT 设备 SDK C API 参考。有关嵌入式 C 的 AWS IoT 设备 SDK 的更多信息，请参阅AWS IoT 适用于嵌入式 C 的设备 SDK。

FreeRTOS 应用程序库

图书馆	API 参考	描述
AWS IoT Device Defender	Device Defender C 开发工具包 API 参考	FreeRTOS 库可将你的 AWS IoT Device Defender FreeRTOS 设备连接到。 AWS IoT Device Defender 有关更多信息，请参阅 AWS IoT Device Defender 图书馆。

图书馆	API 参考	描述
AWS IoT Greengrass	Greengrass API 参考	FreeRTOS 库可将你的 AWS IoT Greengrass FreeRTOS 设备连接到。 AWS IoT Greengrass 有关更多信息，请参阅 AWS IoT Greengrass 探索图书馆。
MQTT	MQTT (v1.x.x) 库 API 参考 MQTT (v1) 代理 API 参考 MQTT (v2.x.x) C 开发工具包 API 参考	coreMQTT 库为 FreeRTOS 设备提供了用于发布和订阅 MQTT 主题的客户端。MQTT 是设备用来与 AWS IoT之交互的协议。 有关 coreMQTT 库版本 3.0.0 的更多信息，请参阅 coreMQTT 库。
coreMQTT 代理	coreMQTT 代理库 API 参考	coreMQTT 代理库是一个高级 API，它可以增加 coreMQTT 库的线程安全性。它允许您创建专用 MQTT 代理任务，该任务在后台管理 MQTT 连接，不需要其他任务的任何干预。该库提供了与 CoreMQTT 等效的线程安全库 APIs，因此可以在多线程环境中使用。 有关 coreMQTT 库的更多信息，请参阅 coreMQTT 代理库。
AWS IoT Device Shadow	设备影子 C 开发工具包 API 参考	AWS IoT Device Shadow 库使你的 FreeRTOS 设备能够与设备阴影进行交互 AWS IoT 。 有关更多信息，请参阅 AWS IoT Device Shadow 库。

配置 FreeRTOS 存储库

FreeRTOS 和嵌入式 C AWS IoT 的设备 SDK 的配置设置定义为 C 预处理器常量。您可以使用全局配置文件设置配置设置，或者使用 gcc 中的 -D 等编译器选项。由于配置设置被定义为编译时常量，因此在更改配置设置时，必须重新构建库。

如果您希望使用全局配置文件来设置配置选项，请创建并保存名为 iot_config.h 的文件，然后将其放在包含路径中。在文件中，使用 #define 指令可配置 FreeRTOS 库、演示和测试。

有关受支持的全局配置选项的更多信息，请参阅全局配置文件参考。

backoffAlgorithm 库

> **Note**
>
> 此页面上的内容可能不是 up-to-date。有关最新更新，请参阅 FreeRTOS.org 库页面。

简介

backoffAlgorithm 库是一个实用工具库，用于分隔同一数据块的反复重传，以避免网络拥塞。该库使用带抖动的指数回退算法计算重试网络操作（例如与服务器的网络连接失败）的回退时间段。

当重试因网络拥塞或服务器负载过高而导致的服务器连接或网络请求失败时，通常使用带抖动的指数回退功能。它用于分散多台设备同时尝试网络连接所创建的重试请求的时间。在连接性不佳的环境中，客户端可随时断开连接；因此，回退策略还可以帮助客户端在不太可能成功时不重复尝试重新连接，从而节省电池电量。

该库使用 C 语言编写，设计符合 ISO C90 和 MISRA C:2012。除了标准 C 库之外，该库不依赖于任何其他库，也没有堆分配，因此适用于 IoT 微控制器，但也完全可移植到其他平台。

该库可免费使用，并根据 MIT 开源许可证分发。

backoffAlgorithm 的代码大小（使用 GCC for ARM Cortex-M 生成的示例）		
文件	使用 -O1 优化	使用 -Os 优化
backoff_algorithm.c	0.1K	0.1K

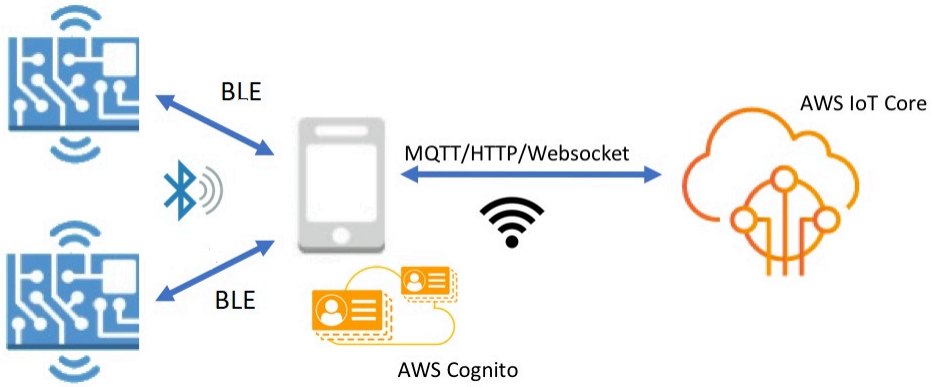
backoffAlgorithm 的代码大小（使用 GCC for ARM Cortex-M 生成的示例）		
文件	使用 -O1 优化	使用 -Os 优化
估计总数	0.1K	0.1K

低功耗蓝牙库

> **Important**
>
> 该库托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

概览

FreeRTOS 支持通过代理设备（例如手机）使用低功耗蓝牙功能发布和订阅消息队列遥测传输 (MQTT) 主题。借助 FreeRTOS 低功耗蓝牙 (BLE) 库，您的微控制器可以安全地与 MQT T 代理进行通信。AWS IoT

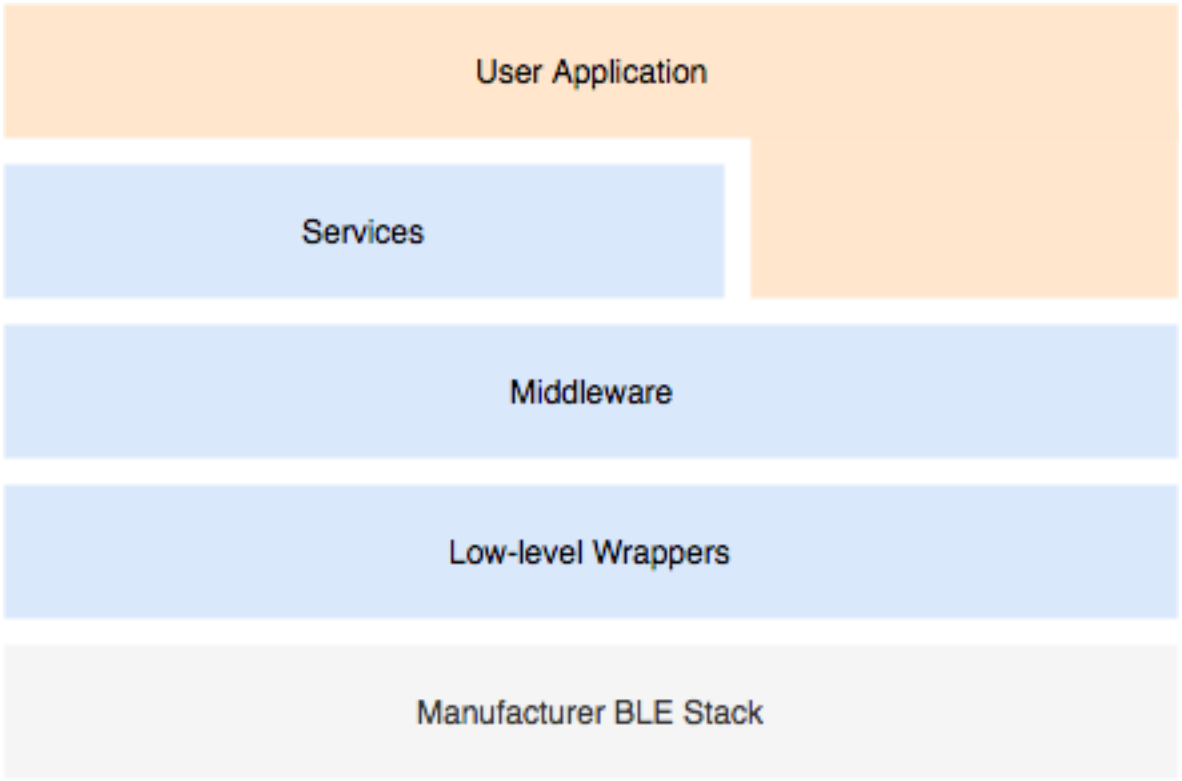


使用 SDKs 适用于 FreeRTOS 蓝牙设备的移动版，您可以编写原生移动应用程序，通过 BLE 与微控制器上的嵌入式应用程序进行通信。有关移动设备的更多信息 SDKs，请参阅 SDKs 适用于 FreeRTOS 蓝牙设备的移动设备。

FreeRTOS BLE 库包含一些用于配置 Wi-Fi 网络，传输大量数据以及通过 BLE 提供网络抽象的服务。FreeRTOS BLE 库还包括中间件和 APIs 较低级别，可以更直接地控制你的 BLE 堆栈。

架构

FreeRTOS BLE 库包含三个层：服务、中间件和低级别包装程序。



服务

FreeRTOS BLE 服务层由四个利用中间件的通用属性 (GATT) 服务组成： APIs

- 设备信息
- Wi-Fi 预置
- 网络抽象
- 大型对象传输

设备信息

设备信息服务可收集有关微控制器的详细信息，其中包括：

- 您的设备正在使用的 FreeRTOS 的版本。
- 设备注册的账户的 AWS IoT 终端节点。
- 低功耗蓝牙最大传输单元 (MTU)。

Wi-Fi 预置

利用 Wi-Fi 预置服务，具有 Wi-Fi 功能的微控制器可执行以下操作：

- 列出范围内的网络。
- 将网络和网络凭证保存到闪存。
- 设置网络优先级。
- 从闪存中删除网络和网络凭证。

网络抽象

网络抽象服务将应用程序的网络连接类型抽象化。通用 API 与您的设备的 Wi-Fi、以太网和低功耗蓝牙硬件堆栈交互，使得应用程序兼容多种连接类型。

大型对象传输

大型对象传输服务向客户端发送数据并从客户端接收数据。Wi-Fi 预置和网络抽象等其他服务使用大型对象传输服务来发送和接收数据。您还可以使用大型对象传输 API 来直接与服务交互。

MQTT over BLE

MQTT over BLE 包含用于通过 BLE 创建 MQTT 代理服务的 GATT 配置文件。MQTT 代理服务允许 MQTT 客户端通过网关设备与 AWS MQTT 代理进行通信。例如，您可以使用代理服务通过智能手机应用程序将运行 FreeRTOS 的设备连接 AWS 到 MQTT。BLE 设备是 GATT 服务器，并且公开了网关设备的服务和特性。使用这些公开的服务和特性，GATT 服务器通过云为设备执行 MQTT 操作。有关详细信息，请参考 附录 A：MQTT over BLE GATT 配置文件。

中间件

FreeRTOS 低功耗蓝牙中间件是从较低级别抽象出来的。 APIs中间件 APIs为低功耗蓝牙堆栈提供了更加用户友好的界面。

使用中间件 APIs，您可以跨多个层向单个事件注册多个回调。初始化低功耗蓝牙中间件还会初始化服务并启动通告。

灵活的回调订阅

假设您的低功耗蓝牙硬件断开连接，并且低功耗蓝牙 MQTT 服务需要检测断开连接事件。您编写的应用程序可能还需要检测相同的断开连接事件。低功耗蓝牙中间件可以将事件路由到已注册回调的代码的各个部分，而不会让较高的层争用低级别资源。

低级别包装程序

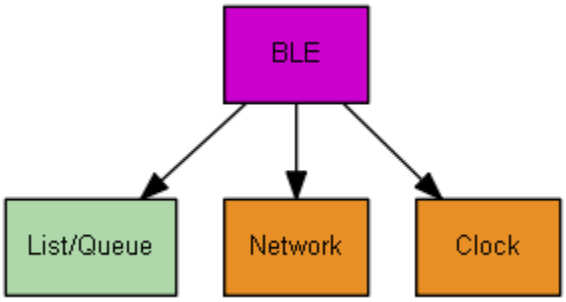
低级别 FreeRTOS 低功耗蓝牙包装器是来自制造商的低功耗蓝牙堆栈的抽象。低级封装器提供了一组 APIs 用于直接控制硬件的通用封装器。低级会 APIs 优化 RAM 使用情况，但功能有限。

使用低功耗蓝牙服务 APIs 与低功耗蓝牙服务进行交互。该服务比低级 APIs服务 APIs 需要更多的资源。

依赖项和要求

低功耗蓝牙库具有以下直接依赖项：

- 线性容器库
- 一个平台层，直接与操作系统交互用于线程管理、计时器、时钟功能和网络访问。



只有 Wi-Fi 预配置服务具有 FreeRTOS 库依赖项：

GATT 服务	依赖关系
Wi-Fi 预置	Wi-Fi 库

要与 AWS IoT MQTT 经纪人通信，您必须拥有一个 AWS 账户，并且必须将您的设备注册为 AWS IoT 事物。有关设置的更多信息，请参阅 AWS IoT 开发人员指南。

在移动设备上，FreeRTOS 低功耗蓝牙使用 Amazon Cognito 进行用户身份验证。要使用 MQTT 代理服务，您必须创建一个 Amazon Cognito 身份和用户池。每个 Amazon Cognito 身份都必须附加适当的策略。有关更多信息，请参阅《Amazon Cognito 开发人员指南》。

库配置文件

使用 FreeRTOS 低功耗蓝牙 MQTT 服务的应用程序必须提供一个定义了配置参数的 `iot_ble_config.h` 标头文件。未定义的配置参数将采用 `iot_ble_config_defaults.h` 中指定的默认值。

一些重要的配置参数包括：

`IOT_BLE_ADD_CUSTOM_SERVICES`

允许用户创建其自己的服务。

`IOT_BLE_SET_CUSTOM_ADVERTISEMENT_MSG`

允许用户自定义通告和扫描响应消息。

有关更多信息，请参阅低功耗蓝牙 API 参考。

优化

在优化主板性能时，请注意以下事项：

- 低级 APIs 使用较少的 RAM，但提供的功能有限。
- 您可以将 `iot_ble_config.h` 标头文件中的 `bleconfigMAX_NETWORK` 参数设置为一个较小的值来减少使用的堆栈量。
- 您可以将 MTU 大小增至其最大值来限制消息缓冲，并加快代码运行速度和减少占用的 RAM。

使用限制

默认情况下，FreeRTOS 低功耗蓝牙库将 `eBTpropertySecureConnectionOnly` 属性设置为 TRUE，这会将设备置于“仅安全连接”模式。按照蓝牙核心规范 5.0 版第 3 册 C 部分 10.2.4 中所述，当设备处于“仅安全连接”模式中时，需要最高 LE 安全模式 1 级别（级别 4）才能访问任何权限高于最低 LE 安全模式 1 级别（级别 1）的属性。在 LE 安全模式 1 级别 4，设备必须具有输入和输出功能才能进行数字比较。

此处列出了支持的模式及其关联属性：

模式 1、级别 1（不安全）

```
/* Disable numeric comparison */
```

```
#define IOT_BLE_ENABLE_NUMERIC_COMPARISON        ( 0 )
#define IOT_BLE_ENABLE_SECURE_CONNECTION         ( 0 )
#define IOT_BLE_INPUT_OUTPUT                     ( eBTIONone )
#define IOT_BLE_ENCRYPTION_REQUIRED               ( 0 )
```

模式 1、级别 2（无身份验证配对，有加密）

```
#define IOT_BLE_ENABLE_NUMERIC_COMPARISON        ( 0 )
#define IOT_BLE_ENABLE_SECURE_CONNECTION         ( 0 )
#define IOT_BLE_INPUT_OUTPUT                     ( eBTIONone )
```

模式 1、级别 3（有身份验证配对，有加密）

不支持此模式。

模式 1、级别 2（有身份验证 LE 安全连接配对，有加密）

默认支持此模式。

有关 LE 安全模式的信息，请参阅蓝牙核心规范 5.0 版第 3 册 C 部分 10.2.1。

初始化

如果您的应用程序通过中间件与低功耗蓝牙堆栈交互，则您只需初始化中间件即可。中间件负责初始化堆栈的较低层。

中间件

初始化中间件

1. 在调用低功耗蓝牙中间件 API 之前初始化任意低功耗蓝牙硬件驱动程序。
2. 启用低功耗蓝牙。
3. 使用 `IotBLE_Init()` 初始化中间件

> **Note**
>
> 如果您正在运行 AWS 演示，则不需要执行此初始化步骤。演示初始化由网络管理器处理，该管理器位于 *freertos*/demos/network_manager。

低级 APIs

如果您不想使用FreeRTOS蓝牙低功耗GATT服务，则可以绕过中间件直接与低级交互以节省资源。APIs

初始化低级 APIs

1. 在调用之前，请初始化所有低功耗蓝牙硬件驱动程序 APIs。驱动程序初始化不是低功耗 APIs蓝牙低功耗的一部分。
2. 低功耗蓝牙低级别 API 提供了对低功耗蓝牙堆栈的启用/禁用调用以优化功率和资源。在调用之前 APIs，必须启用低功耗蓝牙。

```
const BTInterface_t * pxIface = BTGetBluetoothInterface();
xStatus = pxIface->pxEnable( 0 );
```

3. 蓝牙管理器 APIs 包含低功耗蓝牙和经典蓝牙共有的内容。常见管理器的回调必须是第二个初始化的。

```
xStatus = xBTInterface.pxBTInterface->pxBtManagerInit( &xBTManagerCb );
```

4. 低功耗蓝牙适配器位于常见 API 的上方。您必须初始化其回调，就像您初始化常见 API 一样。

```
xBTInterface.pxBTLeAdapterInterface = ( BTBleAdapter_t * )
 xBTInterface.pxBTInterface->pxGetLeAdapter();
xStatus = xBTInterface.pxBTLeAdapterInterface-
>pxBleAdapterInit( &xBTBleAdapterCb );
```

5. 注册新的用户应用程序。

```
xBTInterface.pxBTLeAdapterInterface->pxRegisterBleApp( pxAppUuid );
```

6. 初始化对 GATT 服务器的回调。

```
xBTInterface.pxGattServerInterface = ( BTGattServerInterface_t * )
 xBTInterface.pxBTLeAdapterInterface->ppvGetGattServerInterface();
xBTInterface.pxGattServerInterface->pxGattServerInit( &xBTGattServerCb );
```

在初始化低功耗蓝牙适配器后，您可以添加 GATT 服务器。一次只能注册一个 GATT 服务器。

```
xStatus = xBTInterface.pxGattServerInterface->pxRegisterServer( pxAppUuid );
```

7. 设置应用程序属性，如仅安全连接和 MTU 大小。

```
xStatus = xBTInterface.pxBTInterface-
>pxSetDeviceProperty( &pxProperty[ usIndex ] );
```

API 参考

有关完整 API 参考，请参阅低功耗蓝牙 API 参考。

示例用法

以下示例说明了如何将低功耗蓝牙库用于通告和创建新服务。有关所有 FreeRTOS 低功耗蓝牙演示应用程序，请参阅低功耗蓝牙演示应用程序。

广告

1. 在您的应用程序中，设置通告 UUID：

```
static const BTUuid_t _advUUID =
{
    .uu.uu128 = IOT_BLE_ADVERTISING_UUID,
    .ucType   = eBTuuidType128
};
```

2. 然后，定义 `IotBle_SetCustomAdvCb` 回调函数：

```
void IotBle_SetCustomAdvCb( IotBleAdvertisementParams_t * pAdvParams,
 IotBleAdvertisementParams_t * pScanParams)
{
    memset(pAdvParams, 0, sizeof(IotBleAdvertisementParams_t));
    memset(pScanParams, 0, sizeof(IotBleAdvertisementParams_t));

    /* Set advertisement message */
    pAdvParams->pUUID1 = &_advUUID;
    pAdvParams->nameType = BTGattAdvNameNone;
```

```
    /* This is the scan response, set it back to true. */
    pScanParams->setScanRsp = true;
    pScanParams->nameType = BTGattAdvNameComplete;
}
```

此回调在通告消息中发送 UUID，在扫描响应中发送全名。

3. 打开 vendors/*vendor*/boards/*board*/aws_demos/config_files/iot_ble_config.h 并将 IOT_BLE_SET_CUSTOM_ADVERTISEMENT_MSG 设置为 1。这将触发 IotBle_SetCustomAdvCb 回调。

添加新服务

有关服务的完整示例，请参阅 *freertos*/.../ble/services。

1. UUIDs 为服务的特征和描述符创建：

```
#define xServiceUUID_TYPE \
{\
    .uu.uu128 = gattDemoSVC_UUID, \
    .ucType   = eBTuuidType128 \
}
#define xCharCounterUUID_TYPE \
{\
    .uu.uu128 = gattDemoCHAR_COUNTER_UUID,\
    .ucType   = eBTuuidType128\
}
#define xCharControlUUID_TYPE \
{\
    .uu.uu128 = gattDemoCHAR_CONTROL_UUID,\
    .ucType   = eBTuuidType128\
}
#define xClientCharCfgUUID_TYPE \
{\
    .uu.uu16 = gattDemoCLIENT_CHAR_CFG_UUID,\
    .ucType  = eBTuuidType16\
}
```

2. 创建缓冲区以注册特性和描述符的处理：

```
static uint16_t usHandlesBuffer[egattDemoNbAttributes];
```

3. 创建属性表。为节省一些 RAM，请将表定义为 const。

> **Important**
>
> 始终按顺序创建属性，将服务作为第一个属性。

```
static const BTAttribute_t pxAttributeTable[] = {
     {
          .xServiceUUID =  xServiceUUID_TYPE
     },
    {
          .xAttributeType = eBTDbCharacteristic,
          .xCharacteristic =
          {
               .xUuid = xCharCounterUUID_TYPE,
               .xPermissions = ( IOT_BLE_CHAR_READ_PERM ),
               .xProperties = ( eBTPropRead | eBTPropNotify )
           }
     },
     {
          .xAttributeType = eBTDbDescriptor,
          .xCharacteristicDescr =
          {
              .xUuid = xClientCharCfgUUID_TYPE,
              .xPermissions = ( IOT_BLE_CHAR_READ_PERM | IOT_BLE_CHAR_WRITE_PERM )
           }
     },
    {
          .xAttributeType = eBTDbCharacteristic,
          .xCharacteristic =
          {
               .xUuid = xCharControlUUID_TYPE,
               .xPermissions = ( IOT_BLE_CHAR_READ_PERM | IOT_BLE_CHAR_WRITE_PERM
  ),
               .xProperties = ( eBTPropRead | eBTPropWrite )
           }
     }
};
```

4. 创建回调阵列。此回调阵列必须与以上定义的表阵列采用相同的顺序。

例如，如果在访问 xCharCounterUUID_TYPE 时触发 vReadCounter，并在访问 xCharControlUUID_TYPE 时触发 vWriteCommand，则定义阵列如下：

```
static const IotBleAttributeEventCallback_t pxCallBackArray[egattDemoNbAttributes]
 =
    {
  NULL,
  vReadCounter,
  vEnableNotification,
  vWriteCommand
};
```

5. 创建服务：

```
static const BTService_t xGattDemoService =
{
  .xNumberOfAttributes = egattDemoNbAttributes,
  .ucInstId = 0,
  .xType = eBTServiceTypePrimary,
  .pusHandlesBuffer = usHandlesBuffer,
  .pxBLEAttributes = (BTAttribute_t *)pxAttributeTable
};
```

6. 使用您在上一步中创建的结构调用 API IotBle_CreateService。中间件同步所有服务的创建，因此在触发 IotBle_AddCustomServicesCb 回调时必须已经定义好了所有新服务。

   a. 在 vendors/*vendor*/boards/*board*/aws_demos/config_files/iot_ble_config.h 中将 IOT_BLE_ADD_CUSTOM_SERVICES 设置为 1。

   b. AddCustomServicesCb 在您的应用程序中创建 IotBle _：

```
void IotBle_AddCustomServicesCb(void)
{
    BTStatus_t xStatus;
    /* Select the handle buffer. */
    xStatus = IotBle_CreateService( (BTService_t *)&xGattDemoService,
 (IotBleAttributeEventCallback_t *)pxCallBackArray );
}
```

移植

用户输入和输出外围设备

安全连接需要输入和输出以进行数字比较。可使用事件管理器注册 eBLENumericComparisonCallback 事件：

```
xEventCb.pxNumericComparisonCb = &prvNumericComparisonCb;
xStatus = BLE_RegisterEventCb( eBLENumericComparisonCallback, xEventCb );
```

外围设备必须显示数字密钥，并将比较结果作为输入。

移植 API 实施

要将 FreeRTOS 移植到新目标，必须为 Wi-Fi 配置服务和蓝牙低功耗功能实现 APIs 一些功能。

低功耗蓝牙 APIs

要使用 FreeRTOS 低功耗蓝牙中间件，你必须实现一些。 APIs

APIs 常见于蓝牙经典版的 GAP 和低功耗蓝牙的 GAP 之间

- pxBtManagerInit
- pxEnable
- pxDisable
- pxGetDeviceProperty
- pxSetDeviceProperty（所有选项都是强制性的，eBTpropertyRemoteRssi 和 eBTpropertyRemoteVersionInfo 除外）
- pxPair
- pxRemoveBond
- pxGetConnectionState
- pxPinReply
- pxSspReply
- pxGetTxpower
- pxGetLeAdapter
- pxDeviceStateChangedCb
- pxAdapterPropertiesCb
- pxSspRequestCb

- pxPairingStateChangedCb
- pxTxPowerCb

APIs 专门针对低功耗蓝牙的 GAP

- pxRegisterBleApp
- pxUnregisterBleApp
- pxBleAdapterInit
- pxStartAdv
- pxStopAdv
- pxSetAdvData
- pxConnParameterUpdateRequest
- pxRegisterBleAdapterCb
- pxAdvStartCb
- pxSetAdvDataCb
- pxConnParameterUpdateRequestCb
- pxCongestionCb

GATT 服务器

- pxRegisterServer
- pxUnregisterServer
- pxGattServerInit
- pxAddService
- pxAddIncludedService
- pxAddCharacteristic
- pxSetVal
- pxAddDescriptor
- pxStartService
- pxStopService
- pxDeleteService
- pxSendIndication

- pxSendResponse
- pxMtuChangedCb
- pxCongestionCb
- pxIndicationSentCb
- pxRequestExecWriteCb
- pxRequestWriteCb
- pxRequestReadCb
- pxServiceDeletedCb
- pxServiceStoppedCb
- pxServiceStartedCb
- pxDescriptorAddedCb
- pxSetValCallbackCb
- pxCharacteristicAddedCb
- pxIncludedServiceAddedCb
- pxServiceAddedCb
- pxConnectionCb
- pxUnregisterServerCb
- pxRegisterServerCb

有关将 FreeRTOS 低功耗蓝牙库移植到您的平台的更多信息，请参阅《FreeRTOS 移植指南》中的移植低功耗蓝牙库。

SDKs 适用于 FreeRTOS 蓝牙设备的移动设备

> **Important**
>
> 该库托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

您可以使用 SDKs 适用于 FreeRTOS 蓝牙设备的移动版来创建通过低功耗蓝牙与您的微控制器交互的移动应用程序。移动设备 SDKs 还可以使用Amazon Cognito进行用户身份验证，从而与 AWS 服务进行通信。

适用于 FreeRTOS 蓝牙设备的 Android 开发工具包

使用适用于 FreeRTOS 蓝牙设备的 Android 开发工具包来构建通过低功耗蓝牙与微控制器交互的 Android 移动应用程序。该软件开发工具包已在上线GitHub。

要安装适用于 FreeRTOS 蓝牙设备的 Android 开发工具包，请按照项目的 README.md 文件中“设置开发工具包”的说明进行操作。

有关设置和运行开发工具包中所含演示移动应用程序的信息，请参阅先决条件和 FreeRTOS 低功耗蓝牙移动开发工具包演示应用程序。

适用于 FreeRTOS 蓝牙设备的 iOS 开发工具包

使用适用于 FreeRTOS 蓝牙设备的 iOS 开发工具包来构建通过低功耗蓝牙与微控制器交互的 iOS 移动应用程序。该软件开发工具包已在上线GitHub。

安装 iOS 开发工具包

1. 安装 CocoaPods：

```
$ gem install cocoapods
$ pod setup
```

> **Note**
>
> 您可能需要使用sudo才能安装 CocoaPods。

2. 使用以下命令安装 SDK CocoaPods（将其添加到你的 podfile 中）：

```
$ pod 'FreeRTOS', :git => 'https://github.com/aws/amazon-freertos-ble-ios-sdk.git'
```

有关设置和运行开发工具包中所含演示移动应用程序的信息，请参阅先决条件和 FreeRTOS 低功耗蓝牙移动开发工具包演示应用程序。

附录 A：MQTT over BLE GATT 配置文件

GATT 服务详细信息

MQTT over BLE 使用数据传输 GATT 服务的实例在 FreeRTOS 设备和代理设备之间发送 MQTT 简洁二进制对象表示法 (CBOR) 格式的消息。数据传输服务会公开某些特征，这些特征有助于通过 BLE

GATT 协议发送和接收原始数据。它还可以处理有效负载大于 BLE 最大传输单元 (MTU) 大小的分段和程序集。

服务 UUID

A9D7-166A-D72E-40A9-A002-4804-4CC3-FF00

服务实例

每次与代理进行的 MQTT 会话时都会创建一个 GATT 服务实例。每项服务都有一个用于标识类型的唯一 UUID（两个字节）。每个实例都根据实例 ID 进行区分。

每项服务都作为每个 BLE 服务器设备上的主要服务进行实例化。您可以在指定设备上创建该服务的多个实例。MQTT 代理服务类型具有唯一的 UUID。

特性

特征内容格式：CBOR

最大特征值大小：512 字节

特征	要求	必需属性	可选属性	安全权限	简要描述	UUID
控件	M	写入	无	写入需要加密	用于启动和停止 MQTT 代理。	A9D7-166A-D72E-40A9-A002-4804-4CC3-FF01
TXMessage	M	读取、通知	无	读取需要加密	用于通过透传代理 (Proxy) 向中介代理 (Broker) 发送包含消息的通知。	A9D7-166A-D72E-40A9-A002-4804-4CC3-FF02

特征	要求	必需属性	可选属性	安全权限	简要描述	UUID
RXMessage	M	读写无响应	无	读写需要加密	用于通过透传代理 (Proxy) 接收来自中介代理 (Broker) 的消息。	A9D7-166A-D72E-40A9-A002-4804-4CC3-FF03
TXLarge消息	M	读取、通知	无	读取需要加密	用于通过透传代理 (Proxy) 向中介代理 (Broker) 发送大消息（消息 > BLE MTU 大小）。	A9D7-166A-D72E-40A9-A002-4804-4CC3-FF04
RXLarge消息	M	读写无响应	无	读写需要加密	用于通过透传代理 (Proxy) 从中介代理 (Broker) 接收大消息（消息 > BLE MTU 大小）。	A9D7-166A-D72E-40A9-A002-4804-4CC3-FF05

GATT 过程要求

读取特征值	强制性
读取长特征值	强制性

写入特征值	强制性
写入长特征值	强制性
读取特征描述符	强制性
写入特征描述符	强制性
通知	强制性
指示	强制性

消息类型

交换以下消息类型。

消息类型	消息	使用这些键/值对进行映射
0x01	CONNECT	• 键 =“w”，值 = 类型 0 整数，消息类型 (1) • 键 =“d”，值 = 类型 3，文本字符串，会话的客户端标识符 • 键 =“a”，值 = 类型 3，文本字符串，会话的代理端点 • 键 =“c”，值 = 简单值类型 True/False
0x02	CONNACK	• 键 =“w”，值 = 类型 0 整数，消息类型 (2) • 键 =“s”，值 = 类型 0 整数，状态代码
0x03	发布	• 键 =“w”，值 = 类型 0 整数，消息类型 (3) • 键 =“u”，值 = 类型 3，文本字符串，发布的主题

消息类型	消息	使用这些键/值对进行映射
		• 键 = n”，值 = 类型 0，整数，发布的 QoS • 键 =“i”，值 = 类型 0，整数，消息标识符，仅适用于 QoS 1 发布 • 键 =“k”，值 = 类型 2，字节字符串，发布的有效负载
0x04	PUBACK	• 仅适用于 QoS 1 消息发送。 • 键 =“w”，值 = 类型 0 整数，消息类型 (4) • 键 =“i”，值 = 类型 0，整数，消息标识符
0x08	订阅	• 键 =“w”，值 = 类型 0 整数，消息类型 (8) • 键 =“v”，值 = 类型 4，文本字符串数组，订阅的主题 • 键 =“o”，值 = 类型 4，整数数组，订阅的 QoS • 键 =“i”，值 = 类型 0，整数，消息标识符
0x09	SUBACK	• 键 =“w”，值 = 类型 0 整数，消息类型 (9) • 键 =“i”，值 = 类型 0，整数，消息标识符 • 键 =“s”，值 = 类型 0 整数，订阅的状态代码

消息类型	消息	使用这些键/值对进行映射
0X0A	UNSUBSCRIBE	• 键 =“w”，值 = 类型 0 整数，消息类型 (9) • 键 =“v”，值 = 类型 4，文本字符串数组，取消订阅的主题 • 键 =“i”，值 = 类型 0，整数，消息标识符
0x0B	UNSUBACK	• 键 =“w”，值 = 类型 0 整数，消息类型 (11) • 键 =“i”，值 = 类型 0，整数，消息标识符 • 键 = “s”，值 = 类型 0，整数，状态码 UnSubscription
0X0C	PINGREQ	• 键 =“w”，值 = 类型 0 整数，消息类型 (12)
0x0D	PINGRESP	• 键 =“w”，值 = 类型 0 整数，消息类型 (13)
0x0E	DISCONNNECT	• 键 =“w”，值 = 类型 0 整数，消息类型 (14)

大有效负载传输特征

TXLarge消息

TXLarge设备使用消息发送大于为 BLE 连接协商的 MTU 大小的大型有效负载。

- 设备通过特征将有效负载前面的 MTU 字节作为通知发送。
- 代理根据此特征发送有关剩余字节的读取请求。
- 设备最多发送 MTU 大小或有效负载的剩余字节，以较小者为准。它每次都会根据发送的有效负载大小来增加读取的偏移量。
- 代理将继续读取特征，直到获得长度为零的有效负载或小于 MTU 大小的有效负载。

- 如果设备在指定的超时时间内没有收到读取请求，则传输将失败，代理和网关会释放缓冲区。
- 如果代理在指定的超时时间内没有收到读取响应，则传输将失败，代理会释放缓冲区。

RXLarge消息

RXLarge设备使用消息接收大于 BLE 连接协商的 MTU 大小的大型有效负载。

- 代理使用响应此特征的写入来逐一写入不超过 MTU 大小的消息。
- 设备会缓存消息，直到收到长度为零或长度小于 MTU 大小的写入请求。
- 如果设备在指定的超时时间内没有收到写入请求，则传输将失败，设备会释放缓冲区。
- 如果代理在指定的超时时间内没有收到写入响应，则传输将失败，代理会释放缓冲区。

蜂窝接口库

> **Note**
> 此页面上的内容可能不是 up-to-date。有关最新更新，请参阅 FreeRTOS.org 库页面。

简介

蜂窝接口库实现了一个简单的统一 API，从而隐藏了特定于蜂窝调制解调器的 AT 命令的复杂性，并向 C 程序员提供了类似套接字的接口。

大多数蜂窝调制解调器或多或少会实现 3GPP TS v27.007 标准定义的 AT 命令。该项目在可重复使用的通用组件中提供了此类标准 AT 命令的实现。该项目中的三个蜂窝接口库都利用了这种常见代码。每个调制解调器的库仅实现特定于供应商的 AT 命令，然后公开完整的蜂窝接口库 API。

实现 3GPP TS v27.007 标准的通用组件按照以下代码质量标准编写：

- GNU 复杂度分数不超过 8
- MISRA C:2012 编码标准。与标准的任何偏差都记录在标记有“Coverity”的源代码注释中。

依赖项和要求

蜂窝接口库没有直接依赖关系。但是，以太网、Wi-Fi 和蜂窝网络不能在 FreeRTOS 网络堆栈中共存。开发人员必须选择一个网络接口才能与安全套接字库集成。

移植

有关将蜂窝接口库移植到平台的信息，请参阅《FreeRTOS 移植指南》中的移植蜂窝接口库。

内存使用

蜂窝接口库的代码大小（使用 GCC for ARM Cortex-M 生成的示例）		
文件	使用 -O1 优化	使用 -Os 优化
cellular_3gpp_api.c	6.3K	5.7K
cellular_3gpp_urc_handler.c	0.9K	0.8K
cellular_at_core.c	1.4K	1.2K
cellular_common_api.c	0.5K	0.5K
cellular_common.c	1.6K	1.4K
cellular_pkthandler.c	1.4K	1.2K
cellular_pktio.c	1.8K	1.6K
估计总数	13.9K	12.4K

入门

下载源代码

源代码可以作为 FreeRTOS 库的一部分下载，也可以单独下载。

要使用 HTTPS 从 Github 克隆该库，请执行：

```
git clone https://github.com/FreeRTOS/FreeRTOS-Cellular-Interface.git
```

使用 SSH：

```
git clone git@github.com:FreeRTOS/FreeRTOS-Cellular-Interface.git
```

文件夹结构

在此存储库的根目录下，您将看到以下文件夹：

- `source`：可重复使用的常用代码，用于实现 3GPP TS v27.007 定义的标准 AT 命令
- `doc`：文档
- `test`：单元测试和 CBMC
- `tools`: Coverity 静态分析工具和 CMock

配置和构建库

蜂窝接口库应作为应用程序的一部分进行构建。为此，您必须提供某些配置。FreeRTOS_Cellular_Interface_Windows_Simulator 项目提供了有关如何配置构建的示例。在蜂窝 API 参考中可找到更多信息。

有关更多信息，请参阅蜂窝接口页面。

将蜂窝接口库与 MCU 平台集成

蜂窝接口库继续 MCUs 使用抽象接口，即通信接口，与蜂窝调制解调器通信。还必须在 MCU 平台上实现通信接口。通信接口的最常见实现通过 UART 硬件进行通信，但也可以通过其他物理接口（例如 SPI）实现。在蜂窝库 API 参考中可找到有关通信接口的文档。其中提供了以下通信接口实现示例：

- FreeRTOS Windows 模拟器通信接口
- FreeRTOS Common IO UART 通信接口
- STM32 L475 发现板通信接口
- Sierra Sensor Hub 主板通信接口

通用 I/O

> **Important**
>
> 该库托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

概述

通常，设备驱动程序独立于基础操作系统，并且特定于给定的硬件配置。硬件抽象层 (HAL) 提供了驱动程序和更高级别的应用程序代码之间的通用接口。HAL 提取出特定驱动程序的工作原理的详细信息，并提供一个统一的 API 来控制此类设备。您可以使用相同的方法 APIs 访问基于多个微控制器 (MCU) 的参考板上的各种设备驱动程序。

FreeRTOS 通用 I/O 将充当此硬件抽象层。它为在支持的参考板上访问常用串行设备提供了一套标准 APIs 。它们与这些外围设备进行 APIs 通信和交互，使您的代码能够跨平台运行。如果没有通用 I/O，则编写代码以使用低级设备这一操作是特定于芯片供应商的。

支持的外围设备

- UART
- SPI
- I2C

支持的特征

- 同步读/写 – 此函数在请求的数据量传输完成后才会返回。
- 异步读/写 – 此函数会立即返回，并且数据以异步方式传输。在此操作完成后，将调用已注册用户回调。

外围设备特定的

- I2C – 将多个操作合并到一个事务中。用于在一个事务中依次执行写入操作和读取操作。
- SPI – 在主设备和辅助设备之间传输数据，这意味着写入操作和读取操作会同时进行。

移植

有关更多信息，请参阅《FreeRTOS 移植指南》。

AWS IoT Device Defender 图书馆

> **Note**
>
> 此页面上的内容可能不是 up-to-date。有关最新更新，请参阅 FreeRTOS.org 库页面。

简介

您可以使用该 AWS IoT Device Defender 库将安全指标从您的物联网设备发送到 AWS IoT Device Defender。您可以使用 AWS IoT Device Defender 来持续监控设备的这些安全指标，以确定是否与您为每台设备定义的适当行为存在偏差。如果出现问题，则 AWS IoT Device Defender 会发出警报，以便您可以采取措施解决问题。与之交互 AWS IoT Device Defender 使用 MQTT（一种轻量级的发布-订阅协议）。该库提供了一个 API 来编写和识别 AWS IoT Device Defender使用的 MQTT 主题字符串。

有关更多信息，请参阅 AWS IoT 开发人员指南中的 AWS IoT Device Defender。

该库使用 C 语言编写，设计符合 ISO C90 和 MISRA C:2012。除了标准 C 库之外，该库不依赖于任何其他库。该库没有平台依赖关系，例如线程或同步。它可以与任何 MQTT 库以及任何 JSON 或 CBOR 库一起使用。该库有证据表明内存使用安全，没有堆分配，因此适用于 IoT 微控制器，但也完全可移植到其他平台。

该 AWS IoT Device Defender 库可以免费使用，并根据麻省理工学院的开源许可证进行分发。

Dev AWS IoT ice Defender 的代码大小（使用 GCC 为 ARM Cortex-M 生成的示例）		
文件	使用 -O1 优化	使用 -Os 优化
defender.c	1.1K	0.6K
估计总数	1.1K	0.6K

AWS IoT Greengrass 探索图书馆

> **Note**
>
> 此页面上的内容可能不是 up-to-date。有关最新更新，请参阅 FreeRTOS.org 库页面。

概览

微控制设备使用 AWS IoT Greengrass Discovery 库来发现网络上的 Greengrass 核心。使用 AWS IoT Greengrass Discovery APIs，您的设备可以在找到内核端点后向 Greengrass 核心发送消息。

依赖项和要求

要使用 Greengrass Discovery 库，你必须在 AWS IoT中创建一个东西，包括证书和策略。有关更多信息，请参阅 AWS IoT 入门。

您必须为 *freertos*/demos/include/aws_clientcredential.h 文件中的以下常量设置值：

clientcredentialMQTT_BROKER_ENDPOINT

您的 AWS IoT 终端节点。

clientcredentialIOT_THING_NAME

IoT 事物的名称。

clientcredentialWIFI_SSID

Wi-Fi 网络的 SSID。

clientcredentialWIFI_PASSWORD

Wi-Fi 密码。

clientcredentialWIFI_SECURITY

Wi-Fi 网络所使用的安全类型。

您还必须为 *freertos*/demos/include/aws_clientcredential_keys.h 文件中的以下常量设置值：

keyCLIENT_CERTIFICATE_PEM

与事物关联的证书 PEM。

keyCLIENT_PRIVATE_KEY_PEM

与事物关联的私有密钥 PEM。

必须在控制台中设置 Greengrass 组和核心设备。有关更多信息，请参阅入门 AWS IoT Greengrass。

尽管 Greengrass 连接不需要 coreMQTT 库，我们仍强烈建议您安装。该库可用于在发现 Greengrass 核心后与其进行通信。

API 参考

有关完整 API 参考，请参阅 Greengrass API 参考。

示例用法

Greengrass 工作流

MCU 设备通过向包含 Greengrass 核心连接 AWS IoT 参数的 JSON 文件请求来启动发现过程。可通过以下两种方法在 JSON 文件中检索 Greengrass 核心连接参数：

- 自动选择，循环访问 JSON 文件中列出的所有 Greengrass 核心，并连接到第一个可用核心。
- 手动选择，使用 `aws_ggd_config.h` 中的信息连接到指定的 Greengrass 核心。

如何使用 Greengrass API

Greengrass API 的所有默认配置选项在 `aws_ggd_config_defaults.h` 中定义。

如果只存在一个 Greengrass 核心，可调用 `GGD_GetGGCIPandCertificate` 请求包含 Greengrass 核心连接信息的 JSON 文件。`GGD_GetGGCIPandCertificate` 返回后，`pcBuffer` 参数中包含了 JSON 文件的文本。`pxHostAddressData` 参数中包含了您可以连接的 Greengrass 核心的 IP 地址和端口。

要获得更多自定义选项，例如动态分配证书，则必须调用以下命令 APIs：

`GGD_JSONRequestStart`

向发出 HTTP GET 请求 AWS IoT 以启动发现 Greengrass 核心的发现请求。`GD_SecureConnect_Send`用于将请求发送到 AWS IoT。

`GGD_JSONRequestGetSize`

从 HTTP 响应获取 JSON 文件的大小。

`GGD_JSONRequestGetFile`

获取 JSON 对象字符串。`GGD_JSONRequestGetSize` 和 `GGD_JSONRequestGetFile` 使用 `GGD_SecureConnect_Read` 从套接字获取 JSON 数据。必须调用 `GGD_JSONRequestStart`、`GGD_SecureConnect_Send`、`GGD_JSONRequestGetSize`，从 AWS IoT接收 JSON 数据。

`GGD_GetIPandCertificateFromJSON`

从 JSON 数据中提取 IP 地址和 Greengrass 核心证书。可以通过将 `xAutoSelectFlag` 设置为 `True`，打开自动选择功能。自动选择将找到 FreeRTOS 设备可以连接的第一个核心设备。要连接到 Greengrass 核心，可调用 `GGD_SecureConnect_Connect` 函数，传递核心设备的 IP 地址、端口和证书。要使用手动选择，可设置 `HostParameters_t` 参数的以下字段：

`pcGroupName`

核心所属 Greengrass 组的 ID。可以使用 `aws greengrass list-groups` CLI 命令来查找 Greengrass 组的 ID。

`pcCoreAddress`

要连接的 Greengrass 核心的 ARN。

coreHTTP 库

> **Note**
>
> 此页面上的内容可能不是 up-to-date。有关最新更新，请参阅 FreeRTOS.org 库页面。

适用于小型 IoT 设备（MCU 或小型 MPU）的 HTTP C 客户端库

简介

coreHTTP 库是 HTTP/1.1 标准子集的客户端实现。HTTP 标准提供了一种在 TCP/IP 上运行的无状态协议，通常用于分布式、协作式的超文本信息系统。

coreHTTP 库实现了 HTTP/1.1 协议标准的子集。该库已针对低内存占用空间进行了优化。该库提供了一个完全同步的 API，因此应用程序可以完全管理其并发性。它仅使用固定缓冲区，因此应用程序可以完全控制其内存分配策略。

该库使用 C 语言编写，设计符合 ISO C90 和 MISRA C:2012。该库的唯一依赖项是标准 C 库和 Node.js 的 http-parser 的 LTS 版本 (v12.19.1)。该库有证据表明内存使用安全，没有堆分配，因此适用于 IoT 微控制器，但也完全可移植到其他平台。

在 IoT 应用中使用 HTTP 连接时，我们建议您使用安全的传输接口，例如，使用 TLS 协议的接口，如中 coreHTTP 双向身份验证演示所示。

该库可免费使用，并根据 MIT 开源许可证分发。

coreHTTP 的代码大小（使用 GCC for ARM Cortex-M 生成的示例）		
文件	使用 -O1 优化	使用 -Os 优化
core_http_client.c	3.2K	2.6K

coreHTTP 的代码大小（使用 GCC for ARM Cortex-M 生成的示例）		
文件	使用 -O1 优化	使用 -Os 优化
api.c (llhttp)	2.6K	2.0K
http.c (llhttp)	0.3K	0.3K
http.c (llhttp)	17.9	15.9
估计总数	23.9K	20.7K

coreJSON 库

> **Note**
>
> 此页面上的内容可能不是 up-to-date。有关最新更新，请参阅 FreeRTOS.org 库页面。

简介

JSON（JavaScript 对象表示法）是一种人类可读的数据序列化格式。它被广泛用于交换数据，例如与 Dev AWS IoT ice Shadow 服务交换数据，并且是许多服务的一部分 APIs，例如 GitHub REST API。JSON 由 Ecma International 作为标准进行维护。

coreJSON 库在严格执行 ECMA-404 标准 JSON 数据交换语法的同时提供支持键查找的解析器。该库使用 C 语言编写，设计符合 ISO C90 和 MISRA C:2012。该库有证据表明内存使用安全，没有堆分配，因此适用于 IoT 微控制器，但也完全可移植到其他平台。

内存使用

coreJson 库使用内部堆栈来跟踪 JSON 文档中的嵌套结构。堆栈在单个函数调用期间存在；它不会被保留。堆栈大小可以通过定义宏 `JSON_MAX_DEPTH`（默认为 32 级）来指定。每一级消耗一个字节。

coreJSON 的代码大小（使用 GCC for ARM Cortex-M 生成的示例）		
文件	使用 -O1 优化	使用 -Os 优化
core_json.c	2.9K	2.4K

coreJSON 的代码大小（使用 GCC for ARM Cortex-M 生成的示例）		
文件	使用 -O1 优化	使用 -Os 优化
估计总数	2.9K	2.4K

coreMQTT 库

> **Note**
>
> 此页面上的内容可能不是 up-to-date。有关最新更新，请参阅 FreeRTOS.org 库页面。

简介

coreMQTT 库是 MQTT（消息队列遥测传输）标准的客户端实现。MQTT 标准提供了一种在 TCP/IP 之上运行的轻量级发布/订阅（或 PubSub）消息协议，通常用于机器对机器 (M2M) 和物联网 (IoT) 用例。

coreMQTT 库符合 MQTT 3.1.1 协议标准。该库已针对低内存占用空间进行了优化。该库的设计涵盖了不同的使用案例，从仅使用 QoS 0 MQTT PUBLISH 消息的资源受限平台，到使用 QoS 2 MQTT PUBLISH over TLS（传输层安全）连接的资源丰富平台。该库提供了可组合函数的菜单，可选择和组合这些函数来精确满足特定使用案例的需求。

该库使用 C 语言编写，设计符合 ISO C90 和 MISRA C:2012。除以下库外，此 MQTT 库不依赖任何其他库：

- 标准 C 库
- 客户实现的网络传输接口
- （可选）用户实现的平台时间函数

通过提供简单的发送和接收传输接口规范，该库与底层网络驱动程序分离。应用程序编写者可选择现有传输接口，也可以根据自己的应用程序实现自己的传输接口。

该库提供了一个高级别 API，以便连接 MQTT 代理、订阅/取消订阅主题、向主题发布消息以及接收传入的消息。此 API 将上述传输接口作为参数，并使用该参数向 MQTT 代理发送和接收消息。

该库还公开了低级别 serializer/deserializer API. This API can be used to build a simple IoT application consisting of only the required a subset of MQTT functionality, without any other overhead. The serializer/deserializer API，可以与任何可用的传输层 API（例如套接字）结合使用，用于向代理发送和接收消息。

在 IoT 应用中使用 HTTP 连接时，我们建议您使用安全的传输接口，例如，使用 TLS 协议的接口。

此 MQTT 库没有平台依赖关系，例如线程或同步。该库有证据表明内存使用安全，没有堆分配，因此适用于 IoT 微控制器，但也完全可移植到其他平台。该库可免费使用，并根据 MIT 开源许可证分发。

coreMQTT 的代码大小（使用 GCC for ARM Cortex-M 生成的示例）		
文件	使用 -O1 优化	使用 -Os 优化
core_mqtt.c	4.0K	3.4K
core_mqtt_state.c	1.7K	1.3K
core_mqtt_serializer.c	2.8K	2.2K
估计总数	8.5K	6.9K

coreMQTT 代理库

> **Note**
>
> 此页面上的内容可能不是 up-to-date。有关最新更新，请参阅 FreeRTOS.org 库页面。

简介

coreMQTT 代理库是一个高级别 API，它增加了 coreMQTT 库 的线程安全性。它允许您创建专用 MQTT 代理任务，该任务在后台管理 MQTT 连接，不需要其他任务的任何干预。该库提供了与 CoreMQTT 等效的线程安全库 APIs，因此可以在多线程环境中使用。

MQTT 代理是一个独立的任务（或执行线程）。它是唯一允许访问 MQTT 库的 API 的任务，从而实现了线程安全。它通过隔离对单个任务的所有 MQTT API 调用来序列化访问权限，并且无需使用信号灯或任何其他同步基元。

该库使用线程安全的消息队列（或其他进程间通信机制）来序列化所有调用 MQTT 的请求。 APIs消息传递实现通过消息传递接口与库分离，该接口允许将库移植到其他操作系统。消息传递接口由用于发送和接收指向代理命令结构的指针的函数和用于分配这些命令对象的函数组成，允许应用程序编写者决定适合其应用程序的内存分配策略。

该库使用 C 语言编写，设计符合 ISO C90 和 MISRA C:2012。除了 coreMQTT 库 和标准 C 库之外，该库不依赖于任何其他库。该库有证据表明内存使用安全，没有堆分配，因此可用于 IoT 微控制器，但也完全可移植到其他平台。

该库可免费使用，并根据 MIT 开源许可证分发。

coreMQTT 代理的代码大小（使用 GCC for ARM Cortex-M 生成的示例）		
文件	使用 -O1 优化	使用 -Os 优化
core_mqtt_agent.c	1.7K	1.5K
core_mqtt_agent_command_functions.c	0.3K	0.2K
core_mqtt.c (coreMQTT)	4.0K	3.4K
core_mqtt_state.c (coreMQTT)	1.7K	1.3K
core_mqtt_serializer.c (coreMQTT)	2.8K	2.2K
估计总数	10.5K	8.6K

AWS IoT 空中 (OTA) 图书馆

> **Note**
>
> 此页面上的内容可能不是 up-to-date。有关最新更新，请参阅 FreeRTOS.org 库页面。

简介

AWS IoT Over-the-air (OTA) 更新库使您能够使用 HTTP 或 MQTT 作为协议管理 FreeRTOS 设备的固件更新的通知、下载和验证。通过使用 OTA 代理库，可以从逻辑上将固件更新与设备上运行的应用程

序进行隔离。OTA 代理可以与应用程序共享网络连接。通过共享网络连接，有可能节省大量的 RAM。此外，可使用 OTA 代理库来定义特定于应用程序的逻辑，以测试、提交或回滚固件更新。

物联网 (IoT) 将互联网连接扩展到传统上未连接的嵌入式设备。您可以对这些设备进行编程，使其通过互联网传输可用数据，并且可以进行远程监控和控制。随着技术的进步，这些传统嵌入式设备正在快速获得消费者、工业和企业领域的互联网功能。

IoT 设备的部署规模通常很大，并且通常部署在操作人员难以接近或根本无法接近的位置。想象一下发现可能暴露数据的安全漏洞的场景。在这种情况下，使用安全修复程序快速可靠地更新受影响的设备非常重要。如果无法执行 OTA 更新，那么更新在地理位置上分散的设备也可能很困难。让技术人员更新这些设备既昂贵又耗时，而且往往不切实际。更新这些设备所需的时间会使它们在更长的时间内面临安全漏洞。召回这些设备进行更新也需要付出高昂的代价，并可能因停机而对消费者造成严重干扰。

空中下载 (OTA) 更新使设备固件更新成为了可能，无需进行昂贵的召回或技术人员拜访。该方法增加了以下好处：

- 安全 – 能够快速响应在现场部署设备后发现的安全漏洞和软件错误。
- 创新 – 随着新功能的开发，产品可以经常更新，从而推动创新周期。与传统更新方法相比，这种更新不仅可以快速生效，而且停机时间最短。
- 成本 – 与传统上用于更新这些设备的方法相比，OTA 更新可以显著降低维护成本。

提供 OTA 功能需要考虑以下设计注意事项：

- 安全通信 – 更新必须使用加密的通信渠道，以防止下载内容在传输过程中被篡改。
- 恢复 – 由于网络连接中断或收到无效更新等原因，更新可能会失败。在这些情况下，设备需要能够恢复到稳定状态和避免变砖。
- 作者验证 – 必须验证更新是否来自可信来源，同时还要进行其他验证，例如检查版本和兼容性。

有关使用 FreeRTOS 设置 OTA 更新的更多信息，请参阅 FreeRTOS 更新 Over-the-Air。

AWS IoT 空中 (OTA) 图书馆

AWS IoT OTA 库使您能够管理有关新可用更新的通知、下载更新以及对固件更新进行加密验证。使用 over-the-air (OTA) 客户端库，您可以在逻辑上将固件更新机制与设备上运行的应用程序分开。 over-the-air(OTA) 客户端库可以与应用程序共享网络连接，从而在资源受限的设备中节省内存。此外，over-the-air(OTA) 客户端库还允许您定义特定于应用程序的逻辑，用于测试、提交或回滚固件更新。该库支持不同的应用程序协议，例如消息队列遥测传输 (MQTT) 和超文本传输协议 (HTTP)，并提供各种配置选项，您可以根据自己的网络类型和条件进行微调。

该库 APIs 提供以下主要功能：

- 注册接收通知或投票以获取可用的新更新请求。
- 接收、解析和验证更新请求。
- 根据更新请求中的信息下载并验证文件。
- 在激活收到的更新之前进行自测，以确保更新的功能有效性。
- 更新设备的状态。

该库使用 AWS 服务来管理各种与云相关的功能，例如发送固件更新、监控多个区域的大量设备、缩小错误部署的爆炸半径以及验证更新的安全性。该库可以与任何 MQTT 或 HTTP 库一起使用。

该库的演示演示了在 FreeRTOS 设备上使用 CoreMQTT 库和 AWS 服务进行的完整 over-the-air更新。

特征

下面是完整的 OTA 代理接口：

`OTA_Init`

通过在系统中启动 OTA 代理（“OTA 任务”）来初始化 OTA 引擎。只能存在一个 OTA 代理。

`OTA_Shutdown`

向 OTA 代理发出关闭信号。OTA 代理可以选择取消订阅所有 MQTT 作业通知主题，停止正在进行的 OTA 作业（如果有），并清除所有资源。

`OTA_GetState`

获取 OTA 代理的当前状态。

`OTA_ActivateNewImage`

激活通过 OTA 收到的最新的微控制器固件映像。（详细的作业状态现在应当为“自检”。）

`OTA_SetImageState`

设置当前运行的微控制器固件映像的验证状态（正在测试、已接受或已拒绝）。

`OTA_GetImageState`

获取当前运行的微控制器固件映像的状态（正在测试、已接受或已拒绝）。

OTA_CheckForUpdate

从 OTA 更新服务请求下一个可用的 OTA 更新。

OTA_Suspend

暂停所有 OTA 代理操作。

OTA_Resume

恢复 OTA 代理操作。

OTA_SignalEvent

向 OTA 代理任务发出事件信号。

OTA_EventProcessingTask

OTA 代理事件处理循环。

OTA_GetStatistics

获取 OTA 消息包的统计信息，包括接收、排队、处理和丢弃的数据包数量。

OTA_Err_strerror

OTA 错误的字符串转换错误代码。

OTA_JobParse_strerror

将 OTA 作业解析错误代码转换为字符串。

OTA_PalStatus_strerror

OTA PAL 状态的状态代码转换为字符串。

OTA_OsStatus_strerror

OTA OS 状态的状态代码转换为字符串。

API 参考

有关更多信息，请参阅AWS IoT Over-the-air 更新：函数。

示例用法

使用 MQTT 协议的典型的支持 OTA 的设备应用程序采用以下 API 调用顺序来驱动 OTA 代理。

1. 连接 C AWS IoT oreMQTT 代理。有关更多信息，请参阅 coreMQTT 代理库。

2. 通过调用 `OTA_Init` 来初始化 OTA 代理（包括缓冲区、所需的 OTA 接口、事物名称和应用程序回调）。回调实施特定于应用程序的逻辑，于 OTA 更新作业完成后执行。
3. OTA 更新完成后，FreeRTOS 调用具有以下事件之一的作业完成回调：`accepted`、`rejected` 或 `self test`。
4. 如果新的固件映像已遭拒绝（例如，由于验证错误），应用程序通常可以忽略通知并等待下一次更新。
5. 如果更新有效且标记为“已接受”，可调用 `OTA_ActivateNewImage` 重置设备并启动新的固件映像。

移植

有关将 OTA 功能移植到您的平台的信息，请参阅《FreeRTOS 移植指南》中的移植 OTA 库。

内存使用

AWS IoT OTA 的代码大小（使用 GCC 为 ARM Cortex-M 生成的示例）		
文件	使用 -O1 优化	使用 -Os 优化
ota.c	8.3K	7.5K
ota_interface.c	0.1K	0.1K
ota_base64.c	0.6K	0.6K
ota_mqtt.c	2.4K	2.2K
ota_cbor.c	0.8K	0.6K
ota_http.c	0.3K	0.3K
估计总数	12.5K	11.3K

核心PKCS11 库

> **Note**
>
> 此页面上的内容可能不是 up-to-date。有关最新更新，请参阅 FreeRTOS.org 库页面。

概览

公有密钥加密标准 #11 定义了一个独立于平台的 API，可用于管理和使用加密令牌。PKCS #11 是指标准定义的 API 和标准本身。PKCS #11 加密 API 用于提取密钥存储、加密对象的 get/set 属性以及会话语义。该 API 广泛用于操作常见加密对象，它之所以重要，是因为它指定的函数允许应用程序软件使用、创建、修改和删除加密对象，而无需将这些对象在应用程序的内存中公开这些对象。例如，FreeRTOS 和 AWS eRTOS 参考集成使用 PKCS #11 API 的一小部分来访问创建由传输层安全 (TLS) 协议进行身份验证和保护的网络连接所必需的秘密（私有）密钥，而应用程序“看到”该密钥不会“看见”该密钥。

核心PKCS11 库包含 PKCS #11 接口 (API) 的基于软件的模拟实现，该接口使用 Mbed TLS 提供的加密功能。使用软件模拟可实现快速开发并提高灵活性，但预计您将使用专门针对生产设备中使用的安全密钥存储的实现来取代模拟。通常，安全加密处理器（例如，可信平台模块 (TPM)、硬件安全模块 (HSM)、安全元件或任何其他类型的安全硬件飞地）的供应商会随硬件一起分发 PKCS #11 实现。因此，仅限核心PKCS11 软件的模拟库的目的是提供一种非硬件特定的 PKCS #11 实现，允许在生产设备中切换到加密处理器专用的 PKCS #11 实现之前进行快速原型设计和开发。

仅实现了 PKCS #11 标准的一个子集，重点是涉及非对称密钥、随机数生成和哈希的操作。目标使用案例包括小型嵌入式设备上适用于 TLS 身份验证的证书和密钥管理以及代码签名式签名验证。请参阅 FreeRTOS 源代码存储库中的文件 `pkcs11.h`（从 OASIS 获取，标准正文）。在 FreeRTOS 参考实现中，PKCS#11 API 调用由 TLS 帮助程序接口生成，目的是为了在 `SOCKETS_Connect` 过程中执行 TLS 客户端身份验证。PKCS #11 API 调用也可以由一次性开发人员预置工作流程生成，以将用于身份验证的 TLS 客户端证书和私有密钥导入至 AWS IoT MQTT 代理。以上两个使用案例（预置和 TLS 客户端身份验证）都只需要实施 PKCS #11 接口标准的一小部分。

特征

下面列出了使用的部分 PKCS #11。以下列表按照为实现预置、TLS 客户端身份验证和清除而调用的例程大致排序。有关各个函数的详细说明，请参阅标准委员会提供的 PKCS #11 文档。

常规设置和卸载 API

- `C_Initialize`
- `C_Finalize`
- `C_GetFunctionList`
- `C_GetSlotList`
- `C_GetTokenInfo`
- `C_OpenSession`
- `C_CloseSession`

- C_Login

预置 API

- C_CreateObject CKO_PRIVATE_KEY（对于设备私有密钥）
- C_CreateObject CKO_CERTIFICATE（对于设备证书和代码验证证书）
- C_GenerateKeyPair
- C_DestroyObject

客户端身份验证

- C_GetAttributeValue
- C_FindObjectsInit
- C_FindObjects
- C_FindObjectsFinal
- C_GenerateRandom
- C_SignInit
- C_Sign
- C_VerifyInit
- C_Verify
- C_DigestInit
- C_DigestUpdate
- C_DigestFinal

非对称加密支持

FreeRTOS PKCS#11 参考实施使用 2048 位 RSA（仅限签名）以及具有 NIST P-256 曲线的 ECDSA。以下说明描述了如何基于 P-256 客户端证书创建 AWS IoT 事物。

请确保您使用的是以下（或更新）版本的 AWS CLI 和 OpenSSL：

```
aws --version
aws-cli/1.11.176 Python/2.7.9 Windows/8 botocore/1.7.34
```

```
openssl version
OpenSSL 1.0.2g  1 Mar 2016
```

以下过程假设您使用 `aws configure` 命令来配置 AWS CLI。有关更多信息，请参阅《AWS Command Line Interface 用户指南》中的使用 `aws configure` 进行快速配置。

基于 P-256 客户端证书创建 AWS IoT 事物

1. 创造一个 AWS IoT 东西。

   ```
   aws iot create-thing --thing-name thing-name
   ```

2. 使用 OpenSSL 创建 P-256 密钥。

   ```
   openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -pkeyopt
    ec_param_enc:named_curve -outform PEM -out thing-name.key
   ```

3. 创建证书注册请求，采用第 2 步中创建的密钥进行签名。

   ```
   openssl req -new -nodes -days 365 -key thing-name.key -out thing-name.req
   ```

4. 将证书注册请求提交给 AWS IoT。

   ```
   aws iot create-certificate-from-csr  \
     --certificate-signing-request file://thing-name.req --set-as-active  \
     --certificate-pem-outfile thing-name.crt
   ```

5. 将证书（由 ARN 输出通过上一个命令引用）附加到事物。

   ```
   aws iot attach-thing-principal --thing-name thing-name \
     --principal "arn:aws:iot:us-
   east-1:123456789012:cert/
   86e41339a6d1bbc67abf31faf455092cdebf8f21ffbc67c4d238d1326c7de729"
   ```

6. 创建策略。（该政策过于宽松，只能用于开发目的。）

   ```
   aws iot create-policy --policy-name FullControl --policy-document file://
   policy.json
   ```

   以下是在 `create-policy` 命令中指定的 policy.json 文件的列表。如果不希望运行 FreeRTOS 演示以进行 Greengrass 连接和发现，可以忽略 `greengrass:*` 操作。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:*",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "greengrass:*",
      "Resource": "*"
    }
  ]
}
```

7. 将委托人（证书）和策略附加到事物。

```
aws iot attach-principal-policy --policy-name FullControl \
  --principal "arn:aws:iot:us-
east-1:123456789012:cert/
86e41339a6d1bbc67abf31faf455092cdebf8f21ffbc67c4d238d1326c7de729"
```

接下来，请按照本指南 AWS IoT 入门部分中的步骤进行操作。请记得将创建的证书和私有密钥复制到 aws_clientcredential_keys.h 文件中。将事物名称复制到 aws_clientcredential.h 中。

> **Note**
>
> 仅出于演示目的对证书和私有密钥进行了硬编码。生产级应用程序应将这些文件存储在安全位置。

移植

有关将核心PKCS11 库移植到您的平台的信息，请参阅 FreeRTOS 移植指南中的移植核心PKCS11 库。

内存使用

内核的代码大小PKCS11 （使用 GCC 为 ARM Cortex-M 生成的示例）		
文件	使用 -O1 优化	使用 -Os 优化
core_pkcs11.c	0.8K	0.8K
core_pki_utils.c	0.5K	0.3K
core_pkcs11_mbedtls.c	8.9K	7.5K
估计总数	10.2K	8.6K

安全套接字库

> **Important**
>
> 该库托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

概览

您可以使用 FreeRTOS 安全套接字库来创建可安全通信的嵌入式应用程序。该库旨在使来自各种网络编程背景的软件开发人员能够轻松掌握。

FreeRTOS 安全套接字库基于 Berkeley 套接字接口，并具有通过 TLS 协议进行安全通信的额外选项。有关 FreeRTOS 安全套接字库和 Berkeley 套接字接口之间差异的信息，请参阅安全套接字 API 参考中的 `SOCKETS_SetSockOpt`。

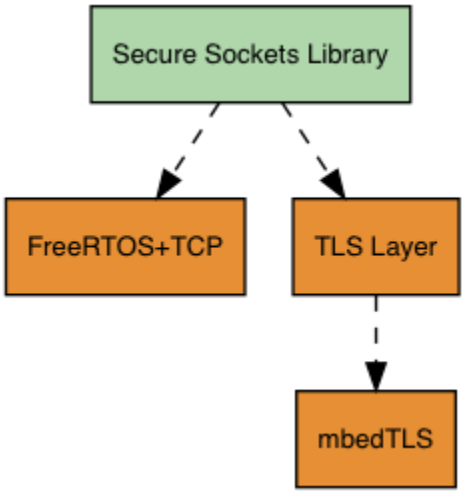
> **Note**
>
> 目前，Free RTOS 安全套接字仅支持客户端 APIs，以及服务器`Bind`端 API 的轻量级 IP (LWIP) 实现。

依赖项和要求

FreeRTOS 安全套接字库依赖于 TCP/IP 堆栈和 TLS 实现。FreeRTOS 的端口通过三种方式之一来满足这些依赖项：

- TCP/IP 和 TLS 的自定义实现
- TCP/IP 以及具有 mbedTLS 的 FreeRTOS TLS 层的自定义实现
- FreeRTOS+TCP 和具有 mbedTLS 的 FreeRTOS TLS 层

以下依赖项图显示了 FreeRTOS 安全套接字库包含的参考实施。此参考实施通过以太网和 Wi-Fi 支持 TLS 和 TCP/IP，并将 FreeRTOS+TCP 和 mbedTLS 作为依赖项。有关 FreeRTOS TLS 层的更多信息，请参阅传输层安全。



特征

FreeRTOS 安全套接字库的功能包括：

- 一个基于 Berkeley 套接字的标准接口
- 线程安全 APIs ，用于发送和接收数据
- Easy-to-enable TLS

故障排除

错误代码

FreeRTOS 安全套接字库返回的错误代码为负值。有关每个错误代码的更多信息，请参阅安全套接字 API 参考中的安全套接字错误代码。

> **Note**
>
> 如果 FreeRTOS 安全套接字 API 返回错误代码，则 coreMQTT 库（依赖于 FreeRTOS 安全套接字库）返回错误代码 `AWS_IOT_MQTT_SEND_ERROR`。

开发人员支持

FreeRTOS 安全套接字库包含两个用于处理 IP 地址的辅助标记宏：

`SOCKETS_inet_addr_quick`

此宏将表示为四个独立数字八位字节的 IP 地址转换为按网络字节顺序表示为 32 位数字的 IP 地址。

`SOCKETS_inet_ntoa`

此宏将按网络字节顺序表示为 32 位数字的 IP 地址转换为用十进制表示的字符串。

使用限制

FreeRTOS 安全套接字库仅支持 TCP 套接字。不支持 UDP 套接字。

除了服务器端 APIs API 的轻量级 IP (LWIP) 实现之外，FreeRTOS 安全套接字库不支持服务器。`Bind`支持 APIs 客户端。

初始化

要使用 FreeRTOS 安全套接字库，您需要初始化该库及其依赖项。要初始化安全套接字库，请在应用程序中使用以下代码：

```
BaseType_t xResult = pdPASS;
xResult = SOCKETS_Init();
```

必须单独初始化相关库。例如，如果 FreeRTOS+TCP 是一个依赖项，则您还需要在应用程序中调用 `FreeRTOS_IPInit`。

API 参考

有关完整 API 参考，请参阅安全套接字 API 参考。

示例用法

以下代码可将客户端连接到服务器。

```
#include "aws_secure_sockets.h"

#define configSERVER_ADDR0                    127
#define configSERVER_ADDR1                    0
#define configSERVER_ADDR2                    0
#define configSERVER_ADDR3                    1
#define configCLIENT_PORT                     443

/* Rx and Tx timeouts are used to ensure the sockets do not wait too long for
 * missing data. */
static const TickType_t xReceiveTimeOut = pdMS_TO_TICKS( 2000 );
static const TickType_t xSendTimeOut = pdMS_TO_TICKS( 2000 );

/* PEM-encoded server certificate */
/* The certificate used below is one of the Amazon Root CAs.\
Change this to the certificate of your choice. */
static const char cTlsECHO_SERVER_CERTIFICATE_PEM[] =
"-----BEGIN CERTIFICATE-----\n"
"MIIBtjCCAVugAwIBAgITBmyf1XSXNmY/Owua2eiedgPySjAKBggqhkjOPQQDAjA5\n"
"MQswCQYDVQQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwFwYDVQQDExBBbWF6b24g\n"
"Um9vdCBDQSAzMB4XDTE1MDUyNjAwMDAwMFoXDTQwMDUyNjAwMDAwMFowOTELMAkG\n"
"A1UEBhMCVVMxDzANBgNVBAoTBkFtYXpvbjEZMBcGA1UEAxMQQW1hem9uIFJvb3Qg\n"
"Q0EgMzBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABCmXp8ZBf8ANm+gBG1bG8lKl\n"
"ui2yEujSLtf6ycXYqm0fc4E7O5hrOXwzpcVOho6AF2hiRVd9RFgdszflZwjrZt6j\n"
"QjBAMA8GA1UdEwEB/wQFMAMBAf8wDgYDVR0PAQH/BAQDAgGGMB0GA1UdDgQWBBSr\n"
"ttvXBp43rDCGB5Fwx5zEGbF4wDAKBggqhkjOPQQDAgNJADBGAiEA4IWSoxe3jfkr\n"
"BqWTrBqYaGFy+uGh0PsceGCmQ5nFuMQCIQCcAu/xlJyzlvnrxir4tiz+OpAUFteM\n"
"YyRIHN8wfdVoOw==\n"
"-----END CERTIFICATE-----\n";

static const uint32_t ulTlsECHO_SERVER_CERTIFICATE_LENGTH =
 sizeof( cTlsECHO_SERVER_CERTIFICATE_PEM );

void vConnectToServerWithSecureSocket( void )
{
    Socket_t xSocket;
    SocketsSockaddr_t xEchoServerAddress;
    BaseType_t xTransmitted, lStringLength;

    xEchoServerAddress.usPort = SOCKETS_htons( configCLIENT_PORT );
```

```
    xEchoServerAddress.ulAddress = SOCKETS_inet_addr_quick( configSERVER_ADDR0,
                                                            configSERVER_ADDR1,
                                                            configSERVER_ADDR2,
                                                            configSERVER_ADDR3 );

    /* Create a TCP socket. */
    xSocket = SOCKETS_Socket( SOCKETS_AF_INET, SOCKETS_SOCK_STREAM,
SOCKETS_IPPROTO_TCP );
    configASSERT( xSocket != SOCKETS_INVALID_SOCKET );

    /* Set a timeout so a missing reply does not cause the task to block indefinitely.
*/
    SOCKETS_SetSockOpt( xSocket, 0, SOCKETS_SO_RCVTIMEO, &xReceiveTimeOut,
sizeof( xReceiveTimeOut ) );
    SOCKETS_SetSockOpt( xSocket, 0, SOCKETS_SO_SNDTIMEO, &xSendTimeOut,
sizeof( xSendTimeOut ) );

    /* Set the socket to use TLS. */
    SOCKETS_SetSockOpt( xSocket, 0, SOCKETS_SO_REQUIRE_TLS, NULL, ( size_t ) 0 );
    SOCKETS_SetSockOpt( xSocket, 0, SOCKETS_SO_TRUSTED_SERVER_CERTIFICATE,
cTlsECHO_SERVER_CERTIFICATE_PEM, ulTlsECHO_SERVER_CERTIFICATE_LENGTH );

    if( SOCKETS_Connect( xSocket, &xEchoServerAddress, sizeof( xEchoServerAddress ) )
== 0 )
    {
        /* Send the string to the socket. */
        xTransmitted = SOCKETS_Send( xSocket,                         /* The socket
receiving. */
                                     ( void * )"some message",        /* The data being
sent. */
                                     12,                              /* The length of
the data being sent. */
                                     0 );                             /* No flags. */

        if( xTransmitted < 0 )
        {
            /* Error while sending data */
            return;
        }

        SOCKETS_Shutdown( xSocket, SOCKETS_SHUT_RDWR );
    }
    else
    {
```

```
        //failed to connect to server
    }

    SOCKETS_Close( xSocket );
}
```

有关完整示例，请参阅安全套接字 Echo 客户端演示。

移植

FreeRTOS 安全套接字依赖于 TCP/IP 堆栈和 TLS 实施。根据您的堆栈，要移植安全套接字库，您可能需要移植以下项目中的部分项目：

- FreeRTOS+TCP TCP/IP 堆栈
- 这些区域有：核心PKCS11 库
- 这些区域有：传输层安全

有关移植的更多信息，请参阅《FreeRTOS 移植指南》中的移植安全套接字库。

AWS IoT Device Shadow 库

> **Note**
>
> 此页面上的内容可能不是 up-to-date。有关最新更新，请参阅 FreeRTOS.org 库页面。

简介

您可以使用 Dev AWS IoT ice Shadow 库来存储和检索每台已注册设备的当前状态（影子）。设备影子是设备的永久性虚拟表示形式，即使设备处于离线状态，您也可以在 Web 应用程序中与之交互。设备状态以影子的形式进行捕获，并保存在 JSON 文档中。您可以通过 MQTT 或 HTTP 向 Device Shadow 服务发送命令以查询最新的已知设备状态或更改状态。 AWS IoT 每个设备影子都由相应事物的名称唯一标识，该名称表示 AWS 云上的特定设备或逻辑实体。有关更多信息，请参阅使用 AWS IoT管理事物。有关影子的更多详细信息，请参阅 AWS IoT 文档。

除了标准 C 库之外，Dev AWS IoT ice Shadow 库不依赖其他库。该库没有平台依赖关系，例如线程或同步。它可以与任何 MQTT 库以及任何 JSON 一起使用。

该库可免费使用，并根据 MIT 开源许可证分发。

Dev AWS IoT ice Shadow 的代码大小（使用 GCC 为 ARM Cortex-M 生成的示例）		
文件	使用 -O1 优化	使用 -Os 优化
shadow.c	1.2K	0.9K
估计总数	1.2K	0.9K

AWS IoT 职位库

> **Note**
>
> 此页面上的内容可能不是 up-to-date。有关最新更新，请参阅 FreeRTOS.org 库页面。

简介

AWS IoT 作业是一项将待处理任务通知一台或多台连接的设备的服务。您可以使用作业来管理设备队列、更新设备上的固件和安全证书，或者执行管理任务，例如重新启动设备和执行诊断。有关更多信息，请参阅《AWS IoT 开发人员指南》中的作业。与 AWS IoT Jobs 服务的交互使用 MQTT，这是一种轻量级的发布订阅协议。该库提供了一个 API 来撰写和识别 AWS IoT Jobs 服务使用的 MQTT 主题字符串。

AWS IoT Jobs 库以 C 语言编写，旨在符合 ISO C90 和 MIS RA C: 2012。除了标准 C 库之外，该库不依赖于任何其他库。它可以与任何 MQTT 库以及任何 JSON 一起使用。该库有证据表明内存使用安全，没有堆分配，因此适用于 IoT 微控制器，但也完全可移植到其他平台。

该库可免费使用，并根据 MIT 开源许可证分发。

AWS IoT 任务的代码大小（使用 GCC 为 ARM Cortex-M 生成的示例）		
文件	使用 -O1 优化	使用 -Os 优化
jobs.c	1.9K	1.6K
估计总数	1.9K	1.6K

传输层安全

> **Important**
>
> 该库托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

FreeRTOS 传输层安全性协议 (TLS) 接口是一个可选的精简包装器，用于从协议堆栈中位于它上层的安全套接字层 (SSL) 接口提取加密实施详细信息。TLS 接口的用途在于，使用 TLS 协议协商和加密基元的其他实施，轻松替换当前软件加密库 mbed TLS。无需对 SSL 接口做任何更改，即可换出 TLS 接口。请参阅 FreeRTOS 源代码存储库中的`iot_tls.h`。

TLS 接口是可选的，因为可以选择从 SSL 直接连接到加密库。接口不能用于包括 TLS 和网络传输全堆栈卸载实施的 MCU 解决方案。

有关移植 TLS 接口的更多信息，请参阅《FreeRTOS 移植指南》中的移植 TLS 库。

Wi-Fi 库

> **Important**
>
> 该库托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

概览

FreeRTOS Wi-Fi 库将端口特定的 Wi-Fi 实现抽象为一个常用 API，后者使用 Wi-Fi 功能简化了针对所有符合 FreeRTOS 要求的主板的应用程序开发和移植。通过使用此常用 API，应用程序可通过常用接口与其低级别无线堆栈进行通信。

依赖项和要求

FreeRTOS Wi-Fi 库需要 FreeRTOS+TCP 内核。

特征

Wi-Fi 库包括以下功能：

- Support 支持 WEP、WPA 和身份 WPA2验证 WPA3
- 接入点扫描
- 电源管理
- 网络分析

有关 Wi-Fi 库的功能的更多信息，请见下文。

Wi-Fi 模式

Wi-Fi 设备可以处于以下三种模式之一：工作站、接入点或 P2P。可以调用 `WIFI_GetMode` 获取 Wi-Fi 设备的当前模式。可通过调用 `WIFI_SetMode` 设置设备的 Wi-Fi 模式。如果设备已经与网络连接，则通过调用 `WIFI_SetMode` 切换模式会断开设备的连接。

工作站模式

将您的设备设置为工作站模式以将主板连接到现有接入点。

接入点 (AP) 模式

将您的设备设置为 AP 模式可使设备成为一个可供其他设备连接到的接入点。当设备处于 AP 模式下时，可以将另一个设备连接到 FreeRTOS 设备，并配置新的 Wi-Fi 凭证。要配置 AP 模式，可调用 `WIFI_ConfigureAP`。要将设备置于 AP 模式下，请调用 `WIFI_StartAP`。要关闭 AP 模式，请调用 `WIFI_StopAP`。

> **Note**
>
> FreeRTOS 库在 AP 模式下不提供 Wi-Fi 配置。您必须提供其他功能，包括 DHCP 和 HTTP 服务器功能，才能完全支持 AP 模式。

P2P 模式

将您的设备设置为 P2P 模式可允许多个设备直接互连，而无需接入点。

安全性

Wi-Fi API 支持 WEP、WPA 和 WPA3 安全类型。 WPA2如果设备处于工作站模式下，则在调用 `WIFI_ConnectAP` 函数时，您必须指定网络安全类型。如果设备处于 AP 模式下，则可以配置设备以使用任何支持的安全类型：

- eWiFiSecurityOpen
- eWiFiSecurityWEP
- eWiFiSecurityWPA
- eWiFiSecurityWPA2
- eWiFiSecurityWPA3

扫描和连接

要扫描附近的接入点，请将设备设置为工作站模式，并调用 WIFI_Scan 函数。如果在扫描中找到了所需的网络，则可以通过调用 WIFI_ConnectAP 并提供网络凭证来连接到该网络。通过调用 WIFI_Disconnect，可以断开 Wi-Fi 设备与网络的连接。有关扫描和连接的更多信息，请参阅示例用法和API 参考。

电源管理

不同 Wi-Fi 设备具有不同的电源要求，具体取决于应用程序和可用电源。设备可能始终通着电以缩短延迟，也可能间歇性连接并在不需要 Wi-Fi 时切换到低功耗模式。接口 API 支持各种电源管理模式，如常开、低功耗和正常模式。可以使用 WIFI_SetPMMode 函数，为设备设置电源模式。可以调用 WIFI_GetPMMode 函数，获取设备当前的电源模式。

网络配置文件

可使用 Wi-Fi 库将网络配置文件保存在设备的非易失性存储中。这样一来，您可以保存网络设置，以便当设备重新连接到 Wi-Fi 网络时可以进行检索，从而无需在设备连接到网络后再次预配置设备。WIFI_NetworkAdd 用于添加网络配置文件。WIFI_NetworkGet 用于检索网络配置文件。WIFI_NetworkDel 用于删除网络配置文件。可以保存的配置文件数量取决于平台。

配置

要使用 Wi-Fi 库，您需要在配置文件中定义几个标识符。有关这些标识符的信息，请参阅API 参考。

> **Note**
>
> 该库不包含所需的配置文件。您必须创建一个配置文件。在创建配置文件时，请务必包含主板所需的任何特定于主板的配置标识符。

初始化

在使用 Wi-Fi 库之前，您需要初始化一些特定于主板的组件以及 FreeRTOS 组件。使用 `vendors/vendor/boards/board/aws_demos/application_code/main.c` 文件作为初始化模板时，请执行以下操作：

1. 如果应用程序处理 Wi-Fi 连接，请删除 `main.c` 中的示例 Wi-Fi 连接逻辑。替换以下 `DEMO_RUNNER_RunDemos()` 函数调用：

   ```
   if( SYSTEM_Init() == pdPASS )
       {
       ...
           DEMO_RUNNER_RunDemos();
       ...
           }
   ```

   对于针对您自己的应用程序的调用：

   ```
   if( SYSTEM_Init() == pdPASS )
       {
       ...
           // This function should create any tasks
           // that your application requires to run.
           YOUR_APP_FUNCTION();
       ...
           }
   ```

2. 调用 `WIFI_On()` 以初始化和启动您的 Wi-Fi 芯片。

   > **Note**
   >
   > 一些主板可能需要额外的硬件初始化。

3. 将已配置的 `WIFINetworkParams_t` 结构传递到 `WIFI_ConnectAP()` 以将主板连接到可用的 Wi-Fi 网络。有关 `WIFINetworkParams_t` 结构的更多信息，请参阅示例用法和API 参考。

API 参考

有关完整 API 参考，请参阅 Wi-Fi API 参考。

示例用法

连接到已知 AP

```
#define clientcredentialWIFI_SSID    "MyNetwork"
#define clientcredentialWIFI_PASSWORD   "hunter2"

WIFINetworkParams_t xNetworkParams;
WIFIReturnCode_t xWifiStatus;

xWifiStatus = WIFI_On(); // Turn on Wi-Fi module

// Check that Wi-Fi initialization was successful
if( xWifiStatus == eWiFiSuccess )
{
    configPRINT( ( "WiFi library initialized.\n") );
}
else
{
    configPRINT( ( "WiFi library failed to initialize.\n" ) );
    // Handle module init failure
}

/* Setup parameters. */
xNetworkParams.pcSSID = clientcredentialWIFI_SSID;
xNetworkParams.ucSSIDLength = sizeof( clientcredentialWIFI_SSID );
xNetworkParams.pcPassword = clientcredentialWIFI_PASSWORD;
xNetworkParams.ucPasswordLength = sizeof( clientcredentialWIFI_PASSWORD );
xNetworkParams.xSecurity = eWiFiSecurityWPA2;

// Connect!
xWifiStatus = WIFI_ConnectAP( &( xNetworkParams ) );

if( xWifiStatus == eWiFiSuccess )
{
    configPRINT( ( "WiFi Connected to AP.\n" ) );
    // IP Stack will receive a network-up event on success
}
else
{
    configPRINT( ( "WiFi failed to connect to AP.\n" ) );
    // Handle connection failure
}
```

正在扫描附近 APs

```
WIFINetworkParams_t xNetworkParams;
WIFIReturnCode_t xWifiStatus;

configPRINT( ("Turning on wifi...\n") );
xWifiStatus = WIFI_On();

configPRINT( ("Checking status...\n") );
if( xWifiStatus == eWiFiSuccess )
{
    configPRINT( ("WiFi module initialized.\n") );
}
else
{
    configPRINTF( ("WiFi module failed to initialize.\n" ) );
    // Handle module init failure
}

WIFI_SetMode(eWiFiModeStation);

/* Some boards might require additional initialization steps to use the Wi-Fi library.
 */

while (1)
{
    configPRINT( ("Starting scan\n") );
    const uint8_t ucNumNetworks = 12; //Get 12 scan results
    WIFIScanResult_t xScanResults[ ucNumNetworks ];
    xWifiStatus = WIFI_Scan( xScanResults, ucNumNetworks ); // Initiate scan

    configPRINT( ("Scan started\n") );

    // For each scan result, print out the SSID and RSSI
    if ( xWifiStatus == eWiFiSuccess )
    {
        configPRINT( ("Scan success\n") );
        for ( uint8_t i=0; i<ucNumNetworks; i++ )
        {
            configPRINTF( ("%s : %d \n", xScanResults[i].cSSID,
 xScanResults[i].cRSSI) );
        }
    } else {
        configPRINTF( ("Scan failed, status code: %d\n", (int)xWifiStatus) );
```

```
    }

    vTaskDelay(200);
}
```

移植

`iot_wifi.c` 实现需要实施 `iot_wifi.h` 中定义的函数。对于任何不必要的或不受支持的函数，此实现至少需要返回 `eWiFiNotSupported`。

有关移植 Wi-Fi 库的更多信息，请参阅《FreeRTOS 移植指南》中的移植 Wi-Fi 库。

FreeRTOS 演示

FreeRTOS 将一些演示应用程序包含在主 FreeRTOS 目录下的 `demos` 文件夹中。可由 FreeRTOS 执行的所有示例均显示在 `demos` 下的 `common` 文件夹中。对于每个符合 FreeRTOS 条件的平台，`demos` 文件夹下还有一个文件夹。

尝试演示应用程序之前，建议您完成 开始使用 FreeRTOS中的教程。它会介绍如何设置和运行 coreMQTT 代理演示。

运行 FreeRTOS 演示

以下主题介绍如何设置和运行 FreeRTOS 演示：

- 低功耗蓝牙演示应用程序
- Microchip 好奇号 MZEF 的演示引导加载程序 PIC32
- AWS IoT Device Defender 演示
- AWS IoT Greengrass V1 发现演示应用程序
- AWS IoT Greengrass V2
- coreHTTP 演示
- AWS IoT 作业库演示
- coreMQTT 演示
- Over-the-air 更新演示应用程序
- 安全套接字 Echo 客户端演示
- AWS IoT Device Shadow 演示应用程序

DEMO_RUNNER_RunDemos 函数位于 *freertos*/demos/demo_runner/iot_demo_runner.c 文件中，用于初始化一个分离的线程，该线程上运行单个演示应用程序。默认情况下，DEMO_RUNNER_RunDemos 仅调用和启动 coreMQTT 代理演示。根据您在下载 FreeRTOS 时选择的配置，以及从哪个位置下载 FreeRTOS，其他示例运行程序函数可能会默认启动。要启用演示应用程序，请打开 *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h 文件，然后定义要运行的演示。

> **Note**
>
> 并非所有示例组合在一起都有效。根据组合，由于内存约束，软件可能无法在所选目标上运行。建议您一次运行一个演示。

配置演示

为您配置了演示，以便快速开始。您可能希望更改项目的一些配置，以创建在您平台上运行的版本。您可在 vendors/*vendor*/boards/*board*/aws_demos/config_files 中找到配置文件。

低功耗蓝牙演示应用程序

> **Important**
>
> 该演示托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

概览

FreeRTOS 低功耗蓝牙包括三个演示应用程序：

- 低功耗蓝牙 MQTT 演示

  此应用程序演示如何使用低功耗蓝牙 MQTT 服务。

- Wi-Fi 预置 演示

  此应用程序演示如何使用低功耗蓝牙预置服务。

- 通用属性服务器 演示

此应用程序演示如何使用 FreeRTOS 低功耗蓝牙 APIs 中间件来创建简单的 GATT 服务器。

> **Note**
>
> 要设置和运行 FreeRTOS 演示，请按照开始使用 FreeRTOS中的步骤操作。

先决条件

要按照这些演示练习，您的微控制器需要具备低功耗蓝牙功能。您还需要 适用于 FreeRTOS 蓝牙设备的 iOS 开发工具包 或 适用于 FreeRTOS 蓝牙设备的 Android 开发工具包。

为 FreeRTOS 低功耗蓝牙设置 AWS IoT 和亚马逊 Cognito

要通过 MQTT 将您的设备连接到 AWS IoT ，您需要设置 AWS IoT 和 Amazon Cognito。

要设置 AWS IoT

1. 在 https://aws.amazon.com/上 AWS 开设一个账户。
2. 打开 AWS IoT 控制台，从导航窗格中，依次选择管理和事物。
3. 选择创建，然后选择创建单个事物。
4. 输入您设备的名称，然后选择下一步。
5. 如果您通过移动设备将微控制器连接到云中，请选择创建没有证书的事物。由于移动设备 SDKs 使用 Amazon Cognito 进行设备身份验证，因此您无需为使用低功耗蓝牙的演示创建设备证书。

   如果您直接通过 Wi-Fi 将微控制器连接到云中，请依次选择创建证书和激活，然后下载事物的证书、公有密钥和私有密钥。
6. 从已注册事物列表中选择您刚刚创建的事物，然后从事物的页面中选择交互。记下 AWS IoT REST API 端点。

有关设置的更多信息，请参阅《入门》 AWS IoT。

创建 Amazon Cognito 用户群体

1. 打开 Amazon Cognito 控制台，然后选择管理用户群体。
2. 选择创建用户池。
3. 指定用户池名称，然后选择查看默认值。

4. 在导航窗格中，选择应用程序客户端，然后选择添加应用程序客户端。
5. 输入应用程序客户端的名称，然后选择创建应用程序客户端。
6. 在导航窗格中，选择 审核，然后选择 创建池。

   记录在您用户池的常规设置页面中显示的池 ID。
7. 在导航窗格中，选择应用程序客户端，然后选择显示详细信息。记录应用程序客户端 ID 和应用程序客户端密钥。

创建 Amazon Cognito 身份池

1. 打开 Amazon Cognito 控制台，然后选择管理身份池。
2. 为身份池输入一个名称。
3. 展开身份验证提供商，选择 Cognito 选项卡，然后输入您的用户池 ID 和应用程序客户端 ID。
4. 选择创建池。
5. 展开查看详细信息，记下两个 IAM 角色名称。要访问 Amazon Cognito，请选择允许，以便为经过身份验证和未经过身份验证的身份创建 IAM 角色。
6. 选择编辑身份池。记录身份池 ID。其格式应为 `us-west-2:12345678-1234-1234-1234-123456789012`。

有关设置 Amazon Cognito 的更多信息，请参阅 Amazon Cognito 入门。

创建 IAM policy 并将其附加到经过身份验证的身份

1. 打开 IAM 控制台，然后在导航窗格中选择角色。
2. 查找并选择您的经过身份验证的角色，然后依次选择附加策略和添加内联策略。
3. 选择 JSON 选项卡，然后粘贴以下 JSON：

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "iot:AttachPolicy",
            "iot:AttachPrincipalPolicy",
            "iot:Connect",
```

```
            "iot:Publish",
            "iot:Subscribe",
            "iot:Receive",
            "iot:GetThingShadow",
            "iot:UpdateThingShadow",
            "iot:DeleteThingShadow"
         ],
         "Resource":[
            "*"
         ]
      }
   ]
}
```

4. 选择查看策略，输入策略的名称，然后选择创建策略。

请随身携 AWS IoT 带您和亚马逊 Cognito 的信息。您需要端点并使用 AWS 云端 IDs 对您的移动应用程序进行身份验证。

为低功耗蓝牙设置 FreeRTOS 环境

要设置环境，您需要在微控制器下载包含低功耗蓝牙库 的 FreeRTOS，并在移动设备上下载和配置适用于 FreeRTOS 蓝牙设备的移动开发工具包。

在微控制器环境中设置 FreeRTOS 低功耗蓝牙

1. 从中下载或克隆 FreeRTOS。GitHub有关说明，请参阅 README.md 文件。
2. 在微控制器上设置 FreeRTOS。

   有关在符合 FreeRTOS 条件的微控制器上开始使用 FreeRTOS 的信息，请参阅 FreeRTOS 入门中有关您主板的指南。

> **Note**
>
> 您可在任何已启用低功耗蓝牙且具有 FreeRTOS 和移植的 FreeRTOS 低功耗蓝牙库的微控制器上运行演示。目前，FreeRTOS 低功耗蓝牙 MQTT 演示项目已完全移植到以下启用低功耗蓝牙的设备：
>
> - Espressif ESP32-DevKit C 和 the ESP-WROVER-KIT
> - 北欧 n RF5284 0-DK

常见组件

FreeRTOS 演示应用程序有两个常见组件：

- Network Manager
- 低功耗蓝牙移动开发工具包演示应用程序

Network Manager

网络管理器管理微控制器的网络连接。它位于您的 FreeRTOS 目录中，路径为：demos/network_manager/aws_iot_network_manager.c。如果为 Wi-Fi 和低功耗蓝牙均启用了网络管理器，则默认情况下通过低功耗蓝牙启动演示。如果低功耗蓝牙连接中断，而您的主板已中断 Wi-Fi-enabled，则网络管理器会切换到可用的 Wi-Fi 连接，以防止您断开与网络的连接。

要在 Network Manager 中启用网络连接类型，请将网络连接类型添加到中的configENABLED_NETWORKS参数中vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_iot_network_config.h（其中，*board*是供应商名称，是您用于运行演示的主板的名称）。*vendor*

例如，如果您同时启用了低功耗蓝牙和 Wi-Fi，则 aws_iot_network_config.h 中以 #define configENABLED_NETWORKS 开头的一行应如下所示：

```
#define  configENABLED_NETWORKS  ( AWSIOT_NETWORK_TYPE_BLE | AWSIOT_NETWORK_TYPE_WIFI )
```

要获取当前支持的网络连接类型列表，请查看 aws_iot_network.h 中以 #define AWSIOT_NETWORK_TYPE 开头的行。

FreeRTOS 低功耗蓝牙移动开发工具包演示应用程序

FreeRTOS 低功耗蓝牙移动 SDK 演示应用程序位于 FreeRTOS 蓝牙设备的 Android SDK 下，FreeRTOS 蓝牙设备amazon-freertos-ble-android-sdk/app的 iOS SDK 位于 GitHub 下方。amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo在本示例中，我们使用 iOS 版本的演示移动应用程序的屏幕截图。

> **Note**
>
> 如果您使用的是 iOS 设备，则需要 Xcode 来构建演示移动应用程序。如果您使用的是 Android 设备，则可使用 Android Studio 来构建演示移动应用程序。

配置 iOS 开发工具包演示应用程序

当您定义配置变量时，使用在配置文件中提供的占位符值的格式。

1. 确认已安装 适用于 FreeRTOS 蓝牙设备的 iOS 开发工具包。
2. 从 amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/ 发布以下命令：

```
$ pod install
```

3. 打开具有 Xcode 的 amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/AmazonFreeRTOSDemo.xcworkspace 项目，将签名开发人员账户更改为您的账户。
4. 在您所在的地区创建 AWS IoT 政策（如果您还没有）。

> **Note**
>
> 此策略不同于为经过 Amazon Cognito 身份验证的身份创建的 IAM policy。

   a. 打开 AWS IoT 管理控制台。
   b. 在导航窗格中依次选择安全、策略和创建。输入用于标识您的策略的名称。在添加语句部分中，选择高级模式。将以下 JSON 复制并粘贴到策略编辑器窗口中。*aws-account*用您的 AWS 地区*aws-region*和账户 ID 替换和。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:Connect",
            "Resource":"arn:aws:iot:region:account-id:*"
        },
        {
            "Effect": "Allow",
            "Action": "iot:Publish",
            "Resource": "arn:aws:iot:region:account-id:*"
        },
        {
             "Effect": "Allow",
             "Action": "iot:Subscribe",
             "Resource": "arn:aws:iot:region:account-id:*"
```

```
        },
        {
             "Effect": "Allow",
             "Action": "iot:Receive",
             "Resource": "arn:aws:iot:region:account-id:*"
        }
    ]
}
```

c. 选择创建。

5. 打开 amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/AmazonFreeRTOSDemo/Amazon/AmazonConstants.swift 并重新定义以下变量：

   - region: 您所在 AWS 的地区。
   - iotPolicyName: 您的 AWS IoT 保单名称。
   - mqttCustomTopic：您要发布到的 MQTT 主题。

6. 打开 amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/AmazonFreeRTOSDemo/Support/awsconfiguration.json。

   在 CognitoIdentity 下，重新定义以下变量：

   - PoolId：您的 Amazon Cognito 身份池 ID。
   - Region: 您所在 AWS 的地区。

   在 CognitoUserPool 下，重新定义以下变量：

   - PoolId：您的 Amazon Cognito 用户群体 ID。
   - AppClientId：您的应用程序客户端 ID。
   - AppClientSecret：您的应用程序客户端密钥。
   - Region: 您所在 AWS 的地区。

配置 Android 开发工具包演示应用程序

当您定义配置变量时，使用在配置文件中提供的占位符值的格式。

1. 确认已安装 适用于 FreeRTOS 蓝牙设备的 Android 开发工具包。
2. 在您所在的地区创建 AWS IoT 政策（如果您还没有）。

> **Note**
>
> 此策略不同于为经过 Amazon Cognito 身份验证的身份创建的 IAM policy。

a. 打开 AWS IoT 管理控制台。

b. 在导航窗格中依次选择安全、策略和创建。输入用于标识您的策略的名称。在添加语句部分中，选择高级模式。将以下 JSON 复制并粘贴到策略编辑器窗口中。*aws-account*用您的 AWS 地区*aws-region*和账户 ID 替换和。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:Connect",
            "Resource":"arn:aws:iot:region:account-id:*"
        },
        {
            "Effect": "Allow",
            "Action": "iot:Publish",
            "Resource": "arn:aws:iot:region:account-id:*"
        },
        {
             "Effect": "Allow",
             "Action": "iot:Subscribe",
             "Resource": "arn:aws:iot:region:account-id:*"
        },
        {
             "Effect": "Allow",
             "Action": "iot:Receive",
             "Resource": "arn:aws:iot:region:account-id:*"
        }
    ]
}
```

c. 选择创建。

3. 打开 https://github.com/aws/amazon-freertos-ble-android-sdk/blob/master/app/src/main/java/software/amazon/freertos/demo/DemoConstants .java 并重新定义以下变量：

- AWS_IOT_POLICY_NAME: 您的 AWS IoT 保单名称。
- AWS_IOT_REGION: 您所在 AWS 的地区。

4. 打开 https://github.com/aws/amazon-freertos-ble-android-sdk/blob/master/app/src/main/res/raw/awsconfiguration .json。

   在 CognitoIdentity 下，重新定义以下变量：

   - PoolId：您的 Amazon Cognito 身份池 ID。
   - Region: 您所在 AWS 的地区。

   在 CognitoUserPool 下，重新定义以下变量：

   - PoolId：您的 Amazon Cognito 用户群体 ID。
   - AppClientId：您的应用程序客户端 ID。
   - AppClientSecret：您的应用程序客户端密钥。
   - Region: 您所在 AWS 的地区。

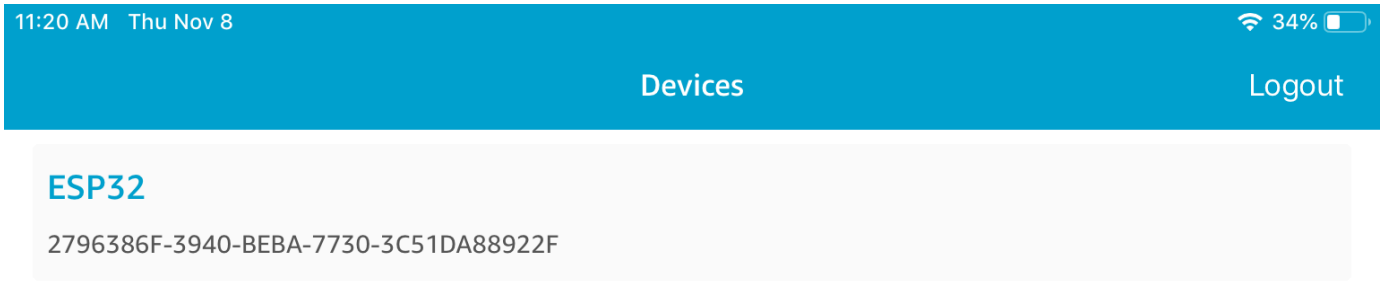
通过低功耗蓝牙发现微控制器并建立安全连接

1. 为了将您的微控制器和移动设备安全配对（步骤 6），您需要一个同时具有输入和输出功能的串行终端仿真器（例如 TeraTerm）。将终端配置为通过串行连接来连接到主板，如安装终端仿真器中所述。
2. 在您的微控制器上运行低功耗蓝牙演示项目。
3. 在移动设备上运行低功耗蓝牙移动开发工具包演示应用程序。

   要在 Android 开发工具包中从命令行启动演示应用程序，请运行以下命令：

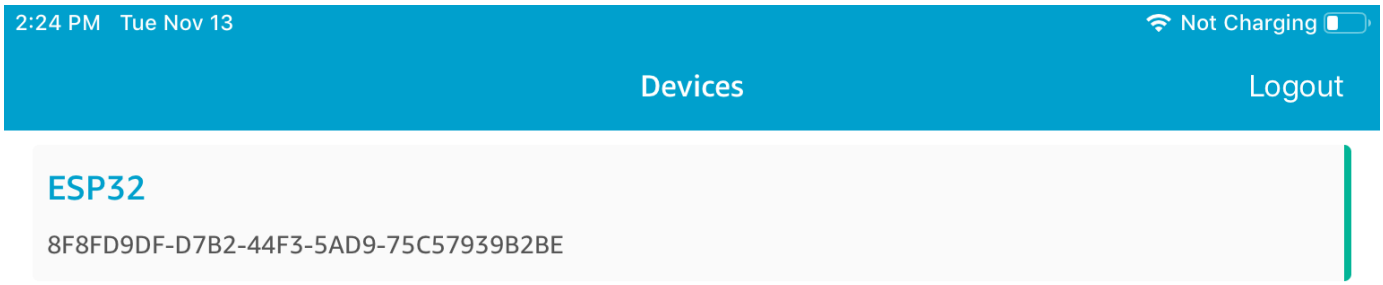
   ```
   $ ./gradlew installDebug
   ```

4. 确认您的微控制器显示在低功耗蓝牙移动开发工具包演示应用程序的 Devices (设备) 下。

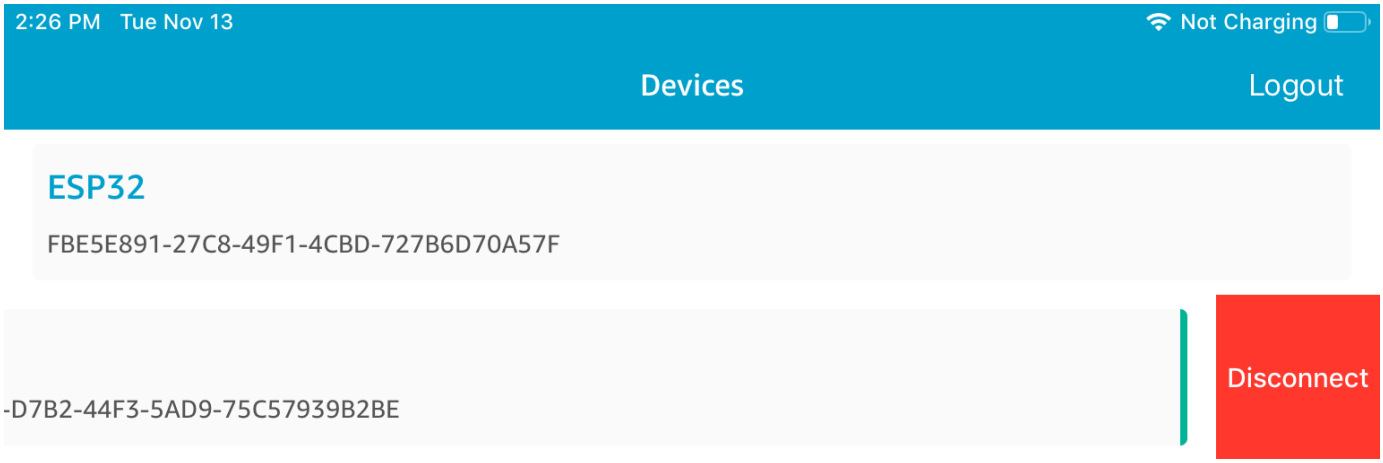


> **Note**
>
> 处于范围内的所有具有 FreeRTOS 的设备以及设备信息服务 (*freertos*/.../device_information) 均会显示在列表中。

5. 从设备列表中选择您的微控制器。应用程序与主板建立连接，在已连接设备的旁边会显示一根绿色线条。

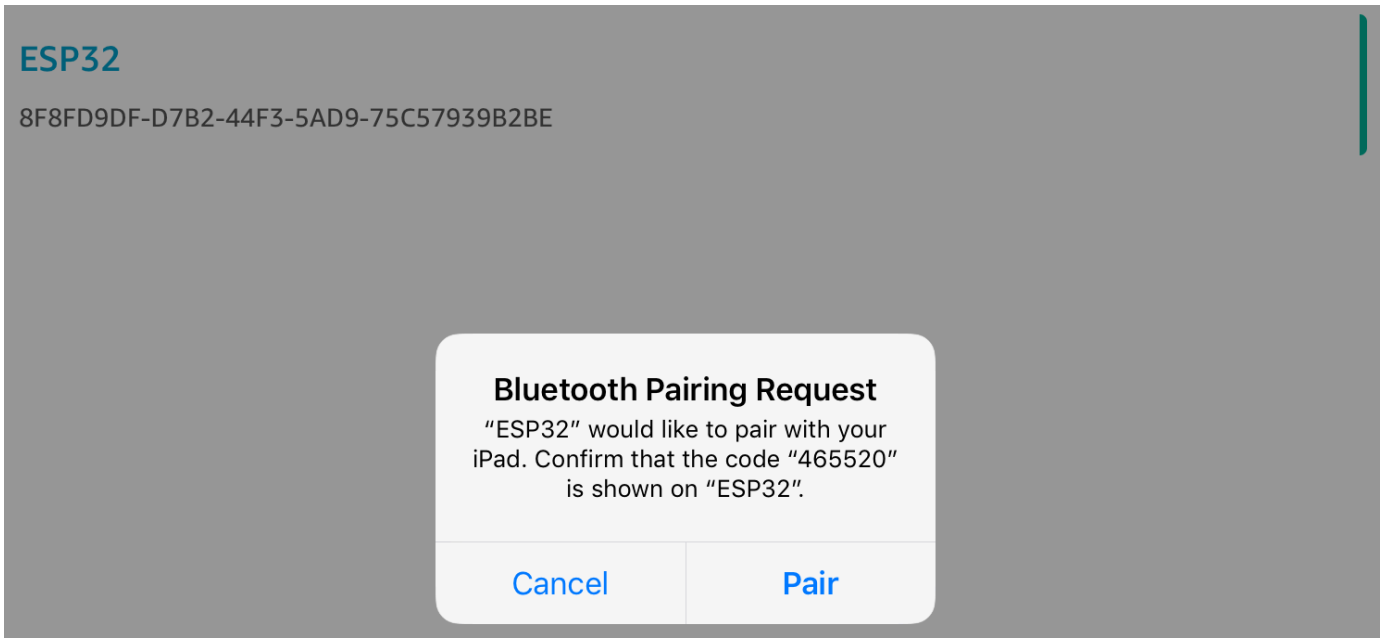


您可以通过将线条拖动到左侧，从微控制器上断开连接。



6. 如果出现提示，请将您的微控制器和移动设备进行配对。

```
24 261107 [Btc_task]  Disconnected from BLE device. Stopping the counter update
25 261108 [Btc_task] Disconnect received for MQTT service instance 0
26 261108 [Btc_task] BLE disconnected with remote device, start advertisement
27 261108 [Btc_task] Started advertisement. Listening for a BLE Connection.
28 261289 [Btc_task] BLE Connected to remote device, connId = 0
E (2614110) BT_GATT: gatts_write_attr_perm_check - GATT_INSUF_AUTHENTICATION: MITM required
E (2614420) BT_SMP: Value for numeric comparison = 465520
29 261412 [uTask] Numeric comparison:465520
30 261412 [uTask] Press 'y' to confirm
```



如果两个设备上用于比较数字的代码相同，则配对这两个设备。

> **Note**
>
> 低功耗蓝牙移动开发工具包演示应用程序使用 Amazon Cognito 进行用户身份验证。确保您已设置 Amazon Cognito 用户和身份池，并且将 IAM policy 附加到了通过身份验证的身份。

低功耗蓝牙 MQTT

在 MQTT 低功耗蓝牙演示中，您的微控制器通过 MQTT 代理向 AWS 云端发布消息。

订阅演示 MQTT 主题

1. 登录 AWS IoT 控制台。
2. 在导航窗格中选择测试，然后选择 MQTT 测试客户端，以便打开 MQTT 客户端。
3. 在 Subscription topic (订阅主题)中，输入 ***thing-name*/example/topic1**，然后选择 Subscribe to topic (订阅主题)。

如果您使用低功耗蓝牙将微控制器与移动设备配对，则通过您移动设备上的低功耗蓝牙移动开发工具包演示应用程序来路由 MQTT 消息。

通过低功耗蓝牙启用演示

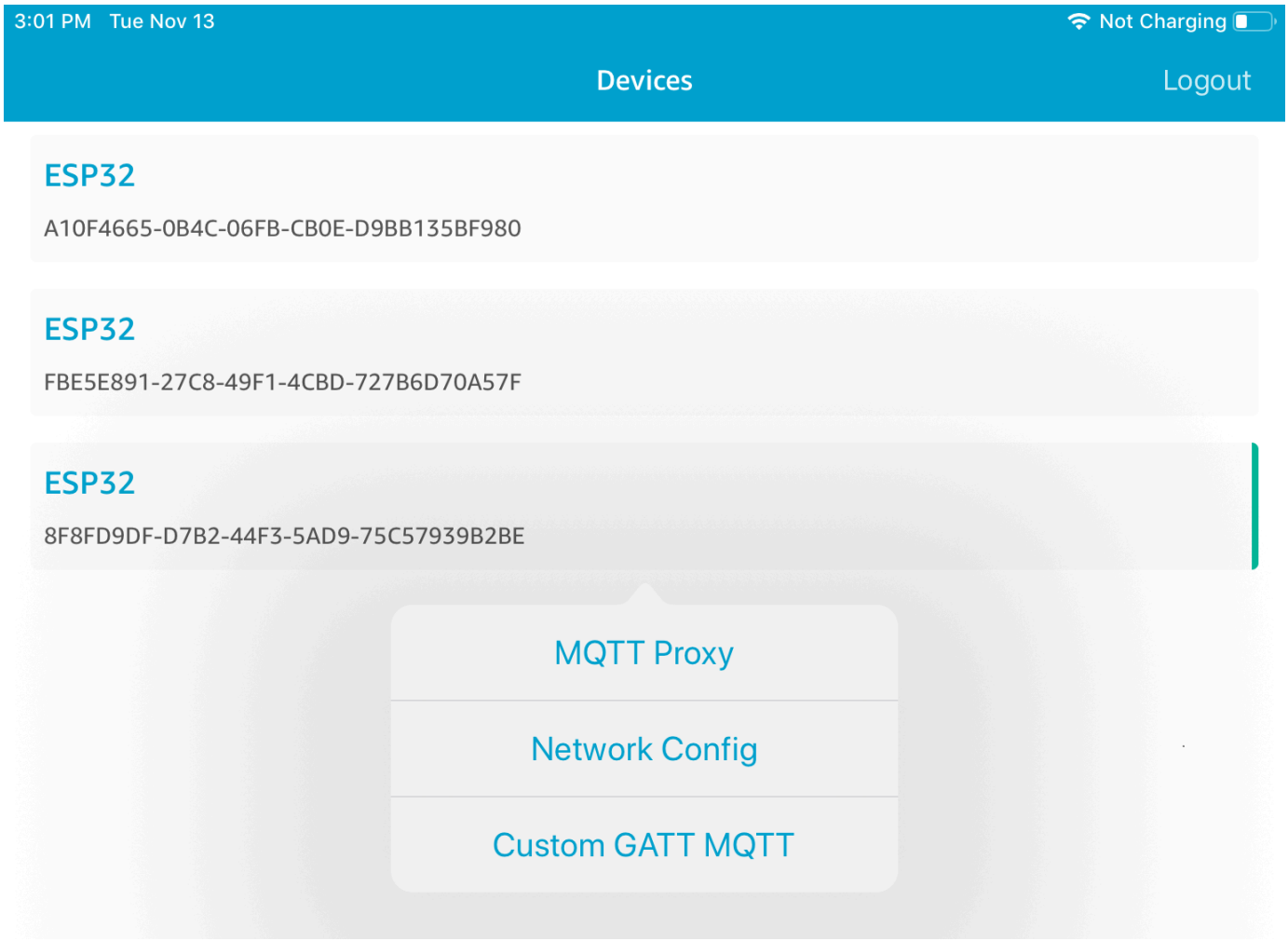
1. 打开 `vendors/vendor/boards/board/aws_demos/config_files/aws_demo_config.h` 并定义 `CONFIG_MQTT_BLE_TRANSPORT_DEMO_ENABLED`。
2. 打开`demos/include/aws_clientcredential.h`，然后`clientcredentialMQTT_BROKER_ENDPOINT`使用 AWS IoT 代理端点进行配置。使用 BLE 微控制器设备的事物名称配置 `clientcredentialIOT_THING_NAME`。 AWS IoT 代理端点可以通过在 AWS IoT 控制台左侧导航窗格中选择 “设置” 获取，也可以通过 CLI 运行命令获取：`aws iot describe-endpoint --endpoint-type=iot:Data-ATS`。

   > **Note**
   >
   > AWS IoT 代理端点和事物名称必须位于配置认知身份和用户池的同一区域。

运行演示

1. 在微控制器上生成并运行演示项目。

2. 确保您已使用 FreeRTOS 低功耗蓝牙移动开发工具包演示应用程序将主板与移动设备配对。
3. 从演示移动应用程序的 Devices (设备) 列表中，选择您的微控制器，然后选择 MQTT Proxy (MQTT 代理) 以打开 MQTT 代理设置。



4. 在启用 MQTT 代理之后，MQTT 消息将显示在 *thing-name*/example/topic1 主题上，并且数据将输出到 UART 终端。

Wi-Fi 预置

Wi-Fi 预置是一项 FreeRTOS 低功耗蓝牙服务，让您可以安全地将 Wi-Fi 网络凭证通过低功耗蓝牙从移动设备发送到微控制器。Wi-Fi 预置服务的源代码位于 *freertos*/.../wifi_provisioning。

> **Note**
>
> Espressi ESP32 f-C 目前支持 Wi-Fi 配置演示 DevKit

启用演示

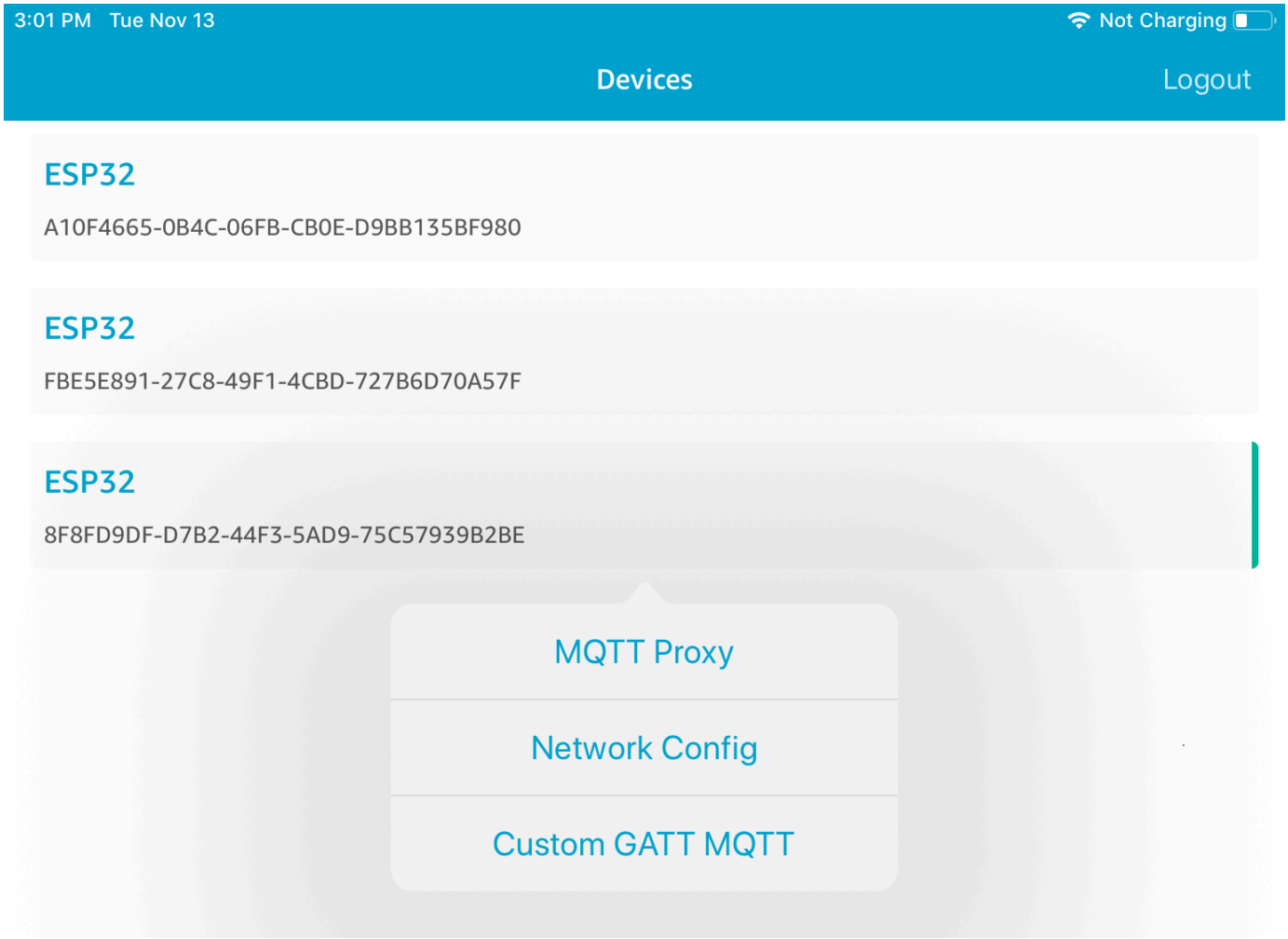
1. 启用 Wi-Fi 预置服务。打开`vendors/vendor/boards/board/aws_demos/config_files/iot_ble_config.h`，然后设置`#define IOT_BLE_ENABLE_WIFI_PROVISIONING`为`1`（其中*vendor*是供应商的名称，*board*是您用来运行演示的主板的名称）。

   > **Note**
   >
   > 默认情况下禁用 Wi-Fi 预置服务。

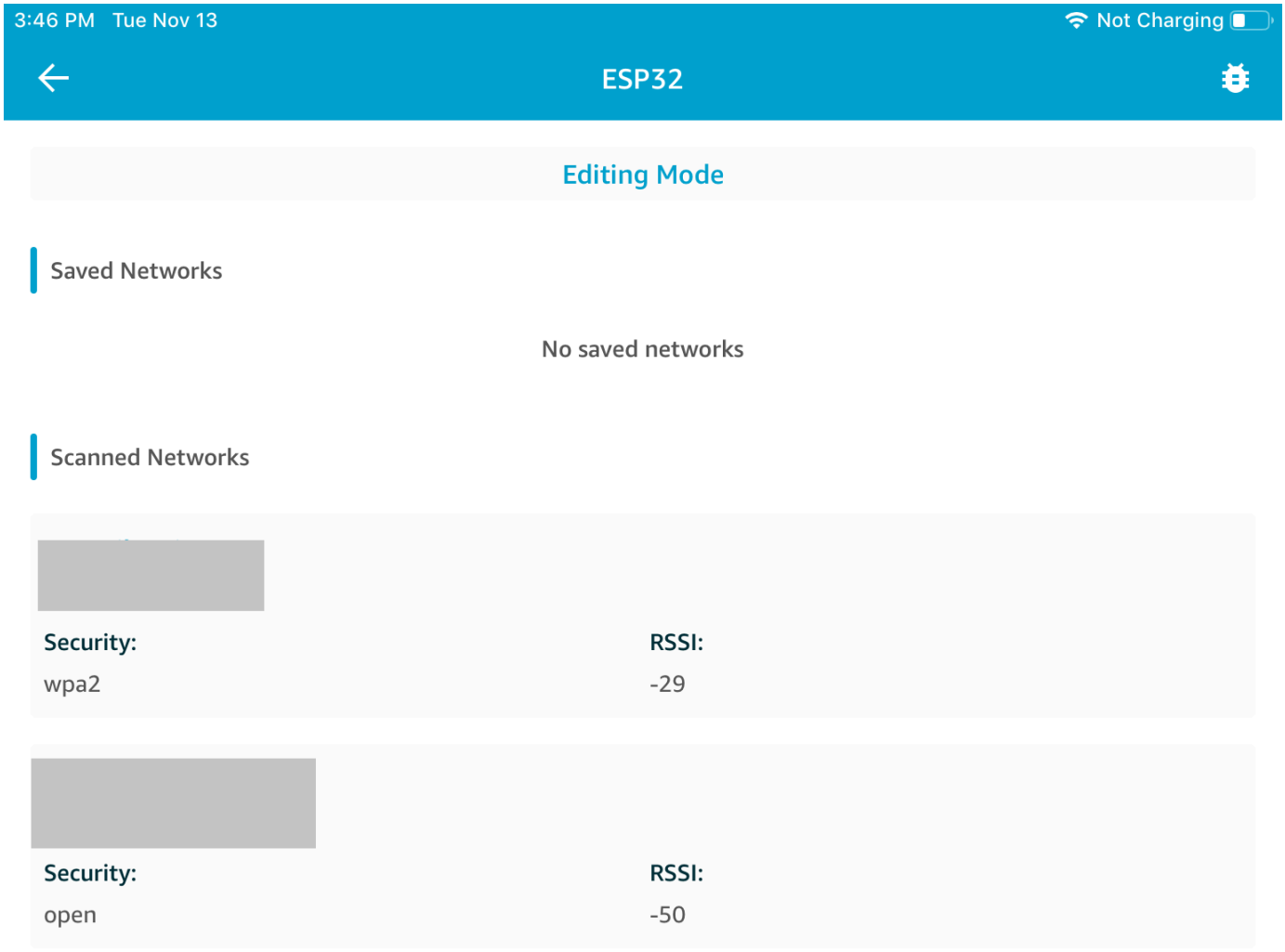
2. 配置 Network Manager 以启用低功耗蓝牙和 Wi-Fi。

运行演示

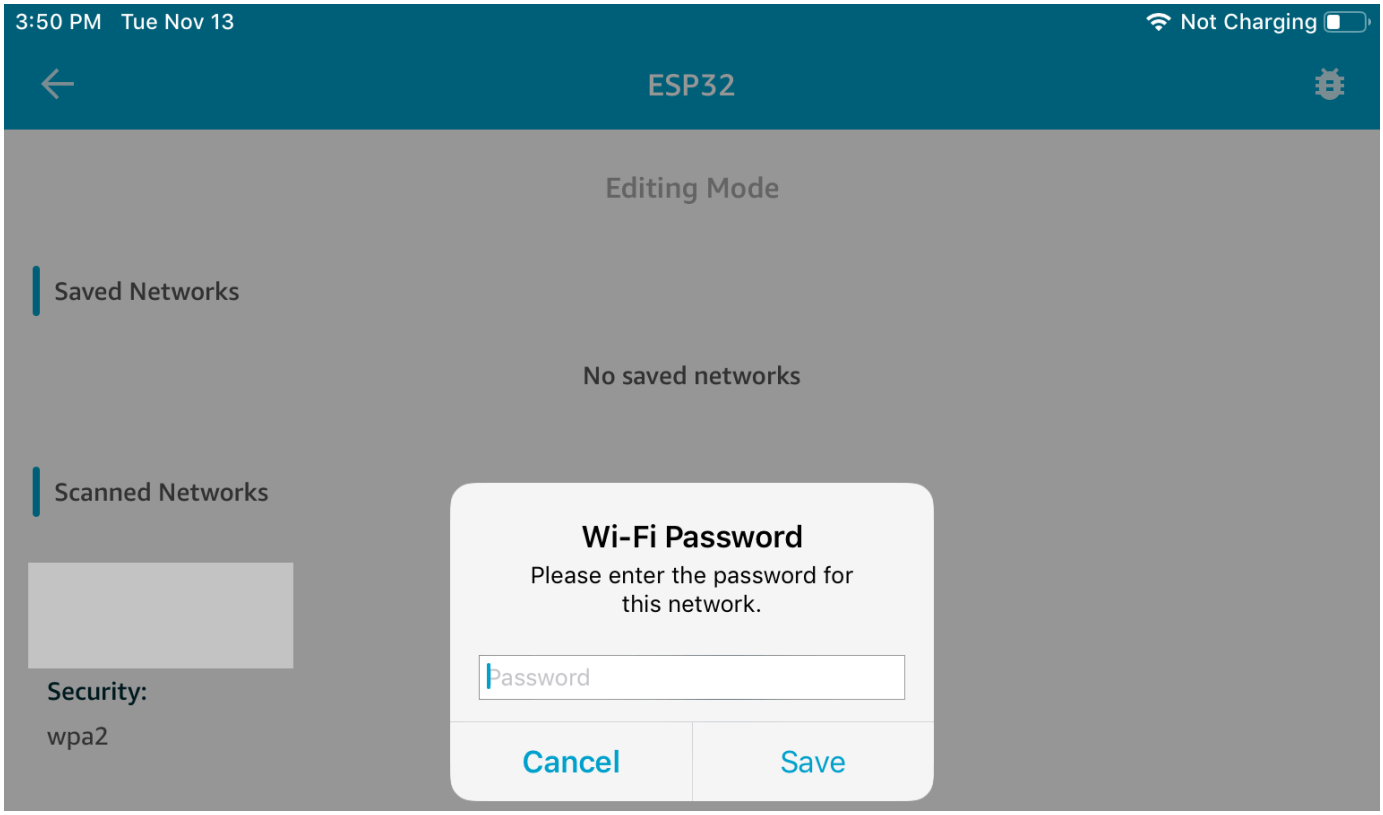
1. 在微控制器上生成并运行演示项目。
2. 确保您已使用 FreeRTOS 低功耗蓝牙移动开发工具包演示应用程序将微控制器与移动设备配对。
3. 从演示移动应用程序的 Devices (设备) 列表中，选择您的微控制器，然后选择 Network Config (网络配置) 以打开网络配置设置。



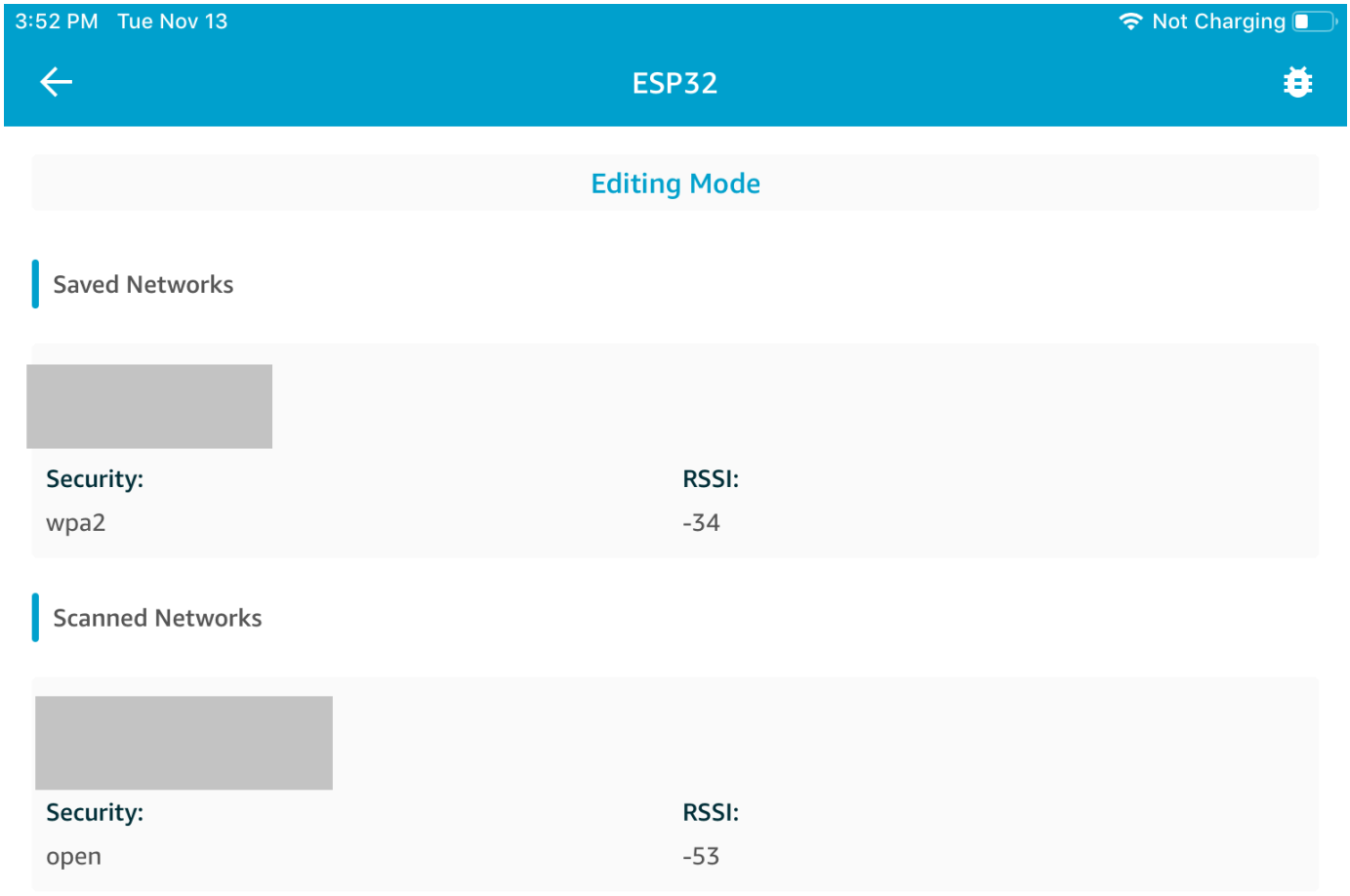
4. 在为主板选择 Network Config (网络配置) 之后，微控制器会将附近的网络列表发送到移动设备。可用 Wi-Fi 网络显示在 Scanned Networks (扫描到的网络) 下的列表中。



从 Scanned Networks (扫描到的网络) 列表中选择网络，然后输入 SSID 和密码（如果需要）。

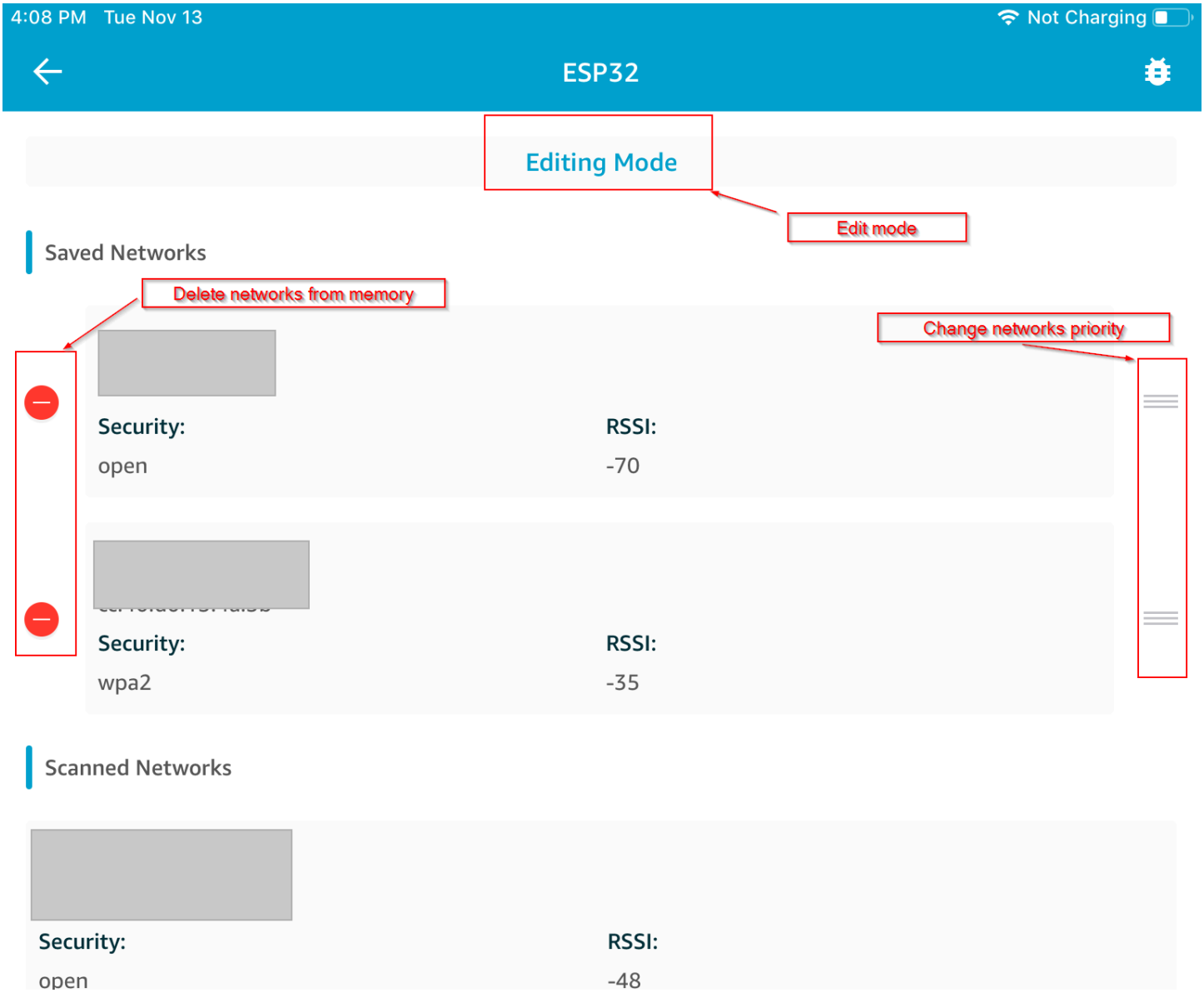


该微控制器连接到网络并保存网络信息。网络显示在 Saved Networks (已保存网络) 下。



您可以在演示移动应用程序中保存多个网络。在您重新启动应用程序和演示时，微控制器会连接到 Saved Networks (已保存网络) 列表中自上而下的第一个可用的已保存网络。

要更改网络优先级顺序或者删除网络，请在 Network Configuration (网络配置) 页面上选择 Editing Mode (编辑模式)。要更改网络优先级顺序，请选择您要重排优先级的网络的右侧，然后向上或向下拖动网络。要删除网络，请选择待删除网络左侧的红色按钮。



通用属性服务器

在本示例中，您的微控制器上的演示通用属性 (GATT) 服务器应用程序发送简单的计数值到 FreeRTOS 低功耗蓝牙移动开发工具包演示应用程序。

使用低功耗蓝牙手机 SDKs，您可以为移动设备创建自己的 GATT 客户端，该客户端通过微控制器连接到 GATT 服务器，并与演示移动应用程序并行运行。

启用演示

1. 启用低功耗蓝牙 GATT 演示。在`vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/iot_ble_config.h`（其中*`vendor`*是供应商的名称，*`board`*是您用来运行演示

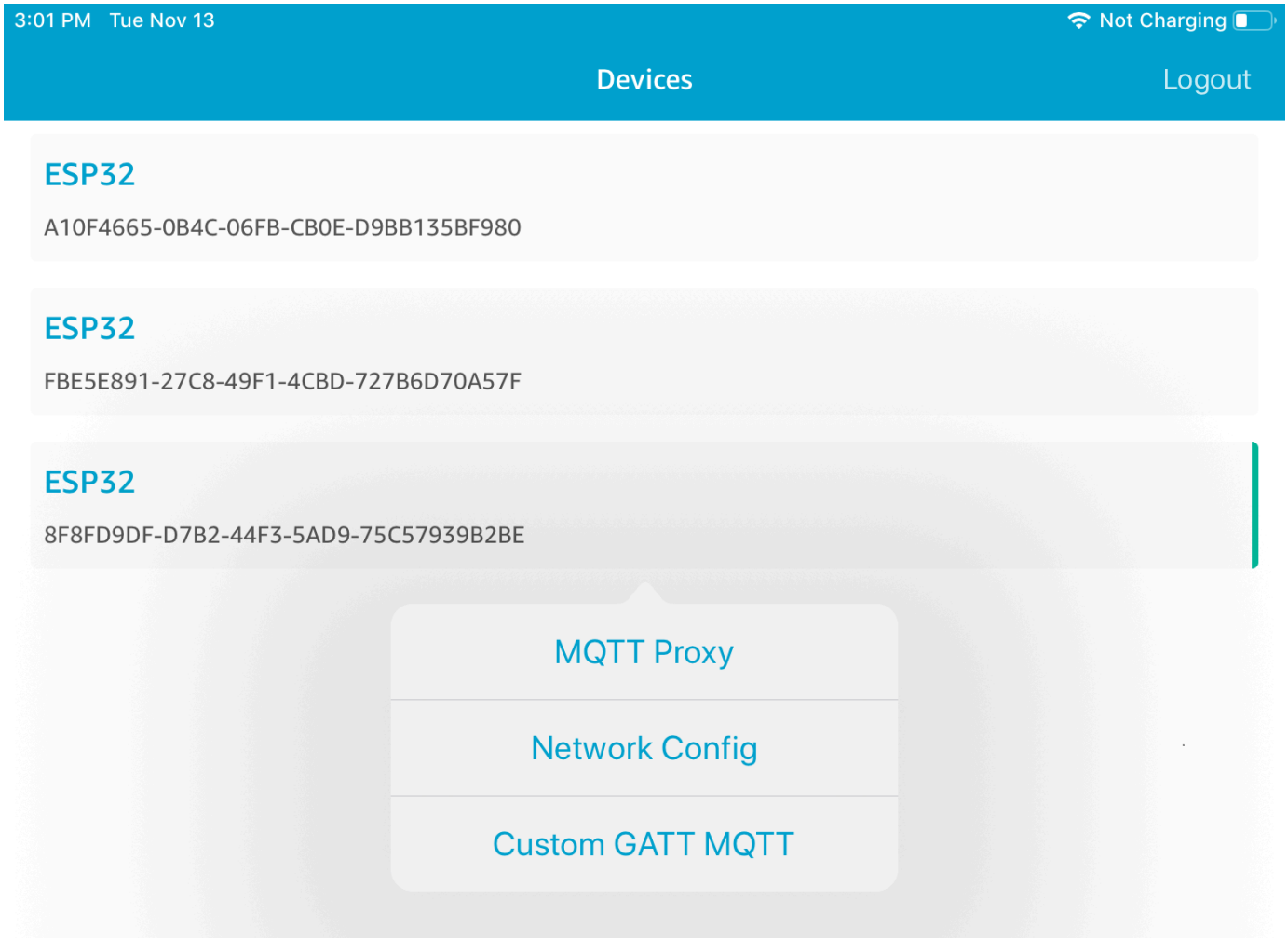
的主板的名称）中，添加`#define IOT_BLE_ADD_CUSTOM_SERVICES ( 1 )`到 define 语句列表中。

> **Note**
>
> 低功耗蓝牙 GATT 演示在默认情况下处于禁用状态。

2. 打开 *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h，注释掉 `#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED` 并定义 `CONFIG_BLE_GATT_SERVER_DEMO_ENABLED`。

运行演示

1. 在微控制器上生成并运行演示项目。
2. 确保您已使用 FreeRTOS 低功耗蓝牙移动开发工具包演示应用程序将主板与移动设备配对。
3. 从应用程序的 Devices (设备) 列表中，选择您的主板，然后选择 MQTT Proxy (MQTT 代理) 以打开 MQTT 代理选项。



4. 返回到 Devices (设备) 列表，选择您的主板，然后选择 Custom GATT MQTT (自定义 GATT MQTT) 以打开自定义 GATT 服务选项。
5. 选择 Start Counter (启动计数器) 以开始将数据发布到 ***your-thing-name*/example/topic** MQTT 主题。

   在启用 MQTT 代理之后，Hello World 和递增计数消息将显示在 *your-thing-name*/example/topic 主题上。

Microchip 好奇号 MZEF 的演示引导加载程序 PIC32

Important

该演示托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

Note

根据Microchip的协议，我们将从FreeRTOS参考集成存储库主分支中删除好奇号 PIC32 MZEF（DM320104），并且将在新版本中不再使用它。Microchip已发布官方通知，不再建议将 PIC32 MZEF（DM320104）用于新设计。 PIC32MZEF 项目和源代码仍然可以通过之前的版本标签进行访问。Microchip 建议客户使用好奇号 PIC32MZ-EF-2.0 开发板 (DM320209) 进行新设计。该 PIC32MZv1 平台仍然可以在 FreeRTOS 参考集成存储库的 v202012. 00 中找到。但是，FreeRTOS 参考的 v202107.00 不再支持该平台。

此演示启动加载程序实施固件版本检查、加密签名验证和应用程序自我测试。这些功能支持 FreeRTOS 的固件更新 over-the-air (OTA)。

固件验证包括验证无线接收的新固件的真实性和完整性。在启动之前，启动加载程序验证应用程序的加密签名。该演示在 SHA-256 之上使用了椭圆曲线数字签名算法 (ECDSA)。提供的实用程序可用于生成签名的应用程序，该应用程序可刷写设备。

启动加载程序支持 OTA 需要的以下功能：

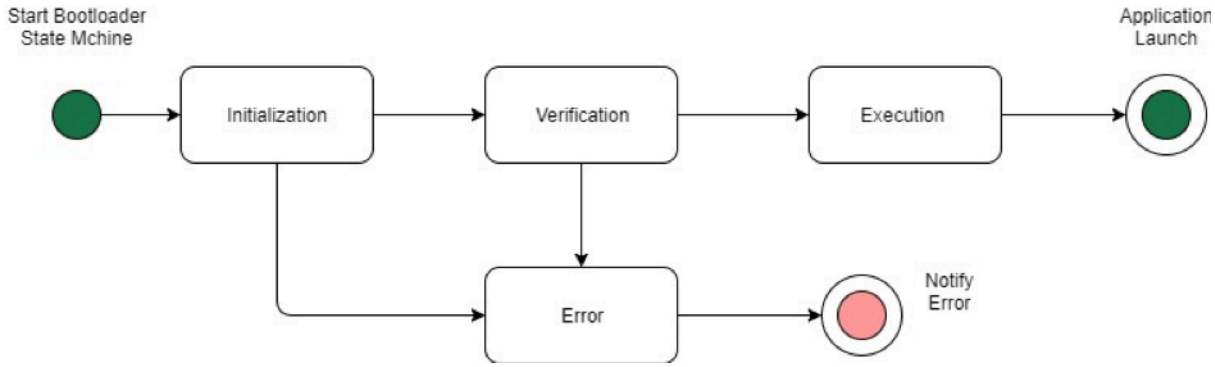
- 在设备上维护应用程序映像，并在这些映像之间切换。
- 允许对收到的 OTA 映像执行自行测试并在出现故障时回退。
- 检查 OTA 更新映像的签名和版本。

Note

要设置和运行 FreeRTOS 演示，请按照开始使用 FreeRTOS中的步骤操作。

启动加载程序状态

以下状态机中显示启动加载程序进程。



下表介绍了启动加载程序状态。

启动加载程序状态	描述
初始化	启动加载程序处于初始化状态。
验证	启动加载程序正在验证设备上存在的映像。
执行映像	启动加载程序正在启动所选映像。
执行默认映像	启动加载程序正在启动默认映像。
错误	启动加载程序处于出错状态。

在前面的示意图中，`Execute Image` 和 `Execute Default` 都显示为 `Execution` 状态。

启动加载程序执行状态

启动加载程序处于 `Execution` 状态，已准备好启动经过验证的选定映像。如果要启动的映像位于较高的库中，则在执行映像之前交换库，因为应用程序始终针对较低的库生成。

启动加载程序默认执行状态

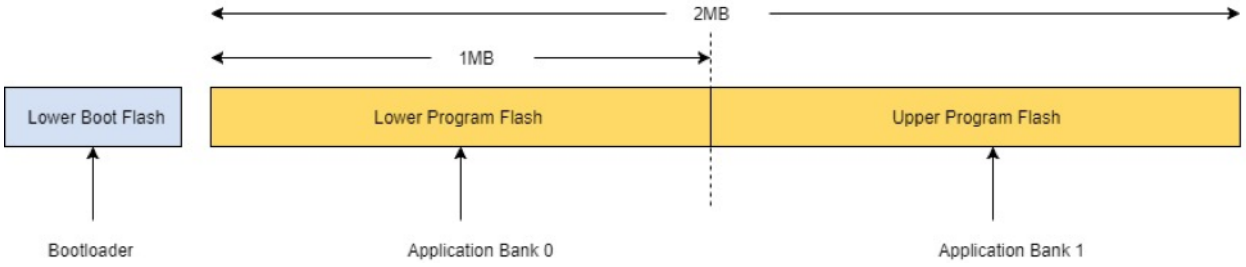
如果启动默认映像的配置选项已启用，则启动加载程序将从默认执行地址启动应用程序。除了在调试期间，否则应禁用此选项。

启动加载程序出错状态

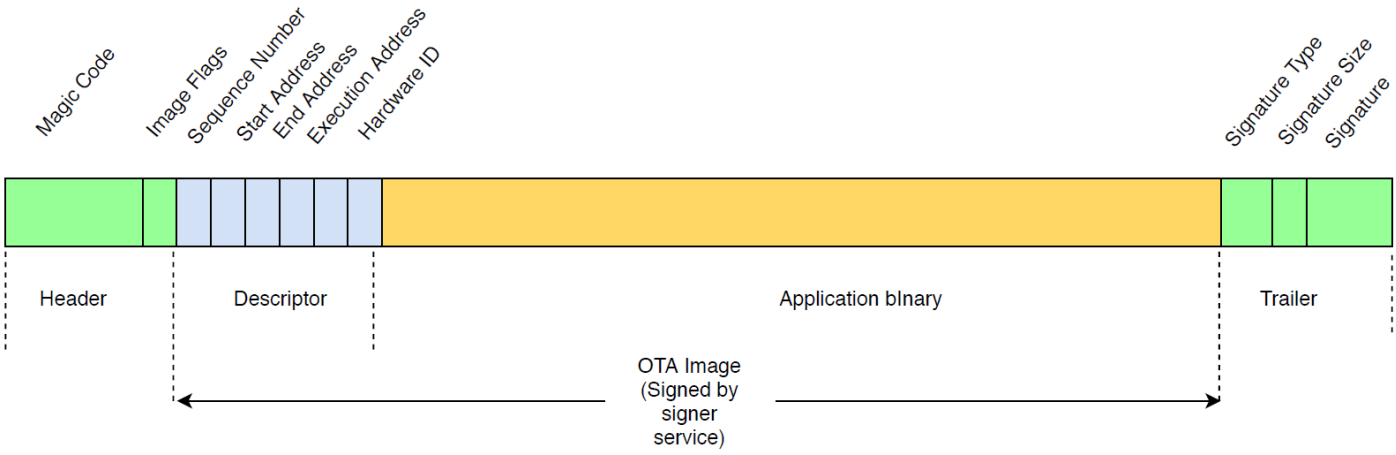
启动加载程序处于出错状态，设备上没有有效的映像。启动加载程序必须通知用户。默认实施发送日志消息到控制台，并无限期快速闪烁主板上的 LED。

闪存设备

Microchip Cu PIC32 riosity MZEF 平台包含一个两兆字节 (MB) 的内部程序闪存，分为两个库。它支持这两个库之间的内存交换映射和实时更新。演示启动加载程序在单独的较低引导闪存区域中编程。



应用程序映像结构



该图显示存储在设备的各个库中应用程序映像的主要组件。

组件	大小（字节）
映像标头	8 字节
映像描述符	24 字节
应用程序二进制文件	< 1 MB - (324)

组件	大小（字节）
Trailer	292 字节

映像标头

设备上的应用程序映像必须以由幻码和映像标志组成的标头为开头。

标头字段	大小（字节）
幻码	7 字节
映像标志	1 字节

幻码

闪存设备上的映像必须以幻码开头。默认幻码为 @AFRTOS。启动加载程序在启动映像之前检查是否存在有效的幻码。这是验证的第一步。

映像标志

映像标志用于存储应用程序映像的状态。标志在 OTA 过程中使用。两个库的映像标志确定了设备的状态。如果正在执行的映像标记为等待提交，这意味着设备处于 OTA 自我测试阶段。即使设备上的映像标记为有效，在每次启动时也会经过相同的验证步骤。如果某个映像被标记为新的，则启动加载程序会将其标记为等待提交，并在验证之后启动它进行自我测试。引导加载程序还会初始化并启动监视程序计时器，以便在新的 OTA 映像自我测试失败时重启设备，此时引导加载程序擦除无效映像来拒绝该映像，并执行之前有效的映像。

设备只能有一个有效的映像。其他映像可以是新 OTA 映像或等待提交（自我测试）。OTA 更新成功后，将从设备上擦除旧映像。

状态	值	描述
新映像	0xFF	应用程序映像是新的，从未执行。
等待提交	0xFE	标记应用程序映像供测试执行。

状态	值	描述
有效	0xFC	应用程序映像标记为有效且已提交。
无效	0xF8	应用程序映像标记为无效。

映像描述符

闪存设备上的应用程序映像必须在映像标头之后包含映像描述符。映像描述符由构建后实用工具生成，该实用工具使用配置文件 (`ota-descriptor.config`) 生成相应的描述符并添加到应用程序二进制文件前面。此构建后步骤的输出是可用于 OTA 的二进制映像。

描述符字段	大小（字节）
序列号	4 字节
开始地址	4 字节
结束地址	4 字节
执行地址	4 字节
硬件 ID	4 字节
预留	4 字节

序列号

序列号必须在生成新 OTA 映像之前递增。请参阅 `ota-descriptor.config` 文件。启动加载程序使用此编号来确定要启动的映像。有效值介于 1 到 4294967295 之间。

开始地址

设备上应用程序映像的开始地址。由于映像描述符附加到应用程序二进制文件的前面，此地址是映像描述符的开头。

结束地址

设备上应用程序映像的结束地址，不包括映像的后缀部分。

执行地址

映像的执行地址。

硬件 ID

启动加载程序用于验证为正确的平台生成了 OTA 映像的唯一硬件 ID。

预留

此项保留供将来使用。

映像后缀

映像后缀附加到应用程序二进制文件。其中包含签名类型字符串、签名大小和映像的签名。

后缀字段	大小（字节）
签名类型	32 字节
签名大小	4 字节
签名	256 字节

签名类型

签名类型是一个字符串，表示使用的加密算法，并用作后缀的标记。启动加载程序支持椭圆曲线数字签名算法 (ECDSA)。默认值为 sig-sha256-ecdsa。

签名大小

加密签名的大小，以字节为单位。

签名

随映像描述符一起附加的应用程序二进制文件加密签名。

启动加载程序配置

基本启动加载程序配置选项在 *freertos*/vendors/microchip/boards/curiosity_pic32mzef/bootloader/config_files/aws_boot_config.h 中提供。一些选项仅提供用于调试目的。

启用默认启动

从默认地址启用应用程序的执行，并且只能为调试启用。映像从默认地址执行而不经过任何验证。

启用加密签名验证

在启动时启用加密签名验证。失败的映像从设备中擦除。此选项仅提供用于调试用途，并且必须在生产中保持启用。

擦除无效映像

如果库上的映像验证失败，则启用该库的完整擦除。此选项仅提供用于调试，并且必须在生产中保持启用。

启用硬件 ID 验证

对在 OTA 映像描述符中的硬件 ID 以及在启动加载程序内编程的硬件 ID 启用验证。此项可选，如果无需硬件 ID 验证，则可以禁用。

启用地址验证

在 OTA 映像的描述符中启用开始地址、结束地址和执行地址的验证。建议您保持启用此选项。

构建启动加载程序

该演示引导加载程序作为可加载项目，包含在位于 FreeRTOS 源代码存储库的 *freertos*/vendors/microchip/boards/curiosity_pic32mzef/aws_demos/mplab/ 中的 aws_demos 项目中。生成 aws_demos 项目时，它先生成引导加载程序，然后生成应用程序。最终输出是一个统一的十六进制映像，包括启动加载程序和应用程序。提供 factory_image_generator.py 实用程序用于生成具有加密签名的统一十六进制映像。启动加载程序实用程序脚本位于 *freertos*/demos/ota/bootloader/utility/ 中。

启动加载程序构建前步骤

此构建前步骤执行名为 codesigner_cert_utility.py 的实用工具脚本，该脚本从代码签名证书中提取公有密钥，并生成包含采用抽象语法表示法 1 (ASN.1) 编码格式公有密钥的 C 标头文件。此标头编译到启动加载程序项目中。生成的标头包含两个常量：公有密钥以及密钥长度的数组。也可以不带 aws_demos 生成启动加载程序项目，并作为普通应用程序进行调试。

AWS IoT Device Defender 演示

> **Important**
>
> 该演示托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

简介

此演示向您展示了如何使用 Devic AWS IoT e Defender 库进行连接AWS IoT Device Defender。该演示使用 CoreMQTT 库通过 TLS（相互身份验证）建立与 MQTT 代理和 CoreJson 库的 AWS IoT MQTT 连接，以验证和解析从该服务收到的响应。 AWS IoT Device Defender 该演示向您展示了如何使用从设备收集的指标构建 JSON 格式的报告，以及如何将构造的报告提交给 AWS IoT Device Defender 服务。该演示还展示了如何在 CoreMQTT 库中注册回调函数，以处理来自 AWS IoT Device Defender 服务的响应，以确认已发送的报告是被接受还是被拒绝。

> **Note**
>
> 要设置和运行 FreeRTOS 演示，请按照开始使用 FreeRTOS中的步骤操作。

功能

此演示创建了一个应用程序任务，演示如何收集指标、构建 JSON 格式的设备防御者报告，以及如何通过与 MQTT 代理的安全 MQTT 连接将其提交给 AWS IoT Device Defender 服务。 AWS IoT 此演示包括标准网络指标和自定义指标。对于自定义指标，此演示包括：

- 一个名为 “`task_numbers`” 的指标，它是 FreeRTOS 任务的列表。 IDs该指标的类型是“数字列表”。
- 一个名为“`stack_high_water_mark`”的指标，它是演示应用程序任务的堆栈高级别水印。该指标的类型是“数字”。

我们如何收集网络指标取决于所使用的 TCP/IP 堆栈。对于 Freertos+TCP 和支持的 LWIP 配置，我们提供了指标收集实现，用于从设备收集真实指标并在报告中提交。 AWS IoT Device Defender 你可以在 freertos+TCP 和 lwIP 上找到。 GitHub

对于使用任何其他 TCP/IP 堆栈的主板，我们提供了指标收集函数的存根定义，这些函数会为所有网络指标返回零。在 *freertos*/demos/device_defender_for_aws/metrics_collector/stub/metrics_collector.c 中为网络实现函数以发送真实指标。该文件也可在 GitHub网站上找到。

对于 ESP32，默认 LWIP 配置不使用核心锁定，因此演示将使用存根指标。如果要使用参考 lwIP 指标收集实现，请在 lwiopts.h 中定义以下宏：

```
#define LINK_SPEED_OF_YOUR_NETIF_IN_BPS 0
#define LWIP_TCPIP_CORE_LOCKING         1
#define LWIP_STATS                      1
#define MIB2_STATS                      1
```

下面是运行演示时的一个输出示例。

```
24 1682 [iot_thread] [INFO] MQTT connection successfully established with broker.


25 1682 [iot_thread] [INFO] A clean MQTT connection is established. Cleaning up all the stored outgoing publishes.


26 1682 [iot_thread] [INFO] Attempt to subscribe to the MQTT topic $aws/things/DemoThing/defender/metrics/json/accepted.27 1682 [
iot_thread] [INFO] SUBSCRIBE sent for topic $aws/things/DemoThing/defender/metrics/json/accepted to broker.
28 1722 [iot_thread] [INFO] Packet received. ReceivedBytes=3.
29 1722 [iot_thread] [INFO] MQTT_PACKET_TYPE_SUBACK.


30 1722 [iot_thread] [INFO] Subscribed to the topic $aws/things/DemoThing/defender/metrics/json/accepted with maximum QoS 1.
31 2322 [iot_thread] [INFO] Attempt to subscribe to the MQTT topic $aws/things/DemoThing/defender/metrics/json/rejected.32 2322 [
iot_thread] [INFO] SUBSCRIBE sent for topic $aws/things/DemoThing/defender/metrics/json/rejected to broker.
33 2382 [iot_thread] [INFO] Packet received. ReceivedBytes=3.
34 2382 [iot_thread] [INFO] MQTT_PACKET_TYPE_SUBACK.


35 2382 [iot_thread] [INFO] Subscribed to the topic $aws/things/DemoThing/defender/metrics/json/rejected with maximum QoS 1.
36 2982 [iot_thread] [INFO] the published payload:{"hed": {"rid": 1109,"v": "1.0"},"met": {"tp": {"pts": [{"pt": 33251}],"t": 1},
"up": {"pts": [],"t": 0},"ns": {"bi": 0,"bo": 0,"pi": 0,"po": 0},"tc": {"ec": {"cs": [{"lp": 33251,"rad": "44.236.152.27:888337 2
982 [iot_thread] [INFO] PUBLISH sent for topic $aws/things/DemoThing/defender/metrics/json to broker with packet ID 3.


38 3102 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
39 3102 [iot_thread] [INFO] Ack packet deserialized with result: MQTTSuccess.
40 3102 [iot_thread] [INFO] State record updated. New state=MQTTPublishDone.
41 3102 [iot_thread] [INFO] PUBACK received for packet id 3.


42 3102 [iot_thread] [INFO] Cleaned up outgoing publish packet with packet id 3.


43 3102 [iot_thread] [INFO] Packet received. ReceivedBytes=141.
44 3102 [iot_thread] [INFO] De-serialized incoming PUBLISH packet: DeserializerResult=MQTTSuccess.
45 3102 [iot_thread] [INFO] State record updated. New state=MQTTPublishDone.
46 3502 [iot_thread] [INFO] UNSUBSCRIBE sent topic $aws/things/DemoThing/defender/metrics/json/accepted to broker.


47 3542 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
48 3542 [iot_thread] [INFO] MQTT_PACKET_TYPE_UNSUBACK.


49 4142 [iot_thread] [INFO] UNSUBSCRIBE sent topic $aws/things/DemoThing/defender/metrics/json/rejected to broker.


50 4202 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
51 4202 [iot_thread] [INFO] MQTT_PACKET_TYPE_UNSUBACK.


52 4802 [iot_thread] [INFO] Disconnected from the broker.
53 4802 [iot_thread] [INFO ][DEMO][4802] memory_metrics::freertos_heap::before::bytes::2088152

54 4802 [iot_thread] [INFO ][DEMO][4802] memory_metrics::freertos_heap::after::bytes::1985556

55 4802 [iot_thread] [INFO ][DEMO][4802] memory_metrics::demo_task_stack::before::bytes::1908

56 4802 [iot_thread] [INFO ][DEMO][4802] memory_metrics::demo_task_stack::after::bytes::1908

57 5802 [iot_thread] [INFO ][DEMO][5802] Demo completed successfully.

58 5804 [iot_thread] [INFO ][INIT][5804] SDK cleanup done.

59 5804 [iot_thread] [INFO ][DEMO][5804] -------DEMO FINISHED-------
```

如果您的主板未使用 FreeRTOS+TCP 或支持的 lwIP 配置，则输出将如下所示。

```
24 1716 [iot_thread] [INFO] MQTT connection successfully established with broker.


25 1716 [iot_thread] [INFO] A clean MQTT connection is established. Cleaning up all the stored outgoing publishes.


26 1716 [iot_thread] [INFO] Attempt to subscribe to the MQTT topic $aws/things/DemoThing/defender/metrics/json/accepted.27 1716
 [iot_thread] [INFO] SUBSCRIBE sent for topic $aws/things/DemoThing/defender/metrics/json/accepted to broker.
28 1756 [iot_thread] [INFO] Packet received. ReceivedBytes=3.
29 1756 [iot_thread] [INFO] MQTT_PACKET_TYPE_SUBACK.


30 1756 [iot_thread] [INFO] Subscribed to the topic $aws/things/DemoThing/defender/metrics/json/accepted with maximum QoS 1.
31 2356 [iot_thread] [INFO] Attempt to subscribe to the MQTT topic $aws/things/DemoThing/defender/metrics/json/rejected.32 2356
 [iot_thread] [INFO] SUBSCRIBE sent for topic $aws/things/DemoThing/defender/metrics/json/rejected to broker.
33 2436 [iot_thread] [INFO] Packet received. ReceivedBytes=3.
34 2436 [iot_thread] [INFO] MQTT_PACKET_TYPE_SUBACK.


35 2436 [iot_thread] [INFO] Subscribed to the topic $aws/things/DemoThing/defender/metrics/json/rejected with maximum QoS 1.
36 3036 [iot_thread] [ERROR] Using stub definition of GetNetworkStats! Please implement for your network stack to get correct m
etrics.
37 3036 [iot_thread] [ERROR] Using stub definition of GetOpenTcpPorts! Please implement for your network stack to get correct m
etrics.
38 3036 [iot_thread] [ERROR] Using stub definition of GetOpenUdpPorts! Please implement for your network stack to get correct m
etrics.
39 3036 [iot_thread] [ERROR] Using stub definition of GetEstablishedConnections! Please implement for your network stack to get
 correct metrics.
40 3036 [iot_thread] [INFO] the published payload:{"hed": {"rid": 1079,"v": "1.0"},"met": {"tp": {"pts": [],"t": 0},"up": {"pts
": [],"t": 0},"ns": {"bi": 0,"bo": 0,"pi": 0,"po": 0},"tc": {"ec": {"cs": [],"t": 0}}},"cmet": {"stack_high_water_mark": [{"num
41 3036 [iot_thread] [INFO] PUBLISH sent for topic $aws/things/DemoThing/defender/metrics/json to broker with packet ID 3.


42 3196 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
43 3196 [iot_thread] [INFO] Ack packet deserialized with result: MQTTSuccess.
44 3196 [iot_thread] [INFO] State record updated. New state=MQTTPublishDone.
45 3196 [iot_thread] [INFO] PUBACK received for packet id 3.


46 3196 [iot_thread] [INFO] Cleaned up outgoing publish packet with packet id 3.


47 3196 [iot_thread] [INFO] Packet received. ReceivedBytes=141.
48 3196 [iot_thread] [INFO] De-serialized incoming PUBLISH packet: DeserializerResult=MQTTSuccess.
49 3196 [iot_thread] [INFO] State record updated. New state=MQTTPublishDone.
50 3596 [iot_thread] [INFO] UNSUBSCRIBE sent topic $aws/things/DemoThing/defender/metrics/json/accepted to broker.


51 3656 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
52 3656 [iot_thread] [INFO] MQTT_PACKET_TYPE_UNSUBACK.


53 4256 [iot_thread] [INFO] UNSUBSCRIBE sent topic $aws/things/DemoThing/defender/metrics/json/rejected to broker.


54 4336 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
55 4336 [iot_thread] [INFO] MQTT_PACKET_TYPE_UNSUBACK.


56 4936 [iot_thread] [INFO] Disconnected from the broker.
57 4936 [iot_thread] [INFO ][DEMO][4936] memory_metrics::freertos_heap::before::bytes::2088152

58 4936 [iot_thread] [INFO ][DEMO][4936] memory_metrics::freertos_heap::after::bytes::1985556

59 4936 [iot_thread] [INFO ][DEMO][4936] memory_metrics::demo_task_stack::before::bytes::1908

60 4936 [iot_thread] [INFO ][DEMO][4936] memory_metrics::demo_task_stack::after::bytes::1908

61 5936 [iot_thread] [INFO ][DEMO][5936] Demo completed successfully.

62 5938 [iot_thread] [INFO ][INIT][5938] SDK cleanup done.

63 5938 [iot_thread] [INFO ][DEMO][5938] -------DEMO FINISHED-------
```

该演示的源代码位于您下载的*freertos*/demos/device_defender_for_aws/目录中或GitHub网站上。

订阅主题 AWS IoT Device Defender

subscribeToDefender主题功能订阅 MQTT 主题，将收到对已发布的 Device Defender 报告的回复。它使用宏 DEFENDER_API_JSON_ACCEPTED 来构造主题字符串，根据该字符串接收接受的 Device Defender 报告的响应。它使用宏 DEFENDER_API_JSON_REJECTED 来构造主题字符串，根据该字符串接收拒绝的 Device Defender 报告的响应。

收集设备指标

该 collectDeviceMetrics函数使用中metrics_collector.h定义的函数收集网络指标。收集的指标包括发送和接收的字节数和数据包数、开放的 TCP 端口数、开放的 UDP 端口数以及建立的 TCP 连接数。

生成 AWS IoT Device Defender 报告

generateDeviceMetrics报告功能使用中定义的函数生成设备防御者报告report_builder.h。该函数获取网络指标和缓冲区，按照预期的格式创建一个 JSON 文档， AWS IoT Device Defender 然后将其写入提供的缓冲区。所期望的 JSON 文档的格式在《AWS IoT 开发者指南》的设备端指标中指定。AWS IoT Device Defender

发布 AWS IoT Device Defender 报告

该 AWS IoT Device Defender 报告以 MQTT 主题发布，用于发布 JSON AWS IoT Device Defender 报告。该报告是使用宏构造的DEFENDER_API_JSON_PUBLISH，如 GitHub 网站上的此代码片段所示。

处理回复的回调

publishCallback 函数处理传入的 MQTT 发布消息。它使用 AWS IoT Device Defender 库中的 Defender_MatchTopic API 来检查传入的 MQTT 消息是否来自 AWS IoT Device Defender 服务。如果消息来自 AWS IoT Device Defender 服务，则它会解析收到的 JSON 响应并在响应中提取报告 ID。然后验证报告 ID 是否与报告中发送的 ID 相同。

AWS IoT Greengrass V1 发现演示应用程序

> **Important**
>
> 该演示托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

在运行 FreeRTOS 的 AWS IoT Greengrass Discovery 演示之前，你需要 AWS设置 AWS IoT Greengrass、和。 AWS IoT要进行设置 AWS，请按照中的说明进行操作设置您的 AWS 账户和权限。要进行设置 AWS IoT Greengrass，你需要创建一个 Greengrass 群组，然后添加一个 Greengrass 核心。有关设置的更多信息 AWS IoT Greengrass，请参阅入门 AWS IoT Greengrass。

设置 AWS 和之后 AWS IoT Greengrass，您需要为配置一些额外的权限 AWS IoT Greengrass。

要设置 AWS IoT Greengrass 权限

1. 转到 IAM 控制台。
2. 在导航窗格中选择 “角色”，然后找到并选择 Greengr ServiceRole ass_。
3. 选择附加政策，选择 AmazonS3 FullAccess 和 A AWSIoTFullcces s，然后选择附加策略。
4. 浏览至 AWS IoT 控制台。
5. 在导航窗格中，选择 Greengrass，选择 Groups (组)，然后选择之前创建的 Greengrass 组。
6. 选择 Settings (设置)，然后选择 Add role (添加角色)。
7. 选择 Greengrass_ ServiceRole，然后选择 “保存”。

将您的主板连接到 FreeRTOS 演示 AWS IoT 并进行配置。

1. 注册您的 MCU 主板 AWS IoT

   注册主板后，您需要创建新的 Greengrass 策略并将其附加到设备证书。

   创建新 AWS IoT Greengrass 政策

   1. 浏览至 AWS IoT 控制台。
   2. 在导航窗格中依次选择安全、策略和创建。
   3. 输入用于标识您的策略的名称。
   4. 在添加语句部分中，选择高级模式。将以下 JSON 复制并粘贴到策略编辑器窗口中：

```
{
      "Effect": "Allow",
      "Action": [
        "greengrass:*"
      ],
      "Resource": "*"
}
```

此策略授予对所有资源的 AWS IoT Greengrass 权限。

5. 选择创建。

将 AWS IoT Greengrass 策略附加到设备的证书

1. 浏览至 AWS IoT 控制台。
2. 在导航窗格中，选择 Manage (管理)，选择 Things (事物)，然后选择之前创建的事物。
3. 选择 Security (安全性)，然后选择附加到设备的证书。
4. 选择 Policies (策略)，选择 Actions (操作)，然后选择 Attach Policy (附加策略)。
5. 查找并选择之前创建的 Greengrass 策略，然后选择 Attach (附加)。

2. 下载 FreeRTOS

> **Note**
>
> 如果您要从 FreeRTOS 控制台下载 FreeRTOS，请选择 “连接到”，而不是 “连接到 AWS IoT Greengrass”。***Platform*** AWS IoT***Platform***

3. 配置 FreeRTOS 演示.

打开 *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h，注释掉 #define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED 并定义 CONFIG_GREENGRASS_DISCOVERY_DEMO_ENABLED。

设置完毕后 AWS IoT Greengrass， AWS IoT 以及下载和配置 FreeRTOS 之后，你可以在设备上构建、刷新和运行 Greengrass 演示。要设置主板的硬件和软件开发环境，请遵循主板特定的入门指南中的说明。

Greengrass 演示向 Greengrass 核心和 MQTT 客户端发布了一系列消息。 AWS IoT 要在 AWS IoT MQTT 客户端中查看消息，请打开AWS IoT 控制台，选择测试，选择 MQTT 测试客户端，然后向添加订阅。freertos/demos/ggd

在 MQTT 客户端中，您应看到下列字符串：

```
Message from Thing to Greengrass Core: Hello world msg #1!
Message from Thing to Greengrass Core: Hello world msg #0!
```

```
Message from Thing to Greengrass Core: Address of Greengrass Core
 found! 123456789012.us-west-2.compute.amazonaws.com
```

使用亚马逊 EC2 实例

如果您使用的是 Amazon EC2 实例

1. 找到与您的亚马逊 EC2 实例关联的公有 DNS (IPv4) — 前往亚马逊 EC2 控制台，然后在左侧导航面板中选择实例。选择您的 Amazon EC2 实例，然后选择描述面板。查找公共 DNS (IPv4) 的条目并记下来。
2. 找到安全组条目，然后选择附加到您的 Amazon EC2 实例的安全组。
3. 选择入站规则选项卡，然后选择编辑入站规则并添加以下规则。

   入站规则

类型	协议	端口范围	来源	描述 - 可选
HTTP	TCP	80	0.0.0.0/0	-
HTTP	TCP	80	::/0	-
SSH	TCP	22	0.0.0.0/0	-
自定义 TCP	TCP	8883	0.0.0.0/0	MQTT 通信
自定义 TCP	TCP	8883	::/0	MQTT 通信
HTTPS	TCP	443	0.0.0.0/0	-
HTTPS	TCP	443	::0/0	-
所有 ICMP-IPv4	ICMP	全部	0.0.0.0/0	-
所有 ICMP-IPv4	ICMP	全部	::0/0	-

4. 在 AWS IoT 控制台中选择 Greengrass，然后选择群组，然后选择你之前创建的 Greengrass 群组。选择设置。将本地连接检测更改为手动管理连接信息。
5. 在导航窗格中，选择核心，然后选择您的组核心。

6. 选择连接，并确保您只有一个核心终端节点（删除其余所有终端节点），并且它不是 IP 地址（因为它可能会发生更改）。最好的选择是使用您在第一步中记下的公共 DNS (IPv4)。
7. 将您创建的 FreeRTOS IoT 物品添加到 GG 组。
   a. 选择返回箭头返回 AWS IoT Greengrass 群组页面。在导航窗格中，选择设备，然后选择添加设备。
   b. 选择选择 IoT 物品。选择您的设备，然后选择完成。
8. 添加必要的订阅 — 在 Greengrass 组页面上，选择订阅，然后选择添加订阅并输入信息，如下所示。

   订阅

来源	目标	主题
TIGG1	IoT 云	freertos/demos/ggd

   其中 “来源” 是您注册看板时在 AWS IoT 控制台中创建的 AWS IoT 东西的名称-在此处给出的示例中TIGG1为 “”。
9. 开始部署您的群 AWS IoT Greengrass 组，并确保部署成功。现在，您应该可以成功运行 AWS IoT Greengrass 发现演示了。

AWS IoT Greengrass V2

与 AWS IoT Greengrass V2 设备的兼容性

AWS IoT Greengrass V2 对客户端设备的支持向后兼容。 AWS IoT Greengrass V1您无需更改应用程序代码即可将 FreeRTOS 客户端设备连接到 V2 核心设备。要使客户端设备能够连接到 V2 核心设备，请执行以下操作。

- 将 Greengrass 软件部署到 Greengrass 核心设备上。请参阅将客户端设备连接到核心设备，从而将设备连接到 AWS IoT Greengrass V2。
- 要在客户端设备、 AWS IoT Core 云服务和 Greengrass 组件之间中继消息（包括 Lambda 函数），请部署和配置 MQTT 桥接组件。
- 部署 IP 探测器组件以自动检测连接信息，或手动管理端点。
- 有关更多信息，请参阅与本地 AWS IoT 设备交互。

有关更多详细信息，请参阅有关在上运行AWS IoT Greengrass V1 应用程序的 AWS文档 AWS IoT Greengrass V2。

coreHTTP 演示

> **Important**
>
> 该演示托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

这些演示可以帮助您学习如何使用 coreHTTP 库。

主题

- coreHTTP 双向身份验证演示
- coreHTTP 基本 Amazon S3 上传演示
- coreHTTP 基本 S3 下载演示
- coreHTTP 基本多线程演示

coreHTTP 双向身份验证演示

> **Important**
>
> 该演示托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

简介

coreMQTT（双向身份验证）演示项目展示了如何使用在客户端与服务器之间具有双向身份验证的 TLS 建立与 HTTP 服务器的连接。此演示使用基于 mbedTLS 的传输接口实现来建立经过服务器和客户端身份验证的 TLS 连接，并演示了 HTTP 中的请求响应工作流程。

> **Note**
>
> 要设置和运行 FreeRTOS 演示，请按照开始使用 FreeRTOS中的步骤操作。

功能

此演示创建一个应用程序任务，其示例演示如何完成以下操作：

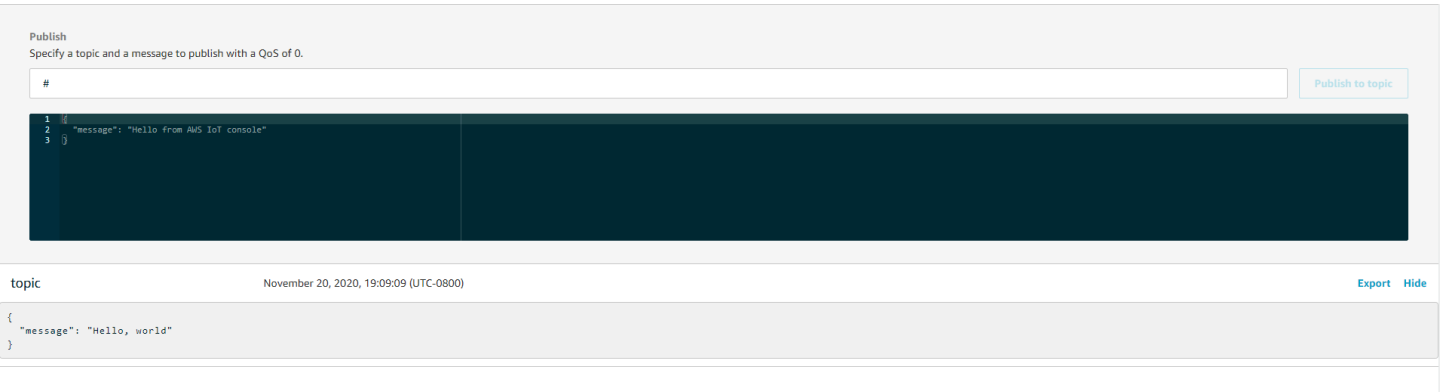
- 连接到 AWS IoT 终端上的 HTTP 服务器。
- 发送 POST 请求。
- 接收响应。
- 断开与服务器的连接。

完成这些步骤后，演示将生成类似于以下屏幕截图的输出。

```
9 1565 [iot_thread] [INFO ][DEMO][1565] ---------STARTING DEMO---------

10 1566 [iot_thread] [INFO ][INIT][1566] SDK successfully initialized.
11 1566 [iot_thread] [INFO ][DEMO][1566] Successfully initialized the demo. Network type for the demo: 4
12 1566 [iot_thread] [INFO] [HTTPDemo] [http_demo_mutual_auth.c:319] 13 1566 [iot_thread] Establishing a TLS session to a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com:8443.14 1566 [iot_thread]
15 1622 [iot_thread] DNS[0x68F5]: The answer to 'a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com' (35.166.102.88) will be stored
16 1622 [iot_thread] DNS[0x68F5]: The answer to 'a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com' (35.166.2.12) will be stored
17 1622 [iot_thread] DNS[0x68F5]: The answer to 'a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com' (44.239.154.164) will be stored
18 1622 [iot_thread] DNS[0x68F5]: The answer to 'a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com' (44.239.97.33) will be stored
19 1622 [iot_thread] DNS[0x68F5]: The answer to 'a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com' (44.238.43.70) will be stored
20 1622 [iot_thread] DNS[0x68F5]: The answer to 'a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com' (52.32.144.134) will be stored
21 2042 [iot_thread] [INFO] [HTTPDemo] [http_demo_mutual_auth.c:393] 22 2042 [iot_thread] Sending HTTP POST request to a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com/topics/topic?qos=1...23 2042 [iot_thread]
24 2082 [iot_thread] [INFO] [HTTPDemo] [http_demo_mutual_auth.c:418] 25 2082 [iot_thread] Received HTTP response from a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com/topics/topic?qos=1...
26 2082 [iot_thread]
27 2082 [iot_thread] [INFO] [HTTPDemo] [http_demo_mutual_auth.c:283] 28 2082 [iot_thread] Demo completed successfully.29 2082 [iot_thread]
30 2082 [iot_thread] [INFO ][DEMO][2082] memory_metrics::freertos_heap::before::bytes::2088152
31 2082 [iot_thread] [INFO ][DEMO][2082] memory_metrics::freertos_heap::after::bytes::1990104
32 2082 [iot_thread] [INFO ][DEMO][2082] memory_metrics::demo_task_stack::before::bytes::1908
33 2082 [iot_thread] [INFO ][DEMO][2082] memory_metrics::demo_task_stack::after::bytes::1908
34 3082 [iot_thread] [INFO ][DEMO][3082] Demo completed successfully.
35 3084 [iot_thread] [INFO ][INIT][3084] SDK cleanup done.
36 3084 [iot_thread] [INFO ][DEMO][3084] -------DEMO FINISHED-------
```

AWS IoT 控制台生成类似于以下屏幕截图的输出。



源代码组织

演示源文件已命名`http_demo_mutual_auth.c`，可以在*`freertos`*`/demos/coreHTTP/`目录和GitHub网站上找到。

正在连接到 AWS IoT HTTP 服务器

该 connectToServerWithBackoffRetries函数尝试与 AWS IoT HTTP 服务器建立相互身份验证的 TLS 连接。如果连接失败，则会在超时后重试。超时值呈指数级增长，直到达到最大尝试次数或达到最大超时值。`RetryUtils_BackoffAndSleep` 函数提供指数级增长的超时值，并在达到最大尝试次数时返回 `RetryUtilsRetriesExhausted`。如果在配置的尝试次数后仍无法建立与代理的 TLS 连接，则 `connectToServerWithBackoffRetries` 函数将返回失败状态。

发送 HTTP 请求和接收响应

prvSendHttp请求函数演示如何向 AWS IoT HTTP 服务器发送 POST 请求。有关向 REST API 发出请求的更多信息 AWS IoT，请参阅设备通信协议-HTTPS。响应是通过相同的 CoreHTTP API 调用 `HTTPClient_Send` 接收。

coreHTTP 基本 Amazon S3 上传演示

> **Important**
>
> 该演示托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

简介

此示例演示如何将 PUT 请求发送到 Amazon Simple Storage Service (Amazon S3) HTTP 服务器以及上传一个小文件。完成上传后，它还会执行 GET 请求来验证文件大小。此示例使用一个网络传输接口，该接口使用 mbedTLS 在运行 coreHTTP 的 IoT 设备客户端与 Amazon S3 HTTP 服务器之间建立双向身份验证的连接。

> **Note**
>
> 要设置和运行 FreeRTOS 演示，请按照开始使用 FreeRTOS中的步骤操作。

单线程与多线程

coreMQTT 有两种使用模式，即单线程和多线程（多任务处理）。尽管本节中的演示在线程中运行 HTTP 库，但它实际上演示了如何在单线程环境中使用 coreHTTP。此演示中只有一个任务使用 HTTP

API。尽管单线程应用程序必须重复调用 HTTP 库，但多线程应用程序可以在后台的代理（或进程守护程序）任务中发送 HTTP 请求。

源代码组织

演示源文件已命名`http_demo_s3_upload.c`，可以在*freertos*`/demos/coreHTTP/`目录和GitHub网站上找到。

配置 Amazon S3 HTTP 服务器连接

此演示使用预签名 URL 连接到 Amazon S3 HTTP 服务器并授权访问要下载的对象。Amazon S3 HTTP 服务器的 TLS 连接仅使用服务器身份验证。在应用程序级别，使用预签名 URL 查询中的参数对对象的访问进行身份验证。请按照以下步骤配置与 AWS的连接。

1. 设置 AWS 账户：
   a. 如果您还没有，请创建一个 AWS 账户。
   b. 账户和权限是使用 AWS Identity and Access Management (IAM) 设置的。您可以使用 IAM 来管理账户中各用户的权限。默认情况下，用户只有在根所有者授予权限后才拥有权限。
      i. 要向您的 AWS 账户添加用户，请参阅 IAM 用户指南。
      ii. 向您的 AWS 账户授予访问FreeRTOS的权限， AWS IoT 并通过添加此政策：
         - 亚马逊 3 FullAccess
2. 按照《Amazon Simple Storage Service 用户指南》的如何创建 S3 存储桶中的步骤，在 Amazon S3 中创建一个存储桶。
3. 按照如何将文件和文件夹上传至 S3 存储桶中的步骤，将文件上传到 Amazon S3 存储桶。
4. 使用 `FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/presigned_urls_gen.py` 文件中的脚本生成预签名 URL。

   有关使用说明，请参阅 `FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/README.md` 文件。

功能

首先，此演示使用 TLS 服务器身份验证连接到 Amazon S3 HTTP 服务器。然后，它会创建一个 HTTP 请求来上传在 `democonfigDEMO_HTTP_UPLOAD_DATA` 中指定的数据。上传文件后，它会通过请求文件的大小来检查文件是否上传成功。该演示的源代码可以在GitHub网站上找到。

连接到 Amazon S3 HTTP 服务器

该 connectToServerWithBackoffRetries函数尝试与 HTTP 服务器建立 TCP 连接。如果连接失败，则会在超时后重试。超时值呈指数级增长，直到达到最大尝试次数或最大超时值。如果在配置的尝试次数后仍无法建立与服务器的 TCP 连接，则 connectToServerWithBackoffRetries 函数将返回失败状态。

prvConnectToServer 函数演示如何仅使用服务器身份验证来建立与 Amazon S3 HTTP 服务器的连接。它使用在 FreeRTOS-Plus/Source/Application-Protocols/network_transport/freertos_plus_tcp/using_mbedtls/using_mbedtls.c 文件中实现的基于 mbedTLS 的传输接口。的定义prvConnectToServer可以在GitHub网站上找到。

上传数据

prvUploadS3ObjectFile 函数演示如何创建 PUT 请求和指定要上传的文件。上传文件的 Amazon S3 存储桶和要上传的文件名在预签名 URL 中指定。为了节省内存，请求标头和接收响应使用相同的缓冲区。使用 HTTPClient_Send API 函数同步接收响应。Amazon S3 HTTP 服务器需要一个 200 OK 响应状态代码。任何其他状态代码均为错误。

的源代码prvUploadS3ObjectFile()可以在 GitHub网站上找到。

验证上传

prvVerifyS3ObjectFileSize 函数调用 prvGetS3ObjectFileSize 来检索 S3 存储桶中对象的大小。目前，Amazon S3 HTTP 服务器不支持使用预签名 URL 的 HEAD 请求，因此请求第 0 个字节。文件的大小包含在响应的 Content-Range 标头字段中。服务器的预期响应为 206 Partial Content。任何其他响应状态代码均为错误。

的源代码prvGetS3ObjectFileSize()可以在 GitHub网站上找到。

coreHTTP 基本 S3 下载演示

> **Important**
>
> 该演示托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

简介

此演示展示了如何使用范围请求来从 Amazon S3 HTTP 服务器下载文件。当您使用 `HTTPClient_AddRangeHeader` 创建 HTTP 请求时，coreHTTP API 原生支持范围请求。对于微控制器环境，强烈建议使用范围要求。通过在不同的范围内下载大文件（而不是在单个请求中下载），可以在不阻塞网络套接字的情况下处理文件的各个部分。范围请求降低了丢弃数据包的风险（而丢弃的数据包要求在 TCP 连接上重新传输），因此可以改善设备的功耗。

此示例使用一个网络传输接口，该接口使用 mbedTL 在运行 CoreHTTP 的 IoT 设备客户端与 Amazon S3 HTTP 服务器之间建立双向身份验证的连接。

> **Note**
>
> 要设置和运行 FreeRTOS 演示，请按照开始使用 FreeRTOS中的步骤操作。

单线程与多线程

coreHTTP 有两种使用模式，即单线程和多线程（多任务处理）。尽管本节中的演示在线程中运行 HTTP 库，但它实际上演示了如何在单线程环境中使用 coreHTTP（演示中只有一个任务使用 HTTP API）。尽管单线程应用程序必须重复调用 HTTP 库，但多线程应用程序可以在后台的代理（或进程守护程序）任务中发送 HTTP 请求。

源代码组织

演示项目已命名`http_demo_s3_download.c`，可以在*`freertos`*`/demos/coreHTTP/`目录和 GitHub网站上找到。

配置 Amazon S3 HTTP 服务器连接

此演示使用预签名 URL 连接到 Amazon S3 HTTP 服务器并授权访问要下载的对象。Amazon S3 HTTP 服务器的 TLS 连接仅使用服务器身份验证。在应用程序级别，使用预签名 URL 查询中的参数对对象的访问进行身份验证。请按照以下步骤配置与 AWS的连接。

1. 设置 AWS 账户：
    a. 如果您还没有，请创建并激活一个 AWS 帐户。
    b. 账户和权限是使用 AWS Identity and Access Management (IAM) 设置的。您可以使用 IAM 来管理账户中各用户的权限。默认情况下，用户只有在根所有者授予权限后才拥有权限。
        i. 要向您的 AWS 账户添加用户，请参阅 IAM 用户指南。

ii. 向您的 AWS 账户授予访问FreeRTOS的权限， AWS IoT 并添加以下政策：

- 亚马逊 3 FullAccess

2. 按照《Amazon Simple Storage Service 控制台用户指南》的如何创建 S3 存储桶中的步骤，在 S3 中创建一个存储桶。
3. 按照如何将文件和文件夹上传至 S3 存储桶中的步骤，将文件上传到 S3。
4. 使用 `FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/presigned_urls_gen.py` 中的脚本生成预签名 URL。有关使用说明，请参阅`FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/README.md`。

功能

首先，该演示检索文件的大小。然后，它会在循环中按顺序请求每个字节范围，范围大小为 `democonfigRANGE_REQUEST_LENGTH`。

该演示的源代码可以在GitHub网站上找到。

连接到 Amazon S3 HTTP 服务器

函数 connectToServerWithBackoffRetries() 尝试与 HTTP 服务器建立 TCP 连接。如果连接失败，则会在超时后重试。超时值呈指数级增长，直到达到最大尝试次数或最大超时值。如果在配置的尝试次数后仍无法建立与服务器的 TCP 连接，则 `connectToServerWithBackoffRetries()` 函数将返回失败状态。

`prvConnectToServer()` 函数演示如何仅使用服务器身份验证来建立与 Amazon S3 HTTP 服务器的连接。它使用基于 mbedtls 的传输接口，该接口在 freeRTOS-_mbedtls.c 文件中实现。Plus/Source/Application-Protocols/network_transport/freertos_plus_tcp/using_mbedtls/using

的源代码`prvConnectToServer()`可以在上找到 GitHub。

创建范围请求

API 函数 `HTTPClient_AddRangeHeader()` 支持将字节范围序列化为 HTTP 请求标头以构建范围请求。此演示使用范围请求来检索文件大小并请求文件的各个部分。

`prvGetS3ObjectFileSize()` 函数检索 S3 存储桶中文件的大小。`Connection: keep-alive` 标头已添加到 Amazon S3 的第一个请求中，以便在发送响应后保持连接畅通。目前，S3 HTTP 服务器

不支持使用预签名 URL 的 HEAD 请求，因此请求第 0 个字节。文件的大小包含在响应的 Content-Range 标头字段中。服务器的预期响应为 206 Partial Content；收到任何其他响应状态代码均为错误。

的源代码prvGetS3ObjectFileSize()可以在上找到 GitHub。

检索文件大小后，此演示会为要下载的文件的每个字节范围创建一个新的范围请求。对于文件的每个部分，它都会使用 HTTPClient_AddRangeHeader()。

发送范围请求和接收响应

函数 prvDownloadS3ObjectFile() 循环发送范围请求，直到整个文件下载完成。API 函数 HTTPClient_Send() 同步发送请求并接收响应。当函数返回时，响应在 xResponse 中接收。然后验证状态代码是否为 206 Partial Content，并且目前已下载的字节数按 Content-Length 标头值递增。

的源代码prvDownloadS3ObjectFile()可以在上找到 GitHub。

coreHTTP 基本多线程演示

> **Important**
>
> 该演示托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

简介

此演示使用 FreeRTOS 的线程安全队列来保存待处理的请求和响应。此演示中有三个任务需要注意。

- 主任务等待请求出现在请求队列中。它将通过网络发送这些请求，然后将响应放入响应队列中。
- 请求任务创建要发送到服务器的 HTTP 库请求对象，并将它们放入请求队列中。每个请求对象都指定了应用程序为下载配置的 S3 文件的字节范围。
- 响应任务等待响应出现在响应队列中。它会记录收到的每一个响应。

该基本多线程演示配置为使用仅具有服务器身份验证的 TLS 连接，这是 Amazon S3 HTTP 服务器所必需的。应用程序层身份验证使用预签名 URL查询中的签名版本 4 参数完成。

源代码组织

演示项目已命名`http_demo_s3_download_multithreaded.c`，可以在*freertos*`/demos/coreHTTP/`目录和 GitHub网站上找到。

构建演示项目

该演示项目使用 Visual Studio 的免费社区版本。构建演示：

1. 在 Visual Studio IDE 中打开 `mqtt_multitask_demo.sln` Visual Studio 解决方案文件。
2. 从 IDE 的构建菜单中选择构建解决方案。

> **Note**
>
> 如果你使用的是 Microsoft Visual Studio 2017 或更早版本，则必须选择与你的版本兼容的平台工具集：项目-> RTOSDemos 属性-> 平台工具集。

配置演示项目

此演示使用 FreeRTOS+TCP TCP/IP 堆栈，因此，请按照为 TCP/IP 入门项目提供的说明进行操作：

1. 安装必备组件（例如 WinPCap）。
2. （可选）设置静态或动态 IP 地址、网关地址和网络掩码。
3. （可选）设置 MAC 地址。
4. 在主机上选择一个以太网网络接口。
5. 重要的是，在尝试运行 HTTP 演示之前，请先测试您的网络连接。

配置 Amazon S3 HTTP 服务器连接

按照 coreHTTP 基本下载演示中有关配置 Amazon S3 HTTP 服务器连接的说明进行操作。

功能

此演示共创建三个任务：

- 其一是通过网络发送请求和接收响应。
- 其二是创建要发送的请求。

- 其三是处理收到的响应。

在此演示中，主要任务：

1. 创建请求和响应队列。
2. 创建与服务器的连接。
3. 创建请求和响应任务。
4. 等待请求队列通过网络发送请求。
5. 将通过网络收到的响应放入响应队列中。

请求任务：

1. 创建每个范围请求。

响应任务：

1. 处理收到的每个响应。

Typedefs

此演示定义以下结构来支持多线程。

请求项目

以下结构定义了要放入请求队列的请求项目。创建 HTTP 请求后，请求任务会将请求项目复制到队列中。

```
/**
 * @brief Data type for the request queue.
 *
 * Contains the request header struct and its corresponding buffer, to be
 * populated and enqueued by the request task, and read by the main task. The
 * buffer is included to avoid pointer inaccuracy during queue copy operations.
 */
typedef struct RequestItem
{
    HTTPRequestHeaders_t xRequestHeaders;
```

```
    uint8_t ucHeaderBuffer[ democonfigUSER_BUFFER_LENGTH ];
} RequestItem_t;
```

响应项目

以下结构定义了要放入响应队列的响应项目。通过网络收到响应后，主 HTTP 任务会将响应项目复制到队列中。

```
/**
 * @brief Data type for the response queue.
 *
 * Contains the response data type and its corresponding buffer, to be enqueued
 * by the main task, and interpreted by the response task. The buffer is
 * included to avoid pointer inaccuracy during queue copy operations.
 */
typedef struct ResponseItem
{
    HTTPResponse_t xResponse;
    uint8_t ucResponseBuffer[ democonfigUSER_BUFFER_LENGTH ];
} ResponseItem_t;
```

主要 HTTP 发送任务

主要应用程序任务：

1. 解析主机地址的预签名 URL 以建立与 Amazon S3 HTTP 服务器的连接。
2. 解析 S3 存储桶中的对象路径的预签名 URL。
3. 使用 TLS 服务器身份验证连接到 Amazon S3 HTTP 服务器。
4. 创建请求和响应队列。
5. 创建请求和响应任务。

`prvHTTPDemoTask()` 函数会进行此设置，并提供演示状态。此函数的源代码可以在 GitHub 上找到。

在函数 `prvDownloadLoop()` 中，主任务会阻止并等待来自请求队列的请求。收到请求时，它会使用 API 函数 `HTTPClient_Send()` 发送请求。如果 API 函数成功，则会将响应放入响应队列中。

`prvDownloadLoop()` 的源代码可以在 GitHub 上找到。

HTTP 请求任务

请求任务在函数 prvRequestTask 中指定。此函数的源代码可以在 GitHub 上找到。

请求任务检索 Amazon S3 存储桶中文件的大小。这在函数 prvGetS3ObjectFileSize 中完成。将“Connection: keep-alive”标头添加到 Amazon S3 的此请求中，以便在发送响应后保持连接畅通。目前，Amazon S3 HTTP 服务器不支持使用预签名 URL 的 HEAD 请求，因此请求第 0 个字节。文件的大小包含在响应的 Content-Range 标头字段中。服务器的预期响应为 206 Partial Content；收到任何其他响应状态代码均为错误。

prvGetS3ObjectFileSize 的源代码可以在 GitHub 上找到。

检索文件大小后，请求任务会继续请求文件的各个范围。将每个范围请求放入请求队列中，以供主任务发送。文件范围由演示用户在宏 democonfigRANGE_REQUEST_LENGTH 中配置。使用函数 HTTPClient_AddRangeHeader，HTTP 客户端库 API 原生支持范围请求。函数 prvRequestS3ObjectRange 演示如何使用 HTTPClient_AddRangeHeader()。

函数 prvRequestS3ObjectRange 的源代码可以在 GitHub 上找到。

HTTP 响应任务

响应任务在响应队列中等待通过网络收到的响应。当主任务成功收到 HTTP 响应时，它会填充响应队列。此任务通过记录状态代码、标头和正文来处理响应。例如，现实世界中的应用程序可以通过将响应正文写入闪存来处理响应。如果响应状态代码不是 206 partial content，则该任务会通知主任务：演示应失败。响应任务在函数 prvResponseTask 中指定。此函数的源代码可以在 GitHub 上找到。

AWS IoT 作业库演示

> **Important**
>
> 该演示托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

简介

AWS IoT 作业库演示向您展示了如何通过 MQTT 连接连接到 J AWS IoT obs 服务 AWS IoT、从设备检索作业并在设备上处理作业。 AWS IoT Jobs 演示项目使用 FreeRTOS Windows 端口，因此可以

在 Windows 上使用 Visual Studio 社区版本进行构建和评估。无需微控制器硬件。该演示使用 TLS 与 AWS IoT MQTT 代理建立安全连接，方式与。coreMQTT 双向身份验证演示

> **Note**
>
> 要设置和运行 FreeRTOS 演示，请按照开始使用 FreeRTOS中的步骤操作。

源代码组织

演示代码在`jobs_demo.c`文件中，可以在GitHub网站或*`freertos`*`/demos/jobs_for_aws/`目录中找到。

配置 AWS IoT MQTT 代理连接

在本演示中，您将使用与 MQTT 代理的 M AWS IoT QTT 连接。此连接的配置方式与 coreMQTT 双向身份验证演示相同。

功能

该演示演示了用于从设备接收任务 AWS IoT 并在设备上处理这些任务的工作流程。该演示是交互式的，需要您使用 AWS IoT 控制台或 AWS Command Line Interface (AWS CLI) 来创建作业。有关创建作业的更多信息，请参阅 AWS CLI 命令参考中的 create-job。该演示要求将作业文档的 `action` 键设置为 `print` 才能将消息输出到控制台。

请参阅此作业文档的以下格式。

```
{
    "action": "print",
    "message": "ADD_MESSAGE_HERE"
}
```

您可以使用创建作业，如以下示例命令所示。 AWS CLI

```
aws iot create-job \
    --job-id t12 \
    --targets arn:aws:iot:region:123456789012:thing/device1 \
    --document '{"action":"print","message":"hello world!"}'
```

该演示还使用将 action 键设置为 publish 的作业文档来将消息重新发布到主题。请参阅此作业文档的以下格式。

```
{
    "action": "publish",
    "message": "ADD_MESSAGE_HERE",
    "topic": "topic/name/here"
}
```

该演示会循环执行，直到收到一个将 action 键设置为 exit 的作业文档才会退出。作业文档的格式如下所示。

```
{
    "action: "exit"
}
```

Jobs 演示的入口点

Jobs 演示入口点函数的源代码可在上找到 GitHub。该函数执行以下操作：

1. 使用 mqtt_demo_helpers.c 中的帮助程序函数建立 MQTT 连接。
2. 使用 mqtt_demo_helpers.c 中的帮助程序函数为 NextJobExecutionChanged API 订阅 MQTT 主题。主题字符串是之前使用 AWS IoT Jobs 库定义的宏汇编而成的。
3. 使用 mqtt_demo_helpers.c 中的帮助程序函数为 StartNextPendingJobExecution API 发布 MQTT 主题。主题字符串是之前使用 AWS IoT Jobs 库定义的宏汇编而成的。
4. 反复调用 MQTT_ProcessLoop 以接收传入的消息，这些消息将交给 prvEventCallback 进行处理。
5. 演示收到退出操作后，使用文件中的帮助程序函数取消订阅 MQTT 主题并断开连接。mqtt_demo_helpers.c

收到的 MQTT 消息的回调

该 prvEventCallback函数Jobs_MatchTopic从 AWS IoT Jobs 库调用，对传入的 MQTT 消息进行分类。如果消息类型对应于新作业，则会调用 prvNextJobHandler()。

prvNextJobHandler 函数及其调用的函数从 JSON 格式的消息中解析作业文档，然后运行作业指定的操作。prvSendUpdateForJob 函数非常让人感兴趣。

为正在运行的作业发送更新

函数 prvSendUpdateForJob() `Jobs_Update()` 从作业库中调用，以填充紧随其后的 MQTT 发布操作中使用的主题字符串。

coreMQTT 演示

> **Important**
>
> 该演示托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

这些演示可帮助您学习如何使用 coreMQTT 库。

主题

- coreMQTT 双向身份验证演示
- coreMQTT 代理连接共享演示

coreMQTT 双向身份验证演示

> **Important**
>
> 该演示托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

简介

coreMQTT 双向身份验证演示项目展示了如何使用在客户端与服务器之间具有双向身份验证的 TLS 建立与 MQTT 代理的连接。此演示使用基于 mbedTLS 的传输接口实现来建立经过服务器和客户端身份验证的 TLS 连接，并演示了 QoS 1 级别的 MQTT 订阅发布工作流程。它会订阅主题筛选条件，然后发布到与筛选条件匹配的主题，并等待从 QoS 1 级别收到服务器发回的消息。这种向代理发布消息并从代理接收相同消息的循环过程会无限期地重复。本演示中的消息按照 QoS 1 发送，这样可以保证至少有一次按照 MQTT 规范传递。

> **Note**
>
> 要设置和运行 FreeRTOS 演示，请按照开始使用 FreeRTOS中的步骤操作。

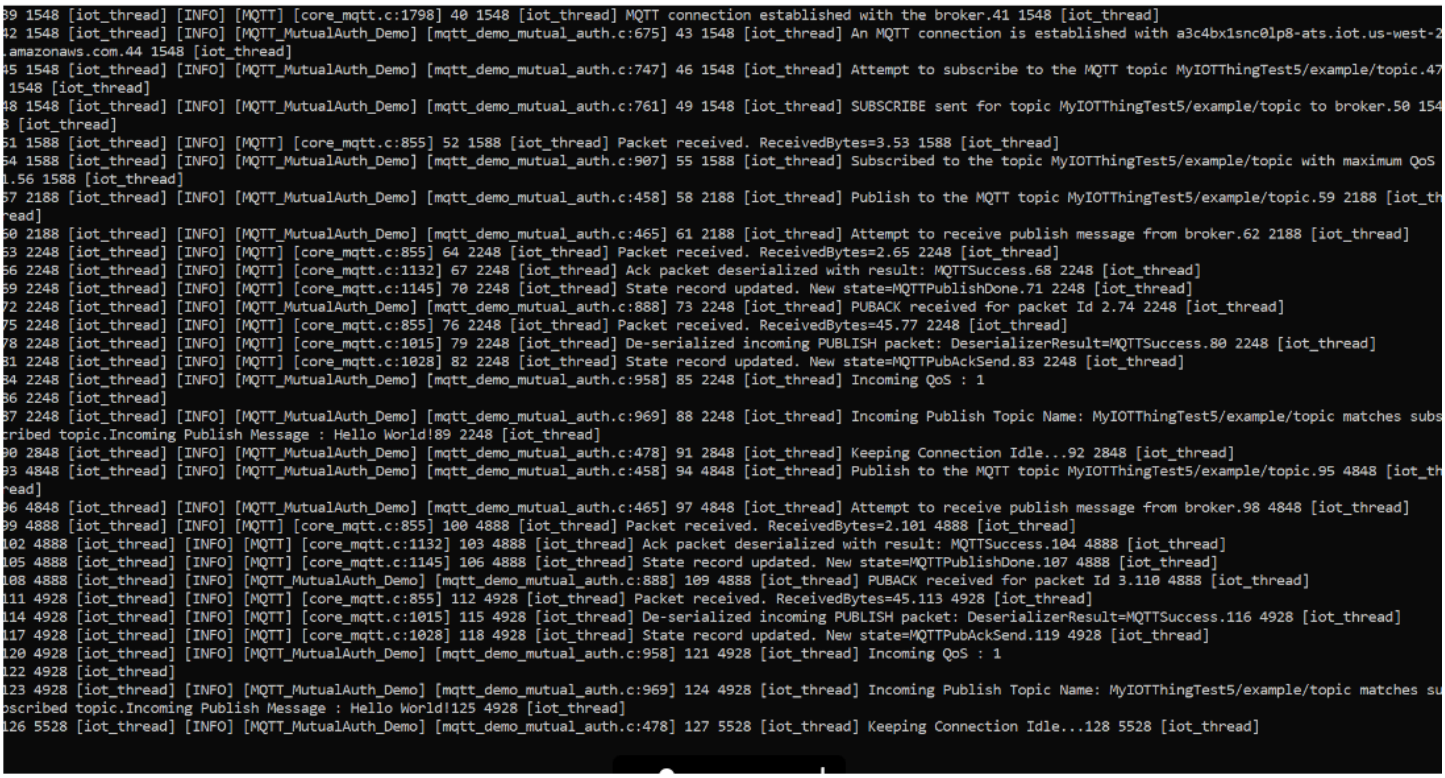
源代码

演示源文件已命名mqtt_demo_mutual_auth.c，可以在*freertos*/demos/coreMQTT/目录和GitHub网站上找到。

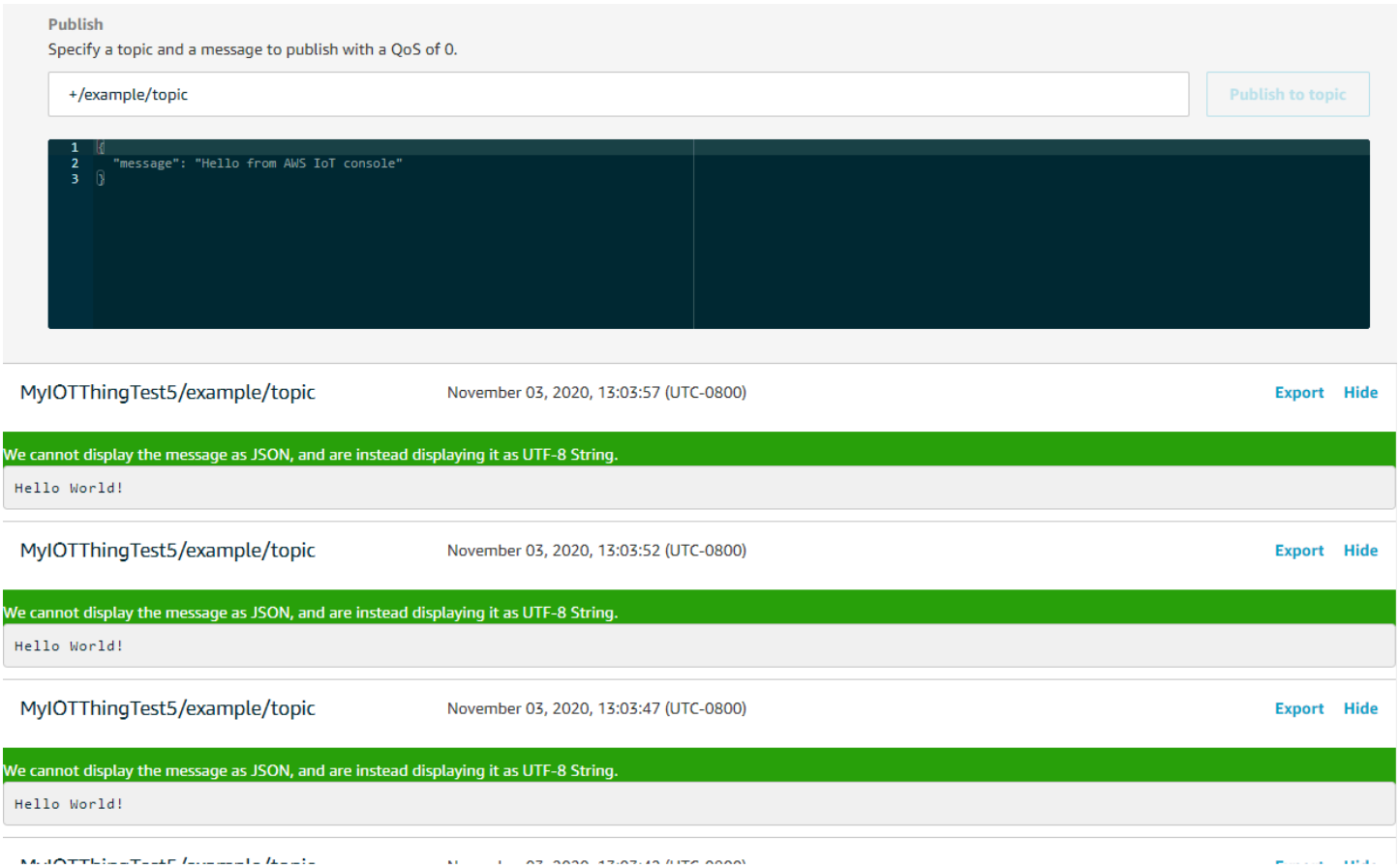
功能

该演示创建了一个应用程序任务，该任务循环访问一组示例，这些示例演示如何连接代理、订阅代理上的主题、发布到代理上的主题，最后断开与代理的连接。该演示应用程序会订阅一个主题的消息，也会向同一个主题发布消息。每次演示向 MQTT 代理发布消息时，代理都会向演示应用程序发送相同的消息。

成功完成演示将生成类似于以下图像的输出。



AWS IoT 控制台将生成类似于下图的输出。



使用指数回退和抖动重试逻辑

prvBackoffFor重试函数显示如何通过指数退避和抖动重试服务器失败的网络操作（例如 TLS 连接或 MQTT 订阅请求）。该函数可计算下一次重试尝试的回退周期，如果重试尝试未用完，则执行回退延迟。由于计算退避周期需要生成随机数，因此该函数使用 PKCS11 模块生成随机数。如果供应商平台支持，则使用该 PKCS11 模块允许访问真随机数生成器 (TRNG)。我们建议您在随机数生成器中加入特定于设备的熵源，这样可以降低设备在连接重试过程中发生冲突的可能性。

连接到 MQTT 代理

该 prvConnectToServerWithBackoffRetries函数尝试与 MQTT 代理建立相互身份验证的 TLS 连接。如果连接失败，则会在回退周期后重试。回退周期将呈指数级增长，直到达到最大尝试次数或达到最大回退周期。`BackoffAlgorithm_GetNextBackoff` 函数提供指数级增长的回退值，并在达到最大尝试次数后返回 `RetryUtilsRetriesExhausted`。如果在配置的尝试次数后仍无法建立与代理的 TLS 连接，则 `prvConnectToServerWithBackoffRetries` 函数将返回失败状态。

prvCreate MQTTConnection WithBroker 函数演示了如何通过干净的会话与 MQTT 代理建立 MQTT 连接。它使用 TLS 传输接口，该接口在 `FreeRTOS-Plus/Source/Application-Protocols/`

platform/freertos/transport/src/tls_freertos.c 文件中实现。请记住，我们正在 xConnectInfo 中为代理设置保持活动状态的秒数。

下一个函数展示了如何使用 MQTT_Init 函数在 MQTT 环境中设置 TLS 传输接口和时间函数，还展示了事件回调函数指针 (prvEventCallback) 是如何设置的。该回调用于报告传入的消息。

订阅 MQTT 主题

p rv MQTTSubscribe WithBackoffRetries 函数演示如何在 MQTT 代理上订阅主题过滤器。该示例演示了如何订阅一个主题筛选条件，还可以在同一个订阅 API 调用中传递主题筛选条件列表来订阅多个主题筛选条件。此外，对于 RETRY_MAX_ATTEMPTS，如果 MQTT 代理拒绝订阅请求，则订阅将以指数回退的方式重试。

向主题发布

p rv MQTTPublish ToTopic 函数演示如何在 MQTT 代理上发布到某个主题。

接收传入的消息

如前所述，该应用程序会在连接到代理之前注册一个事件回调函数。prvMQTTDemoTask 函数调用 MQTT_ProcessLoop 函数来接收传入的消息。当收到传入的 MQTT 消息时，它会调用应用程序注册的事件回调函数。该 prvEventCallback函数就是这样一个事件回调函数的示例。 prvEventCallback检查传入的数据包类型并调用相应的处理程序。在下面的示例中，该函数要么调用 prvMQTTProcessIncomingPublish() 来处理传入的发布消息，要么调用 prvMQTTProcessResponse() 来处理确认消息 (ACK)。

处理传入的 MQTT 发布数据包

prv MQTTProcess IncomingPublish 函数演示如何处理来自 MQTT 代理的发布数据包。

取消订阅主题

该工作流程的最后一步是取消订阅主题，这样代理就不会从 mqttexampleTOPIC 发送任何已发布的消息。这是函数 prv MQTTUnsubscribe FromTopic 的定义。

更改演示中使用的根 CA

默认情况下，FreeRTOS 演示使用 Amazon Root CA 1 证书（RSA 2048 位密钥）向服务器进行身份验证。 AWS IoT Core 您可以使用其他 CA 证书来进行服务器身份验证，包括 Amazon Root CA 3 证书（ECC 256 位密钥）。要更改 coreMQTT 双向身份验证演示的根 CA，请执行以下操作：

1. 在文本编辑器中，打开 *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/mqtt_demo_mutual_auth_config.h文件。
2. 在文件中找到以下行。

   ```
   * #define democonfigROOT_CA_PEM    "...insert here..."
   ```

   取消注释此行，如有必要，将其移到注释块末尾的 */ 之后。
3. 复制要使用的 CA 证书，然后将其粘贴到文本 "...insert here..." 处。结果应该类似以下示例。

   ```
   #define democonfigROOT_CA_PEM   "-----BEGIN CERTIFICATE-----\n"\
   "MIIBtjCCAVugAwIBAgITBmyf1XSXNmY/Owua2eiedgPySjAKBggqhkjOPQQDAjA5\n"\
   "MQswCQYDVQQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwFwYDVQQDExBBbWF6b24g\n"\
   "Um9vdCBDQSAzMB4XDTE1MDUyNjAwMDAwMFoXDTQwMDUyNjAwMDAwMFowOTELMAkG\n"\
   "A1UEBhMCVVMxDzANBgNVBAoTBkFtYXpvbjEZMBcGA1UEAxMQQW1hem9uIFJvb3Qg\n"\
   "Q0EgMzBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABCmXp8ZBf8ANm+gBG1bG8lKl\n"\
   "ui2yEujSLtf6ycXYqm0fc4E7O5hrOXwzpcVOho6AF2hiRVd9RFgdszflZwjrZt6j\n"\
   "QjBAMA8GA1UdEwEB/wQFMAMBAf8wDgYDVR0PAQH/BAQDAgGGMB0GA1UdDgQWBBSr\n"\
   "ttvXBp43rDCGB5Fwx5zEGbF4wDAKBggqhkjOPQQDAgNJADBGAiEA4IWSoxe3jfkr\n"\
   "BqWTrBqYaGFy+uGh0PsceGCmQ5nFuMQCIQCcAu/xlJyzlvnrxir4tiz+OpAUFteM\n"\
   "YyRIHN8wfdVoOw==\n"\
   "-----END CERTIFICATE-----\n"
   ```
4. （可选）您可以更改其他演示的根 CA。对每个 *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/*demo-name*_config.h 文件重复步骤 1 到 3。

coreMQTT 代理连接共享演示

> **Important**
>
> 该演示托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

简介

CoreMQTT 连接共享演示项目向您展示了如何使用多线程应用程序通过 TLS 建立与 AWS MQTT 代理的连接，并在客户端和服务器之间进行相互身份验证。此演示使用基于 mbedTLS 的传输接口实现来

建立经过服务器和客户端身份验证的 TLS 连接，并演示了 QoS 1 级别的 MQTT 订阅发布工作流程。该演示订阅主题筛选条件，发布与筛选条件匹配的主题，然后等待从 QoS 1 级别的服务器接收这些消息。对于每个创建的任务，这种向代理发布消息并从代理接收相同消息的循环过程会无限期地重复。本演示中的消息按照 QoS 1 发送，这样可以保证至少有一次按照 MQTT 规范传递。

> **Note**
>
> 要设置和运行 FreeRTOS 演示，请按照开始使用 FreeRTOS中的步骤操作。

此演示使用线程安全队列来保存与 MQTT API 交互的命令。此演示中有两个任务需要注意。

- MQTT 代理（主）任务处理命令队列中的命令，而其他任务则将它们排入队列。此任务会进入循环，它会在此期间处理命令队列中的命令。如果收到终止命令，则此任务将退出循环。
- 演示 subpub 任务会创建对 MQTT 主题的订阅，然后创建发布操作并将其推送到命令队列中。然后，这些发布操作由 MQTT 代理任务运行。演示 subpub 任务会等待发布完成（通过执行命令完成回调来指示），然后在开始下一次发布之前进入短暂的延迟。此任务展示了应用程序任务如何使用 coreMQTT 代理 API 的示例。

对于传入的发布消息，coreMQTT 代理会调用单个回调函数。此演示还包括一个订阅管理器，它允许任务指定一个回调，以便为它们已订阅的主题的传入发布消息进行调用。在此演示中，代理的传入发布回调会调用订阅管理器，将发布分散到任何已注册订阅的任务中。

此演示使用具有双向身份验证的 TLS 连接到 AWS。如果网络在演示过程中意外断开连接，则客户端会尝试使用指数回退逻辑重新连接。如果客户端重新连接成功，但代理无法恢复之前的会话，则客户端将重新订阅与上一个会话相同的主题。

单线程与多线程

coreMQTT 有两种使用模式，即单线程和多线程（多任务处理）。单线程模型使用来自仅一个线程的 coreMQTT 库，并要求您在 MQTT 库中重复进行显式调用。多线程使用案例可以在代理（或进程守护程序）任务的后台运行 MQTT 协议，如此处记录的演示所示。在代理任务中运行 MQTT 协议时，无需显式管理任何 MQTT 状态或调用 `MQTT_ProcessLoop` API 函数。此外，如果您使用代理任务，则多个应用程序任务可共享单个 MQTT 连接，而无需同步原语（例如，互斥锁）。

源代码

演示源文件已命名`mqtt_agent_task.c`，可以在*`freertos`*`/demos/coreMQTT_Agent/`目录和GitHub网站上找到。`simple_sub_pub_demo.c`

功能

此演示至少创建两个任务：一个用于处理 MQTT API 调用请求的主要任务，以及用于创建这些请求的可配置子任务数量。在此演示中，主任务会创建子任务，调用处理循环，然后进行清理。主任务创建与代理的单个 MQTT 连接，该连接在子任务之间共享。子任务通过代理创建 MQTT 订阅，然后向其发布消息。每个子任务的发布都使用唯一的主题。

主任务

主应用程序任务 RunCoreMQTTAgentDemo 建立 MQTT 会话，创建子任务，并运行处理循环 MQTTAgent_，CommandLoop直到收到终止命令。如果网络意外断开连接，演示将在后台重新连接到代理，并与代理重新建立订阅。处理循环终止后，它会断开与代理的连接。

命令

当您调用 coreMQTT 代理 API 时，它会创建一个发送到代理任务队列的命令，然后在 MQTTAgent_CommandLoop() 中进行处理。在创建命令时，可以传递可选的完成回调和上下文参数。相应的命令完成后，将使用传递的上下文以及作为命令的结果创建的任何返回值来调用完成回调。完成回调的签名如下：

```
typedef void (* MQTTAgentCommandCallback_t )( void * pCmdCallbackContext,
                                              MQTTAgentReturnInfo_t * pReturnInfo );
```

命令完成上下文是用户定义的；对于这个演示，它是：struct。 MQTTAgent CommandContext

在以下情况下即认为命令已完成：

- 订阅、取消订阅和发布 (QoS > 0)：收到相应的确认数据包后。
- 所有其他操作：调用相应的 coreMQTT API 后。

命令使用的任何结构（包括发布信息、订阅信息和完成上下文）都必须保持在作用范围内，直到命令完成。在调用完成回调之前，调用任务不得重用命令的任何结构。请注意，由于完成回调由 MQTT 代理调用，因此，它将使用代理任务的线程上下文运行，而不是创建命令的任务。进程间通信机制（例如，任务通知或队列）可用于向调用任务发出命令完成的信号。

运行命令循环

命令在MQTTAgent_CommandLoop() 中持续处理。如果没有要处理的命令，则循环将等待向队列中添加一个命令的最长等待时间 MQTT_AGENT_MAX_EVENT_QUEUE_WAIT_TIME；如果未添加任何命

令，则循环将运行 MQTT_ProcessLoop() 的一次迭代。这样既可确保管理 MQTT Keep-Alive，又可确保即使队列中没有命令，也能收到传入的发布。

在以下原因下，命令循环函数将返回：

- 命令返回除 MQTTSuccess 之外的任何状态代码。命令循环返回错误状态，因此您可以决定如何处理。在此演示中，重新建立了 TCP 连接并尝试重新连接。如果出现任何错误，则可以在后台进行重新连接，而没有其他使用 MQTT 的任务产生的任何干预。
- 处理了断开连接命令 (MQTTAgent_Disconnect)。退出了命令循环，以便断开 TCP 连接。
- 处理了终止命令 (MQTTAgent_Terminate)。此命令还将任何仍在队列中或正在等待确认数据包的命令标记为错误，返回代码为 MQTTRecvFailed。

订阅管理器

由于该演示使用多个主题，因此，订阅管理器是将订阅的主题与唯一的回调或任务相关联的便捷方式。此演示中的订阅管理器是单线程的，因此不应同时用于多个任务。在此演示中，订阅管理器函数只能从传递给 MQTT 代理的回调函数中调用，并且只能在代理任务的线程上下文中运行。

简单的订阅-发布任务

的每个实例都会 prvSimpleSubscribePublishTask创建对 MQTT 主题的订阅，并为该主题创建发布操作。为了演示多种发布类型，偶数编号的任务使用 QoS 0（发送发布数据包后即完成），奇数任务使用 QoS 1（收到 PUBACK 数据包后即完成）。

Over-the-air 更新演示应用程序

FreeRTOS 包含一个演示应用程序，用于演示 (OTA) 库 over-the-air的功能。OTA 演示应用程序位于 *freertos*/demos/ota/ota_demo_core_mqtt/ota_demo_core_mqtt.c 或 *freertos*/demos/ota/ota_demo_core_http/ota_demo_core_http.c 文件中。

OTA 演示应用程序执行以下操作：

1. 初始化 FreeRTOS 网络堆栈和 MQTT 缓冲池。
2. 使用 vRunOTAUpdateDemo() 创建任务来执行 OTA 库。
3. 使用 _establishMqttConnection() 创建 MQTT 客户端。
4. 使用连接到 AWS IoT MQTT 代理IotMqtt_Connect()并注册 MQTT 断开连接回调:。prvNetworkDisconnectCallback
5. 调用 OTA_AgentInit() 创建 OTA 任务，并注册在 OTA 任务完成时使用的回调。

6. 使用 `xOTAConnectionCtx.pvControlClient = _mqttConnection;` 重用 MQTT 连接。
7. 如果 MQTT 断开连接，应用程序将暂停 OTA 代理，尝试使用带有抖动的指数延迟重新连接，然后恢复 OTA 代理。

在使用 OTA 更新之前，请完成 FreeRTOS 更新 Over-the-Air 中的所有先决条件

在完成 OTA 更新的设置后，在支持 OTA 功能的平台上下载、构建、刷写并运行 FreeRTOS OTA 演示。设备特定的演示说明可用于以下取得 FreeRTOS 认证的设备：

- 德州仪器 CC322 0SF-LAUNCHXL
- 微芯片好奇号 PIC32 MZEF
- Espressif ESP32
- 在瑞萨 N 上下载、构建、刷新并运行 FreeRTOS OTA 演示 RX65

在设备上构建、刷新和运行 OTA 演示应用程序后，您可以使用 AWS IoT 控制台或创建 OTA 更新任务。 AWS CLI 在您创建 OTA 更新任务之后，连接到终端仿真器以查看 OTA 更新的进度。记下此过程中生成的任何错误。

成功 OTA 更新作业会显示类似以下内容的输出。为简便起见，在本示例中，我们删除了列表中的一些行。

```
    249 21207 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT]  Received: 0
 Queued: 0   Processed: 0   Dropped: 0
    250 21247 [MQTT Agent Task] [core_mqtt.c:886] [INFO] [MQTT] Packet received.
 ReceivedBytes=601.
    251 21247 [MQTT Agent Task] [core_mqtt.c:1045] [INFO] [MQTT] De-serialized incoming
 PUBLISH packet: DeserializerResult=MQTTSuccess.
    252 21248 [MQTT Agent Task] [core_mqtt.c:1058] [INFO] [MQTT] State record updated.
 New state=MQTTPubAckSend.
    253 21249 [MQTT Agent Task] [ota_demo_core_mqtt.c:976] [INFO] [MQTT] Received job
 message callback, size 548.
    254 21252 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[execution.jobId: AFR_OTA-9702f1a3-b747-4c3e-a0eb-a3b0cf83ddbb]
    255 21253 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[execution.jobDocument.afr_ota.streamname: AFR_OTA-945d320b-a18b-441b-
b435-4a18d4e7671f]
    256 21255 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[execution.jobDocument.afr_ota.protocols: ["MQTT"]]
    257 21256 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[filepath: aws_demos.bin]
```

```
    258 21257 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
 value]=[filesize: 1164016]
    259 21258 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
 value]=[fileid: 0]
    260 21259 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[certfile: ecdsa-sha256-signer.crt.pem]
    261 21260 [OTA Agent Task] [ota.c:1575] [INFO] [OTA] Extracted parameter [ sig-
sha256-ecdsa: MEQCIE1SFkIHHiZAvkPpu6McJtx7SYoD... ]
    262 21261 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
 value]=[fileType: 0]
    263 21262 [OTA Agent Task] [ota.c:2199] [INFO] [OTA] Job document was accepted.
 Attempting to begin the update.
    264 21263 [OTA Agent Task] [ota.c:2323] [INFO] [OTA] Job parsing success:
 OtaJobParseErr_t=OtaJobParseErrNone, Job name=AFR_OTA-9702f1a3-b747-4c3e-a0eb-
a3b0cf83ddbb
    265 21318 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT]  Received: 0
 Queued: 0   Processed: 0   Dropped: 0
    266 21418 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT]  Received: 0
 Queued: 0   Processed: 0   Dropped: 0
    267 21469 [OTA Agent Task] [ota.c:938] [INFO] [OTA] Setting OTA data interface.
    268 21470 [OTA Agent Task] [ota.c:2839] [INFO] [OTA] Current State=[CreatingFile],
 Event=[ReceivedJobDocument], New state=[CreatingFile]
    269 21482 [MQTT Agent Task] [core_mqtt.c:886] [INFO] [MQTT] Packet received.
 ReceivedBytes=3.
    270 21483 [OTA Agent Task] [ota_demo_core_mqtt.c:1503] [INFO] [MQTT] SUBSCRIBED
 to topic $aws/things/__test_infra_thing71/streams/AFR_OTA-945d320b-a18b-441b-
b435-4a18d4e7671f/data/cbor to bro
    271 21484 [OTA Agent Task] [ota.c:2839] [INFO] [OTA] Current
 State=[RequestingFileBlock], Event=[CreateFile], New state=[RequestingFileBlock]
    272 21518 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT]  Received: 0
 Queued: 0   Processed: 0   Dropped: 0
    273 21532 [MQTT Agent Task] [core_mqtt_agent_command_functions.c:76] [INFO] [MQTT]
 Publishing message to $aws/things/__test_infra_thing71/streams/AFR_OTA-945d320b-
a18b-441b-b435-4a18d4e7671f/
    274 21534 [OTA Agent Task] [ota_demo_core_mqtt.c:1553] [INFO] [MQTT] Sent PUBLISH
 packet to broker $aws/things/__test_infra_thing71/streams/AFR_OTA-945d320b-a18b-441b-
b435-4a18d4e7671f/get/cbor
    275 21534 [OTA Agent Task] [ota_mqtt.c:1112] [INFO] [OTA] Published to MQTT
 topic to request the next block: topic=$aws/things/__test_infra_thing71/streams/
AFR_OTA-945d320b-a18b-441b-b435-4a1
    276 21537 [OTA Agent Task] [ota.c:2839] [INFO] [OTA] Current
 State=[WaitingForFileBlock], Event=[RequestFileBlock], New state=[WaitingForFileBlock]
    277 21558 [MQTT Agent Task] [core_mqtt.c:886] [INFO] [MQTT] Packet received.
 ReceivedBytes=4217.
```

```
   278 21559 [MQTT Agent Task] [core_mqtt.c:1045] [INFO] [MQTT] De-serialized incoming
PUBLISH packet: DeserializerResult=MQTTSuccess.
   279 21560 [MQTT Agent Task] [core_mqtt.c:1058] [INFO] [MQTT] State record updated.
New state=MQTTPublishDone.
   280 21561 [MQTT Agent Task] [ota_demo_core_mqtt.c:1026] [INFO] [MQTT] Received data
message callback, size 4120.
   281 21563 [OTA Agent Task] [ota.c:2464] [INFO] [OTA] Received valid file block:
Block index=0, Size=4096
   282 21566 [OTA Agent Task] [ota.c:2683] [INFO] [OTA] Number of blocks remaining:
284

   ... // Output removed for brevity

   3672 42745 [OTA Agent Task] [ota.c:2464] [INFO] [OTA] Received valid file block:
Block index=284, Size=752
   3673 42747 [OTA Agent Task] [ota.c:2633] [INFO] [OTA] Received final block of the
update.
   (428298) ota_pal: No such certificate file: ecdsa-sha256-signer.crt.pem. Using
certificate in ota_demo_config.h.
   3674 42818 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT]  Received: 285
Queued: 285   Processed: 284   Dropped: 0
   3675 42918 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT]  Received: 285
Queued: 285   Processed: 284   Dropped: 0

   ... // Output removed for brevity

   3678 43197 [OTA Agent Task] [ota.c:2654] [INFO] [OTA] Received entire update and
validated the signature.
   3685 43215 [OTA Agent Task] [ota_demo_core_mqtt.c:862] [INFO] [MQTT] Received
OtaJobEventActivate callback from OTA Agent.

   ... // Output removed for brevity

   2 39 [iot_thread] [INFO ][DEMO][390] ---------STARTING DEMO---------

   [0;32mI (3633) WIFI: WIFI_EVENT_STA_CONNECTED
   [0;32mI (4373) WIFI: SYSTEM_EVENT_STA_GOT_IP

   ... // Output removed for brevity

   4 351 [sys_evt] [INFO ][DEMO][3510] Connected to WiFi access point, ip address:
255.255.255.0.
   5 351 [iot_thread] [INFO ][DEMO][3510] Successfully initialized the demo. Network
type for the demo: 1
```

```
    6 351 [iot_thread] [ota_demo_core_mqtt.c:1902] [INFO] [MQTT] OTA over MQTT demo,
 Application version 0.9.1
    7 351 [iot_thread] [ota_demo_core_mqtt.c:1323] [INFO] [MQTT] Creating a TLS
 connection to <endpoint>-ats.iot.us-west-2.amazonaws.com:8883.
    9 718 [iot_thread] [core_mqtt.c:886] [INFO] [MQTT] Packet received.
 ReceivedBytes=2.
    10 718 [iot_thread] [core_mqtt_serializer.c:970] [INFO] [MQTT] CONNACK session
 present bit not set.
    11 718 [iot_thread] [core_mqtt_serializer.c:912] [INFO] [MQTT] Connection accepted.

    ... // Output removed for brevity

    17 736 [OTA Agent Task] [ota_demo_core_mqtt.c:1503] [INFO] [MQTT] SUBSCRIBED to
 topic $aws/things/__test_infra_thing71/jobs/notify-next to broker.
    18 737 [OTA Agent Task] [ota_mqtt.c:381] [INFO] [OTA] Subscribed to MQTT topic:
 $aws/things/__test_infra_thing71/jobs/notify-next
    30 818 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT]  Received: 0
 Queued: 0   Processed: 0   Dropped: 0
    31 819 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[execution.jobId: AFR_OTA-9702f1a3-b747-4c3e-a0eb-a3b0cf83ddbb]
    32 820 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
 value]=[execution.statusDetails.updatedBy: 589824]
    33 822 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[execution.jobDocument.afr_ota.streamname: AFR_OTA-945d320b-a18b-441b-
b435-4a18d4e7671f]
    34 823 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[execution.jobDocument.afr_ota.protocols: ["MQTT"]]
    35 824 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[filepath: aws_demos.bin]
    36 825 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
 value]=[filesize: 1164016]
    37 826 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
 value]=[fileid: 0]
    38 827 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[certfile: ecdsa-sha256-signer.crt.pem]
    39 828 [OTA Agent Task] [ota.c:1575] [INFO] [OTA] Extracted parameter [ sig-sha256-
ecdsa: MEQCIE1SFkIHHiZAvkPpu6McJtx7SYoD... ]
    40 829 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
 value]=[fileType: 0]
    41 830 [OTA Agent Task] [ota.c:2102] [INFO] [OTA] In self test mode.
    42 830 [OTA Agent Task] [ota.c:1936] [INFO] [OTA] New image has a higher version
 number than the current image: New image version=0.9.1, Previous image version=0.9.0
    43 832 [OTA Agent Task] [ota.c:2120] [INFO] [OTA] Image version is valid: Begin
 testing file: File ID=0
```

```
    53 896 [OTA Agent Task] [ota.c:794] [INFO] [OTA] Beginning self-test.
    62 971 [OTA Agent Task] [ota_demo_core_mqtt.c:1553] [INFO] [MQTT] Sent PUBLISH
 packet to broker $aws/things/__test_infra_thing71/jobs/AFR_OTA-9702f1a3-b747-4c3e-
a0eb-a3b0cf83ddbb/update to br63 971 [MQTT Agent Task] [core_mqtt.c:1045] [INFO] [MQTT]
 De-serialized incoming PUBLISH packet: DeserializerResult=MQTTSuccess.
    65 973 [MQTT Agent Task] [core_mqtt.c:1058] [INFO] [MQTT] State record updated. New
 state=MQTTPublishDone.
    64 973 [OTA Agent Task] [ota_demo_core_mqtt.c:902] [INFO] [MQTT] Successfully
 updated with the new image.
```

Over-the-air 演示配置

OTA 演示配置是在 `aws_iot_ota_update_demo.c` 中提供的特定于演示的配置选项。这些配置与在 OTA 库配置文件中提供的 OTA 库配置不同。

OTA_DEMO_KEEP_ALIVE_SECONDS

对于 MQTT 客户端，此配置是从完成一个控制数据包的传输到开始发送下一个控制数据包之间可能经过的最大时间间隔。如果没有控制数据包，则发送 PINGREQ。如果时间达到此保持活动时间间隔的一倍半而内未发送消息或 PINGREQ 数据包，则代理必须断开客户端。请根据应用程序的要求调整此配置。

OTA_DEMO_CONN_RETRY_BASE_INTERVAL_SECONDS

重试网络连接之前的基本间隔（以秒为单位）。OTA 演示将在此基本间隔后尝试重新连接。每次尝试失败后，间隔都会增加一倍。间隔中还会添加一个随机延迟，最高为该基本延迟的最大值。

OTA_DEMO_CONN_RETRY_MAX_INTERVAL_SECONDS

重试网络连接之前的最大间隔（以秒为单位）。每次尝试失败时，重新连接延迟都会增加一倍，但最多只能达到这一最大值加同一间隔的抖动。

在德州仪器 0SF-LAUNCHXL 上下载、构建、刷新和运行 FreeRTOS OTA 演示 CC322

> **Important**
>
> 该参考集成托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

下载 FreeRTOS 和 OTA 演示代码

- 你可以在 https://github.com/FreeRTOS/FreeRTOS GitHub 网站上下载源代码。

构建演示应用程序

1. 按照中的开始使用 FreeRTOS说明将aws_demos项目导入 Code Composer Studio，配置您的 AWS IoT 终端节点、Wi-Fi SSID 和密码以及主板的私钥和证书。
2. 打开 *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h，注释掉 #define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED 并定义 CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED 或 CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED。
3. 构建解决方案，并确保其构建没有错误。
4. 启动终端模拟器，并使用以下设置连接到主板：
   - 波特率：115200
   - 数据位：8
   - 奇偶校验：无
   - 停止位：1
5. 在主板上运行项目，确认它可以连接到 Wi-Fi 和 AWS IoT MQTT 消息代理。

终端模拟器在运行时应当显示类似以下内容的文本：

```
 0 1000 [Tmr Svc] Simple Link task created
Device came up in Station mode
1 2534 [Tmr Svc] Write certificate...
2 5486 [Tmr Svc] [ERROR] Failed to destroy object. PKCS11_PAL_DestroyObject failed.
3 5486 [Tmr Svc] Write certificate...
4 5776 [Tmr Svc] Security alert threshold = 15
5 5776 [Tmr Svc] Current number of alerts = 1
6 5778 [Tmr Svc] Running Demos.
7 5779 [iot_thread] [INFO ][DEMO][5779] ---------STARTING DEMO---------
8 5779 [iot_thread] [INFO ][INIT][5779] SDK successfully initialized.
Device came up in Station mode
[WLAN EVENT] STA Connected to the AP: afrlab-pepper , BSSID: 74:83:c2:b4:46:27
[NETAPP EVENT] IP acquired by the device
Device has connected to afrlab-pepper
```

```
    Device IP Address is 192.168.36.176
    9 8283 [iot_thread] [INFO ][DEMO][8282] Successfully initialized the demo. Network
 type for the demo: 1
    10 8283 [iot_thread] [INFO] OTA over MQTT demo, Application version 0.9.0
    11 8283 [iot_thread] [INFO] Creating a TLS connection to <endpoint>-ats.iot.us-
west-2.amazonaws.com:8883.
    12 8852 [iot_thread] [INFO] Creating an MQTT connection to <endpoint>-ats.iot.us-
west-2.amazonaws.com.
    13 8914 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
    14 8914 [iot_thread] [INFO] CONNACK session present bit not set.
    15 8914 [iot_thread] [INFO] Connection accepted.
    16 8914 [iot_thread] [INFO] Received MQTT CONNACK successfully from broker.
    17 8914 [iot_thread] [INFO] MQTT connection established with the broker.
    18 8915 [iot_thread] [INFO]  Received: 0   Queued: 0   Processed: 0   Dropped: 0
    19 8953 [OTA Agent T] [INFO] Current State=[RequestingJob], Event=[Start], New
 state=[RequestingJob]
    20 9008 [MQTT Agent ] [INFO] Packet received. ReceivedBytes=3.
    21 9015 [OTA Agent T] [INFO] SUBSCRIBED to topic $aws/things/__test_infra_thing73/
jobs/notify-next to broker.
    22 9015 [OTA Agent T] [INFO] Subscribed to MQTT topic: $aws/things/
__test_infra_thing73/jobs/notify-next
    23 9504 [MQTT Agent ] [INFO] Publishing message to $aws/things/
__test_infra_thing73/jobs/$next/get.
    24 9535 [MQTT Agent ] [INFO] Packet received. ReceivedBytes=2.
    25 9535 [MQTT Agent ] [INFO] Ack packet deserialized with result: MQTTSuccess.
    26 9536 [MQTT Agent ] [INFO] State record updated. New state=MQTTPublishDone.
    27 9537 [OTA Agent T] [INFO] Sent PUBLISH packet to broker $aws/things/
__test_infra_thing73/jobs/$next/get to broker.
    28 9537 [OTA Agent T] [WARN] OTA Timer handle NULL for Timerid=0, can't stop.
    29 9537 [OTA Agent T] [INFO] Current State=[WaitingForJob],
 Event=[RequestJobDocument], New state=[WaitingForJob]
    30 9539 [MQTT Agent ] [INFO] Packet received. ReceivedBytes=120.
    31 9539 [MQTT Agent ] [INFO] De-serialized incoming PUBLISH packet:
 DeserializerResult=MQTTSuccess.
    32 9540 [MQTT Agent ] [INFO] State record updated. New state=MQTTPublishDone.
    33 9540 [MQTT Agent ] [INFO] Received job message callback, size 62.
    34 9616 [OTA Agent T] [INFO] Failed job document content check: Required job
 document parameter was not extracted: parameter=execution
    35 9616 [OTA Agent T] [INFO] Failed job document content check: Required job
 document parameter was not extracted: parameter=execution.jobId
    36 9617 [OTA Agent T] [INFO] Failed job document content check: Required job
 document parameter was not extracted: parameter=execution.jobDocument
    37 9617 [OTA Agent T] [INFO] Failed job document content check: Required job
 document parameter was not extracted: parameter=execution.jobDocument.afr_ota
```

```
   38 9617 [OTA Agent T] [INFO] Failed job document content
check: Required job document parameter was not extracted:
parameter=execution.jobDocument.afr_ota.protocols
   39 9618 [OTA Agent T] [INFO] Failed job document content check: Required job
document parameter was not extracted: parameter=execution.jobDocument.afr_ota.files
   40 9618 [OTA Agent T] [INFO] Failed job document content check: Required job
document parameter was not extracted: parameter=filesize
   41 9618 [OTA Agent T] [INFO] Failed job document content check: Required job
document parameter was not extracted: parameter=fileid
   42 9619 [OTA Agent T] [INFO] Failed to parse JSON document as AFR_OTA job:
DocParseErr_t=7
   43 9619 [OTA Agent T] [INFO] No active job available in received job document:
OtaJobParseErr_t=OtaJobParseErrNoActiveJobs
   44 9619 [OTA Agent T] [ERROR] Failed to execute state transition handler: Handler
returned error: OtaErr_t=OtaErrJobParserError
   45 9620 [OTA Agent T] [INFO] Current State=[WaitingForJob],
Event=[ReceivedJobDocument], New state=[CreatingFile]
   46 9915 [iot_thread] [INFO]  Received: 0   Queued: 0   Processed: 0   Dropped: 0
   47 10915 [iot_thread] [INFO]  Received: 0   Queued: 0   Processed: 0   Dropped: 0
   48 11915 [iot_thread] [INFO]  Received: 0   Queued: 0   Processed: 0   Dropped: 0
   49 12915 [iot_thread] [INFO]  Received: 0   Queued: 0   Processed: 0   Dropped: 0
   50 13915 [iot_thread] [INFO]  Received: 0   Queued: 0   Processed: 0   Dropped: 0
   51 14915 [iot_thread] [INFO]  Received: 0   Queued: 0   Processed: 0   Dropped: 0
```

在 Microchip Curiosity MZEF 上下载、构建、刷新和运行 FreeRTOS OTA 演示 PIC32

Important

该参考集成托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

Note

根据Microchip的协议，我们将从FreeRTOS参考集成存储库主分支中删除好奇号 PIC32 MZEF（DM320104），并且将在新版本中不再使用它。Microchip已发布官方通知，不再建议将 PIC32 MZEF（DM320104）用于新设计。 PIC32MZEF 项目和源代码仍然可以通过之前的版本标签进行访问。Microchip 建议客户使用好奇号 PIC32MZ-EF-2.0 开发板 (DM320209) 进

行新设计。该 PIC32MZv1 平台仍然可以在 FreeRTOS 参考集成存储库的 v202012. 00 中找到。但是，FreeRTOS 参考的 v202107.00 不再支持该平台。

下载 FreeRTOS OTA 演示代码

- 你可以在 https://github.com/FreeRTOS/FreeRTOS GitHub 网站上下载源代码。

构建 OTA 更新演示应用程序

1. 按照中的开始使用 FreeRTOS说明将aws_demos项目导入 MPLAB X IDE，配置您的 AWS IoT 端点、Wi-Fi SSID 和密码以及主板的私钥和证书。
2. 打开 vendors/*vendor*/boards/*board*/aws_demos/config_files/ota_demo_config.h 文件，然后输入您的证书。

   ```
   [] = "your-certificate-key";
   ```

3. 将代码签名证书的内容粘贴到此处：

   ```
   #define otapalconfigCODE_SIGNING_CERTIFICATE [] = "your-certificate-key";
   ```

   采用与 aws_clientcredential_keys.h 相同的格式，每一行必须以换行符 ('\n') 结束，并用引号括起来。

   例如，证书看起来应当如下所示：

   ```
   "-----BEGIN CERTIFICATE-----\n"
   "MIIBXTCCAQOgAwIBAgIJAM4DeybZcTwKMAoGCCqGSM49BAMCMCExHzAdBgNVBAMM\n"
   "FnRlc3Rf62lnbmVyQGFtYXpvbi5jb20wHhcNMTcxMTAzMTkxODM1WhcNMTgxMTAz\n"
   "MTkxODM2WjAhMR8wHQYDVQBBZZZ0ZXN0X3NpZ25lckBhbWF6b24uY29tMFkwEwYH\n"
   "KoZIzj0CAQYIKoZIzj0DAQcDQgAERavZfvwL1X+E4dIF7dbkVMUn4IrJ1CAsFkc8\n"
   "gZxPzn683H40XMKltDZPEwr9ng78w9+QYQg7ygnr2stz8yhh06MkMCIwCwYDVR0P\n"
   "BAQDAgeAMBMGA1UdJQQMMAoGCCsGAQUFBwMDMAoGCCqGSM49BAMCA0gAMEUCIF0R\n"
   "r5cb7rEUNtWOvGd05MacrgOABfSoVYvBOK9fP63WAqt5h3BaS123coKSGg84twlq\n"
   "TkO/pV/xEmyZmZdV+HxV/OM=\n"
   "-----END CERTIFICATE-----\n";
   ```

4. 安装 Python 3 或更高版本。
5. 运行 pip install pyopenssl 来安装 pyOpenSSL。

6. 复制路径 demos/ota/bootloader/utility/codesigner_cert_utility/ 下 .pem 格式的代码签名证书。重命名证书文件 aws_ota_codesigner_certificate.pem。
7. 打开 *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h，注释掉 #define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED 并定义 CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED 或 CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED。
8. 构建解决方案，并确保其构建没有错误。
9. 启动终端模拟器，并使用以下设置连接到主板：
   - 波特率：115200
   - 数据位：8
   - 奇偶校验：无
   - 停止位：1
10. 从开发板上拔下调试器的插头，然后在主板上运行该项目，以确认它可以连接到 Wi-Fi 和 AWS IoT MQTT 消息代理。

运行项目时，MPLAB X IDE 应当打开输出窗口。确保选中该ICD4选项卡。您应当看到如下输出。

```
Bootloader version 00.09.00
[prvBOOT_Init] Watchdog timer initialized.
[prvBOOT_Init] Crypto initialized.

[prvValidateImage] Validating image at Bank : 0
[prvValidateImage] No application image or magic code present at: 0xbd000000
[prvBOOT_ValidateImages] Validation failed for image at 0xbd000000

[prvValidateImage] Validating image at Bank : 1
[prvValidateImage] No application image or magic code present at: 0xbd100000
[prvBOOT_ValidateImages] Validation failed for image at 0xbd100000

[prvBOOT_ValidateImages] Booting default image.


>0 36246 [IP-task] vDHCPProcess: offer ac140a0eip
                                                    1 36297 [IP-task] vDHCPProcess: offer
 ac140a0eip
                2 36297 [IP-task]
```

```
IP Address: 172.20.10.14
3 36297 [IP-task] Subnet Mask: 255.255.255.240
4 36297 [IP-task] Gateway Address: 172.20.10.1
5 36297 [IP-task] DNS Server Address: 172.20.10.1


6 36299 [OTA] OTA demo version 0.9.2
7 36299 [OTA] Creating MQTT Client...
8 36299 [OTA] Connecting to broker...
9 38673 [OTA] Connected to broker.
10 38793 [OTA Task] [prvSubscribeToJobNotificationTopics] OK: $aws/things/devthingota/
jobs/$next/get/accepted
11 38863 [OTA Task] [prvSubscribeToJobNotificationTopics] OK: $aws/things/devthingota/
jobs/notify-next
12 38863 [OTA Task] [OTA_CheckForUpdate] Request #0
13 38964 [OTA] [OTA_AgentInit] Ready.
14 38973 [OTA Task] [prvParseJSONbyModel] Extracted parameter [ clientToken:
 0:devthingota ]
15 38973 [OTA Task] [prvParseJSONbyModel] parameter not present: execution
16 38973 [OTA Task] [prvParseJSONbyModel] parameter not present: jobId
17 38973 [OTA Task] [prvParseJSONbyModel] parameter not present: jobDocument
18 38973 [OTA Task] [prvParseJSONbyModel] parameter not present: streamname
19 38973 [OTA Task] [prvParseJSONbyModel] parameter not present: files
20 38975 [OTA Task] [prvParseJSONbyModel] parameter not present: filepath
21 38975 [OTA Task] [prvParseJSONbyModel] parameter not present: filesize
22 38975 [OTA Task] [prvParseJSONbyModel] parameter not present: fileid
23 38975 [OTA Task] [prvParseJSONbyModel] parameter not present: certfile
24 38975 [OTA Task] [prvParseJSONbyModel] parameter not present: sig-sha256-ecdsa
25 38975 [OTA Task] [prvParseJobDoc] Ignoring job without ID.
26 38975 [OTA Task] [prvOTA_Close] Context->0x8003b620
27 38975 [OTA Task] [prvPAL_Abort] Abort - OK
28 39964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
29 40964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
30 41964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
31 42964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
32 43964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
33 44964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
34 45964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
35 46964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
36 47964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
```

终端模拟器应当显示类似以下内容的文本：

```
AWS Validate: no valid signature in descr: 0xbd000000
AWS Validate: no valid signature in descr: 0xbd100000


>AWS Launch:  No Map performed. Running directly from address: 0x9d000020?
AWS Launch:  wait for app at: 0x9d000020
WILC1000: Initializing...
0 0

>[None] Seed for randomizer: 1172751941
1 0 [None] Random numbers: 00004272 00003B34 00000602 00002DE3
Chip ID 1503a0

[spi_cmd_rsp][356][nmi spi]: Failed cmd response read, bus error...

[spi_read_reg][1086][nmi spi]: Failed cmd response, read reg (0000108c)...

[spi_read_reg][1116]Reset and retry 10 108c

Firmware ver. : 4.2.1

Min driver ver : 4.2.1

Curr driver ver: 4.2.1

WILC1000: Initialization successful!

Start Wi-Fi Connection...
Wi-Fi Connected
2 7219 [IP-task] vDHCPProcess: offer c0a804beip
3 7230 [IP-task] vDHCPProcess: offer c0a804beip
4 7230 [IP-task]

IP Address: 192.168.4.190
5 7230 [IP-task] Subnet Mask: 255.255.240.0
6 7230 [IP-task] Gateway Address: 192.168.0.1
7 7230 [IP-task] DNS Server Address: 208.67.222.222


8 7232 [OTA] OTA demo version 0.9.0
9 7232 [OTA] Creating MQTT Client...
10 7232 [OTA] Connecting to broker...
11 7232 [OTA] Sending command to MQTT task.
```

```
12 7232 [MQTT] Received message 10000 from queue.
13 8501 [IP-task] Socket sending wakeup to MQTT task.
14 10207 [MQTT] Received message 0 from queue.
15 10256 [IP-task] Socket sending wakeup to MQTT task.
16 10256 [MQTT] Received message 0 from queue.
17 10256 [MQTT] MQTT Connect was accepted. Connection established.
18 10256 [MQTT] Notifying task.
19 10257 [OTA] Command sent to MQTT task passed.
20 10257 [OTA] Connected to broker.
21 10258 [OTA Task] Sending command to MQTT task.
22 10258 [MQTT] Received message 20000 from queue.
23 10306 [IP-task] Socket sending wakeup to MQTT task.
24 10306 [MQTT] Received message 0 from queue.
25 10306 [MQTT] MQTT Subscribe was accepted. Subscribed.
26 10306 [MQTT] Notifying task.
27 10307 [OTA Task] Command sent to MQTT task passed.
28 10307 [OTA Task] [OTA] Subscribed to topic: $aws/things/Microchip/jobs/$next/get/
accepted

29 10307 [OTA Task] Sending command to MQTT task.
30 10307 [MQTT] Received message 30000 from queue.
31 10336 [IP-task] Socket sending wakeup to MQTT task.
32 10336 [MQTT] Received message 0 from queue.
33 10336 [MQTT] MQTT Subscribe was accepted. Subscribed.
34 10336 [MQTT] Notifying task.
35 10336 [OTA Task] Command sent to MQTT task passed.
36 10336 [OTA Task] [OTA] Subscribed to topic: $aws/things/Microchip/jobs/notify-next

37 10336 [OTA Task] [OTA] Check For Update #0
38 10336 [OTA Task] Sending command to MQTT task.
39 10336 [MQTT] Received message 40000 from queue.
40 10366 [IP-task] Socket sending wakeup to MQTT task.
41 10366 [MQTT] Received message 0 from queue.
42 10366 [MQTT] MQTT Publish was successful.
43 10366 [MQTT] Notifying task.
44 10366 [OTA Task] Command sent to MQTT task passed.
45 10376 [IP-task] Socket sending wakeup to MQTT task.
46 10376 [MQTT] Received message 0 from queue.
47 10376 [OTA Task] [OTA] Set job doc parameter [ clientToken: 0:Microchip ]
48 10376 [OTA Task] [OTA] Missing job parameter: execution
49 10376 [OTA Task] [OTA] Missing job parameter: jobId
50 10376 [OTA Task] [OTA] Missing job parameter: jobDocument
51 10378 [OTA Task] [OTA] Missing job parameter: ts_ota
52 10378 [OTA Task] [OTA] Missing job parameter: files
```

```
53 10378 [OTA Task] [OTA] Missing job parameter: streamname
54 10378 [OTA Task] [OTA] Missing job parameter: certfile
55 10378 [OTA Task] [OTA] Missing job parameter: filepath
56 10378 [OTA Task] [OTA] Missing job parameter: filesize
57 10378 [OTA Task] [OTA] Missing job parameter: sig-sha256-ecdsa
58 10378 [OTA Task] [OTA] Missing job parameter: fileid
59 10378 [OTA Task] [OTA] Missing job parameter: attr
60 10378 [OTA Task] [OTA] Returned buffer to MQTT Client.
61 11367 [OTA] [OTA] Queued: 1   Processed: 1   Dropped: 0
62 12367 [OTA] [OTA] Queued: 1   Processed: 1   Dropped: 0
63 13367 [OTA] [OTA] Queued: 1   Processed: 1   Dropped: 0
64 14367 [OTA] [OTA] Queued: 1   Processed: 1   Dropped: 0
65 15367 [OTA] [OTA] Queued: 1   Processed: 1   Dropped: 0
66 16367 [OTA] [OTA] Queued: 1   Processed: 1   Dropped: 0
```

此输出显示 Microchip Curi PIC32 osity MZEF 可以连接 AWS IoT 和订阅 OTA 更新所需的 MQTT 主题。出现 `Missing job parameter` 消息在预料之中，因为没有待处理的 OTA 更新作业。

在 Espressif 上下载、构建、刷新和运行 FreeRTOS OTA 演示 ESP32

> **Important**
>
> 该参考集成托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

1. 从以下网址下载 FreeRTOS 源代码。GitHub有关说明，请参阅 README.md 文件。在 IDE 中创建一个项目，其中包含所有所需的源和库。
2. 按照 Espressif 入门中的说明，设置所需的基于 GCC 的工具链。
3. 打开 *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h，注释掉 `#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED` 并定义 `CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` 或 `CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`。
4. 在 `vendors/espressif/boards/esp32/aws_demos` 目录中运行 `make`，构建演示项目。您可以按照 Espressif 入门中所述，刷写演示程序并通过运行 `make flash monitor` 来验证其输出。
5. 运行 OTA 更新演示之前：

- 打开 *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h，注释掉 #define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED 并定义 CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED 或 CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED。
- 打开 vendors/*vendor*/boards/*board*/aws_demos/config_files/ota_demo_config.h 并在下面复制您的 SHA-256/ECDSA 代码签名证书：

```
#define otapalconfigCODE_SIGNING_CERTIFICATE [] = "your-certificate-key";
```

在瑞萨 N 上下载、构建、刷新并运行 FreeRTOS OTA 演示 RX65

> **Important**
>
> 该参考集成托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

本章向您展示如何在瑞萨 N 上下载、构建、刷新和运行 FreeRTOS OTA 演示应用程序。 RX65

主题

- 设置操作环境
- 设置您的 AWS 资源
- 导入、配置头文件并构建 aws_demos 和 boot_loader

设置操作环境

在本节中，此过程使用以下环境：

- IDE：e^2 studio 7.8.0、e^2 studio 2020-07
- 工具链：CCRX Compiler v3.0.1
- 目标设备： RSKRX65N-2MB
- 调试程序：E^2、E^2 Lite 模拟器
- 软件：Renesas Flash Programmer、Renesas Secure Flash Programmer.exe、Tera Term

设置硬件

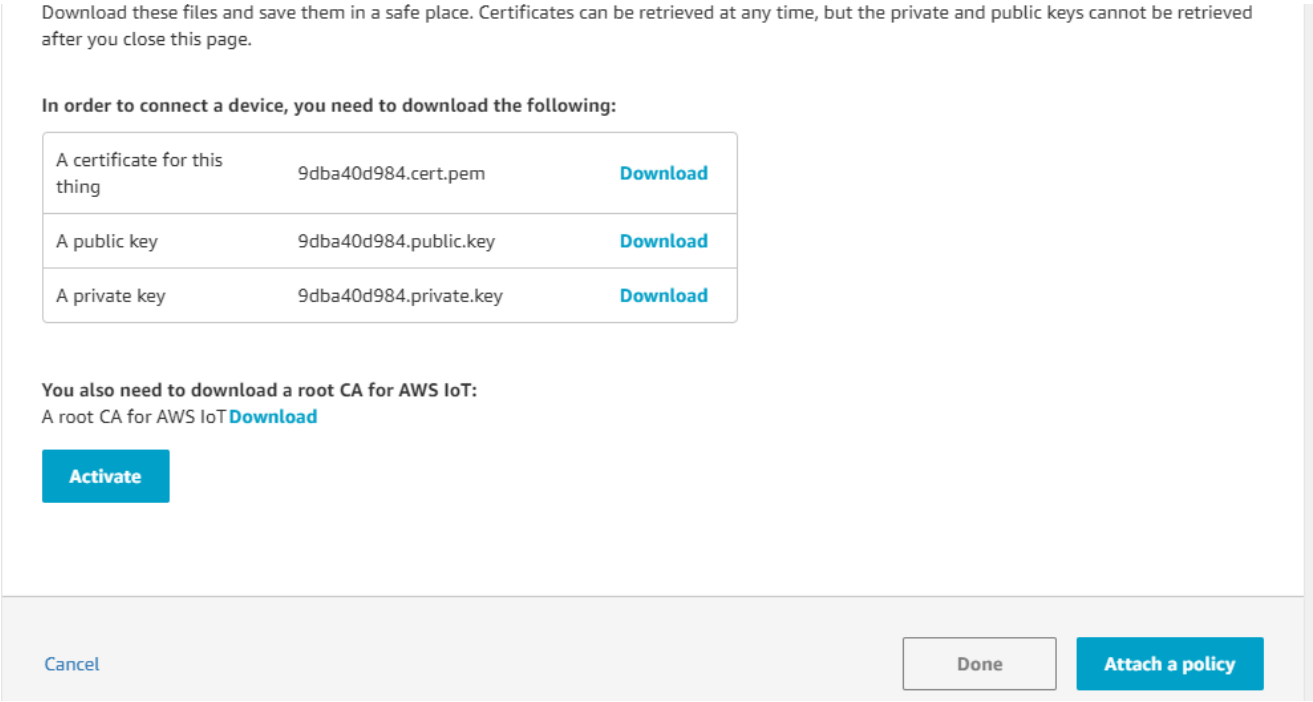
1. 将 E 2 Lite 仿真器和 USB 串行端口连接到 RX65 N 主板和电脑。
2. 将电源连接到 RX65 N。

设置您的 AWS 资源

1. 要运行 FreeRTOS 演示，您必须拥有 AWS 一个拥有有权访问服务的 IAM 用户的账户。 AWS IoT 如果尚未执行此操作，请按照设置您的 AWS 账户和权限中的步骤操作。
2. 要设置 OTA 更新，请按照 OTA 更新先决条件中的步骤操作。特别是，请按照使用 MQTT 的 OTA 更新的先决条件中的步骤操作。
3. 打开 AWS IoT 管理控制台。
4. 在左侧导航窗格中选择管理，然后选择事物，以便创建事物。

   事物是 AWS IoT中的特定设备或逻辑实体的表示形式。它可以是物理设备或传感器（例如，灯泡或墙壁上的开关）。它也可以是一个逻辑实体，例如应用程序或物理实体的实例 AWS IoT，它不连接但与连接的设备相关（例如，带有发动机传感器或控制面板的汽车）。 AWS IoT 提供了一个事物注册表，可帮助您管理事物。

   a. 选择创建，然后选择创建单个事物。
   b. 输入事物的名称，然后选择下一步。
   c. 选择创建证书。
   d. 下载创建的三个文件，然后选择激活。
   e. 选择附加策略。



f. 选择在 设备策略 中创建的策略。

每台使用 MQTT 接收 OTA 更新的设备都必须注册为事物，AWS IoT 并且该设备必须具有与列出的策略类似的附加策略。您可以在 AWS IoT 核心策略操作和 AWS IoT 核心操作资源中找到有关 "Action" 和 "Resource" 对象中项目的更多信息。

备注

- 这些iot:Connect权限允许您的设备通过 MQTT AWS IoT 进行连接。
- 利用 AWS IoT 作业 (.../jobs/*) 主题的 iot:Subscribe 和 iot:Publish 权限，连接的设备能够接收作业通知和作业文档，并发布作业执行的完成状态。
- AWS IoT OTA 直播主题的iot:Subscribe和iot:Publish权限 (.../streams/*) 允许连接的设备从中获取 OTA 更新数据 AWS IoT。在通过 MQTT 执行固件更新时，需要这些权限。
- 这些iot:Receive权限 AWS IoT Core 允许将有关这些主题的消息发布到连接的设备。每次传输 MQTT 消息时，都将检查此权限。您可以使用此权限，撤消对当前订阅主题的客户端的访问权限。

5. 创建代码签名配置文件并在上注册代码签名证书。 AWS

a. 要创建密钥和认证，请参阅瑞萨微控制器固件更新设计政策中的第 7.3 节 “使用 OpenSSL 生成 ECDSA-SHA256 密钥对”。

b. 打开 AWS IoT 管理控制台。在左导航窗格中，选择管理，然后选择作业。选择创建作业，然后选择创建 OTA 更新作业。

c. 在选择要更新的设备下，选择选择，然后选择之前创建的事物。选择下一步。

d. 在创建 FreeRTOS OTA 更新作业页面上，执行以下操作：

   i. 在选择固件映像传输协议中，选择 MQTT。

   ii. 对于选择并签署固件映像，选择为我签署一个新的固件映像。

   iii. 对于代码签名配置文件，选择创建。

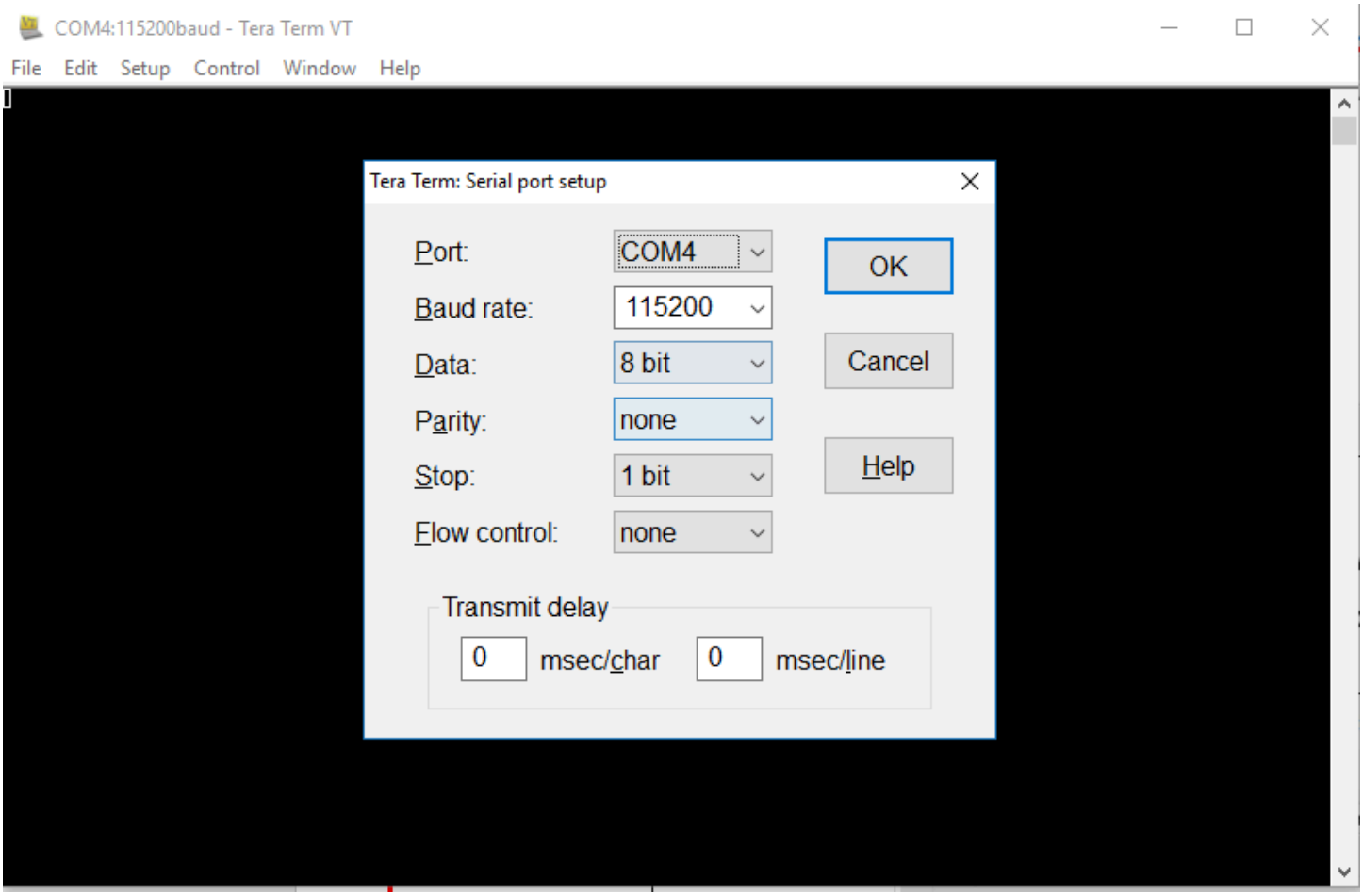
   iv. 在创建代码签名配置文件窗口中，输入配置文件名称。对于设备硬件平台，选择 Windows 模拟器。对于代码签名证书，选择导入。

   v. 浏览以选择证书 (`secp256r1.crt`)、证书私钥 (`secp256r1.key`) 和证书链 (`ca.crt`)。

   vi. 输入设备上代码签名证书的路径名。然后选择创建。

6. 要授予对的代码签名的访问权限 AWS IoT，请按照中的步骤操作授予对代码签名的访问权限 AWS IoT。

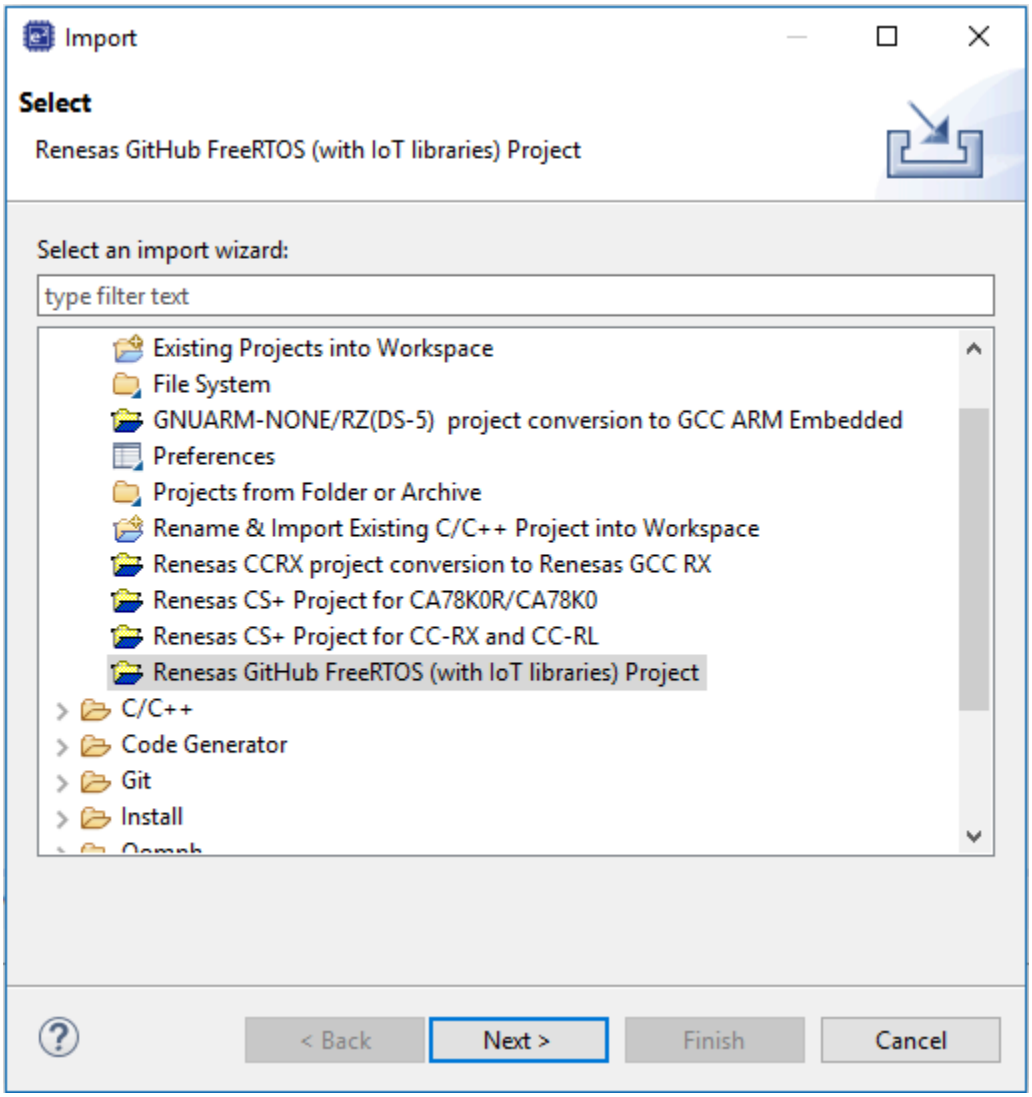
如果你的电脑上没有安装 Tera Term，你可以从 https://ttssh2.osdn.jp/index.html.en 下载并按此处所示进行设置。确保将设备的 USB 串行端口插入 PC。



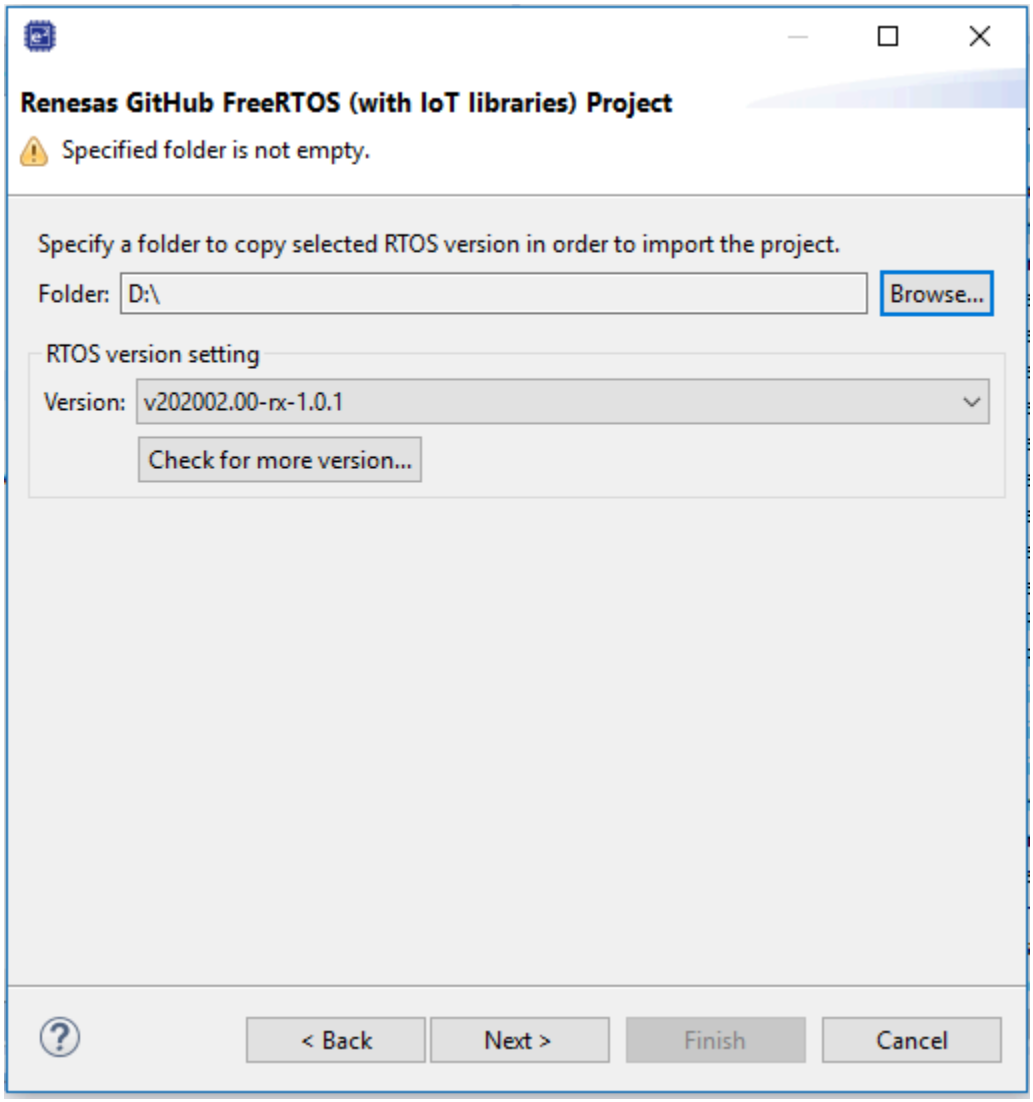
导入、配置头文件并构建 aws_demos 和 boot_loader

首先，您需要选择最新版本的 FreeRTOS 软件包，该软件包将 GitHub 从项目中下载并自动导入到项目中。这样，您就可以专注于配置 FreeRTOS 和编写应用程序代码。

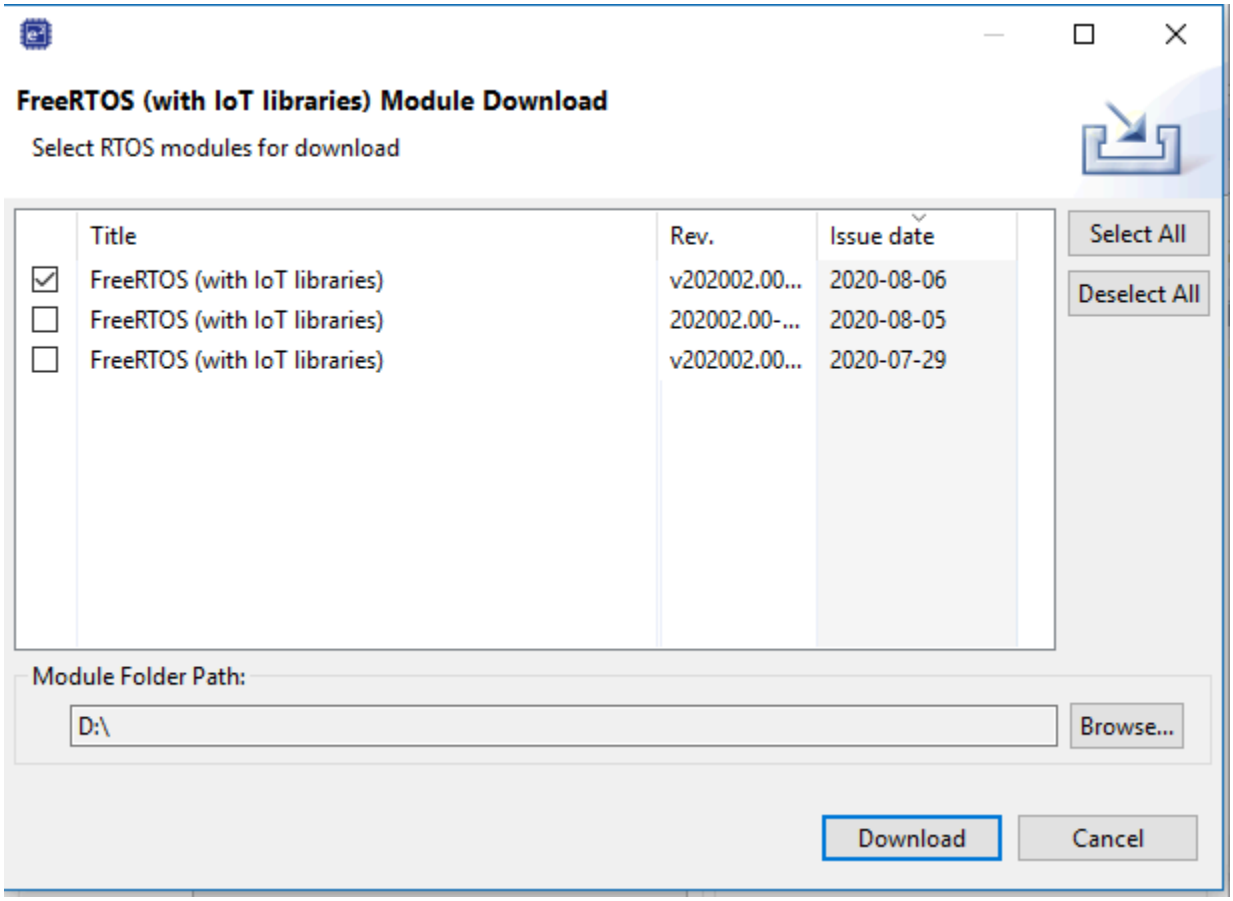
1. 启动 e^2 Studio。
2. 选择文件，然后选择导入…。
3. 选择瑞萨电子 FreeRTO GitHub S（含物联网库）项目。



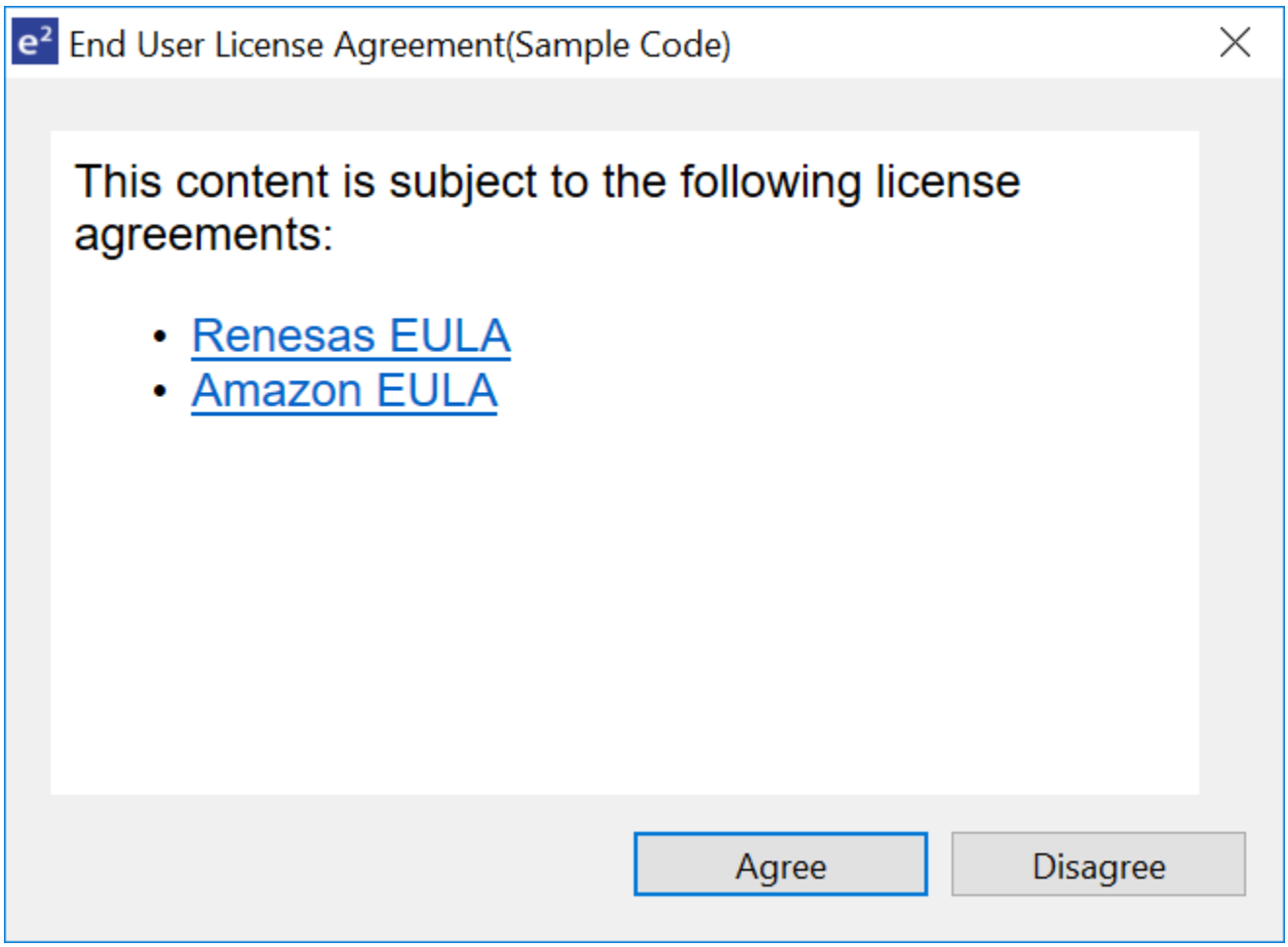
4. 选择查看更多版本...以显示下载对话框。


Renesas GitHub FreeRTOS (with IoT libraries) Project
Specified folder is not empty.
Specify a folder to copy selected RTOS version in order to import the project.
Folder: D:\
Browse...
RTOS version setting
Version: v202002.00-rx-1.0.1
Check for more version...
< Back
Next >
Finish
Cancel

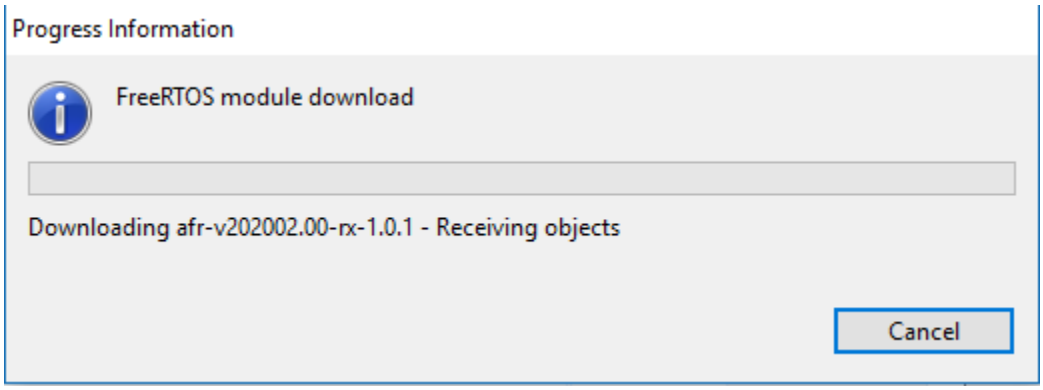
5. 选择最新的程序包。



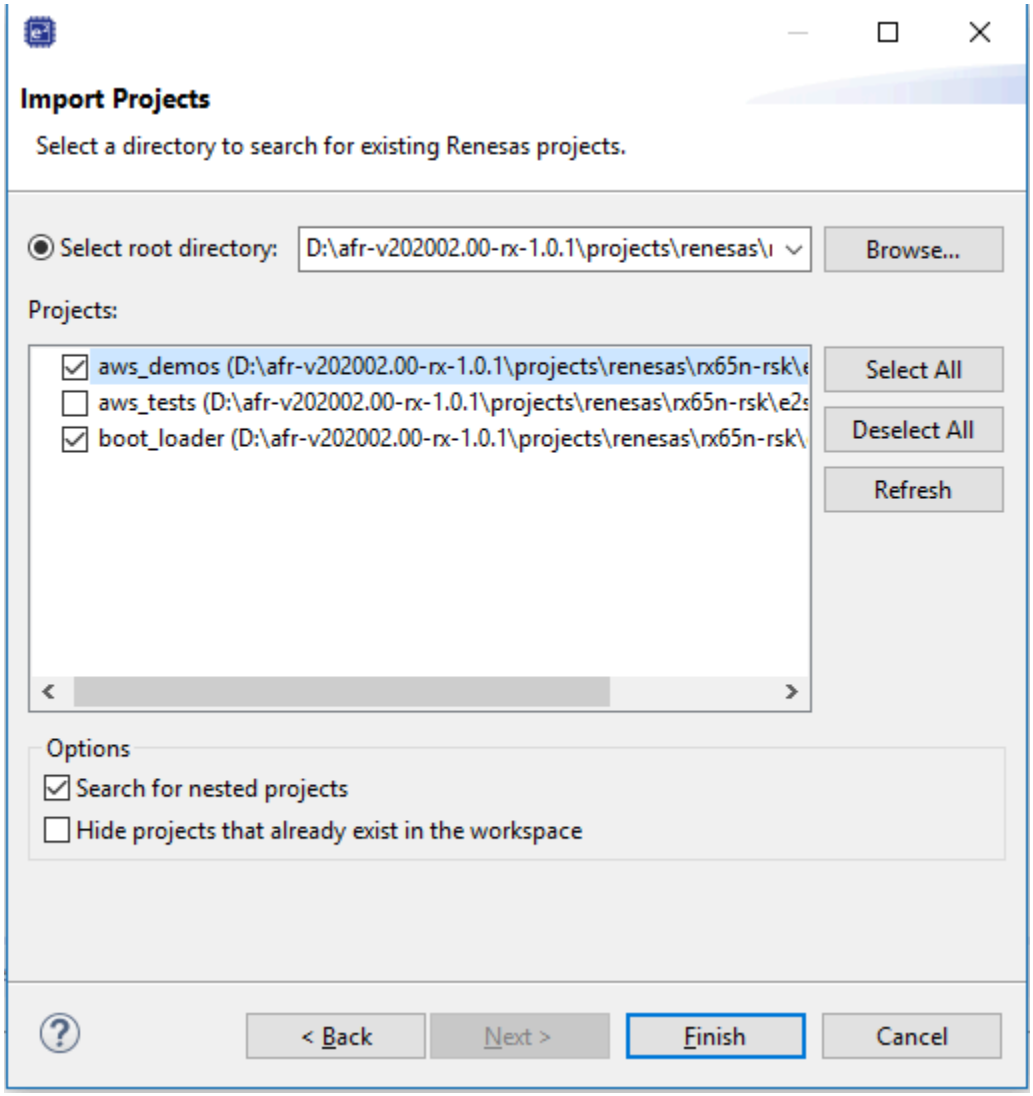
6. 选择同意以接受最终用户许可协议。



7. 等待下载完成。



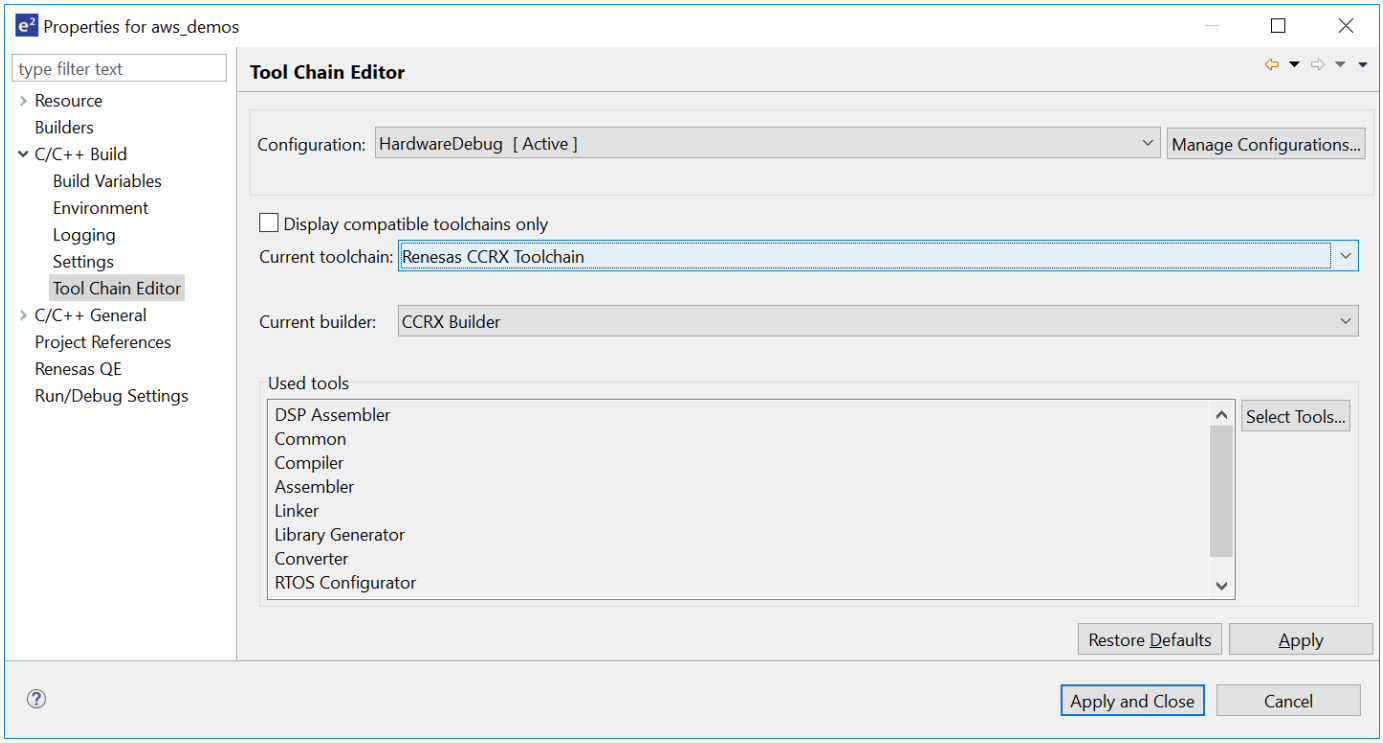
8. 选择 aws_demos 和 boot_loader 项目，然后选择完成以将其导入。



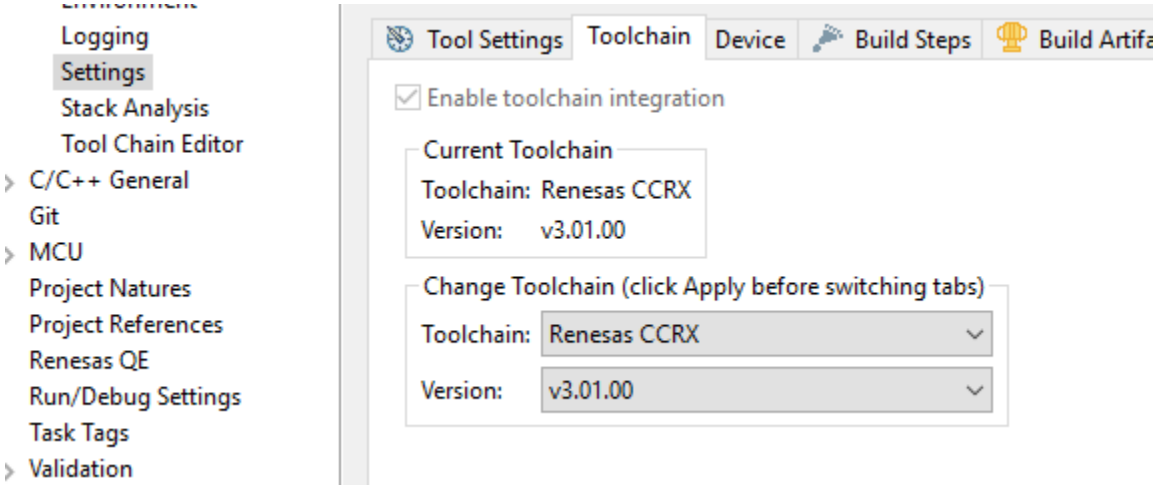
9. 对于这两个项目，打开项目属性。在导航窗格中，选择工具链编辑器。

    a. 选择当前工具链。

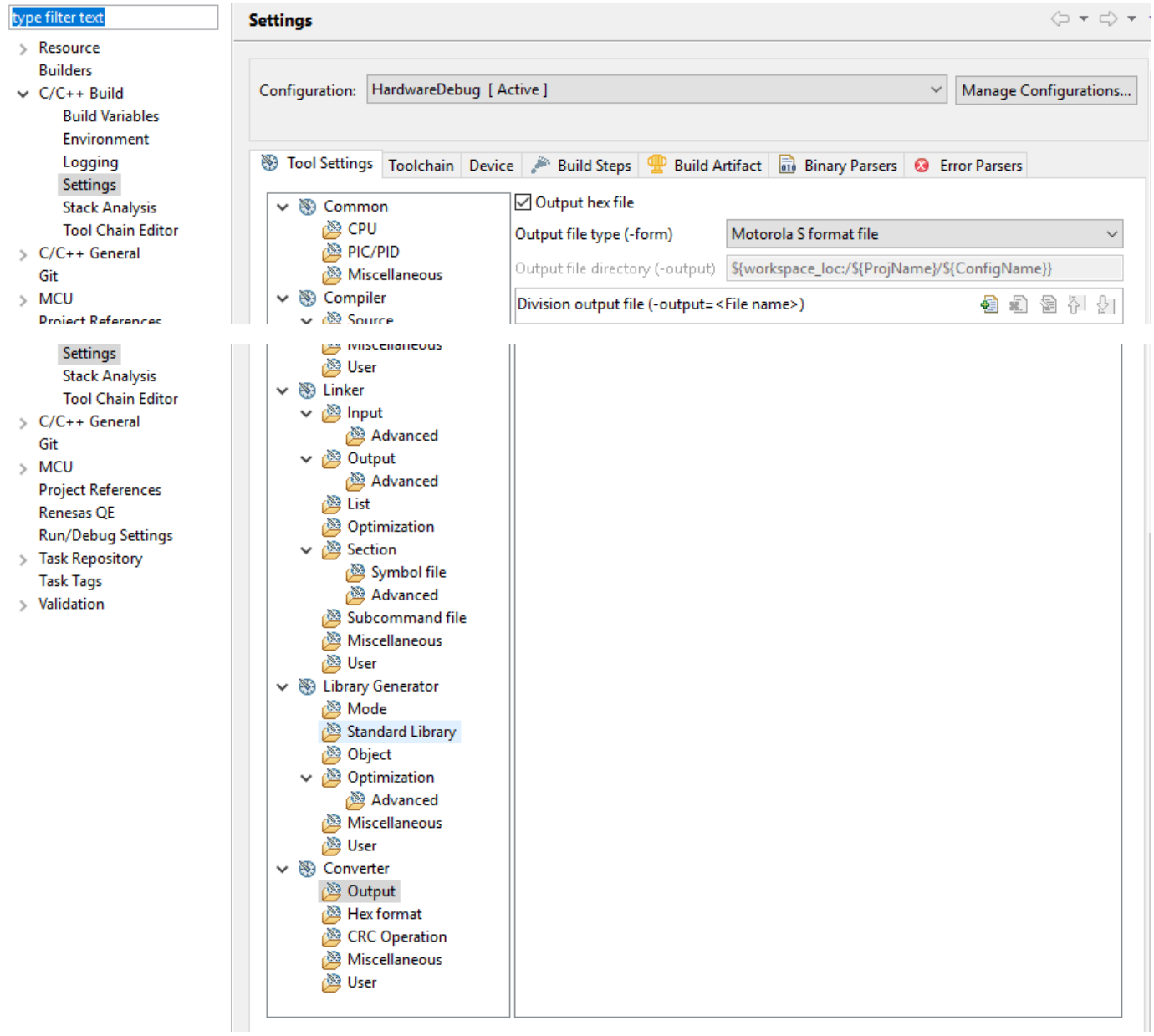
    b. 选择当前生成器。



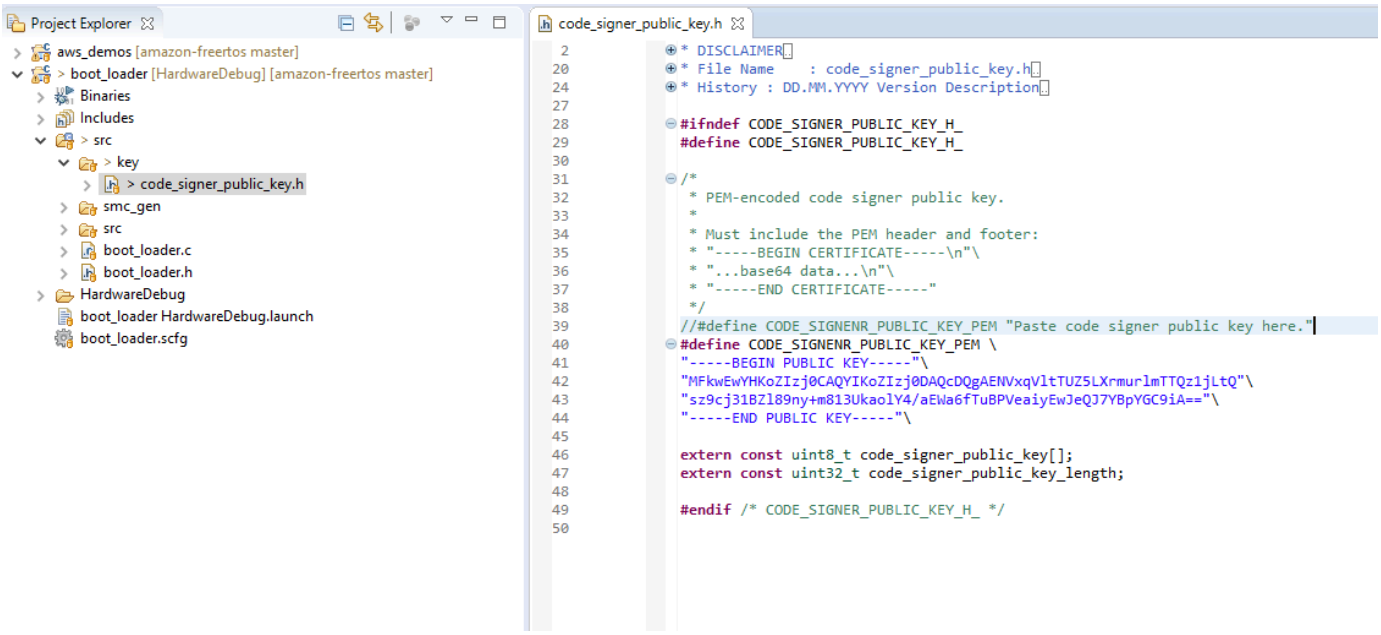
10. 在导航窗格中，选择 Settings（设置）。选择工具链选项卡，然后选择工具链版本。



选择工具设置选项卡，展开转换器，然后选择输出。在主窗口中，确保选中输出十六进制文件，然后选择输出文件类型。



11. 在引导加载程序项目中，打开 `projects\renesas\rx65n-rsk\e2studio\boot_loader\src\key\code_signer_public_key.h` 并输入公钥。有关如何创建公钥的信息，请参阅如何在 N 上使用 Amazon Web Services 实现 FreeRTOS OTA 和瑞萨 MCU 固件更新设计政策中的第 7.3 节 “使用 OpenSSL 生成 ECDSA-SHA256 密钥对”。 RX65



然后构建项目以创建 boot_loader.mot。

12. 打开 aws_demos 项目。

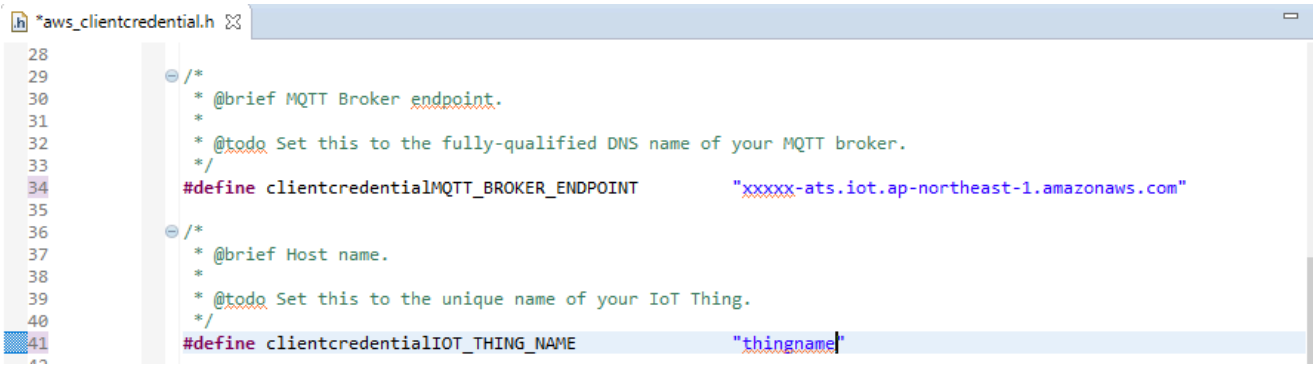
    a. 打开 AWS IoT 管理控制台。

    b. 在左侧导航窗格中，选择 设置。在设备数据端点文本框中记下自定义端点。

    c. 选择管理，然后选择事物。记下你的棋盘上的 AWS IoT 事物名称。

    d. 在 aws_demos 项目中，打开 demos/include/aws_clientcredential.h 并指定以下值。

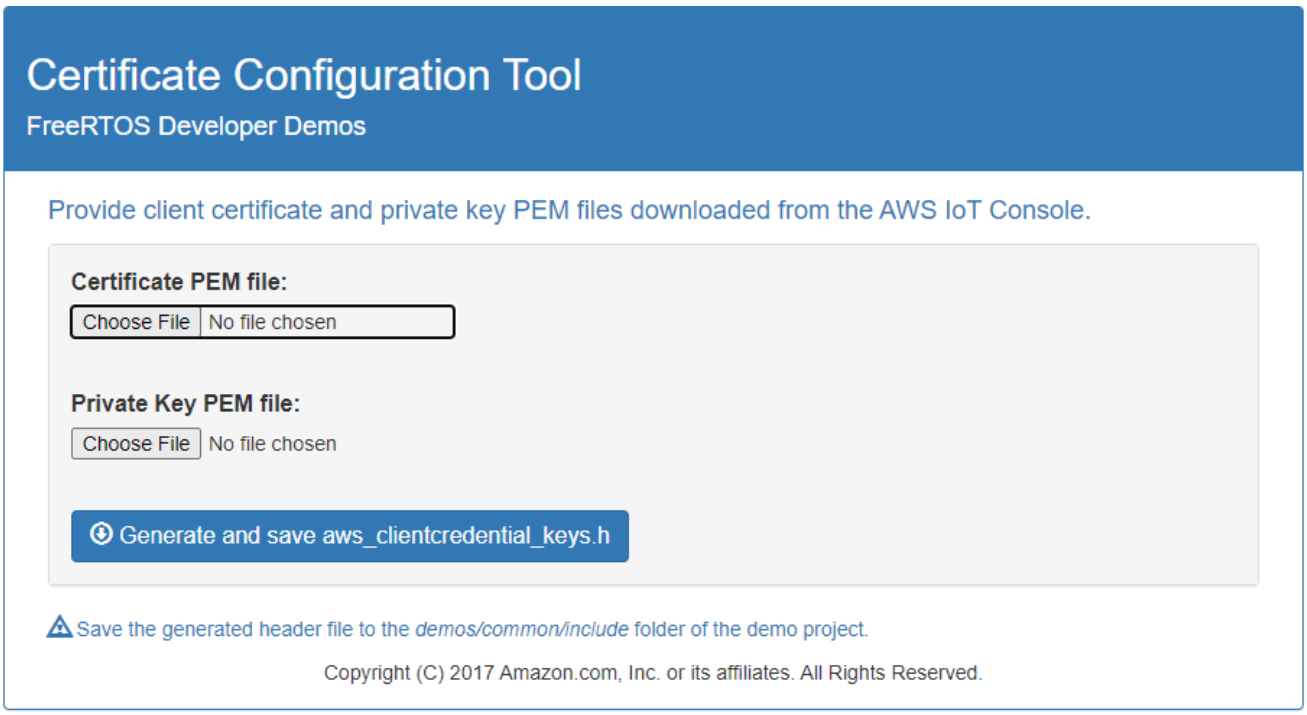
```
#define clientcredentialMQTT_BROKER_ENDPOINT[] = "Your AWS IoT endpoint";
#define clientcredentialIOT_THING_NAME "The AWS IoT thing name of your board"
```



    e. 打开 tools/certificate_configuration/CertificateConfigurator.html文件。

f. 导入之前下载的证书 PEM 文件和私钥 PEM 文件。

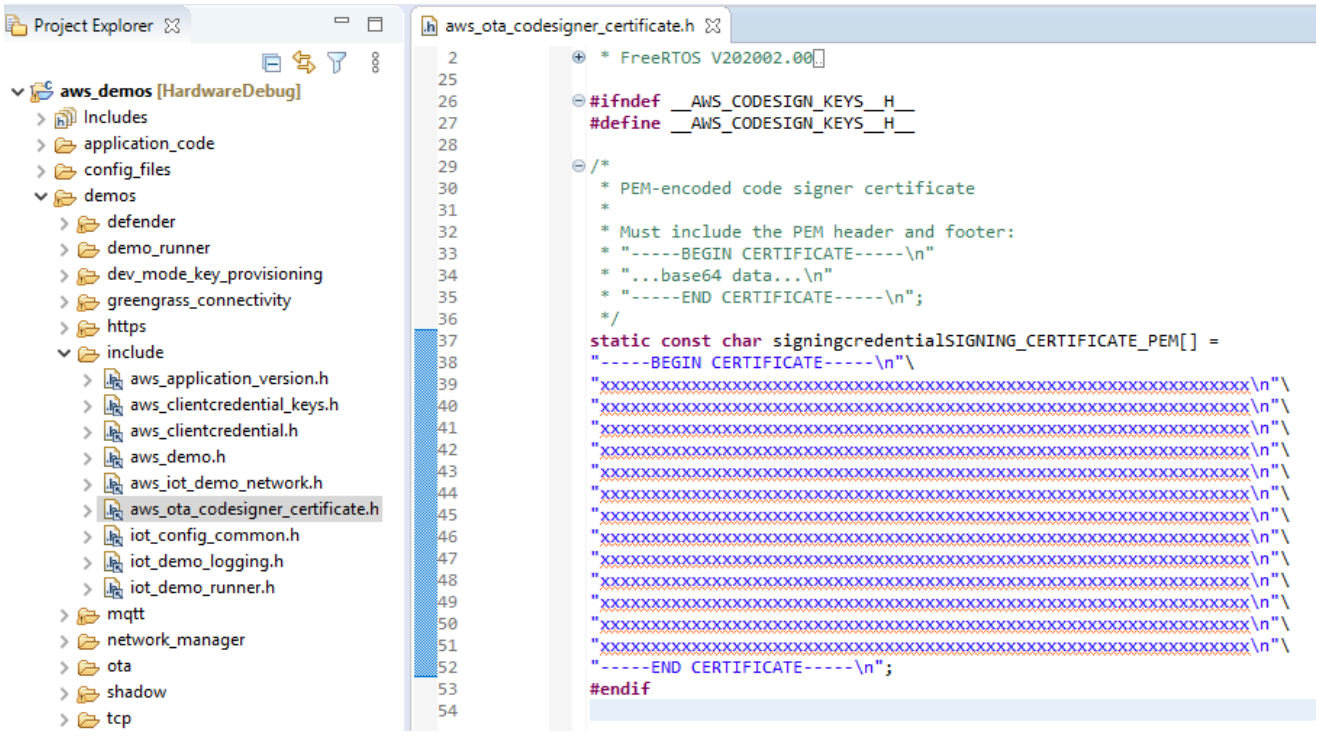
g. 选择生成并保存 aws_clientcredential_keys.h，并在 `demos/include/` 目录下替换此文件。



h. 打开 `vendors/renesas/boards/rx65n-rsk/aws_demos/config_files/ota_demo_config.h` 文件，并指定这些值。

```
#define otapalconfigCODE_SIGNING_CERTIFICATE [] = "your-certificate-key";
```

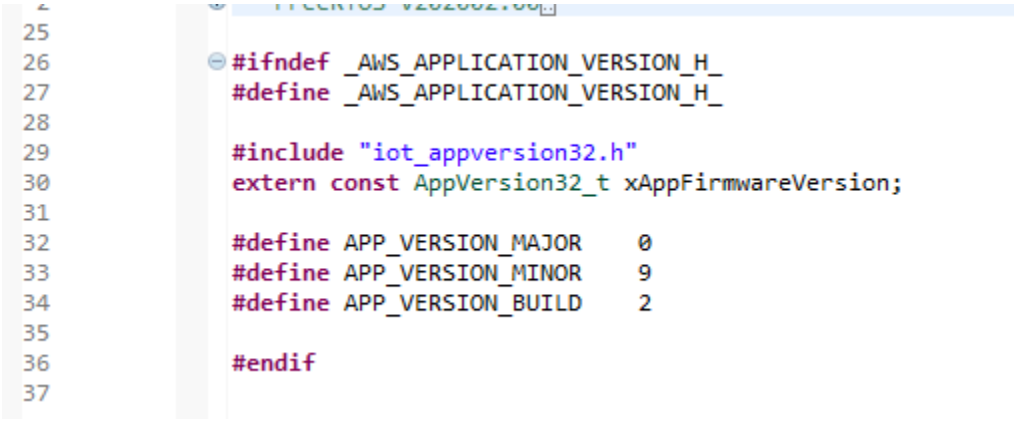
文件中的*your-certificate-key*值在哪里`secp256r1.crt`。记得在证书的每一行后面加上“\”。有关创建`secp256r1.crt`文件的更多信息，请参阅如何在 N 上使用 Amazon Web Services 实现 FreeRTOS OTA 和瑞萨 MCU 固件更新设计政策中的第 7.3 节 “使用 OpenSSL 生成 ECDSA-SHA256 密钥对”。 RX65



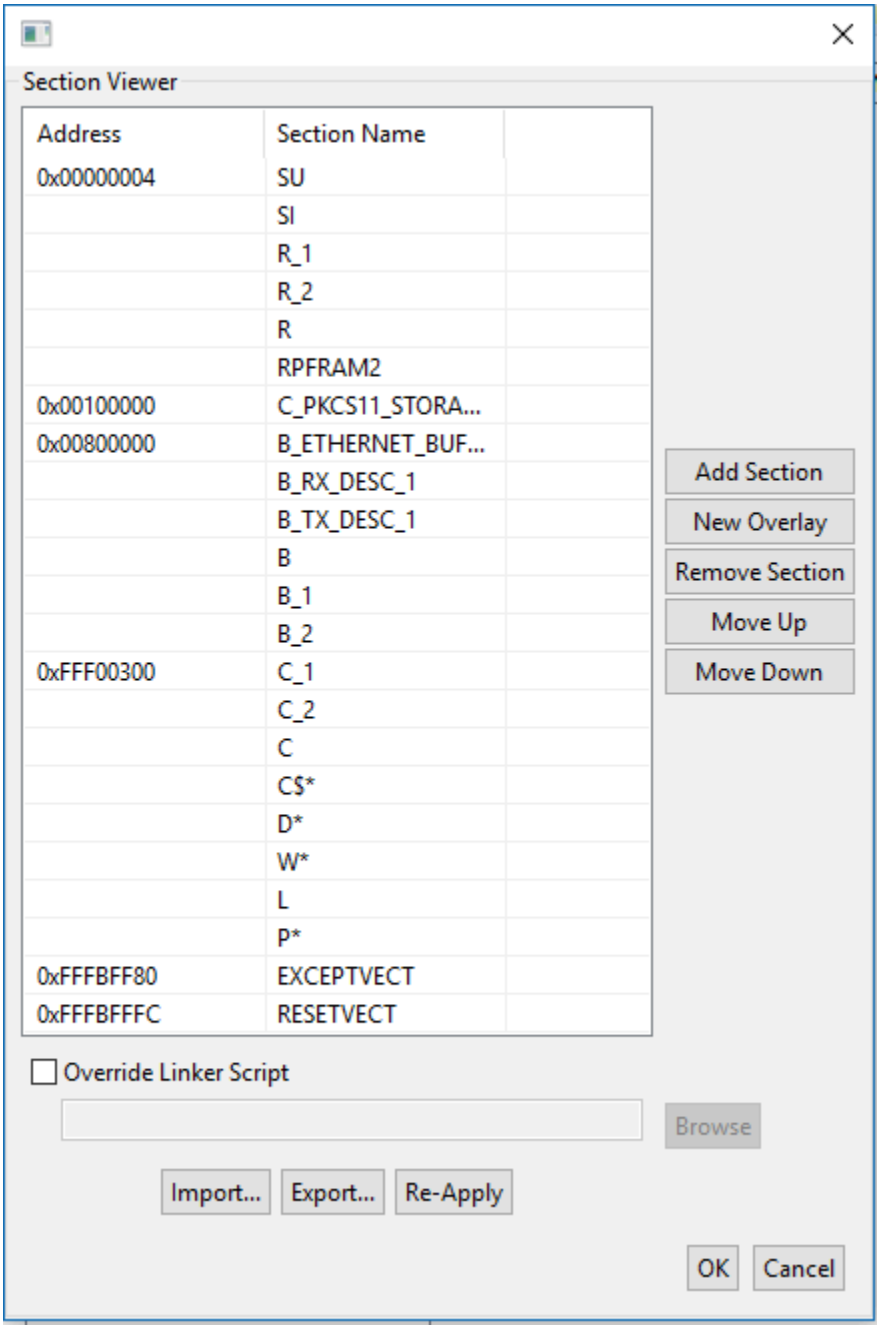
13. 任务 A：安装固件的初始版本

    a. 打开 vendors/renesas/boards/board/aws_demos/config_files/aws_demo_config.h 文件，注释掉 #define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED，然后定义 CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED 或 CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED。

    b. 打开 demos/include/ aws_application_version.h 文件，并将固件的初始版本设置为 0.9.2。



    c. 在部分查看器中更改以下设置。



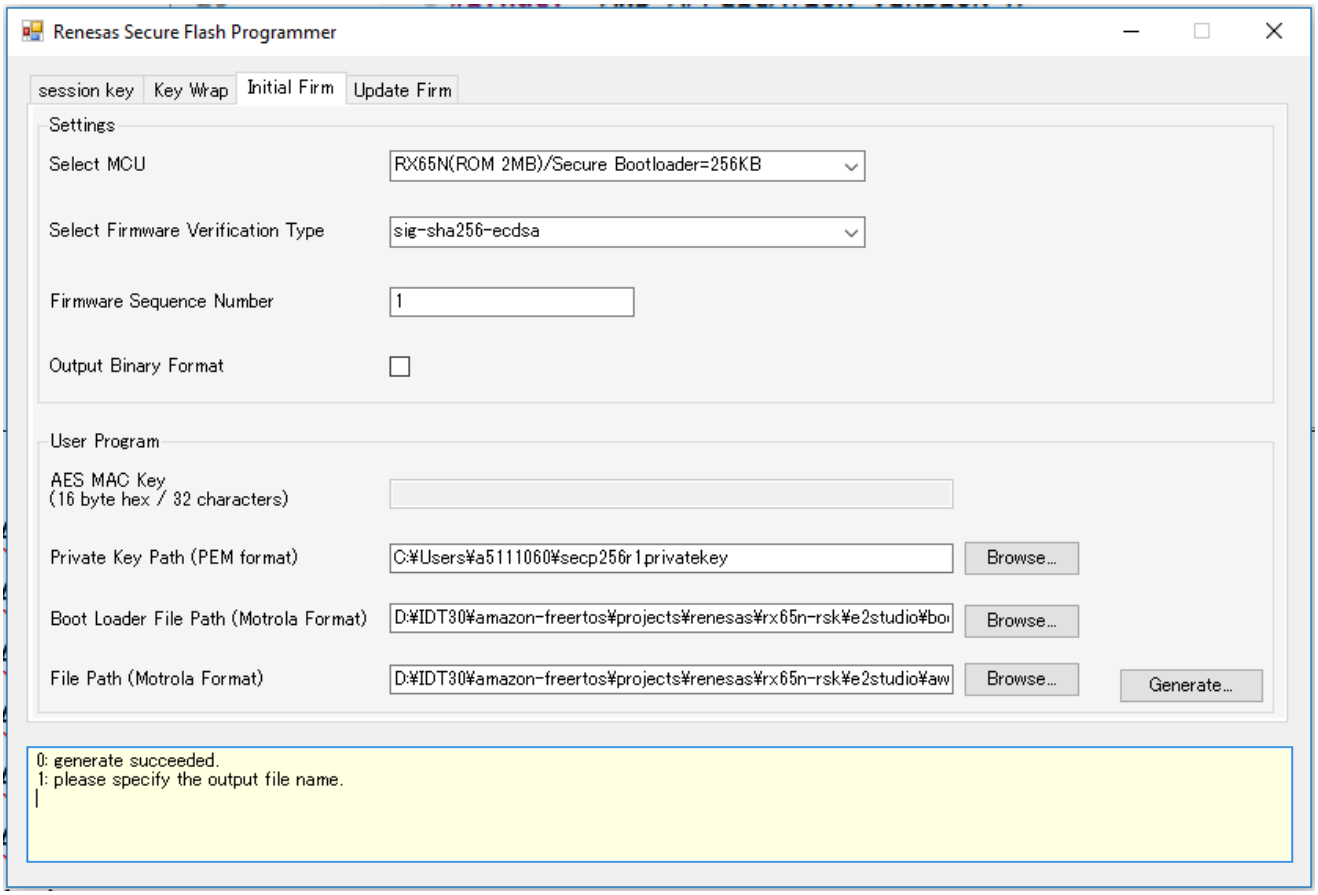
d. 选择构建以创建 `aws_demos.mot` 文件。

14. 使用 Renesas Secure Flash Programmer 创建文件 `userprog.mot`。`userprog.mot` 是 `aws_demos.mot` 和 `boot_loader.mot` 的组合。您可以将此文件刷新到 RX65 N-RSK 以安装初始固件。

    a. 下载 https://github.com/renesas/亚马逊 FreerTOS-Too ls 并打开。`Renesas Secure Flash Programmer.exe`

    b. 选择初始固件选项，然后设置以下参数：

- 私钥路径 – `secp256r1.privatekey` 的位置。
- 引导加载程序文件路径 – `boot_loader.mot` (`projects\renesas\rx65n-rsk\e2studio\boot_loader\HardwareDebug`) 的位置。
- 文件路径 – `aws_demos.mot` (`projects\renesas\rx65n-rsk\e2studio\aws_demos\HardwareDebug`) 的位置。



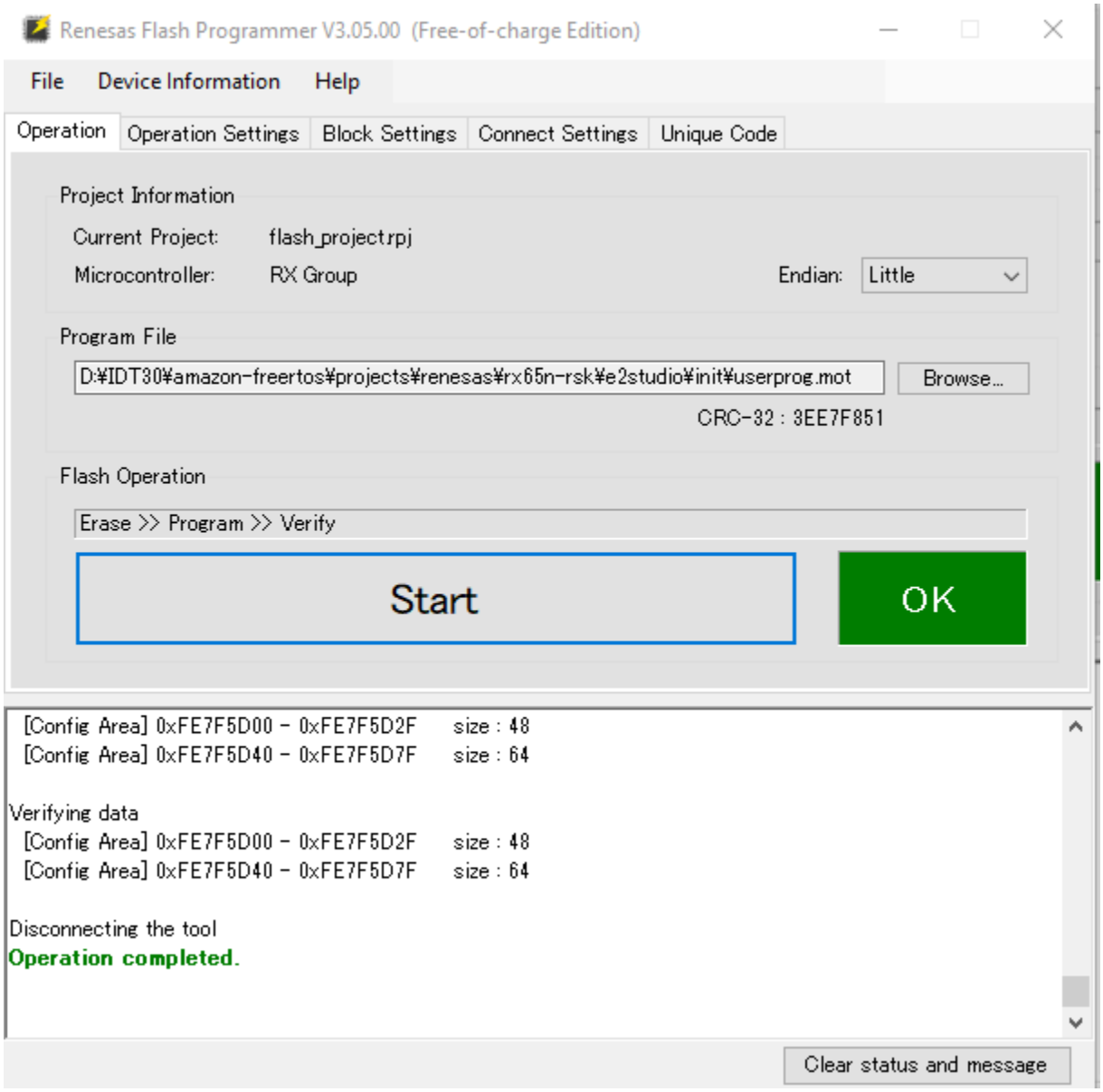
c. 创建一个名为 `init_firmware` 的目录，生成 `userprog.mot`，并将其保存到 `init_firmware` 目录中。验证生成是否成功。

15. 在 RX65 N-RSK 上刷新初始固件。

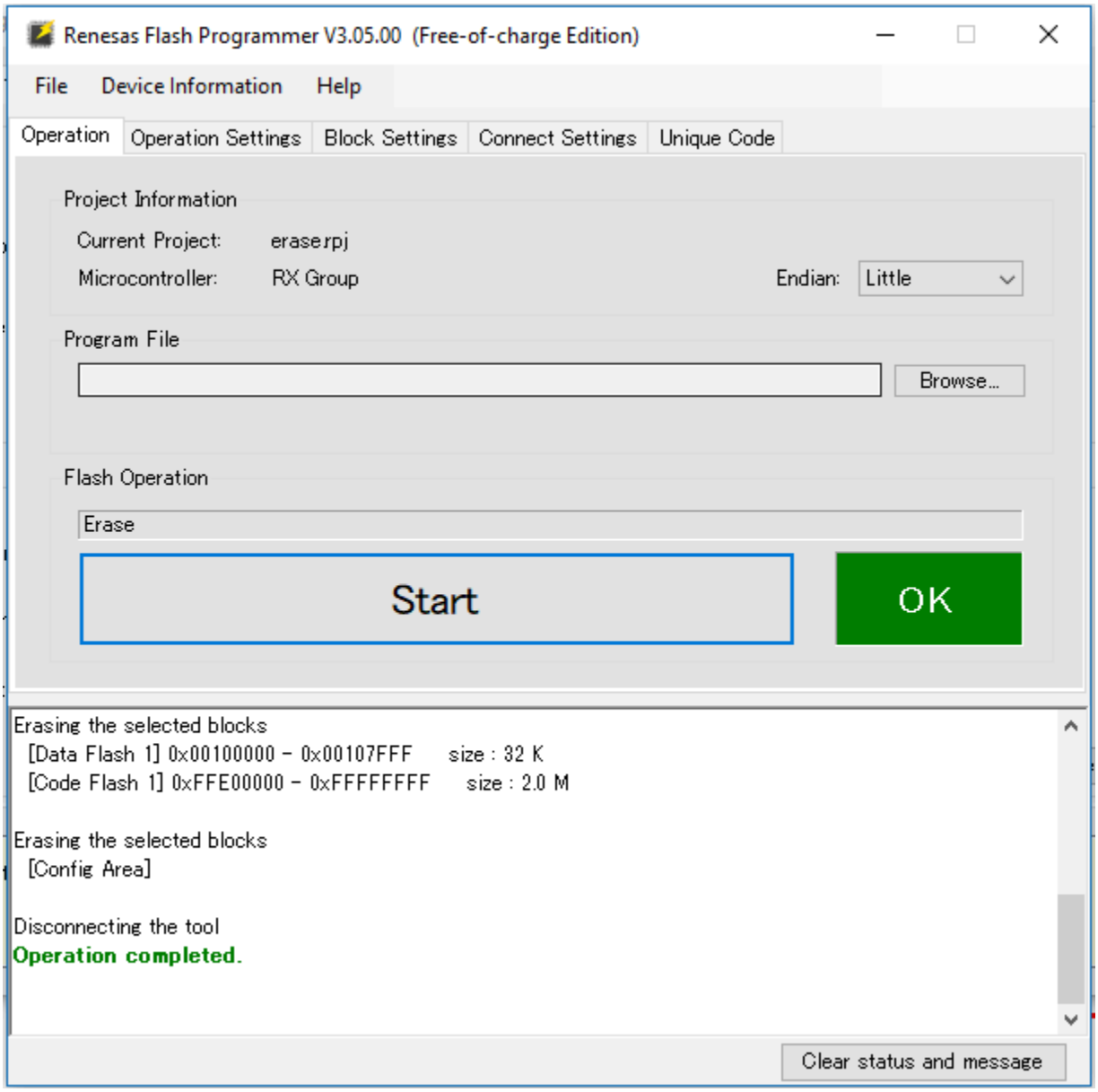
a. 从 https://www.renesas.com/tw/en/products/software-tools/tools/programmer/renesas-flash-programmer-programming-gui .html 下载最新版本的瑞萨 Flash Programmer（编程 GUI）。

b. 打开 `vendors\renesas\rx_mcu_boards\boards\rx65n-rsk\aws_demos\flash_project\erase_from_bank\ erase.rpj` 文件以擦除库中的数据。

c. 选择开始以擦除库。



d. 要刷写 `userprog.mot`，请选择浏览...，导航到 `init_firmware` 目录，选择 `userprog.mot` 文件，然后选择开始。



16. 固件的 0.9.2 版（初始版本）已安装到您 RX65的 N-RSK 上。 RX65N-RSK 董事会现在正在监听 OTA 更新。如果您在 PC 上打开 Tera Term，则当初始固件运行时，您会看到类似以下的内容。

```
------------------------------------------------
RX65N secure boot program
------------------------------------------------
Checking flash ROM status.
bank 0 status = 0xff [LIFECYCLE_STATE_BLANK]
bank 1 status = 0xfc [LIFECYCLE_STATE_INSTALLING]
bank info = 1. (start bank = 0)
start installing user program.
copy secure boot (part1) from bank0 to bank1...OK
```

```
copy secure boot (part2) from bank0 to bank1...OK
update LIFECYCLE_STATE from [LIFECYCLE_STATE_INSTALLING] to [LIFECYCLE_STATE_VALID]
bank1(temporary area) block0 erase (to update LIFECYCLE_STATE)...OK
bank1(temporary area) block0 write (to update LIFECYCLE_STATE)...OK
swap bank...
-----------------------------------------------
RX65N secure boot program
-----------------------------------------------
Checking flash ROM status.
bank 0 status = 0xf8 [LIFECYCLE_STATE_VALID]
bank 1 status = 0xff [LIFECYCLE_STATE_BLANK]
bank info = 0. (start bank = 1)
integrity check scheme = sig-sha256-ecdsa
bank0(execute area) on code flash integrity check...OK
jump to user program
#0 1 [ETHER_RECEI] Deferred Interrupt Handler Task started
1 1 [ETHER_RECEI] Network buffers: 3 lowest 3
2 1 [ETHER_RECEI] Heap: current 234192 lowest 234192
3 1 [ETHER_RECEI] Queue space: lowest 8
4 1 [IP-task] InitializeNetwork returns OK
5 1 [IP-task] xNetworkInterfaceInitialise returns 0
6 101 [ETHER_RECEI] Heap: current 234592 lowest 233392
7 2102 [ETHER_RECEI] prvEMACHandlerTask: PHY LS now 1
8 3001 [IP-task] xNetworkInterfaceInitialise returns 1
9 3092 [ETHER_RECEI] Network buffers: 2 lowest 2
10 3092 [ETHER_RECEI] Queue space: lowest 7
11 3092 [ETHER_RECEI] Heap: current 233320 lowest 233320
12 3193 [ETHER_RECEI] Heap: current 233816 lowest 233120
13 3593 [IP-task] vDHCPProcess: offer c0a80a09ip
14 3597 [ETHER_RECEI] Heap: current 233200 lowest 233000
15 3597 [IP-task] vDHCPProcess: offer c0a80a09ip
16 3597 [IP-task] IP Address: 192.168.10.9
17 3597 [IP-task] Subnet Mask: 255.255.255.0
18 3597 [IP-task] Gateway Address: 192.168.10.1
19 3597 [IP-task] DNS Server Address: 192.168.10.1
20 3600 [Tmr Svc] The network is up and running
21 3622 [Tmr Svc] Write certificate...
22 3697 [ETHER_RECEI] Heap: current 232320 lowest 230904
23 4497 [ETHER_RECEI] Heap: current 226344 lowest 225944
24 5317 [iot_thread] [INFO ][DEMO][5317] ---------STARTING DEMO---------

25 5317 [iot_thread] [INFO ][INIT][5317] SDK successfully initialized.
26 5317 [iot_thread] [INFO ][DEMO][5317] Successfully initialized the demo. Network
 type for the demo: 4
```

```
27 5317 [iot_thread] [INFO ][MQTT][5317] MQTT library successfully initialized.
28 5317 [iot_thread] [INFO ][DEMO][5317] OTA demo version 0.9.2

29 5317 [iot_thread] [INFO ][DEMO][5317] Connecting to broker...

30 5317 [iot_thread] [INFO ][DEMO][5317] MQTT demo client identifier is rx65n-gr-
rose (length 13).
31 5325 [ETHER_RECEI] Heap: current 206944 lowest 206504
32 5325 [ETHER_RECEI] Heap: current 206440 lowest 206440
33 5325 [ETHER_RECEI] Heap: current 206240 lowest 206240
38 5334 [ETHER_RECEI] Heap: current 190288 lowest 190288
39 5334 [ETHER_RECEI] Heap: current 190088 lowest 190088
40 5361 [ETHER_RECEI] Heap: current 158512 lowest 158168
41 5363 [ETHER_RECEI] Heap: current 158032 lowest 158032
42 5364 [ETHER_RECEI] Network buffers: 1 lowest 1
43 5364 [ETHER_RECEI] Heap: current 156856 lowest 156856
44 5364 [ETHER_RECEI] Heap: current 156656 lowest 156656
46 5374 [ETHER_RECEI] Heap: current 153016 lowest 152040
47 5492 [ETHER_RECEI] Heap: current 141464 lowest 139016
48 5751 [ETHER_RECEI] Heap: current 140160 lowest 138680
49 5917 [ETHER_RECEI] Heap: current 138280 lowest 138168
59 7361 [iot_thread] [INFO ][MQTT][7361] Establishing new MQTT connection.
62 7428 [iot_thread] [INFO ][MQTT][7428] (MQTT connection 81cfc8, CONNECT operation
 81d0e8) Wait complete with result SUCCESS.
63 7428 [iot_thread] [INFO ][MQTT][7428] New MQTT connection 4e8c established.
64 7430 [iot_thread] [OTA_AgentInit_internal] OTA Task is Ready.
65 7430 [OTA Agent T] [prvOTAAgentTask] Called handler. Current State [Ready] Event
 [Start] New state [RequestingJob]
66 7431 [OTA Agent T] [INFO ][MQTT][7431] (MQTT connection 81cfc8) SUBSCRIBE
 operation scheduled.
67 7431 [OTA Agent T] [INFO ][MQTT][7431] (MQTT connection 81cfc8, SUBSCRIBE
 operation 818c48) Waiting for operation completion.
68 7436 [ETHER_RECEI] Heap: current 128248 lowest 127992
69 7480 [OTA Agent T] [INFO ][MQTT][7480] (MQTT connection 81cfc8, SUBSCRIBE
 operation 818c48) Wait complete with result SUCCESS.
70 7480 [OTA Agent T] [prvSubscribeToJobNotificationTopics] OK: $aws/things/rx65n-
gr-rose/jobs/$next/get/accepted
71 7481 [OTA Agent T] [INFO ][MQTT][7481] (MQTT connection 81cfc8) SUBSCRIBE
 operation scheduled.
72 7481 [OTA Agent T] [INFO ][MQTT][7481] (MQTT connection 81cfc8, SUBSCRIBE
 operation 818c48) Waiting for operation completion.
73 7530 [OTA Agent T] [INFO ][MQTT][7530] (MQTT connection 81cfc8, SUBSCRIBE
 operation 818c48) Wait complete with result SUCCESS.
```

```
74 7530 [OTA Agent T] [prvSubscribeToJobNotificationTopics] OK: $aws/things/rx65n-
gr-rose/jobs/notify-next
75 7530 [OTA Agent T] [prvRequestJob_Mqtt] Request #0
76 7532 [OTA Agent T] [INFO ][MQTT][7532] (MQTT connection 81cfc8) MQTT PUBLISH
 operation queued.
77 7532 [OTA Agent T] [INFO ][MQTT][7532] (MQTT connection 81cfc8, PUBLISH
 operation 818b80) Waiting for operation completion.
78 7552 [OTA Agent T] [INFO ][MQTT][7552] (MQTT connection 81cfc8, PUBLISH
 operation 818b80) Wait complete with result SUCCESS.
79 7552 [OTA Agent T] [prvOTAAgentTask] Called handler. Current State
 [RequestingJob] Event [RequestJobDocument] New state [WaitingForJob]
80 7552 [OTA Agent T] [prvParseJSONbyModel] Extracted parameter [ clientToken:
 0:rx65n-gr-rose ]
81 7552 [OTA Agent T] [prvParseJSONbyModel] parameter not present: execution
82 7552 [OTA Agent T] [prvParseJSONbyModel] parameter not present: jobId
83 7552 [OTA Agent T] [prvParseJSONbyModel] parameter not present: jobDocument
84 7552 [OTA Agent T] [prvParseJSONbyModel] parameter not present: afr_ota
85 7552 [OTA Agent T] [prvParseJSONbyModel] parameter not present: protocols
86 7552 [OTA Agent T] [prvParseJSONbyModel] parameter not present: files
87 7552 [OTA Agent T] [prvParseJSONbyModel] parameter not present: filepath
99 7651 [ETHER_RECEI] Heap: current 129720 lowest 127304
100 8430 [iot_thread] [INFO ][DEMO][8430] State: Ready  Received: 1   Queued: 0
 Processed: 0   Dropped: 0
101 9430 [iot_thread] [INFO ][DEMO][9430] State: WaitingForJob  Received: 1
 Queued: 0   Processed: 0   Dropped: 0
102 10430 [iot_thread] [INFO ][DEMO][10430] State: WaitingForJob  Received: 1
 Queued: 0   Processed: 0   Dropped: 0
103 11430 [iot_thread] [INFO ][DEMO][11430] State: WaitingForJob  Received: 1
 Queued: 0   Processed: 0   Dropped: 0
104 12430 [iot_thread] [INFO ][DEMO][12430] State: WaitingForJob  Received: 1
 Queued: 0   Processed: 0   Dropped: 0
105 13430 [iot_thread] [INFO ][DEMO][13430] State: WaitingForJob  Received: 1
 Queued: 0   Processed: 0   Dropped: 0
106 14430 [iot_thread] [INFO ][DEMO][14430] State: WaitingForJob  Received: 1
 Queued: 0   Processed: 0   Dropped: 0
107 15430 [iot_thread] [INFO ][DEMO][15430] State: WaitingForJob  Received: 1
 Queued: 0   Processed: 0   Dropped: 0
```

17. 任务 B：更新固件版本

    a. 打开 demos/include/aws_application_version.h 文件并将 APP_VERSION_BUILD 令牌值增加至 0.9.3。

    b. 重新构建项目。

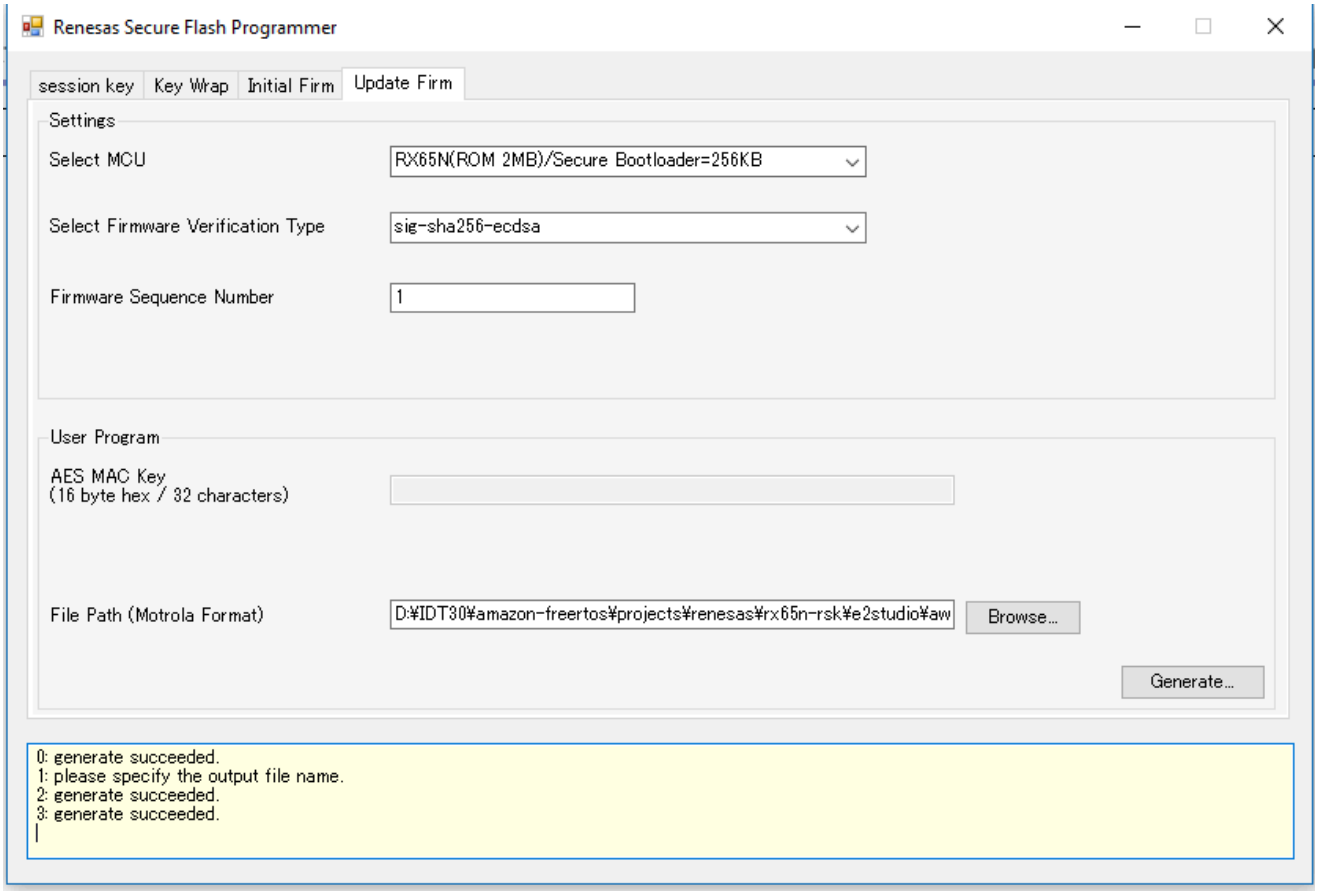
18. 使用 Renesas Secure Flash Programmer 创建 `userprog.rsu` 文件以更新固件版本。

    a. 打开 `Amazon-FreeRTOS-Tools\Renesas Secure Flash Programmer.exe`文件。

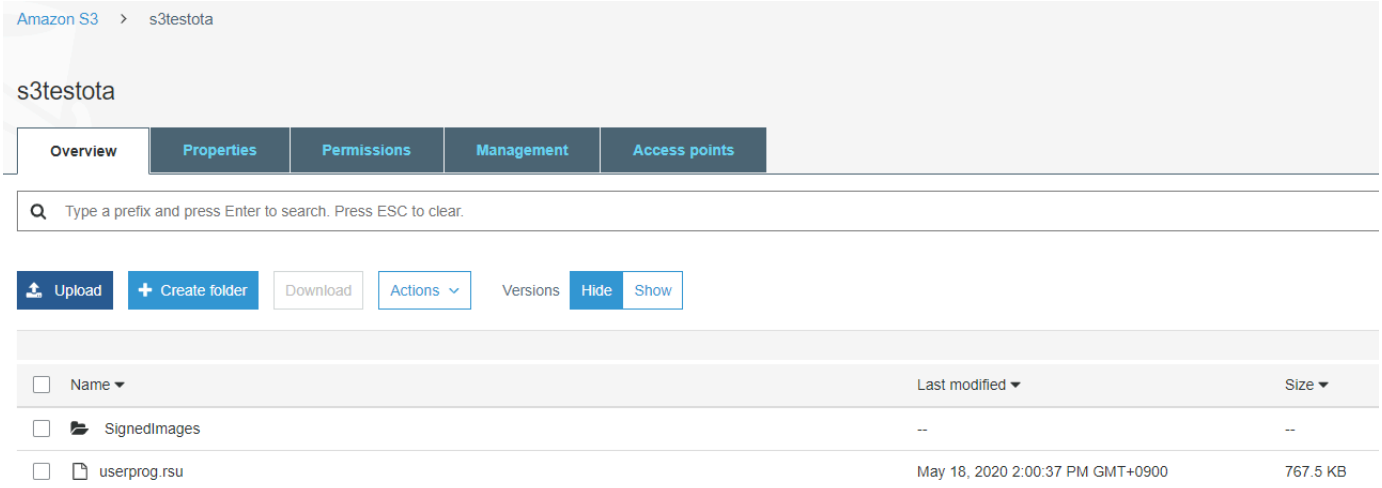
    b. 选择更新固件选项卡并设置以下参数：

    - 文件路径 – `aws_demos.mot` 文件 (`projects\renesas\rx65n-rsk\e2studio\aws_demos\HardwareDebug`) 的位置。

    c. 创建名为 `update _firmware` 的目录。生成 `userprog.rsu` 并将其保存到 `update_firmware` 目录。验证生成是否成功。



19. 如创建 Amazon S3 存储桶以存储更新中所述，将固件更新 `userproj.rsu` 上传到 Amazon S3 存储桶。



20. 创建任务以更新 RX65 N-RSK 上的固件。

    AWS IoT Jobs 是一项向一台或多台连接的设备通知待处理任务的服务。作业可用于管理设备队列、更新设备上的固件和安全证书，或者执行管理任务，例如重新启动设备和执行诊断。

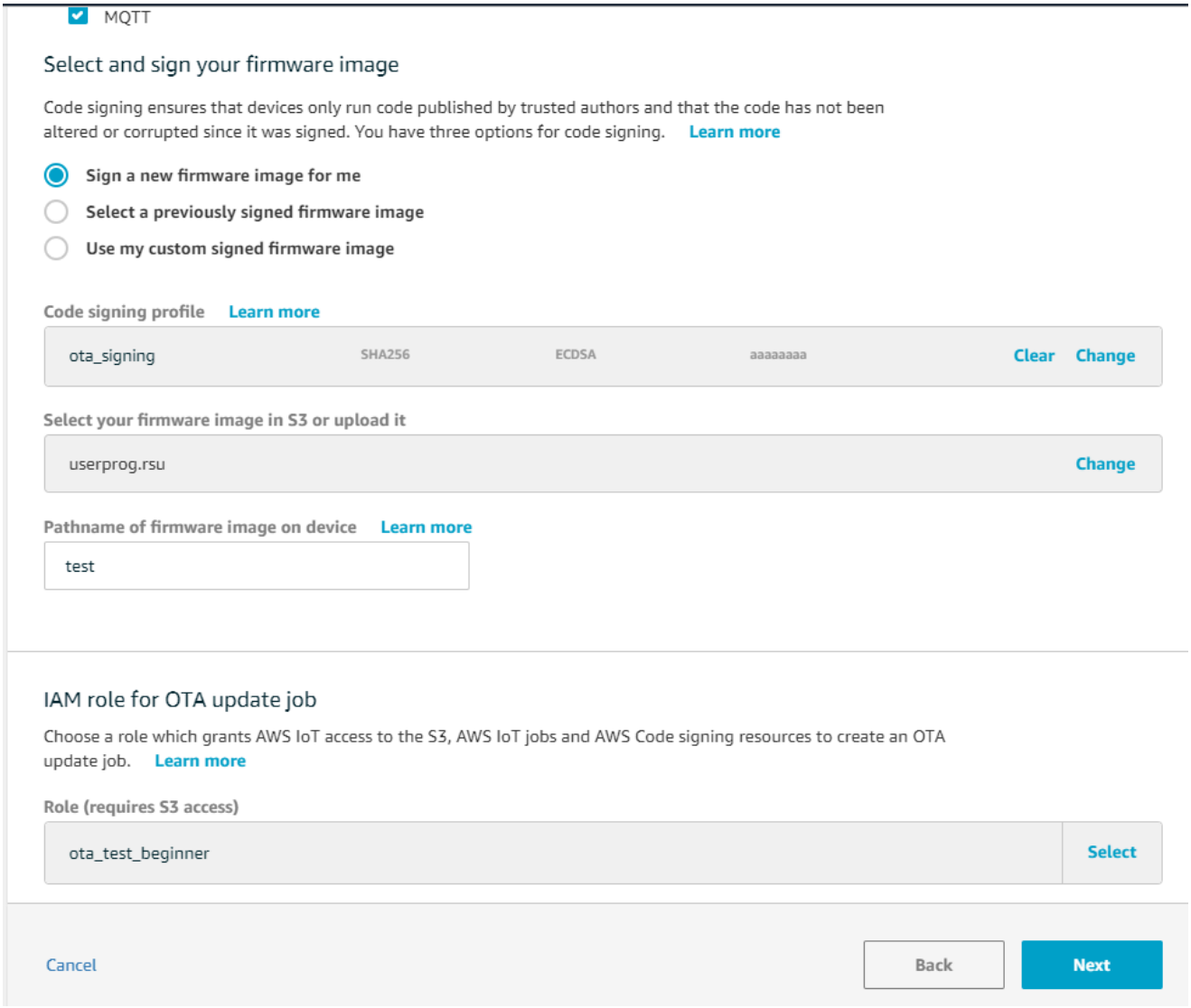
    a. 登录 AWS IoT 控制台。在导航窗格中，选择管理，然后选择作业。

    b. 选择创建作业，然后选择创建 OTA 更新作业。选择一个事物，然后选择下一步。

    c. 按照以下方式创建 FreeRTOS OTA 更新作业：

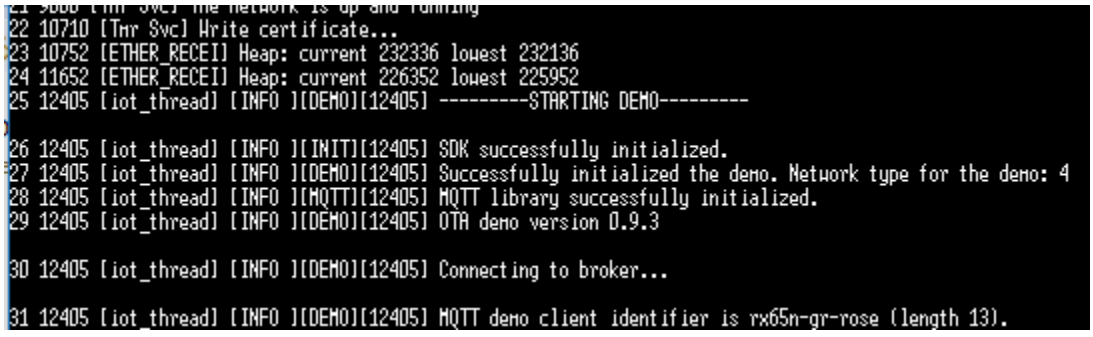
    - 选择 MQTT。
    - 选择在上一节中创建的代码签名配置文件。
    - 选择上传到 Amazon S3 存储桶的固件映像。
    - 对于设备上固件映像的路径名，请输入**`test`**。
    - 选择在上一部分中创建的 IAM 角色。

    d. 选择下一步。

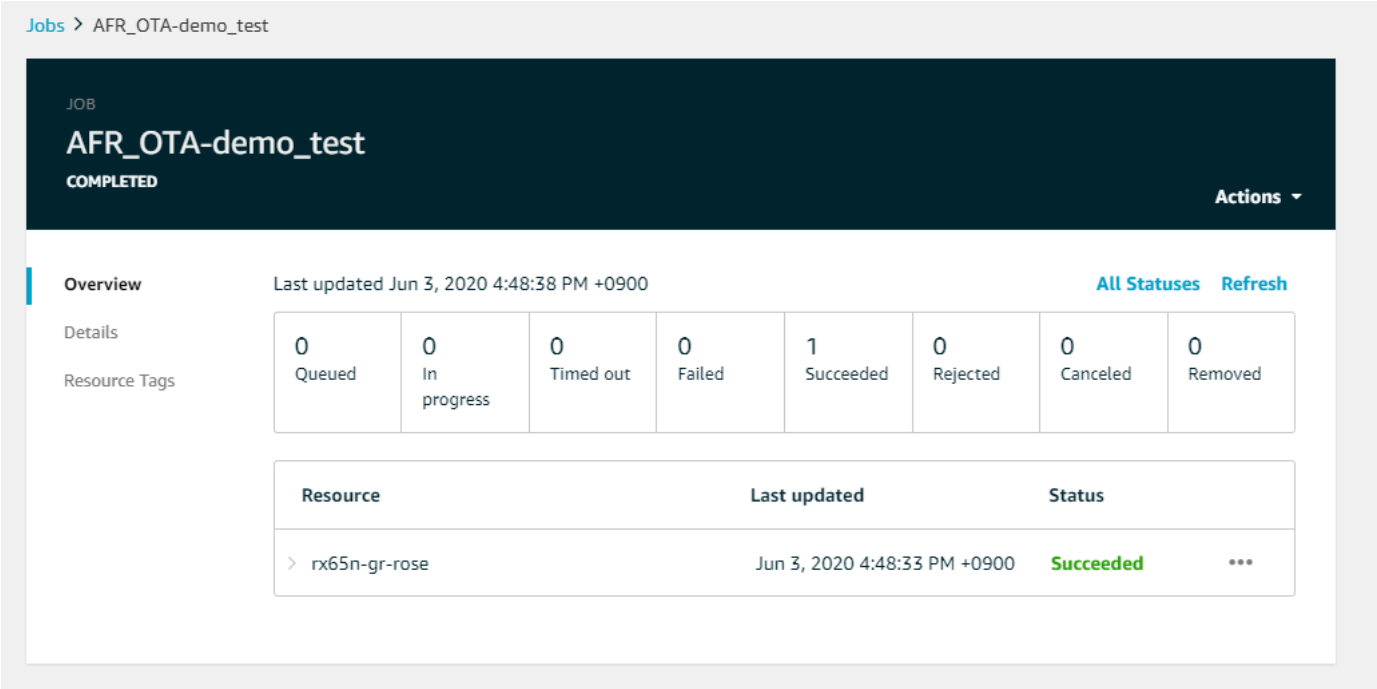


e. 输入一个 ID，然后选择创建。

21. 重新打开 Tera Term 以验证固件是否已成功更新到 OTA 演示版 0.9.3。



22. 在 AWS IoT 控制台上，验证任务状态是否为 “成功”。



教程：使用 ESP32 FreeRTOS 低功耗蓝牙在 Espressif 上执行 OTA 更新

> **Important**
>
> 该参考集成托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

本教程向您展示如何在 Android 设备上更新连接到 MQTT 低功耗蓝牙代理的乐鑫 ESP32 微控制器。它使用 AWS IoT Over-the-air (OTA) 更新任务更新设备。设备 AWS IoT 使用在安卓演示应用程序中输入的 Amazon Cognito 凭证进行连接。经过身份验证的操作员从云端启动 OTA 更新。当设备通过 Android 演示应用程序进行连接时，OTA 更新即会启动，设备上的固件也会更新。

FreeRTOS 版本 2019.06.00 主版本及更高版本包括蓝牙低功耗 MQTT 代理支持，可用于 Wi-Fi 配置和服务安全连接。 AWS IoT 通过使用低功耗蓝牙功能，您可以构建低功耗设备，这些设备无需 Wi-Fi 即可与移动设备配对，从而进行连接。设备可以通过使用通用接入配置文件 (GAP) 和通用属性 (GATT) 配置文件的 Android 或 iOS 低功耗 SDKs 蓝牙进行连接，从而使用 MQTT 进行通信。

为了允许通过低功耗蓝牙进行 OTA 更新，我们将按照以下步骤操作：

1. 配置存储：创建 Amazon S3 存储桶和策略，并配置可执行更新的用户。

2. 创建代码签名证书：创建签名证书并允许用户签署固件更新。
3. 配置 Amazon Cognito 身份验证：创建凭证提供程序、用户群体和应用程序对用户群体的访问权限。
4. 配置 FreeRTOS：设置低功耗蓝牙、客户端凭证和代码签名公共证书。
5. 配置 Android 应用程序：设置凭证提供程序、用户群体，并将应用程序部署到 Android 设备。
6. 运行 OTA 更新脚本：要启动 OTA 更新，请使用 OTA 更新脚本。

有关更新工作的更多信息，请参阅 FreeRTOS 更新 Over-the-Air。有关如何设置低功耗蓝牙 MQTT 代理功能的更多信息，请参阅 Richard Kang 在 Espressif 上的《将低功耗蓝牙与 FreeRTOS 配合使用 FreeRTOS》 ESP32一文。

先决条件

要执行本教程中的步骤，您需要以下资源：

- ESP32 开发板。
- 一根 MicroUSB 转 USB A 电缆。
- 一个 AWS 账户（免费套餐就足够了）。
- 一台使用 Android v 6.0 或更高版本和蓝牙版本 4.2或更高版本的 Android 手机。

在开发计算机上，您需要：

- 有足够的磁盘空间（约 500 Mb）来存放 Xtensa 工具链和 FreeRTOS 源代码和示例。
- 已安装 Android Studio
- 已安装 AWS CLI。
- 已安装 Python3。
- 适用于 Py thon 的 boto3 AWS 软件开发套件 (SDK)。

本教程中的步骤假设 Xtensa 工具链、ESP-IDF 和 FreeRTOS 代码已安装在主目录的 `/esp` 目录中。您必须将 `~/esp/xtensa-esp32-elf/bin` 添加到 `$PATH` 变量中。

步骤 1：配置存储

1. 创建 Amazon S3 存储桶以存储更新 并启用版本控制来保留固件映像。

2. 创建 OTA 更新服务角色 并为角色添加以下托管策略：

   - AWSIot正在记录
   - AWSIotRuleActions
   - AWSIotThingsRegistration
   - AWSFreeRTOSOTAUpdate

3. 创建可执行 OTA 更新的用户。该用户可以在账户中的 IoT 设备上签名和部署固件更新，并且有权在所有设备上执行 OTA 更新。访问权限应仅限于可信实体。
4. 按照步骤创建 OTA 用户策略并将其附加到您的用户。

步骤 2：创建代码签名证书

1. 创建启用版本控制的 Amazon S3 存储桶来保留固件映像。
2. 创建可用于对固件进行签名的代码签名证书。导入证书时，记录证书的 Amazon 资源名称 (ARN)。

```
aws acm import-certificate --profile=ota-update-user --certificate file://
ecdsasigner.crt --private-key file://ecdsasigner.key
```

输出示例：

```
{
"CertificateArn": "arn:aws:acm:us-east-1:<account>:certificate/<certid>"
}
```

稍后您将使用 ARN 创建签名配置文件。如果需要，您可以立即使用以下命令创建该配置文件：

```
aws signer put-signing-profile --profile=ota-update-user --profile-
name esp32Profile --signing-material certificateArn=arn:aws:acm:us-
east-1:account:certificate/certid --platform AmazonFreeRTOS-Default --signing-
parameters certname=/cert.pem
```

输出示例：

```
{
"arn": "arn:aws:signer::<account>:/signing-profiles/esp32Profile"
}
```

步骤 3：Amazon Cognito 身份验证配置

创建 AWS IoT 策略

1. 登录 AWS IoT 控制台。
2. 在控制台的右上角，选择我的账户。在账户设置下，记下您的 12 位账户 ID。
3. 在左侧导航窗格中，选择 设置。在设备数据端点下，记下端点值。该端点的值类似于 `xxxxxxxxxxxxxx.iot.us-west-2.amazonaws.com`。在本示例中， AWS 区域为“us-west-2”。
4. 在左导航窗格中依次选择安全、策略和创建。如果您的账户中没有任何策略，则会看到“您还没有任何政策”消息，并且您可以选择创建政策。
5. 输入策略的名称，例如，“esp32_mqtt_proxy_iot_policy”。
6. 在添加语句部分中，选择高级模式。将以下 JSON 复制并粘贴到策略编辑器窗口中。将 `aws-account-id` 替换为您的账户 ID，并将 `aws-region` 替换为您的区域（例如，“us-west-2”）。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:Connect",
      "Resource": "arn:aws:iot:aws-region:aws-account-id:*"
    },
    {
      "Effect": "Allow",
      "Action": "iot:Publish",
      "Resource": "arn:aws:iot:aws-region:aws-account-id:*"
    },
    {
      "Effect": "Allow",
      "Action": "iot:Subscribe",
      "Resource": "arn:aws:iot:aws-region:aws-account-id:*"
    },
    {
      "Effect": "Allow",
      "Action": "iot:Receive",
      "Resource": "arn:aws:iot:aws-region:aws-account-id:*"
    }
  ]
}
```

7. 选择创建。

创建 AWS IoT 事物

1. 登录 AWS IoT 控制台。
2. 在左侧导航窗格中选择管理，然后选择事物。
3. 在右上角，选择创建。如果您的账户中没有注册任何事物，则会显示消息“您还没有任何事物”，并且您可以选择注册事物。
4. 在创建 AWS IoT 事物页面上，选择创建单个事物。
5. 在将设备添加到事物注册表页面上，为您的事物输入一个名称（例如，“esp32-ble”）。仅允许使用字母数字字符、连字符 (-) 和下划线 (_) 字符。选择下一步。
6. 在添加事物的证书页面上的跳过证书并创建事物下，选择创建没有证书的事物。由于我们使用 BLE 代理移动应用程序，该应用程序使用 Amazon Cognito 凭证进行身份验证和授权，因此不需要设备证书。

创建 Amazon Cognito 应用程序客户端

1. 登录 Amazon Cognito 控制台。
2. 在右上角的导航横幅中选择创建用户群体。
3. 输入群体名称（例如，“esp32_mqtt_proxy_user_pool”）。
4. 选择 Review defaults (查看原定设置)。
5. 在应用程序客户端中，选择添加应用程序客户端，然后选择添加应用程序客户端。
6. 输入应用程序客户端名称（例如，“mqtt_app_client”）。
7. 确保选中生成客户端密钥。
8. 选择创建应用程序客户端。
9. 选择 Return to pool details（返回池详细信息）。
10. 在用户群体的审核页面上，选择创建群体。您应该会看到一条“已成功创建用户池”的消息。记下该群体 ID。
11. 在导航窗格中选择应用程序客户端。
12. 选择显示详细信息。记录应用程序客户端 ID 和应用程序客户端密钥。

创建一个 Amazon Cognito 身份池

1. 登录 Amazon Cognito 控制台。
2. 选择创建新身份池。
3. 输入身份池的名称（例如，“mqtt_proxy_identity_pool”）。
4. 展开身份验证提供程序。
5. 选择 Cognito 选项卡。
6. 输入在前面的步骤中记下的用户群体 ID 和应用程序客户端 ID。
7. 选择创建池。
8. 在下一页上，要为经过身份验证和未经过身份验证的身份创建新角色，请选择允许。
9. 记下身份池 ID，其格式为 `us-east-1:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`。

将 IAM policy 附加到经过身份验证的身份。

1. 打开 Amazon Cognito 控制台。
2. 选择刚才创建的身份池（例如，“mqtt_proxy_identity_pool”）。
3. 选择编辑身份池。
4. 记下分配给经过身份验证的角色的 IAM 角色（例如，“cognito_mqtt_proxy_identity_poolauth_Role”）。
5. 打开 IAM 控制台。
6. 在导航窗格中，选择角色。
7. 搜索分配的角色（例如，“cognito_mqtt_proxy_identity_poolauth_Role”），然后将其选中。
8. 选择添加内联策略，然后选择 JSON。
9. 输入以下策略：

```
{
       "Version": "2012-10-17",
       "Statement": [
       {
          "Effect": "Allow",
          "Action": [
             "iot:AttachPolicy",
             "iot:AttachPrincipalPolicy",
             "iot:Connect",
             "iot:Publish",
```

```
            "iot:Subscribe"
        ],
        "Resource": "*"
    }]
}
```

10. 选择 Review Policy（查看策略）。
11. 输入策略名称（例如，“mqttProxyCognito策略”）。
12. 选择创建策略。

步骤 4：配置 Amazon FreeRTOS

1. 从 FreeRTOS 存储库下载最新版本的亚马逊 FreeRTOS 代码。 GitHub
2. 要启用 OTA 更新演示，请按照开始使用 Espressif ESP32-C 和 DevKit ESP-WROVER-KIT中的步骤操作。
3. 在以下文件中进行这些修改：

   a. 打开 `vendors/espressif/boards/esp32/aws_demos/config_files/aws_demo_config.h` 并定义 `CONFIG_OTA_UPDATE_DEMO_ENABLED`。

   b. 打开 `vendors/espressif/boards/esp32/aws_demos/common/config_files/aws_demo_config.h` 并将 `democonfigNETWORK_TYPES` 更为为 `AWSIOT_NETWORK_TYPE_BLE`。

   c. 打开 `demos/include/aws_clientcredential.h` 并输入 `clientcredentialMQTT_BROKER_ENDPOINT` 的端点 URL。

      输入 `clientcredentialIOT_THING_NAME` 的事物名称（例如，“esp32-ble”）。使用 Amazon Cognito 凭证时，无需添加证书。

   d. 打开 `vendors/espressif/boards/esp32/aws_demos/config_files/aws_iot_network_config.h` 并更改 `configSUPPORTED_NETWORKS` 和 `configENABLED_NETWORKS`，以便仅包含 `AWSIOT_NETWORK_TYPE_BLE`。

   e. 打开 `vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/ota_demo_config.h` 文件，然后输入您的证书。

      ```
      #define otapalconfigCODE_SIGNING_CERTIFICATE [] = "your-certificate-key";
      ```

   应用程序应该会启动并输出演示版本：

```
11 13498 [iot_thread] [INFO ][DEMO][134980] Successfully initialized the demo.
 Network type for the demo: 2
12 13498 [iot_thread] [INFO ][MQTT][134980] MQTT library successfully initialized.
13 13498 [iot_thread] OTA demo version 0.9.20
14 13498 [iot_thread] Creating MQTT Client...
```

步骤 5：配置 Android 应用程序

1. 从 amazon-freertos-ble-android-sdk 存储库中下载 Android 低功耗蓝牙 SDK GitHub 和示例应用程序。
2. 打开文件app/src/main/res/raw/awsconfiguration.json并填写池 ID AppClientId、区域，然后 AppClientSecret 按照以下 JSON 示例中的说明进行操作。

```
{
  "UserAgent": "MobileHub/1.0",
  "Version": "1.0",
  "CredentialsProvider": {
    "CognitoIdentity": {
      "Default": {
        "PoolId": "Cognito->Manage Identity Pools->Federated Identities-
>mqtt_proxy_identity_pool->Edit Identity Pool->Identity Pool ID",
        "Region": "Your region (for example us-east-1)"
      }
    }
  },

  "IdentityManager": {
    "Default": {}
  },

  "CognitoUserPool": {
    "Default": {
      "PoolId": "Cognito-> Manage User Pools -> esp32_mqtt_proxy_user_pool ->
 General Settings -> PoolId",
      "AppClientId": "Cognito-> Manage User Pools -> esp32_mqtt_proxy_user_pool ->
 General Settings -> App clients ->Show Details",
      "AppClientSecret": "Cognito-> Manage User Pools -> esp32_mqtt_proxy_user_pool
 -> General Settings -> App clients ->Show Details",
      "Region": "Your region (for example us-east-1)"
    }
```

```
  }
}
```

3. 打开app/src/main/java/software/amazon/freertos/DemoConstants.java并输入您之前创建的策略名称（例如*esp32_mqtt_proxy_iot_policy*）和区域（例如*us-east-1*）。
4. 构建和安装演示应用程序。
   a. 在 Android Studio 中，选择构建，然后选择创建模块应用程序。
   b. 选择运行，然后选择运行应用程序。您可以转到 Android Studio 中的 logcat 窗格来监控日志消息。
   c. 在 Android 设备上，通过登录屏幕创建一个账户。
   d. 创建用户。如果用户已存在，请输入凭证。
   e. 允许 Amazon FreeRTOS 演示访问设备的位置。
   f. 扫描低功耗蓝牙设备。
   g. 将找到的设备的滑块移至开。
   h. 在串行端口调试控制台上按 y 键 ESP32。
   i. 选择配对并连接。
5. 建立连接后，更多...链接会变为活动状态。连接完成后，Android 设备 logcat 中的连接状态会更改为“BLE_CONNECTED”：

```
2019-06-06 20:11:32.160 23484-23497/software.amazon.freertos.demo I/FRD: BLE
 connection state changed: 0; new state: BLE_CONNECTED
```

6. 在传输消息之前，Amazon FreeRTOS 设备和 Android 设备会协商 MTU。您会在 logcat 中看到以下输出：

```
2019-06-06 20:11:46.720 23484-23497/software.amazon.freertos.demo I/FRD:
 onMTUChanged : 512 status: Success
```

7. 设备连接到应用程序并开始使用 MQTT 代理发送 MQTT 消息。要确认设备可通信，请确保 MQTT_CONTROL 特征数据值已更改为 01：

```
2019-06-06 20:12:28.752 23484-23496/software.amazon.freertos.demo D/FRD: <-<-<-
 Writing to characteristic: MQTT_CONTROL with data: 01
2019-06-06 20:12:28.839 23484-23496/software.amazon.freertos.demo D/FRD:
 onCharacteristicWrite for: MQTT_CONTROL; status: Success; value: 01
```

8. 设备配对后，您将在 ESP32 主机上看到一条提示。要启用 BLE，请按 y。在执行此步骤后，演示才能运行。

```
E (135538) BT_GATT: GATT_INSUF_AUTHENTICATION: MITM Required
W (135638) BT_L2CAP: l2cble_start_conn_update, the last connection update command
 still pending.
E (135908) BT_SMP: Value for numeric comparison = 391840
15 13588 [InputTask] Numeric comparison:391840
16 13589 [InputTask] Press 'y' to confirm
17 14078 [InputTask] Key accepted
W (146348) BT_SMP: FOR LE SC LTK IS USED INSTEAD OF STK
18 16298 [iot_thread] Connecting to broker...
19 16298 [iot_thread] [INFO ][MQTT][162980] Establishing new MQTT connection.
20 16298 [iot_thread] [INFO ][MQTT][162980] (MQTT connection 0x3ffd5754, CONNECT
 operation 0x3ffd586c) Waiting for operation completion.
21 16446 [iot_thread] [INFO ][MQTT][164450] (MQTT connection 0x3ffd5754, CONNECT
 operation 0x3ffd586c) Wait complete with result SUCCESS.
22 16446 [iot_thread] [INFO ][MQTT][164460] New MQTT connection 0x3ffc0ccc
 established.
23 16446 [iot_thread] Connected to broker.
```

步骤 6：运行 OTA 更新脚本

1. 要安装先决条件，请运行下面的命令：

```
pip3 install boto3
```

```
pip3 install pathlib
```

2. 在 `demos/include/aws_application_version.h` 中增加 FreeRTOS 应用程序的版本。
3. 构建一个新的 .bin 文件。
4. 下载 python 脚本 start_ota.py。要查看脚本的帮助内容，请在终端窗口中运行以下命令：

```
python3 start_ota.py -h
```

您应看到类似于以下内容的信息：

```
usage: start_ota.py [-h] --profile PROFILE [--region REGION]
                    [--account ACCOUNT] [--devicetype DEVICETYPE] --name NAME
```

```
                    --role ROLE --s3bucket S3BUCKET --otasigningprofile
                    OTASIGNINGPROFILE --signingcertificateid
                    SIGNINGCERTIFICATEID [--codelocation CODELOCATION]
Script to start OTA update
optional arguments:
-h, --help            show this help message and exit
--profile PROFILE     Profile name created using aws configure
--region REGION       Region
--account ACCOUNT     Account ID
--devicetype DEVICETYPE thing|group
--name NAME           Name of thing/group
--role ROLE           Role for OTA updates
--s3bucket S3BUCKET   S3 bucket to store firmware updates
--otasigningprofile OTASIGNINGPROFILE
                      Signing profile to be created or used
--signingcertificateid SIGNINGCERTIFICATEID
                      certificate id (not arn) to be used
--codelocation CODELOCATION
                      base folder location (can be relative)
```

5. 如果您使用提供的 AWS CloudFormation 模板创建资源，请运行以下命令：

```
python3 start_ota_stream.py --profile otausercf --name esp32-ble --role
 ota_ble_iot_role-sample --s3bucket afr-ble-ota-update-bucket-sample --
otasigningprofile abcd --signingcertificateid certificateid
```

你应该在 ESP32 调试控制台中看到更新开始：

```
38 2462 [OTA Task] [prvParseJobDoc] Job was accepted. Attempting to start transfer.
---
49 2867 [OTA Task] [prvIngestDataBlock] Received file block 1, size 1024
50 2867 [OTA Task] [prvIngestDataBlock] Remaining: 1290
51 2894 [OTA Task] [prvIngestDataBlock] Received file block 2, size 1024
52 2894 [OTA Task] [prvIngestDataBlock] Remaining: 1289
53 2921 [OTA Task] [prvIngestDataBlock] Received file block 3, size 1024
54 2921 [OTA Task] [prvIngestDataBlock] Remaining: 1288
55 2952 [OTA Task] [prvIngestDataBlock] Received file block 4, size 1024
56 2953 [OTA Task] [prvIngestDataBlock] Remaining: 1287
57 2959 [iot_thread] State: Active  Received: 5   Queued: 5   Processed: 5
 Dropped: 0
```

6. OTA 更新完成后，设备将按照 OTA 更新过程的要求重新启动。然后，它会尝试使用更新的固件进行连接。如果升级成功，则会将更新后的固件标记为活动状态，您会在控制台中看到更新的版本：

```
13 13498 [iot_thread] OTA demo version 0.9.21
```

AWS IoT Device Shadow 演示应用程序

> **Important**
>
> 该演示托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

简介

此演示展示了如何使用 Dev AWS IoT ice Shadow 库连接到 Dev AWS ice Shadow 服务。它使用 TLS（相互身份验证）与 MQTT 代理建立 M AWS IoT QTT 连接，并使用 CoreJson 库解析器来解析从 Shadow 服务收到的影子文档。coreMQTT 库 AWS 该演示展示了基本的影子操作，例如如何更新影子文档和如何删除影子文档。该演示还展示了如何在 coreMQTT 库中注册回调函数来处理影子消息，例如，影子 `/update` 和从 AWS IoT Device Shadow 服务发送的 `/update/delta` 消息等。

此演示仅用于学习练习，因为更新影子文档（状态）的请求和更新响应是由同一个应用程序完成的。在现实生产场景中，即使设备当前未连接，外部应用程序也会请求远程更新设备的状态。设备连接后将确认更新请求。

> **Note**
>
> 要设置和运行 FreeRTOS 演示，请按照开始使用 FreeRTOS中的步骤操作。

功能

该演示创建了一个应用程序任务，该任务会循环执行一组示例，这些示例演示了影子 `/update` 和 `/update/delta` 回调，从而模拟切换远程设备的状态。它会发送带有新的 `desired` 状态的影子更新，并等待设备根据新的 `reported` 状态更改其 `desired` 状态。此外，它还使用影子 `/update` 回调来输出不断变化的影子状态。此演示还使用了与 MQTT 代理的安全的 AWS IoT MQTT 连接，并假设设备影子中存在power0n状态。

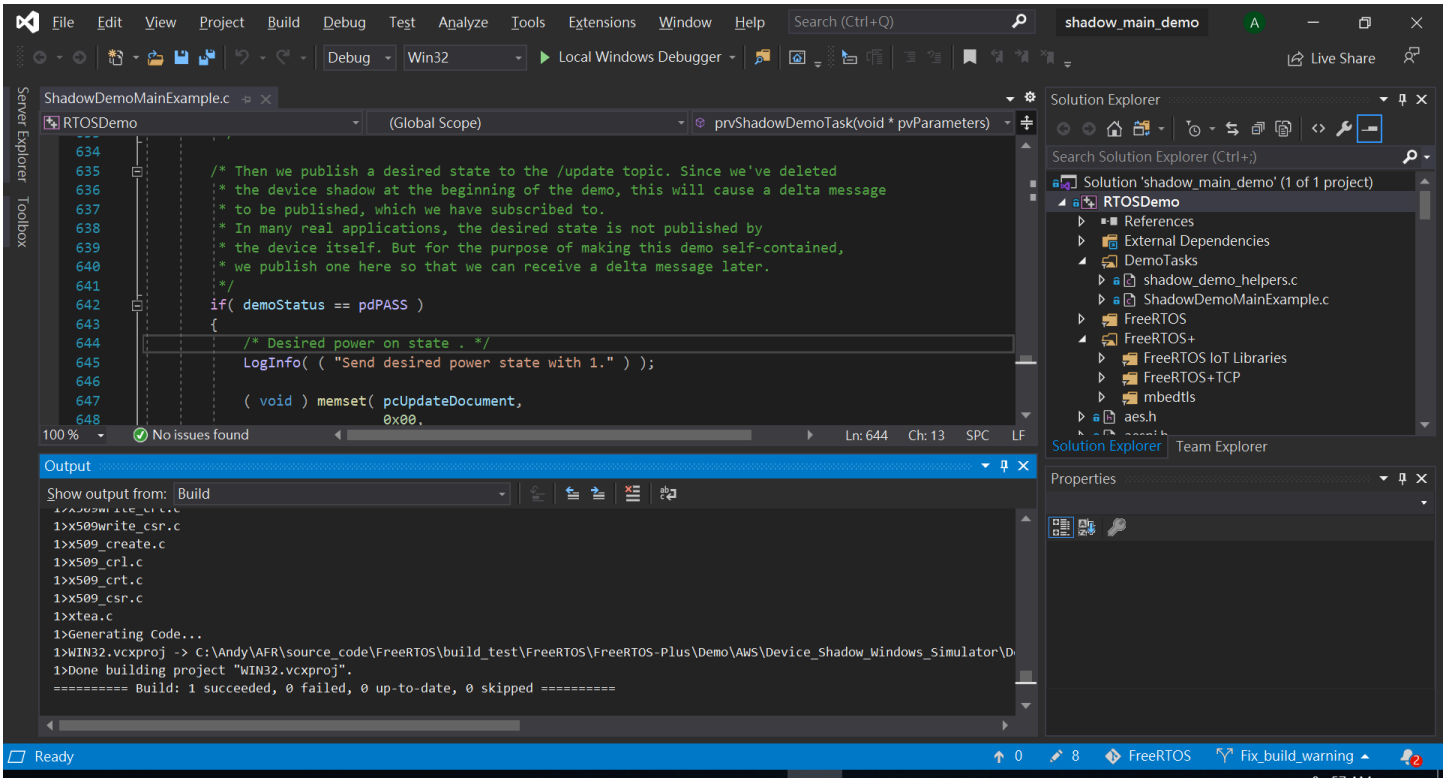
该演示执行以下操作：

1. 使用 shadow_demo_helpers.c 中的帮助程序函数建立 MQTT 连接。
2. 使用 AWS IoT Device Shadow 库定义的宏汇编用于设备影子操作的 MQTT 主题字符串。
3. 发布到用于删除设备影子的 MQTT 主题以删除任何现有设备影子。
4. 使用 shadow_demo_helpers.c 中的帮助程序函数为 /update/delta、/update/accepted 和 /update/rejected 订阅 MQTT 主题。
5. 使用 shadow_demo_helpers.c 中的帮助程序函数发布 powerOn 的所需状态。这将导致向设备发送 /update/delta 消息。
6. 使用 Device Shadow 库定义的函数（Shadow_MatchTopic）处理传入的 MQTT 消息，并确定该消息是否与 AWS IoT 设备影子相关。prvEventCallback如果该消息是设备影子 /update/delta 消息，则主演示函数将发布第二条消息，将报告的状态更新为 powerOn。如果收到一条 /update/accepted 消息，请确认该消息与之前在更新消息中发布的消息具有相同的 clientToken。这标志演示的结束。

```
82 9136 [ShadowDemo] [INFO] [SHADOW] [prvShadowDemoTask:641] 83 9136 [ShadowDemo] Send desired power state with 1.84 9136 [ShadowDemo]
85 9296 [ShadowDemo] [INFO] [SHADOW] [prvEventCallback:482] 86 9296 [ShadowDemo] pPublishInfo->pTopicName:$aws/things/testClient16:34:41/shadow/update/delta.87 9296 [Shad
owDemo]
88 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateDeltaHandler:256] 89 9296 [ShadowDemo] /update/delta json payload:{"version":1,"timestamp":1602751002,"state":{"powerOn":1}
,"metadata":{"powerOn":{"timestamp":1602751002}},"clientToken":"009136"}.90 9296 [ShadowDemo]
91 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateDeltaHandler:298] 92 9296 [ShadowDemo] version: 193 9296 [ShadowDemo]
94 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateDeltaHandler:308] 95 9296 [ShadowDemo] version:1, ulCurrentVersion:0
96 9296 [ShadowDemo]
97 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateDeltaHandler:342] 98 9296 [ShadowDemo] The new power on state newState:1, ulCurrentPowerOnState:0
99 9296 [ShadowDemo]
100 9296 [ShadowDemo] [INFO] [SHADOW] [prvEventCallback:482] 101 9296 [ShadowDemo] pPublishInfo->pTopicName:$aws/things/testClient16:34:41/shadow/update/accepted.102 9296
 [ShadowDemo]
103 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:376] 104 9296 [ShadowDemo] /update/accepted json payload:{"state":{"desired":{"powerOn":1}},"metadata":{"d
esired":{"powerOn":{"timestamp":1602751002}}},"version":1,"timestamp":1602751002,"clientToken":"009136"}.105 9296 [ShadowDemo]
106 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:424] 107 9296 [ShadowDemo] clientToken: 009136108 9296 [ShadowDemo]
109 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:429] 110 9296 [ShadowDemo] receivedToken:9136, clientToken:0
111 9296 [ShadowDemo]
112 9296 [ShadowDemo] [WARN] [SHADOW] [prvUpdateAcceptedHandler:442] 113 9296 [ShadowDemo] The received clientToken=9136 is not identical with the one=0 we sent 114 9296
[ShadowDemo]
115 9696 [ShadowDemo] [INFO] [SHADOW] [prvShadowDemoTask:670] 116 9696 [ShadowDemo] Report to the state change: 1117 9696 [ShadowDemo]
118 9856 [ShadowDemo] [INFO] [SHADOW] [prvEventCallback:482] 119 9856 [ShadowDemo] pPublishInfo->pTopicName:$aws/things/testClient16:34:41/shadow/update/accepted.120 9856
 [ShadowDemo]
121 9856 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:376] 122 9856 [ShadowDemo] /update/accepted json payload:{"state":{"reported":{"powerOn":1}},"metadata":{"
reported":{"powerOn":{"timestamp":1602751003}}},"version":2,"timestamp":1602751003,"clientToken":"009696"}.123 9856 [ShadowDemo]
124 9856 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:424] 125 9856 [ShadowDemo] clientToken: 009696126 9856 [ShadowDemo]
127 9856 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:429] 128 9856 [ShadowDemo] receivedToken:9696, clientToken:9696
129 9856 [ShadowDemo]
130 9856 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:437] 131 9856 [ShadowDemo] Received response from the device shadow. Previously published update with clie
ntToken=9696 has been accepted. 132 9856 [ShadowDemo]
133 10256 [ShadowDemo] [INFO] [SHADOW] [prvShadowDemoTask:698] 134 10256 [ShadowDemo] Start to unsubscribe shadow topics and disconnect from MQTT.
135 10256 [ShadowDemo]
136 12036 [ShadowDemo] [INFO] [SHADOW] [prvShadowDemoTask:747] 137 12036 [ShadowDemo] Demo completed successfully.138 12036 [ShadowDemo]
139 12036 [ShadowDemo] [INFO] [SHADOW] [prvShadowDemoTask:756] 140 12036 [ShadowDemo] Deleting Shadow Demo task.141 12036 [ShadowDemo]
```

该演示可以在文件中*freertos*/demos/device_shadow_for_aws/shadow_demo_main.c或上找到 GitHub。

以下屏幕截图显示了演示成功时的预期输出。



连接 AWS IoT MQTT 经纪商

要连接到 AWS IoT MQTT 代理，我们使用与MQTT_Connect()中相同的coreMQTT 双向身份验证演示方法。

删除影子文档

要删除影子文档，请使用 D xPublishToTopic evice Shadow 库定义的 AWS IoT 宏调用一条空消息。这会使用 MQTT_Publish 来发布到 /delete 主题。以下代码部分显示了如何在函数 prvShadowDemoTask 中执行此操作。

```
/* First of all, try to delete any Shadow document in the cloud. */
returnStatus = PublishToTopic( SHADOW_TOPIC_STRING_DELETE( THING_NAME ),
                               SHADOW_TOPIC_LENGTH_DELETE( THING_NAME_LENGTH ),
                               pcUpdateDocument,
                               0U );
```

订阅影子主题

订阅 Device Shadow 主题，接收 AWS IoT 代理发送的有关影子变更的通知。Device Shadow 主题由 Device Shadow 库中定义的宏汇编而成。以下代码部分显示了如何在函数 `prvShadowDemoTask` 中执行此操作。

```
/* Then try to subscribe shadow topics. */
if( returnStatus == EXIT_SUCCESS )
{
    returnStatus = SubscribeToTopic(
                     SHADOW_TOPIC_STRING_UPDATE_DELTA( THING_NAME ),
                     SHADOW_TOPIC_LENGTH_UPDATE_DELTA( THING_NAME_LENGTH ) );
}

if( returnStatus == EXIT_SUCCESS )
{
    returnStatus = SubscribeToTopic(
                     SHADOW_TOPIC_STRING_UPDATE_ACCEPTED( THING_NAME ),
                     SHADOW_TOPIC_LENGTH_UPDATE_ACCEPTED( THING_NAME_LENGTH ) );
}

if( returnStatus == EXIT_SUCCESS )
{
    returnStatus = SubscribeToTopic(
                     SHADOW_TOPIC_STRING_UPDATE_REJECTED( THING_NAME ),
                     SHADOW_TOPIC_LENGTH_UPDATE_REJECTED( THING_NAME_LENGTH ) );
}
```

发送影子更新

要发送影子更新，该演示使用由 Device Shadow 库定义的宏，通过一条 JSON 格式的消息调用 `xPublishToTopic`。这会使用 `MQTT_Publish` 来发布到 `/delete` 主题。以下代码部分显示了如何在函数 `prvShadowDemoTask` 中执行此操作。

```
#define SHADOW_REPORTED_JSON    \
    "{"                         \
    "\"state\":{"               \
    "\"reported\":{"            \
    "\"powerOn\":%01d"          \
    "}"                         \
```

```
    "},"                          \
    "\"clientToken\":\"%06lu\"" \
    "}"
snprintf( pcUpdateDocument,
          SHADOW_REPORTED_JSON_LENGTH + 1,
          SHADOW_REPORTED_JSON,
           ( int ) ulCurrentPowerOnState,
           ( long unsigned ) ulClientToken );

xPublishToTopic( SHADOW_TOPIC_STRING_UPDATE( THING_NAME ),
                 SHADOW_TOPIC_LENGTH_UPDATE( THING_NAME_LENGTH ),
                 pcUpdateDocument,
                 ( SHADOW_DESIRED_JSON_LENGTH + 1 ) );
```

处理阴影增量消息和影子更新消息

使用 MQTT_Init 函数注册到 coreMQTT 客户端库的用户回调函数将通知我们传入数据包的事件。请参阅 prvEventCallback上的回调函数 GitHub。

回调函数确认传入的数据包的类型为 MQTT_PACKET_TYPE_PUBLISH，并使用 Device Shadow 库 API Shadow_MatchTopic 确认传入的消息是否为影子消息。

如果传入的消息是带有类型的影子消息ShadowMessageTypeUpdateDelta，则我们调用 prvUpdateDeltaHandler 来处理此消息。处理程序 prvUpdateDeltaHandler 使用 coreJson 库解析消息，以便获取 powerOn 状态的增量值，并将其与本地维护的当前设备状态进行比较。如果不同，则会更新本地设备状态以反映影子文档中 powerOn 状态的新值。

如果传入的消息是带有类型的影子消息ShadowMessageTypeUpdateAccepted，则我们调用 prvUpdateAcceptedHandler 来处理此消息。处理程序 prvUpdateAcceptedHandler 使用 coreJson 库解析消息，以便从消息获取 clientToken。此处理函数会检查 JSON 消息中的客户端令牌是否与应用程序使用的客户端令牌匹配。如果不匹配，则此函数会记录警告消息。

安全套接字 Echo 客户端演示

> **Important**
>
> 该演示托管在已弃用的 Amazon-FreeRTOS 存储库中。当您创建新项目时，我们建议从此处开始。如果您已经有一个基于现已弃用的 Amazon-FreeRTOS 存储库的 FreeRTOS 项目，请参阅 Amazon-FreeRTOS Github 存储库迁移指南。

以下示例使用单个 RTOS 堆栈。可以在 `demos/tcp/aws_tcp_echo_client_single_task.c` 中找到本示例的源代码。

在开始之前，请确认您已将 FreeRTOS 下载到微控制器，并且已构建和运行 FreeRTOS 演示项目。你可以从中克隆或下载 FreeRTOS。GitHub有关说明，请参阅 README.md 文件。

运行演示

> **Note**
>
> 要设置和运行 FreeRTOS 演示，请按照开始使用 FreeRTOS中的步骤操作。
> Cypress 07 F 和 CYW9439 07 AEVAL1 AEVAL1 F 开发套件目前不支持 TCP 服务器和 CYW9549客户端演示。

1. 按照《FreeRTOS 移植指南》中设置 TLS Echo 服务器的说明进行操作。

   TLS Echo 服务器应已运行并在端口 9000 上侦听。

   在设置期间，您应已生成四个文件：

   - `client.pem`（客户端证书）
   - `client.key`（客户端私有密钥）
   - `server.pem`（服务器证书）
   - `server.key`（服务器私有密钥）
2. 使用工具 `tools/certificate_configuration/CertificateConfigurator.html` 将客户端证书 (`client.pem`) 和客户端私有密钥 (`client.key`) 复制到 `aws_clientcredential_keys.h`。
3. 打开 `FreeRTOSConfig.h`文件。
4. 将 `configECHO_SERVER_ADDR0`、`configECHO_SERVER_ADDR1`、`configECHO_SERVER_ADDR2` 和 `configECHO_SERVER_ADDR3` 变量设置为 4 个整数，它们构成了 TLS Echo 服务器在其中运行的 IP 地址。
5. 将 `configTCP_ECHO_CLIENT_PORT` 变量设置为 `9000`（TLS Echo 服务器所侦听的端口）。
6. 将 `configTCP_ECHO_TASKS_SINGLE_TASK_TLS_ENABLED` 变量设置为 1。

7. 使用工具 tools/certificate_configuration/PEMfileToCString.html 将服务器证书 (server.pem) 复制到 aws_tcp_echo_client_single_task.c 文件中的 cTlsECHO_SERVER_CERTIFICATE_PEM。
8. 打开 *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h，注释掉 #define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED 并定义 CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED 或 CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED。

微控制器和 TLS Echo 服务器应位于同一网络中。在演示开始时 (main.c)，您应看到一条日志消息，其内容为 Received correct string from echo server。

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。