
Amazon FreeRTOS

用户指南



Amazon FreeRTOS: 用户指南

Copyright © 2019 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

什么是 Amazon FreeRTOS ?	1
FreeRTOS 内核	1
Amazon FreeRTOS 库	1
Amazon FreeRTOS 控制台	1
下载 Amazon FreeRTOS 源代码	2
无线更新	2
开发工作流程	2
其他资源	3
Amazon FreeRTOS 入门	4
先决条件	4
AWS 账户和先决条件	4
Amazon FreeRTOS 支持的硬件平台	5
将您的 MCU 主板注册到 AWS IoT	5
安装终端仿真器	7
开始使用 Texas Instruments CC3220SF-LAUNCHXL	8
设置环境	8
下载并配置 Amazon FreeRTOS	10
生成并运行 Amazon FreeRTOS 演示项目	12
问题排查	13
开始使用 STMicroelectronics STM32L4 Discovery Kit IoT Node	13
设置环境	13
下载并配置 Amazon FreeRTOS	14
生成并运行 Amazon FreeRTOS 演示项目	15
问题排查	16
开始使用 NXP LPC54018 IoT Module	17
设置环境	17
下载并配置 Amazon FreeRTOS	18
生成并运行 Amazon FreeRTOS 演示项目	19
问题排查	20
Microchip Curiosity PIC32MZEF 入门	21
设置 Microchip Curiosity PIC32MZEF 硬件	21
设置环境	22
下载并配置 Amazon FreeRTOS	23
生成并运行 Amazon FreeRTOS 演示项目	24
问题排查	26
Espressif ESP32-DevKitC 和 ESP-WROVER-KIT 入门	26
设置 Espressif 硬件	26
设置环境	26
下载并配置 Amazon FreeRTOS	27
生成并运行 Amazon FreeRTOS 演示项目	28
问题排查	30
开始使用 Infineon XMC4800 IoT Connectivity Kit	34
设置环境	34
下载并配置 Amazon FreeRTOS	35
生成并运行 Amazon FreeRTOS 演示项目	36
Xilinx Avnet MicroZed 工业 IoT 工具包入门	38
设置 MicroZed 硬件	39
设置环境	39
下载并配置 Amazon FreeRTOS	41
生成并运行 Amazon FreeRTOS 演示项目	42
问题排查	50
Renesas Starter Kit+ for RX65N-2MB 入门	51
设置 Renesas 硬件	51
设置环境	51

下载并配置 Amazon FreeRTOS	52
生成并运行 Amazon FreeRTOS 示例	53
MediaTek MT7697Hx 开发工具包入门	57
设置环境	57
下载并配置 Amazon FreeRTOS	57
使用 Keil MDK 生成并运行 Amazon FreeRTOS 演示项目	59
在 Keil µVision 中调试 Amazon FreeRTOS 项目	60
开始使用 FreeRTOS Windows 仿真器	60
设置环境	60
下载并配置 Amazon FreeRTOS	61
生成并运行 Amazon FreeRTOS 演示项目	62
Nordic nRF52840-DK 入门	63
设置 Nordic 硬件	63
设置环境	63
下载并配置 Amazon FreeRTOS	64
生成并运行 Amazon FreeRTOS 演示项目	65
Amazon FreeRTOS 开发人员指南	66
Amazon FreeRTOS 架构	66
FreeRTOS 内核基础知识	66
FreeRTOS 内核计划程序	67
内存管理	67
任务间协调	68
软件计时器	70
低功耗支持	70
Amazon FreeRTOS 库	70
Amazon FreeRTOS 移植库	71
Amazon FreeRTOS 应用程序库	79
低功耗蓝牙	84
AWS IoT Device Defender	94
AWS IoT Greengrass	97
MQTT (测试版)	99
MQTT (传统)	101
无线 (OTA) 代理	104
公有密钥加密标准 (PKCS) #11	106
安全套接字	108
AWS IoT Device Shadow	112
传输层安全性 (TLS)	114
Wi-Fi	114
Amazon FreeRTOS 无线更新	118
无线更新先决条件	119
OTA 教程	132
OTA Update Manager 服务	152
将 OTA 代理集成到应用程序中	153
OTA 安全性	155
OTA 故障排除	156
Amazon FreeRTOS 控制台	161
预定义的 Amazon FreeRTOS 配置	161
自定义 Amazon FreeRTOS 配置	162
快速连接工作流程	163
问题排查	163
Amazon FreeRTOS 演示项目	164
浏览演示应用程序	164
目录和文件组织结构	164
配置文件	164
低功耗蓝牙演示应用程序 (测试版)	165
概述	165
先决条件	165

常见组件	167
MQTT over BLE	170
Wi-Fi 预配置	172
通用属性服务器	174
安全套接字 Echo 客户端演示	175
设备影子演示应用程序	176
Greengrass Discovery 示例应用程序	177
OTA 演示应用程序	178
Microchip Curiosity PIC32MZEF 的演示启动加载程序	181
启动加载程序状态	181
闪存设备	182
应用程序映像结构	182
映像标头	183
映像描述符	183
映像后缀	184
启动加载程序配置	185
生成启动加载程序	185
Amazon FreeRTOS 移植指南	186
启动加载程序	186
日志记录	186
日志记录配置	186
连接	187
Wi-Fi 管理	187
套接字	187
安全性	188
TLS	188
PKCS #11	188
将自定义库与 Amazon FreeRTOS 配合使用	189
OTA 可移植抽象层	189
Amazon FreeRTOS 资格审查计划	191
适用于 Amazon FreeRTOS 的 AWS IoT Device Tester 用户指南	192
先决条件	192
下载 Amazon FreeRTOS	192
下载适用于 Amazon FreeRTOS 的 AWS IoT Device Tester。	192
创建和配置 AWS 账户	193
安装 AWS 命令行界面 (CLI)	193
首次进行测试，以确认您的微控制器主板是否符合要求	193
添加库移植层	193
配置您的 AWS 凭证	194
在 AWS IoT Device Tester 中创建设备池	195
配置生成、刷入和测试设置	197
运行 Amazon FreeRTOS 资格套件	202
AWS IoT Device Tester 命令	202
结果和日志	203
查看结果	203
用于重新确定资格的测试	205
问题排查	205
排查设备配置问题	205
权限策略模板	208

什么是 Amazon FreeRTOS ?

Amazon FreeRTOS 包括以下组件：

- 基于 FreeRTOS 内核的微控制器操作系统
- Amazon FreeRTOS 库，用于连接、安全性和无线 (OTA) 更新。
- 控制台，通过它您可以下载包含开始使用 Amazon FreeRTOS 所需全部内容的 zip 文件。
- 无线 (OTA) 更新。

FreeRTOS 内核

FreeRTOS 内核是一个实时操作系统内核，支持多种架构，适合用于构建嵌入式微控制器应用程序。该内核提供：

- 多任务计划程序。
- 多个内存分配选项（包括创建静态分配系统的功能）。
- 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。

Amazon FreeRTOS 库

Amazon FreeRTOS 提供的库让您可以：

- 使用 MQTT 和设备影子安全地将设备连接到 AWS IoT 云。
- 发现并连接到 AWS IoT Greengrass 内核。
- 管理 Wi-Fi 连接。
- 监听并处理无线 (OTA) 更新。

有关更多信息，请参阅 [Amazon FreeRTOS 库](#)。

Amazon FreeRTOS 控制台

[Amazon FreeRTOS 控制台](#) 让您可以配置和下载程序包，其中包含为基于微控制器的设备编写程序所需的所有内容：

- FreeRTOS 内核
- Amazon FreeRTOS 库
- 平台支持库
- 硬件驱动程序

您可以下载具有预定义配置的程序包，或者通过选择硬件平台和应用程序所需的库来创建自己的配置。这些配置保存在 AWS 中，可供随时下载。

Amazon FreeRTOS 控制台是 AWS IoT 控制台的一部分。您可以通过选择以上链接或者浏览到 AWS IoT 控制台来找到它。

打开 Amazon FreeRTOS 控制台

1. 浏览到 AWS IoT 控制台。
2. 从导航窗格中选择软件。
3. 在 Amazon FreeRTOS 设备软件下，选择配置下载。

有关更多信息，请参阅 [Amazon FreeRTOS 控制台](#)。

下载 Amazon FreeRTOS 源代码

您可以从 [Amazon FreeRTOS 控制台](#) 或 [GitHub](#) 下载 RTOS 内核和软件库。

无线更新

连接到 Internet 的设备可能会长时间使用，必须定期更新来修复错误和改进功能。通常这些设备必须在现场更新，并且必须远程或者“无线”更新。Amazon FreeRTOS 无线 (OTA) 更新服务让您可以：

- 在部署前对固件进行数字签名。
- 安全地部署新固件映像到单个设备、一组设备或者整个队组。
- 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。
- 部署到设备之后，验证新固件的真实性和完整性。
- 监控部署进度。
- 调试失败的部署。

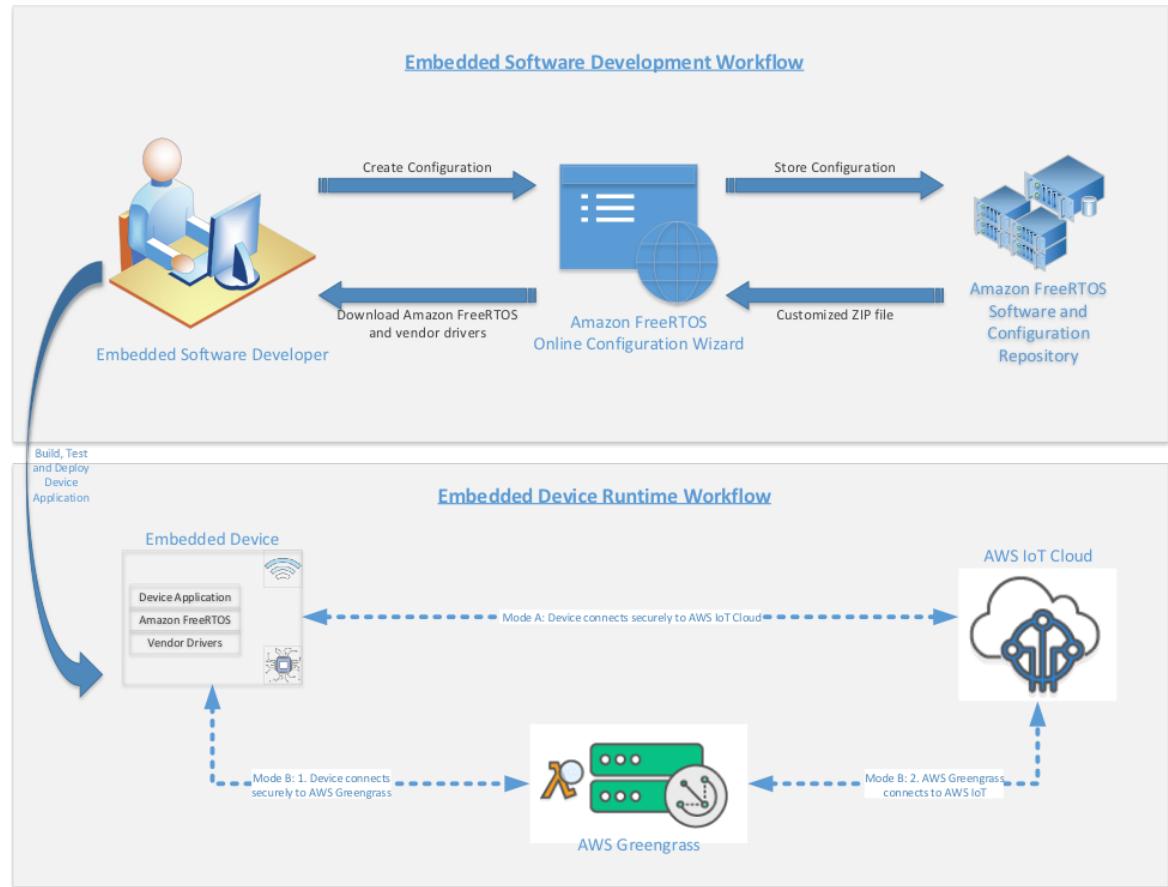
在通过无线发送文件时，最佳实践是对文件进行数字签名，以便接收文件的设备可以验证文件在传输途中未经篡改。可以使用 Code Signing for AWS IoT 来签署和加密文件，也可以使用自己的代码签名工具来签署文件。有关 Code Signing for AWS IoT 的更多信息，请参阅 [Code Signing for AWS IoT 开发人员指南](#)。

有关 OTA 更新的更多信息，请参阅：

- [Amazon FreeRTOS 无线更新 \(p. 118\)](#)
- [OTA 演示应用程序 \(p. 178\)](#)

开发工作流程

您可通过下载 Amazon 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。部署的设备可以连接到 AWS IoT 服务或 AWS IoT Greengrass，作为完善的 IoT 解决方案的一部分。下图显示了基于 Amazon FreeRTOS 的设备上的开发工作流程以及后续的连接。



有关使用 Amazon FreeRTOS 开发应用程序的更多信息，请参阅 [Amazon FreeRTOS 开发人员指南](#)。

其他资源

如果您对 AWS 或 Amazon FreeRTOS 有其他疑问，可能会发现以下资源很有用。

资源	描述
GitHub 上的 Amazon FreeRTOS	如果您要向 Amazon FreeRTOS 工程团队询问有关 Amazon FreeRTOS 的问题，可以在 Amazon FreeRTOS GitHub 页面上提出问题。
AWS 论坛	要与 AWS 社区讨论有关 AWS 和 Amazon FreeRTOS 的技术问题，请访问开发论坛。
AWS 支持中心	要获取 AWS 的技术支持，请访问支持中心。
联系我们	要就 AWS 账单、账户服务、事件、滥用和其他 AWS 相关问题联系我们，请访问“联系我们”页面。

Amazon FreeRTOS 入门

本节介绍如何下载和配置 Amazon FreeRTOS，以及在符合条件的微控制器主板之一上运行演示应用程序。在本教程中，我们假定您熟悉 AWS IoT 和 AWS IoT 控制台。如果您不熟悉，建议您从 [AWS IoT 入门](#) 教程开始。

主题

- [先决条件 \(p. 4\)](#)
- [开始使用 Texas Instruments CC3220SF-LAUNCHXL \(p. 8\)](#)
- [开始使用 STMicroelectronics STM32L4 Discovery Kit IoT Node \(p. 13\)](#)
- [开始使用 NXP LPC54018 IoT Module \(p. 17\)](#)
- [Microchip Curiosity PIC32MZEF 入门 \(p. 21\)](#)
- [Espressif ESP32-DevKitC 和 ESP-WROVER-KIT 入门 \(p. 26\)](#)
- [开始使用 Infineon XMC4800 IoT Connectivity Kit \(p. 34\)](#)
- [Xilinx Avnet MicroZed 工业 IoT 工具包入门 \(p. 38\)](#)
- [Renesas Starter Kit+ for RX65N-2MB 入门 \(p. 51\)](#)
- [MediaTek MT7697Hx 开发工具包入门 \(p. 57\)](#)
- [开始使用 FreeRTOS Windows 仿真器 \(p. 60\)](#)
- [Nordic nRF52840-DK 入门 \(p. 63\)](#)

先决条件

要按本教程进行操作，您需要一个 AWS 账户、一个有权访问 AWS IoT 和 Amazon FreeRTOS 的 IAM 用户，以及一个受支持的硬件平台。

AWS 账户和先决条件

要创建 AWS 账户，请参阅[创建和激活 AWS 账户](#)。

要将 IAM 用户添加到您的 AWS 账户，请参阅[IAM User Guide](#)。要授予您的 IAM 用户账户访问 AWS IoT 和 Amazon FreeRTOS 的权限，请将以下 IAM 策略附加到您的 IAM 用户账户：

- [AmazonFreeRTOSFullAccess](#)
- [AWSIoTFullAccess](#)

将 AmazonFreeRTOSFullAccess 策略附加到您的 IAM 用户

1. 浏览到 [IAM 控制台](#)，在导航窗格中，选择用户。
2. 在搜索文本框中输入您的用户名，然后从列表中选择该名称。
3. 选择 Add permissions。
4. 选择直接附加现有策略。

5. 在搜索框中，输入 **AmazonFreeRTOSFullAccess**，从列表中选择，然后选择下一步：审核。
6. 选择 Add permissions。

将 AWSIoTFullAccess 策略附加到您的 IAM 用户

1. 浏览到 [IAM 控制台](#)，在导航窗格中，选择用户。
2. 在搜索文本框中输入您的用户名，然后从列表中选择该名称。
3. 选择 Add permissions。
4. 选择直接附加现有策略。
5. 在搜索框中，输入 **AWSIoTFullAccess**，从列表中选择，然后选择下一步：审核。
6. 选择 Add permissions。

有关 IAM 和用户账户的更多信息，请参阅 [IAM User Guide](#)。

有关策略的更多信息，请参阅 [IAM 权限和策略](#)。

Amazon FreeRTOS 支持的硬件平台

您需要一块受支持的 MCU 主板：

- STMicroelectronics STM32L4 发现工具包 IoT 节点
- Texas Instruments CC3220SF-LAUNCHXL
- NXP LPC54018 IoT Module
- Microchip Curiosity PIC32MZEF Bundle
- Espressif ESP32-DevKitC
- Espressif ESP-WROVER-KIT
- Infineon XMC4800 IoT 连接工具包
- Xilinx Avnet MicroZed 工业 IoT 工具包
- MediaTek MT7697Hx 开发工具包
- Renesas RX65N RSK IoT Module
- Microsoft Windows 7 或更高版本，至少双核处理器以及有线以太网连接
- Nordic nRF52840-DK [BETA]

将您的 MCU 主板注册到 AWS IoT

您的主板必须已向 AWS IoT 注册才能与 AWS 云通信。

如果您刚开始使用，可以在 [Amazon FreeRTOS 控制台](#) 中使用快速连接工作流程来向 AWS IoT 快速注册您的主板。

要手动注册主板，您必须创建：

- AWS IoT 策略。
AWS IoT 策略向您的设备授予访问 AWS IoT 资源的权限。
- IoT 事物。

通过 IoT 事物，您可以在 AWS IoT 中管理设备。

- 私有密钥和 X.509 证书。

通过私有密钥和证书，您可以用 AWS IoT 对设备进行身份验证。

创建 AWS IoT 策略

1. 要创建 IAM 策略，您需要知道自己的 AWS 区域和 AWS 账号。

要查找 AWS 账号，请在 AWS 管理控制台的右上角，选择 My Account (我的账户)。您的账户 ID 显示在 Account Settings (账户设置) 下。

要查找您的 AWS 账户的 AWS 区域，请使用 AWS Command Line Interface。要安装 AWS CLI，请按照 [AWS Command Line Interface 用户指南](#) 中的说明操作。在安装 AWS CLI 后，打开命令提示符窗口并输入以下命令：

```
aws iot describe-endpoint
```

输出应该如下所示：

```
{  
    "endpointAddress": "xxxxxxxxxxxxxx.iot.us-west-2.amazonaws.com"  
}
```

在此示例中，区域为 us-west-2。

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。将 `aws-region` 和 `aws-account` 替换为您的区域和账户 ID。

```
{  
    "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>:*"  
        }  
    ]  
}
```

此策略授以下权限：

`iot:Connect`

授予设备连接到 AWS IoT 消息代理的权限。

`iot:Publish`

授予设备在 `freertos/demos/echo` MQTT 主题上发布 MQTT 消息的权限。

`iot:Subscribe`

授予设备订阅到 `freertos/demos/echo` MQTT 主题筛选条件的权限。

`iot:Receive`

授予设备从 AWS IoT 消息代理接收消息的权限。

6. 选择 Create。

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 Create。
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。记录证书 ID。稍后您在向证书附加策略时需要它。
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. 选择附加策略以将策略附加到证书，授予您的设备访问 AWS IoT 操作的权限。
10. 选择您刚刚创建的策略，然后选择注册事物。

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	无
停止	1 位
流控制	无

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/ttys*
```

macOS

```
ls /dev/cu.*
```

开始使用 Texas Instruments CC3220SF-LAUNCHXL

在开始之前，请参阅[先决条件 \(p. 4\)](#)。

如果您没有 Texas Instruments (TI) CC3220SF-LAUNCHXL Development Kit，请访问 AWS 合作伙伴设备目录来从我们的[合作伙伴](#)购买一个。

设置环境

Amazon FreeRTOS 支持 TI CC3220SF-LAUNCHXL 开发工具包的两个 IDE：Code Composer Studio 和 IAR Embedded Workbench。

有关安装 Code Composer Studio 的信息，请参阅[安装 Code Composer Studio \(p. 8\)](#)。

有关安装 IAR Embedded Workbench 的信息，请参阅[安装 IAR Embedded Workbench \(p. 8\)](#)。

您还需要[安装 SimpleLink CC3220 开发工具包 \(p. 9\)](#)、[安装 Uniflash \(p. 9\)](#)、[配置 Wi-Fi 预配置 \(p. 9\)](#)和[安装最新的 Service Pack \(p. 9\)](#)。

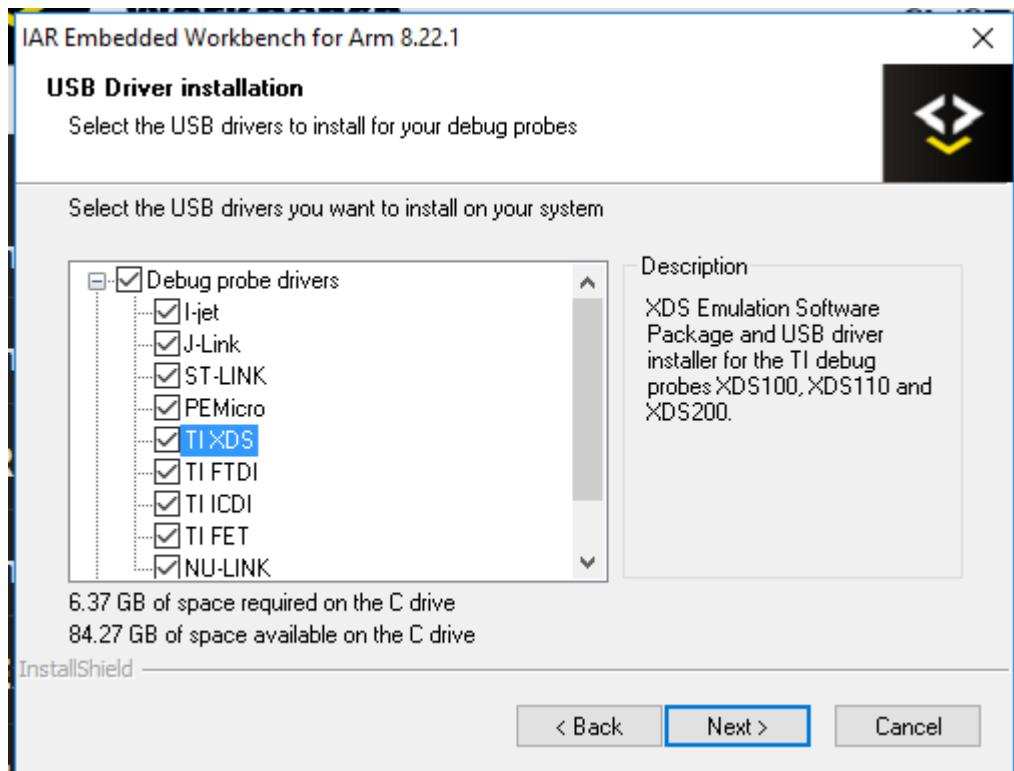
安装 Code Composer Studio

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 7.3.0 版脱机安装程序。
3. 解压缩并运行脱机安装程序。按照提示操作。
4. 对于 Product Families to Install (要安装的产品系列)，请选择 SimpleLink Wi-Fi CC32xx Wireless MCUs (SimpleLink Wi-Fi CC32xx 无线 MCU)。
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 Development Tools Support](#)、[Code Composer Studio 常见问题](#)和[Code Composer Studio 故障排除](#)。

安装 IAR Embedded Workbench

1. 浏览到[IAR Embedded Workbench for ARM](#)。
2. 下载并运行 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：



- 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 CC3220 开发工具包

安装 [SimpleLink CC3220 开发工具包](#)。SimpleLink Wi-Fi CC3200 开发工具包包含适用于 CC3220SF 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 是一个独立的工具，用于对 TI MCU 上的板载闪存进行编程。Uniflash 具有一个 GUI、命令行和脚本编写界面。

配置 Wi-Fi 预配置

要为您的主板配置 Wi-Fi 设置，请执行以下操作之一：

- 配置[配置项目 \(p. 10\)](#)中所述的 Amazon FreeRTOS 演示应用程序。
- 使用来自 Texas Instruments 的 [SmartConfig](#)。

安装最新的 Service Pack

- 在您的 TI CC3220SF-LAUNCHXL 上，将 SOP 跳线放在中间的一组针脚（位置 = 1）上并重置主板。
- 启动 Uniflash，然后从配置列表中选择 CC3220SF-LAUNCHXL。选择 Start Image Creator (启动映像创建器)。
- 选择 New Project (新项目)。

4. 在 Start new project (启动新项目) 页面上，输入项目名称。对于 Device Type (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 Browse (浏览)，然后导航至您安装 CC3220SF SimpleLink 开发工具包的位置。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。

下载并配置 Amazon FreeRTOS

设置环境后，您可以下载 Amazon FreeRTOS。

下载 Amazon FreeRTOS

1. 转到 [Amazon FreeRTOS 控制台](#)。
2. 在 Predefined configurations (预定义配置) 下，找到 Connect to AWS IoT- TI (连接到 AWS IoT - TI)，然后：

如果您在使用 Code Composer Studio，则选择下载。

如果您在使用 IAR Embedded Workbench，则选择连接到 AWS IoT - TI。在硬件平台下，选择编辑。在集成开发环境 (IDE) 下，选择 IAR Embedded Workbench。确保编译器设置为 IAR。保留其他配置选项的默认值，然后选择 Create and Download (创建并下载)。

3. 将下载的文件解压缩到硬盘驱动器。在解压缩后，您会得到一个名为 AmazonFreeRTOS 的目录。

Note

Microsoft Windows 上的文件路径最大长度为 260 个字符。Amazon FreeRTOS 下载中的最长路径为 122 个字符。为了适应 Amazon FreeRTOS 项目中的文件，请确保 AmazonFreeRTOS 目录的路径长度少于 98 个字符。例如，`C:\Users\Username\Dev\AmazonFreeRTOS` 可以正常工作，但 `C:\Users\Username\Documents\Development\Projects\AmazonFreeRTOS` 会导致生成失败。

在本教程中，目录的路径称为 AmazonFreeRTOS。

配置项目

要运行演示，您必须将项目配置为使用 AWS IoT (需要您将主板注册为 AWS IoT 事物)。将您的 MCU 主板注册到 AWS IoT (p. 5) 是先决条件 (p. 4) 中的一个步骤。

配置 AWS IoT 终端节点

1. 浏览至 [AWS IoT 控制台](#)。
2. 在导航窗格中，选择 Settings。

您的 AWS IoT 终端节点将显示在终端节点中。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。

您的设备应具有 AWS IoT 事物名称。记下此名称。

4. 在您的 IDE 中，打开 `<BASE_FOLDER>\demos\common\include\aws_clientcredential.h` 并为以下 `#define` 常量指定值：

- `clientcredentialMQTT_BROKER_ENDPOINT ## AWS_IOT #####`
- `clientcredentialIOT_THING_NAME ### AWS_IOT #####`

配置 Wi-Fi 设置

1. 打开 `aws_clientcredential.h` 文件。
2. 为以下 `#define` 常量指定值：

- `clientcredentialWIFI_SSID Wi-Fi ### SSID`
- `clientcredentialWIFI_PASSWORD Wi-Fi #####`
- `clientcredentialWIFI_SECURITY Wi-Fi #####`

有效安全类型如下：

- `eWiFiSecurityOpen` (开放，不安全)
- `eWiFiSecurityWEP` (WEP 安全性)
- `eWiFiSecurityWPA` (WPA 安全性)
- `eWiFiSecurityWPA2` (WPA2 安全性)

配置您的 AWS IoT 凭证

Note

要配置 AWS IoT 凭证，您需要您在注册设备时从 AWS IoT 控制台下载的私有密钥和证书。在将设备注册为 AWS IoT 事物之后，可从 AWS IoT 控制台检索设备证书，但无法检索私有密钥。

Amazon FreeRTOS 是 C 语言项目，证书和私有密钥必须进行特别的格式设置才能添加到该项目中。您必须设置您的设备的证书和私有密钥的格式。

1. 在浏览器窗口中，打开 `<BASE_FOLDER>\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 `<BASE_FOLDER>\demos\common\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。

生成并运行 Amazon FreeRTOS 演示项目

在 TI Code Composer 中生成并运行 Amazon FreeRTOS 演示项目

将 Amazon 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 `<BASE_FOLDER>\demos\ti\cc3220_launchpad\ccs`。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 Project (项目) 中，选择 Build Project (生成项目) 以确保项目成功编译，没有错误或警报。

订阅 MQTT 主题

Note

运行 Amazon FreeRTOS 示例之前，请执行以下操作：

1. 确保 Texas Instruments CC3220SF-LAUNCHXL 上的 Sense On Power (SOP) 跳线处于位置 0。有关更多信息，请参阅 [CC3220 SimpleLink 用户指南](#)。
2. 使用 USB 电缆将 Texas Instruments CC3220SF-LAUNCHXL 连接到您的计算机。
3. 登录 [AWS IoT 控制台](#)。
4. 在导航窗格中，选择测试以打开 MQTT 客户端。
5. 在订阅主题中，输入 `freertos/demos/echo`，然后选择订阅主题。

在 TI Code Composer 中运行 Amazon FreeRTOS 演示

1. 重新生成您的项目。
2. 在 TI Code Composer 中，从 Run (运行)，选择 Debug (调试)。
3. 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 AWS IoT 控制台的 MQTT 客户端中，您应看到设备发送的 MQTT 消息。

在 IAR Embedded Workbench 中生成并运行 Amazon FreeRTOS 演示项目

将 Amazon 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 `<BASE_FOLDER>\demos\ti\cc3220_launchpad\iar`，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)。

订阅 MQTT 主题

1. 确保 Texas Instruments CC3220SF-LAUNCHXL 上的 Sense On Power (SOP) 跳线处于位置 0。有关更多信息，请参阅 [CC3220 SimpleLink 用户指南](#)。

2. 使用 USB 电缆将 Texas Instruments CC3220SF-LAUNCHXL 连接到您的计算机。
3. 登录 [AWS IoT 控制台](#)。
4. 在导航窗格中，选择测试以打开 MQTT 客户端。
5. 在订阅主题中，输入 `freertos/demos/echo`，然后选择订阅主题。

在 IAR Embedded Workbench 中运行 Amazon FreeRTOS 演示

1. 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)。

2. 从 Project (项目) 菜单，选择 Download and Debug (下载并调试)。如果显示“Warning: Failed to initialize EnergyTrace (警告: 无法初始化 EnergyTrace)”，可以忽略该警告。有关 EnergyTrace 的更多信息，请参阅 [MSP EnergyTrace Technology](#)。
3. 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)。

在 AWS IoT 控制台的 MQTT 客户端中，您应看到设备发送的 MQTT 消息。

问题排查

如果您在 AWS IoT 控制台的 MQTT 客户端中未看到消息，则可能需要为主板配置调试设置。

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。

开始使用 STMicroelectronics STM32L4 Discovery Kit IoT Node

在开始之前，请参阅[先决条件 \(p. 4\)](#)。

如果您还没有 STMicroelectronics STM32L4 Discovery Kit IoT Node，请访问 AWS 合作伙伴设备目录来从我们的[合作伙伴](#)购买一个。

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 kit IoT node](#)、[低能耗无线](#)、[BLE](#)、[NFC](#)、[SubGHz](#)、[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))。

设置环境

为 STM32 安装 System Workbench

1. 浏览到 [OpenSTM32.org](#)。

2. 在 OpenSTM32 网页上注册。您需要登录以下载 System Workbench。
3. 浏览到 [System Workbench for STM32 installer](#) 以下载并安装 System Workbench。

如果您在安装期间遇到问题，请参阅 [System Workbench 网站](#) 上的“常见问题”。

下载并配置 Amazon FreeRTOS

设置环境后，您可以下载 Amazon FreeRTOS。

下载 Amazon FreeRTOS

1. 在 AWS IoT 控制台中，浏览到 [Amazon FreeRTOS 页面](#)。
2. 在导航窗格中，选择 Software (软件)。
3. 在 Amazon FreeRTOS 设备软件下，选择配置下载。
4. 选择下载 FreeRTOS 软件。
5. 在软件配置下，找到连接到 AWS IoT - ST，然后选择下载。
6. 将下载的文件解压缩到 AmazonFreeRTOS 文件夹，然后记录文件夹的路径。

Note

Microsoft Windows 上的文件路径最大长度为 260 个字符。Amazon FreeRTOS 下载中的最长路径为 122 个字符。为了适应 Amazon FreeRTOS 项目中的文件，请确保 AmazonFreeRTOS 目录的路径长度少于 98 个字符。例如，C:\Users\Username\Dev\AmazonFreeRTOS 可以正常工作，但 C:\Users\Username\Documents\Development\Projects\AmazonFreeRTOS 会导致生成失败。

在本教程中，目录的路径称为 AmazonFreeRTOS。

配置项目

要运行演示，您必须将项目配置为使用 AWS IoT（需要您将主板注册为 AWS IoT 事物）。[将您的 MCU 主板注册到 AWS IoT \(p. 5\)](#) 是先决条件 (p. 4) 中的一个步骤。

配置 AWS IoT 终端节点

1. 浏览至 [AWS IoT 控制台](#)。
2. 在导航窗格中，选择 Settings。

您的 AWS IoT 终端节点将显示在终端节点中。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。

您的设备应具有 AWS IoT 事物名称。记下此名称。

4. 在您的 IDE 中，打开 <BASE_FOLDER>\demos\common\include\aws_clientcredential.h 并为以下 #define 常量指定值：
 - clientcredentialMQTT_BROKER_ENDPOINT ## AWS_IOT #####
 - clientcredentialIOT_THING_NAME ### AWS_IOT #####

配置 Wi-Fi 设置

1. 打开 aws_clientcredential.h 文件。
2. 为以下 #define 常量指定值：

- clientcredentialWIFI_SSID *Wi-Fi* ### *SSID*
- clientcredentialWIFI_PASSWORD *Wi-Fi* #####
- clientcredentialWIFI_SECURITY *Wi-Fi* #####

有效安全类型如下：

- eWiFiSecurityOpen (开放，不安全)
- eWiFiSecurityWEP (WEP 安全性)
- eWiFiSecurityWPA (WPA 安全性)
- eWiFiSecurityWPA2 (WPA2 安全性)

配置您的 AWS IoT 凭证

Note

要配置 AWS IoT 凭证，您需要您在注册设备时从 AWS IoT 控制台下载的私有密钥和证书。在将设备注册为 AWS IoT 事物之后，可从 AWS IoT 控制台检索设备证书，但无法检索私有密钥。

Amazon FreeRTOS 是 C 语言项目，证书和私有密钥必须进行特别的格式设置才能添加到该项目中。您必须设置您的设备的证书和私有密钥的格式。

1. 在浏览器窗口中，打开 *<BASE_FOLDER>\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 *<BASE_FOLDER>\demos\common\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。

生成并运行 Amazon FreeRTOS 演示项目

将 Amazon FreeRTOS 演示导入 STM32 System Workbench

1. 打开 STM32 System Workbench，然后输入新工作区的名称。
2. 从 File (文件) 菜单，选择 Import (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 *<BASE_FOLDER>\demos\st\stm321475_discovery\ac6*。
4. 默认情况下应选中项目 aws_demos。
5. 选择 Finish (完成) 以将项目导入 STM32 System Workbench。
6. 从 Project (项目) 菜单，选择 Build All (全部生成)。确认项目成功编译，没有任何错误或警告。

运行 Amazon FreeRTOS 演示项目

1. 使用 USB 线缆将您的 STMicroelectronics STM32L4 Discovery Kit IoT Node 连接到计算机。

2. 重新生成您的项目。
3. 登录 [AWS IoT 控制台](#)。
4. 在导航窗格中，选择测试以打开 MQTT 客户端。
5. 在订阅主题中，输入 `freertos/demos/echo`，然后选择订阅主题。
6. 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 Ac6 STM32 C/C++ Application (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 Run (运行) 菜单，选择 Debug Configurations (调试配置)。
2. 选择 `aws_demos` Debug (aws_demos 调试)。（您可能需要展开 Ac6 STM32 Debugging (Ac6 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。
7. 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 控制台的 MQTT 客户端中，您应看到设备发送的 MQTT 消息。

运行低功耗蓝牙演示

Amazon FreeRTOS 对低功耗蓝牙功能的支持为公开测试版。BLE 演示可能会发生变化。

Note

要运行 BLE 演示，您需要 STM32L475 发现工具包的 SPBTLE-1S BLE 模块。

Amazon FreeRTOS 支持[低功耗蓝牙 \(BLE\)](#)连接。您可以使用 BLE 从 [GitHub](#) 下载 Amazon FreeRTOS。Amazon FreeRTOS BLE 库仍为公共测试版，因此，您需要切换分支以访问您主板的代码。查看名为 `feature/ble-beta` 的分支。

要跨 BLE 运行 Amazon FreeRTOS 演示项目，您需要在 iOS 或 Android 移动设备上运行 Amazon FreeRTOS BLE 移动开发工具包演示应用程序。

设置 Amazon FreeRTOS BLE 移动开发工具包演示应用程序

1. 按照[适用于 Amazon FreeRTOS 蓝牙设备的移动开发工具包](#)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[Amazon FreeRTOS BLE 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 MQTT over BLE 演示的说明，请参阅 [MQTT over BLE 演示应用程序](#)。

问题排查

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: xxxxxxxxxxxxxxxx
[Tmr Svc] [WARN] WiFi firmware needs to be updated.
```

要下载最新的 Wi-Fi 固件，请参阅 [STM32L4 Discovery kit IoT node、低能耗无线、BLE、NFC、SubGHz、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。

开始使用 NXP LPC54018 IoT Module

在开始之前，请参阅[先决条件 \(p. 4\)](#)。

如果您没有 NXP LPC54018 IoT Module，请访问 AWS 合作伙伴设备目录来从我们的[合作伙伴](#)购买一个。使用 USB 电缆将您的 NXP LPC54018 IoT Module 连接到您的计算机。

设置环境

Amazon FreeRTOS 支持 NXP LPC54018 IoT Module 的两个 IDE：IAR Embedded Workbench 和 MCUXpresso。

在开始之前，请安装其中一个 IDE。

安装 IAR Embedded Workbench for ARM

1. 浏览到 [Software for NXP Kits \(适用于 NXP Kits 的软件\)](#)，然后选择 Download Software (下载软件)。

Note

IAR Embedded Workbench for ARM 需要 Microsoft Windows。

2. 解压缩并运行安装程序。按照提示操作。
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)。

从 NXP 安装 MCUXpresso

1. 从 [NXP](#) 下载并运行 MCUXpresso 安装程序。
2. 浏览到 [MCUXpresso SDK \(MCUXpresso 开发工具包\)](#) 并选择 Build your SDK (生成您的开发工具包)。
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 Boards (主板) 下，选择 LPC54018-IoT-Module。
6. 验证硬件详细信息，然后选择 Build MCUXpresso SDK (生成 MCUXpresso 开发工具包)。
7. 使用 MCUXpresso IDE 的适用于 Windows 的开发工具包已生成。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 Installed SDKs (已安装开发工具包) 选项卡。
9. 将下载的开发工具包存档文件拖放到 Installed SDKs (已安装开发工具包) 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](#)。

连接 JTAG 调试器

您需要使用 JTAG 调试器来启动并调试在 NXP LPC54018 主板上运行的代码。Amazon FreeRTOS 已使用 Segger J-Link 探测器进行了测试。有关支持的调试器的更多信息，请参阅 [NXP LPC54018 用户指南](#)。

Note

如果您使用的是 Segger J-Link 调试器，则需要使用转换电缆将 20 针连接器从调试器连接到 NXP IoT Module 上的 10 针连接器。

下载并配置 Amazon FreeRTOS

设置环境后，您可以下载 Amazon FreeRTOS。

下载 Amazon FreeRTOS

1. 浏览到 AWS IoT 控制台中的 [Amazon FreeRTOS 页面](#)。
2. 在导航窗格中，选择 Software (软件)。
3. 在 Amazon FreeRTOS 设备软件下，选择配置下载。
4. 选择下载 FreeRTOS 软件。
5. 在软件配置下，找到连接到 AWS IoT - NXP，然后：

如果您使用的是 IAR Workbench，请选择下载。

如果使用的是 MCUXpresso：

- a. 在软件配置中，找到连接到 AWS IoT - NXP。选择连接到 AWS IoT - NXP，但不选择下载。
 - b. 在硬件平台下，选择编辑。
 - c. 在集成开发环境 (IDE) 下，选择 MCUXpresso。
 - d. 在编译器下，选择 GCC。
 - e. 在页面底部，选择创建和下载。
6. 将下载的文件解压缩到 AmazonFreeRTOS 文件夹，然后记录文件夹的路径。

Note

Microsoft Windows 上的文件路径最大长度为 260 个字符。Amazon FreeRTOS 下载中的最长路径为 122 个字符。为了适应 Amazon FreeRTOS 项目中的文件，请确保 AmazonFreeRTOS 目录的路径长度少于 98 个字符。例如，C:\Users\Username\Dev\AmazonFreeRTOS 可以正常工作，但 C:\Users\Username\Documents\Development\Projects\AmazonFreeRTOS 会导致生成失败。

在本教程中，目录的路径称为 AmazonFreeRTOS。

配置项目

要运行演示，您必须将项目配置为使用 AWS IoT（需要您将主板注册为 AWS IoT 事物）。[将您的 MCU 主板注册到 AWS IoT \(p. 5\)](#) 是先决条件 (p. 4) 中的一个步骤。

配置 AWS IoT 终端节点

1. 浏览至 [AWS IoT 控制台](#)。
2. 在导航窗格中，选择 Settings。

您的 AWS IoT 终端节点将显示在终端节点中。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。

您的设备应具有 AWS IoT 事物名称。记下此名称。

4. 在您的 IDE 中，打开 <BASE_FOLDER>\demos\common\include\aws_clientcredential.h 并为以下 #define 常量指定值：

- clientcredentialMQTT_BROKER_ENDPOINT ## AWS IoT #####
- clientcredentialIOT_THING_NAME #### AWS IoT #####

配置 Wi-Fi 设置

1. 打开 aws_clientcredential.h 文件。
2. 为以下 #define 常量指定值：

- clientcredentialWIFI_SSID *Wi-Fi* ### SSID
- clientcredentialWIFI_PASSWORD *Wi-Fi* #####
- clientcredentialWIFI_SECURITY *Wi-Fi* #####

有效安全类型如下：

- eWiFiSecurityOpen (开放，不安全)
- eWiFiSecurityWEP (WEP 安全性)
- eWiFiSecurityWPA (WPA 安全性)
- eWiFiSecurityWPA2 (WPA2 安全性)

配置您的 AWS IoT 凭证

Note

要配置 AWS IoT 凭证，您需要您在注册设备时从 AWS IoT 控制台下载的私有密钥和证书。在将设备注册为 AWS IoT 事物之后，可从 AWS IoT 控制台检索设备证书，但无法检索私有密钥。

Amazon FreeRTOS 是 C 语言项目，证书和私有密钥必须进行特别的格式设置才能添加到该项目中。您必须设置您的设备的证书和私有密钥的格式。

1. 在浏览器窗口中，打开 *<BASE_FOLDER>\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 *<BASE_FOLDER>\demos\common\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。

生成并运行 Amazon FreeRTOS 演示项目

将 Amazon FreeRTOS 演示导入 IDE

将 Amazon 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 *<BASE_FOLDER>\demos\nxp\lpc54018_iot_module\iar*，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 Amazon FreeRTOS 示例代码导入 MCUXpresso IDE

1. 打开 MCUXpresso，从 File (文件) 菜单，选择 Open Projects From File System (从文件系统打开项目)。
2. 在 Directory (目录) 文本框中，输入 `<BASE_FOLDER>\demos\nxp\lpc54018_iot_module\mcuxpresso`，然后选择 Finish (完成)
3. 从 Project (项目) 菜单，选择 Build All (全部生成)。

运行 Amazon FreeRTOS 演示项目

要在 NXP LPC54018 IoT Module 主板上运行 Amazon FreeRTOS 演示，请将 NXP IoT Module 上的 USB 端口连接到您的主机，打开终端程序，然后连接到标识作为 USB 串行设备的端口。

1. 重新生成您的项目。
2. 登录 [AWS IoT 控制台](#)。
3. 在导航窗格中，选择测试以打开 MQTT 客户端。
4. 在订阅主题中，输入 `freertos/demos/echo`，然后选择订阅主题。
5. 在您的 IDE 中，从 Project (项目) 菜单，选择 Build (生成)。
6. 使用 mini-USB 到 USB 线缆，将 NXP IoT Module 和 Segger J-Link 调试器连接到计算机上的 USB 端口。
7. 如果您使用的是 IAR Embedded Workbench：
 - a. 从 Project (项目) 菜单，选择 Download and Debug (下载并调试)。
 - b. 从 Debug (调试) 菜单，选择 Start Debugging (启动调试)。
 - c. 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。

8. 如果使用的是 MCUXpresso：
 - a. 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。
 - b. 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  以重置调试会话。(由于 MCUXpresso 调试器的 NXP54018-IoT-Module 错误，必须执行此操作。)

9. 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)。

在 AWS IoT 控制台的 MQTT 客户端中，您应看到设备发送的 MQTT 消息。

问题排查

如果 AWS IoT 控制台中未显示消息，请尝试以下步骤：

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。

Microchip Curiosity PIC32MZEF 入门

在开始之前，请参阅[先决条件 \(p. 4\)](#)。

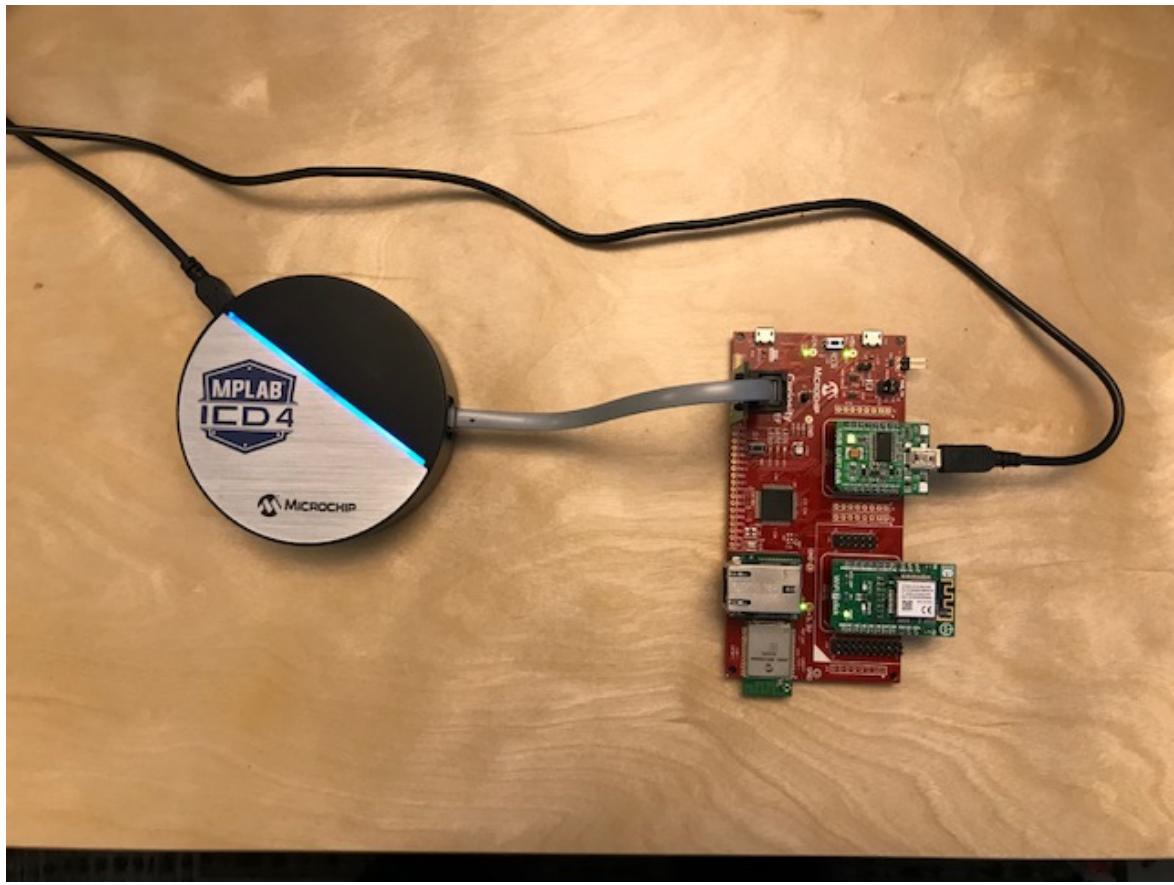
如果您没有 Microchip Curiosity PIC32MZEF bundle，请访问 AWS 合作伙伴设备目录来从我们的[合作伙伴](#)购买一个。您需要以下物品：

- MikroElectronika USB UART Click Board
- RJ-11 到 ICSP 适配器
- MPLAB ICD 4 电路内置调试器
- PIC32 LAN8720 PHY 子板
- MikroElectronika WiFi 7 Click Board

设置 Microchip Curiosity PIC32MZEF 硬件

1. 将 [MikroElectronika USB UART Click Board](#) 连接到 Microchip Curiosity PIC32MZEF 上的 microBUS 1 连接器。
2. 将 [PIC32 LAN8720 PHY 子板](#) 连接到 Microchip Curiosity PIC32MZEF 上的 J18 接头。
3. 使用 USB A 到 USB mini-B 线缆将 [MikroElectronika USB UART Click Board](#) 连接到计算机。
4. 将 [MikroElectronika WiFi 7 Click Board](#) 连接到 Microchip Curiosity PIC32MZEF 上的 microBUS 2 连接器。
5. 将 [RJ-11 到 ICSP 适配器](#) 连接到 Microchip Curiosity PIC32MZEF。
6. 使用 RJ-11 线缆将 MPLAB ICD 4 电路内置调试器连接到 Microchip Curiosity PIC32MZEF。
7. 使用 USB A 到 USB mini-B 线缆将 ICD 4 电路内置调试器连接到计算机。
8. 将 RJ-11 到 ICSP 适配器 J2 插入 Microchip Curiosity PIC32MZEF 上位于 J16 的 ICSP 接头。
9. 将以太网电缆的一端连接到 LAN8720 PHY 子板。将另一端连接到路由器或其他 Internet 端口。

下图显示 Microchip Curiosity PIC32MZEF 以及所有已经组装了的必需的外围设备。



电路内置调试器准备好后，上面的 LED 会变为纯蓝色。

设置环境

1. 安装最新的 [Java SE 开发工具包](#)。
2. 安装 [Python 版本 3.x 或更高版本](#)。
3. 安装 MPLAB X IDE 的最新版本：
 - [MPLAB X Integrated Development Environment for Windows](#)
 - [MPLAB X Integrated Development Environment for macOS](#)
 - [MPLAB X Integrated Development Environment for Linux](#)
4. 安装 MPLAB X XC32 Compiler 的最新版本：
 - [MPLAB XC32/32++ Compiler for Windows](#)
 - [MPLAB XC32/32++ Compiler for macOS](#)
 - [MPLAB XC32/32++ Compiler for Linux](#)
5. 安装 MPLAB Harmony Integrated Software Framework 的最新版本（可选）：
 - [MPLAB Harmony Integrated Software Framework for Windows](#)
 - [MPLAB Harmony Integrated Software Framework for macOS](#)
 - [MPLAB Harmony Integrated Software Framework for Linux](#)
6. 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

下载并配置 Amazon FreeRTOS

设置环境后，您可以下载 Amazon FreeRTOS。

下载 Amazon FreeRTOS

1. 浏览到 AWS IoT 控制台中的 [Amazon FreeRTOS 页面](#)。
2. 在导航窗格中，选择 Software (软件)。
3. 在 Amazon FreeRTOS 设备软件下，选择配置下载。
4. 选择下载 FreeRTOS 软件。
5. 在软件配置下，找到连接到 AWS IoT - Microchip，然后选择下载。
6. 将下载的文件解压缩到 AmazonFreeRTOS 文件夹，然后记录文件夹的路径。

Note

Microsoft Windows 上的文件路径最大长度为 260 个字符。Amazon FreeRTOS 下载中的最长路径为 122 个字符。为了适应 Amazon FreeRTOS 项目中的文件，请确保 AmazonFreeRTOS 目录的路径长度少于 98 个字符。例如，C:\Users\Username\Dev\AmazonFreeRTOS 可以正常工作，但 C:\Users\Username\Documents\Development\Projects\AmazonFreeRTOS 会导致生成失败。

在本教程中，目录的路径称为 AmazonFreeRTOS。

配置项目

要运行演示，您必须将项目配置为使用 AWS IoT（需要您将主板注册为 AWS IoT 事物）。[将您的 MCU 主板注册到 AWS IoT \(p. 5\)](#) 是先决条件 (p. 4) 中的一个步骤。

配置 AWS IoT 终端节点

1. 浏览至 [AWS IoT 控制台](#)。
2. 在导航窗格中，选择 Settings。

您的 AWS IoT 终端节点将显示在终端节点中。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。

您的设备应具有 AWS IoT 事物名称。记下此名称。

4. 在您的 IDE 中，打开 <BASE_FOLDER>\demos\common\include\aws_clientcredential.h 并为以下 #define 常量指定值：
 - clientcredentialMQTT_BROKER_ENDPOINT ## AWS_IOT #####
 - clientcredentialIOT_THING_NAME ### AWS_IOT #####

配置 Wi-Fi 设置

1. 打开 aws_clientcredential.h 文件。

2. 为以下 #define 常量指定值：

- clientcredentialWIFI_SSID *Wi-Fi* ### *SSID*
- clientcredentialWIFI_PASSWORD *Wi-Fi* #####
- clientcredentialWIFI_SECURITY *Wi-Fi* #####

有效安全类型如下：

- eWiFiSecurityOpen (开放，不安全)
- eWiFiSecurityWEP (WEP 安全性)
- eWiFiSecurityWPA (WPA 安全性)
- eWiFiSecurityWPA2 (WPA2 安全性)

配置您的 AWS IoT 凭证

Note

要配置 AWS IoT 凭证，您需要您在注册设备时从 AWS IoT 控制台下载的私有密钥和证书。在将设备注册为 AWS IoT 事物之后，可从 AWS IoT 控制台检索设备证书，但无法检索私有密钥。

Amazon FreeRTOS 是 C 语言项目，证书和私有密钥必须进行特别的格式设置才能添加到该项目中。您必须设置您的设备的证书和私有密钥的格式。

1. 在浏览器窗口中，打开 *<BASE_FOLDER>\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 *<BASE_FOLDER>\demos\common\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。

生成并运行 Amazon FreeRTOS 演示项目

在 MPLAB IDE 中打开 Amazon FreeRTOS 演示

1. 在 MPLAB IDE 中，从 File (文件) 菜单选择 Open Project (打开项目)。
2. 浏览到并打开 *<BASE_FOLDER>\demos\microchip\curiosity_pic32mzef\mplab*。
3. 选择 Open project (打开项目)。

Note

首次打开项目时，您可以忽略如下所示的警告消息：

```
warning: Configuration "pic32mz_ef_curiosity" builds with "XC32", but indicates no
toolchain directory.
warning: Configuration "pic32mz_ef_curiosity" refers to file "AmazonFreeRTOS/lib/
third_party/mcu_vendor/microchip/harmony/framework/bootloader/src/bootloader.h"
which does not exist in the disk. The make process might not build correctly.
```

运行 Amazon FreeRTOS 演示项目

1. 重新生成您的项目。
2. 登录 [AWS IoT 控制台](#)。
3. 在导航窗格中，选择测试以打开 MQTT 客户端。
4. 在订阅主题中，输入 `freertos/demos/echo`，然后选择订阅主题。
5. 在项目选项卡上，右键单击 `aws_demos` 顶级文件夹，然后选择调试。
6. 首次调试示例时，将显示未找到 ICD 4 对话框。在树视图中的 ICD 4 节点下，选择 ICD4 序列号，然后单击确定。
7. 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)。

ICD 4 在编程设备过程中将转变为半黄色，在运行时将转变为半绿色。IDE 中显示 ICD4 选项卡。成功的编程应如下所示：

```
*****
Connecting to MPLAB ICD 4...

Currently loaded versions:
Application version.....01.02.00
Boot version.....01.00.00
FPGA version.....01.00.00
Script version.....00.02.18
Script build number.....fd44437f19
Application build number.....0123456789

Connecting to MPLAB ICD 4...

Currently loaded versions:
Boot version.....01.00.00
Updating firmware application...
Connecting to MPLAB ICD 4...

Currently loaded versions:
Application version.....01.02.16
Boot version.....01.00.00
FPGA version.....01.00.00
Script version.....00.02.18
Script build number.....fd44437f19
Application build number.....0123456789

Target voltage detected
Target device PIC32MZ2048EFM100 found.
Device Id Revision = 0xA1
Serial Number:
Num0 = ec4f6d3c
Num1 = 6b845410

Erasing...

The following memory area(s) will be programmed:
program memory: start address = 0x1d000000, end address = 0x1d07bfff
program memory: start address = 0x1d1fc000, end address = 0x1d1fffff
configuration memory
boot config memory

Programming/Verify complete
```

Running

Note

建议您使用 MPLAB 电路内置调试器而不是 USB 端口进行调试。ICD 4 让您可以更快地逐步调试代码和添加断点，而无需重新启动调试器。

在 AWS IoT 控制台的 MQTT 客户端中，您应看到设备发送的 MQTT 消息。

问题排查

如果 AWS IoT 控制台中未显示消息，请尝试以下步骤：

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。

Espressif ESP32-DevKitC 和 ESP-WROVER-KIT 入门

如果您没有 Espressif ESP32-DevKitC，请访问 AWS 合作伙伴设备目录来从我们的[合作伙伴](#)购买一个。
如果您没有 Espressif ESP32-WROVER-KIT，请访问 AWS 合作伙伴设备目录来从我们的[合作伙伴](#)购买一个。Amazon FreeRTOS 支持这两种设备。要检查您已有的开发模块，请参阅 [ESP32 模块和主板](#)。

Note

目前，ESP32-WROVER-KIT 和 ESP DevKitC 的 Amazon FreeRTOS 端口不支持以下功能：

- 轻型 IP。
- 对称多处理 (SMP)。

设置 Espressif 硬件

有关设置 ESP32-DevKitC 开发主板硬件的信息，请参阅 [ESP32-DevKitC 入门指南](#)。

有关设置 ESP-WROVER-KIT 开发主板硬件的信息，请参阅 [ESP-WROVER-KIT 入门指南](#)。

Note

请勿继续执行 Espressif 指南的入门部分。改为按以下步骤操作。

设置环境

建立串行连接

要在您的主机和 ESP32-DevKitC 之间建立串行连接，您需要安装 CP210x USB to UART Bridge VCP 驱动程序。您可以从 [Silicon Labs](#) 下载这些驱动程序。

要在您的主机和 ESP32-WROVER-KIT 之间建立串行连接，您需要安装一些 FTDI 虚拟 COM 端口驱动程序。您可以从 [FTDI](#) 下载这些驱动程序。

有关与 Espressif 主板建立串行连接的更多信息，请参阅[使用 ESP32 建立串行连接](#)。建立串行连接后，记下主板连接的串行端口。在构建演示时需要用到它。

设置 Toolchain

您需要设置 Espressif 工具链以与主板通信。要设置工具链，请按照适用于您的主机操作系统的说明操作：

- [适用于 Windows 的工具链标准设置](#)
- [适用于 Mac OS 的工具链标准设置](#)
- [适用于 Linux 的工具链标准设置](#)

Note

在按照工具链设置说明操作时，请勿继续执行 Next Steps (后续步骤) 下的 Get ESP-IDF (获取 ESP-IDF) 说明。改为继续按照此页面上的说明操作。

下载并配置 Amazon FreeRTOS

设置环境后，您可以从 GitHub 下载 Amazon FreeRTOS。不可从 Amazon FreeRTOS 控制台使用 Espressif 主板的 Amazon FreeRTOS 配置。

下载 Amazon FreeRTOS

从 [GitHub](#) 克隆 amazon-freertos 存储库。

在本教程中，amazon-freertos 目录的路径称为 BASE_FOLDER。

Note

Microsoft Windows 上的文件路径最大长度为 260 个字符。Amazon FreeRTOS 下载中的最长路径为 122 个字符。为了适应 Amazon FreeRTOS 项目中的文件，请确保 amazon-freertos 目录的路径长度少于 98 个字符。例如，C:\Users\Username\Dev\amazon-freertos 可以使用，但 C:\Users\Username\Documents\Development\Projects\amazon-freertos 会导致生成失败。

配置项目

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 mingw32.exe。
2. 安装 Python 2.7.10 或更高版本。
3. 如果您运行的是 Windows，请使用 easy_install awscli 在 mingw32 环境中安装 AWS CLI。

如果您运行的是 macOS 或 Linux，请确保 AWS CLI 已安装在您的系统上。有关更多信息，请参阅[安装 AWS 命令行接口](#)。

4. 运行 aws configure 并使用 AWS 访问密钥 ID、私有访问密钥和默认区域名称配置 AWS CLI。有关更多信息，请参阅[配置 AWS CLI](#)。
5. 使用以下命令安装 boto3 Python 模块：
 - 在 Windows 上的 mingw32 环境中，使用 easy_install boto3
 - 在 macOS 或 Linux 上，使用 pip install boto3

Amazon 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 `<BASE_FOLDER>/tools/aws_config_quick_start/configure.json` 并设置以下属性：

afr_source_dir

计算机上的 `amazon-freertos` 目录的完整路径。确保您使用正斜杠来指定此路径。

thing_name

要分配给代表您的主板的 AWS IoT 事物的名称。

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。

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 安全性)

运行配置脚本

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 `mingw32.exe`。
2. 转到 `<BASE_FOLDER>/tools/aws_config_quick_start` 目录 , 然后运行以下命令 :

```
python SetupAWS.py setup
```

此脚本创建一个 IoT 事物、证书和策略。它将 IoT 策略附加到证书 , 将证书附加到 IoT 事物。它还会使用您的 AWS IoT 终端节点 , Wi-Fi SSID 和凭证填充 `aws_clientcredential.h` 文件。最后 , 它会格式化您的证书和私有密钥 , 然后将其写入 `aws_clientcredential.h` 标头文件。有关脚本的更多信息 , 请参阅 `<BASE_FOLDER>/tools/aws_config_quick_start` 目录中的 `README.md`。

生成并运行 Amazon FreeRTOS 演示项目

配置主板的连接以刷入演示

1. 将主板连接到主机。
2. 如果您使用的是 macOS 或 Linux , 请打开终端。如果您使用的是 Windows , 请打开 `mingw32.exe` (从 `mysys` 工具链下载)。
3. 导航到 `<BASE_FOLDER>/demos/espressif/esp32_devkitc_esp_wrover_kit/make` 并输入以下命令以创建并打开“Espressif IoT Development Framework Configuration”(Espressif IoT Development Framework 配置) 菜单 :

```
make menuconfig
```

要确认菜单中的选择 , 请选择 `Select` (选择)。要保存配置 , 请选择 `Save` (保存)。要退出菜单 , 请选择 `Exit` (退出)。

4. 在“Espressif IoT Development Framework Configuration”(Espressif IoT Development Framework 配置) 菜单中 , 导航到 `Serial flasher config` (串行刷入配置)。

选择 Default serial port (默认串行端口) 来配置串行端口。您在此处配置的串行端口用于将演示应用程序写入主板。在 Windows 上，串行端口的名称类似于 COM1。在 macOS 上，它们以 /dev/cu 开头。在 Linux 上，它们以 /dev/tty 开头。

选择 Default baud rate (默认波特率) 来更改在与主板通信时使用的默认波特率。增加波特率可以减少刷入主板所需的时间。根据您的硬件，您可以将默认波特率增加到 921600。

5. 在设置主板的配置后，保存并退出菜单。

要生成和刷入固件（包括启动加载程序和分区表）并监控串行控制台输出，请打开终端（macOS 和 Linux）或 mingw32.exe（Windows）。将目录更改为 `<BASE_FOLDER>/demos/espressif/esp32_devkitc_esp_wrover_kit/make` 并运行以下命令：

```
make flash monitor
```

要监控演示，您可以订阅 MQTT 主题，在其中，演示应用程序发布消息 `freertos/demos/echo`。

订阅 MQTT 主题

1. 登录 AWS IoT 控制台。
2. 在导航窗格中，选择测试以打开 MQTT 客户端。
3. 在订阅主题中，输入 `freertos/demos/echo`，然后选择订阅主题。

当您从其中发出 `make flash monitor` 命令的终端或命令提示符返回类似以下内容的输出时，Hello World `number` 和 Hello World `number` ACK 消息会显示在 MQTT 客户端页面底部：

```
12 1350 [MQTTEcho] Echo successfully published 'Hello World 0'  
13 1357 [Echoing] Message returned with ACK: 'Hello World 0 ACK'  
14 1852 [MQTTEcho] Echo successfully published 'Hello World 1'  
15 1861 [Echoing] Message returned with ACK: 'Hello World 1 ACK'  
16 2355 [MQTTEcho] Echo successfully published 'Hello World 2'  
17 2361 [Echoing] Message returned with ACK: 'Hello World 2 ACK'  
18 2857 [MQTTEcho] Echo successfully published 'Hello World 3'  
19 2863 [Echoing] Message returned with ACK: 'Hello World 3 ACK'
```

当演示运行完时，您应在终端或命令提示符中看到类似以下内容的输出：

```
32 6380 [MQTTEcho] Echo successfully published 'Hello World 10'  
33 6386 [Echoing] Message returned with ACK: 'Hello World 10 ACK'  
34 6882 [MQTTEcho] Echo successfully published 'Hello World 11'  
35 6889 [Echoing] Message returned with ACK: 'Hello World 11 ACK'  
36 7385 [MQTTEcho] MQTT echo demo finished.  
37 7385 [MQTTEcho] ----Demo finished----
```

运行低功耗蓝牙演示

Amazon FreeRTOS 对低功耗蓝牙功能的支持为公开测试版。BLE 演示可能会发生变化。

Amazon FreeRTOS 支持 [低功耗蓝牙 \(BLE\)](#) 连接。您可以使用 BLE 从 [GitHub](#) 下载 Amazon FreeRTOS。Amazon FreeRTOS BLE 库仍为公共测试版，因此，您需要切换分支以访问您主板的代码。查看名为 `feature/ble-beta` 的分支。

要跨 BLE 运行 Amazon FreeRTOS 演示项目，您需要在 iOS 或 Android 移动设备上运行 Amazon FreeRTOS BLE 移动开发工具包演示应用程序。

设置 Amazon FreeRTOS BLE 移动开发工具包演示应用程序

1. 按照适用于 Amazon FreeRTOS 蓝牙设备的移动开发工具包中的说明，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 Amazon FreeRTOS BLE 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 MQTT over BLE 演示的说明，请参阅 [MQTT over BLE 演示应用程序](#)。

有关如何在主板上运行 Wi-Fi 预配置演示的说明，请参阅 [Wi-Fi 预配置演示应用程序](#)。

问题排查

- 如果您使用的是 Mac，它不会识别您的 ESP-WROVER-KIT，请确保您未安装 D2XX 驱动程序。要卸载它们，请按照适用于 Mac OS X 的 FTDI 驱动程序安装指南中的说明操作。
- ESP-IDF 提供的监控实用程序（使用 make monitor 调用）可帮助您解码地址。因此，在应用程序崩溃时，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](#)。
- 还可以启用 GDBstub 来通过 gdb 通信，无需任何特殊 JTAG 硬件。有关更多信息，请参阅 [为 GDBStub 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 [JTAG 调试](#)。
- 如果无法使用 pip 在 macOS 上安装 pyserial，请从 [pyserial](#) 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。

其他说明

- ESP-IDF 中必需的库包括在 Amazon FreeRTOS 中，因此无需从外部下载它们。如果已设置 IDF_PATH 环境变量，我们建议您在生成 Amazon FreeRTOS 之前清除它。
- 在 Window 系统上，生成项目可能需要 3-4 分钟。您可在 make 命令上使用 -j4 开关来缩短生成时间：

```
make flash monitor -j4
```

在 Espressif ESP32-DevKitC 和 ESP-WROVER-KIT 上调试代码

您需要 JTAG 到 USB 电缆。我们使用 USB 到 MPSSE 电缆（例如，[FTDI C232HM-DDHSL-0](#)）。

ESP-DevKitC JTAG 设置

对于 FTDI C232HM-DDHSL-0 电缆，以下是与 ESP32 DevkitC 的连接：

C232HM-DDHSL-0 电线颜色	ESP32 GPIO 针脚	JTAG 信号名称
棕色（针脚 5）	IO14	TMS
黄色（针脚 3）	IO12	TDI
黑色（针脚 10）	GND	GND
橙色（针脚 2）	IO13	TCK

C232HM-DDHSL-0 电线颜色	ESP32 GPIO 针脚	JTAG 信号名称
绿色 (针脚 4)	IO15	TDO

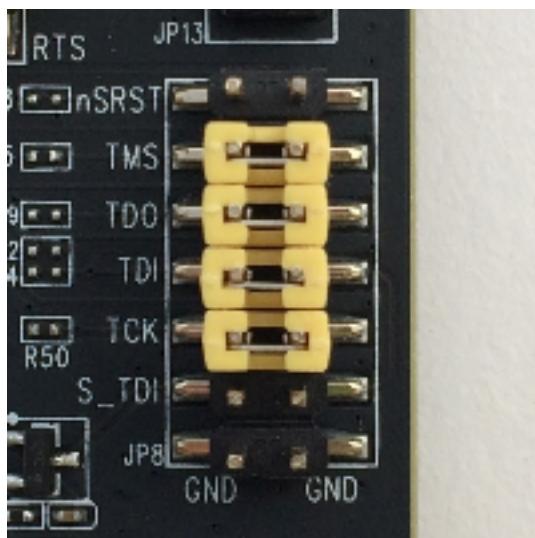
ESP-WROVER-KIT JTAG 设置

对于 FTDI C232HM-DDHSL-0 电缆，以下是与 ESP32-WROVER-KIT 的连接：

C232HM-DDHSL-0 电线颜色	ESP32 GPIO 针脚	JTAG 信号名称
棕色 (针脚 5)	IO14	TMS
黄色 (针脚 3)	IO12	TDI
橙色 (针脚 2)	IO13	TCK
绿色 (针脚 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 ESP-WROVER-KIT 上启用 JTAG，请将跳线放在 TMS、TDO、TDI、TCK 和 S_TDI 针脚上，如此处所示：



Windows 上的调试

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-DevKitC 和 ESP-WROVER-KIT 上调试代码 \(p. 30\)](#)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. 从 USB 控制器列表中，右键单击 FTDI C232HM-DDHSL-0 设备（制造商为 FTDI），然后选择属性。在属性窗口中，选择详细信息选项卡以查看设备的属性。如果未列出设备，请安装[适用于 FTDI C232HM-DDHSL-0 的 Windows 驱动程序](#)。
3. 验证设备管理器中显示的供应商 ID 和产品 ID 与 `demos\espressif\esp32_devkitc_esp_wrover_kit\esp32_devkitj_v1.cfg` 中的 ID 匹配。ID 在以 `ftdi_vid_pid` 开头的行中指定，后跟供应商 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。如需安装：
 - 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框中，选择 WinUSB。
 - e. 从目标驱动程序框下方的下拉框中，选择箭头，然后选择 Install Driver (安装驱动程序)。选择 Replace Driver (替换驱动程序)。
7. 打开命令提示符窗口，导航到 `<BASE_FOLDER>\demos\espressif\esp32_devkitc_esp_wrover_kit\make` 并运行：

```
openocd.exe -f esp32_devkitj_v1.cfg -f esp-wroom-32.cfg
```

保持此命令提示符窗口处于打开状态。

8. 打开一个新的命令提示符窗口，导航到您的 msys32 目录，然后运行 mingw32.exe。在 mingw32 终端内，导航到 `<BASE_FOLDER>\demos\espressif\esp32_devkitc_esp_wrover_kit\make` 并运行 make flash monitor。
9. 打开另一个 mingw32 终端，导航到 `<BASE_FOLDER>\demos\espressif\esp32_devkitc_esp_wrover_kit\make` 并运行 xtensa-esp32-elf-gdb -x gdbinit build/aws_demos.elf。程序应在 main 函数中停止。

Note

ESP32 支持最多两个断点。

在 macOS 上调试

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. 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
```

7. 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)
```

8. 打开 demos/espressif/esp32_devkitc_esp_wrover_kit/esp32_devkitj_v1.cfg。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 ID 以匹配上一步骤 system_profiler 输出中的 ID。
9. 打开终端窗口，导航到 <BASE_FOLDER>/demos/espressif/esp32_devkitc_esp_wrover_kit/make，然后使用以下命令运行 OpenOCD：

```
openocd -f esp32_devkitj_v1.cfg -f esp-wroom-32.cfg
```

10. 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
```

11. 导航到 <BASE_FOLDER>/demos/espressif/esp32_devkitc_esp_wrover_kit/make 并运行以下命令：

```
make flash monitor
```

12. 打开另一个新终端，导航到 <BASE_FOLDER>/demos/espressif/esp32_devkitc_esp_wrover_kit/make 并运行以下命令：

```
xtensa-esp32-elf-gdb -x gdbinit build/aws_demos.elf
```

程序应在 main() 停止。

在 Linux 上调试

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。您应该可以看到类似于如下所示的输出内容：

```
$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 group-owner 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。其他接口路由到 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，导航到 `<BASE_FOLDER>/demos/espressif/esp32_devkitc_esp_wrover_kit/make`，然后使用以下命令运行 OpenOCD：

```
openocd -f esp32_devkitj_v1.cfg -f esp-wroom-32.cfg
```

6. 打开另一个终端，导航到 `<BASE_FOLDER>/demos/espressif/esp32_devkitc_esp_wrover_kit/make` 并运行以下命令：

```
make flash monitor
```

7. 打开另一个终端，导航到 `<BASE_FOLDER>/demos/espressif/esp32_devkitc_esp_wrover_kit/make` 并运行以下命令：

```
xtensa-esp32-elf-gdb -x gdbinit build/aws_demos.elf
```

程序应在 `main()` 中停止。

开始使用 Infineon XMC4800 IoT Connectivity Kit

在开始之前，请参阅[先决条件 \(p. 4\)](#)。

如果您没有 Infineon XMC4800 IoT Connectivity Kit，请访问 AWS 合作伙伴设备目录来从我们的[合作伙伴](#)购买一个。

如果您要建立与主板的串行连接来查看日志记录和调试信息，则除了 XMC4800 IoT Connectivity Kit 之外，还需要 3.3V USB/串口转换器。CP2104 是一种常见的 USB/串口转换器，广泛用于各种主板，例如 Adafruit 的[CP2104 Friend](#)。

设置环境

Amazon FreeRTOS 使用 Infineon 的 DAVE 开发环境来编程 XMC4800。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 XMC4800 Relax EtherCAT 主板的板载调试探测器通信，您需要 J-Link Software and Documentation Pack 中附带的驱动程序。您可以从 Segger 的 [J-Link 软件下载](#) 页面下载 J-Link Software and Documentation Pack。

设置串行连接

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 演示应用程序在针脚 P0.0 和 P0.1 上使用 UART 串行连接，这些数字标注在 XMC4800 Relax EtherCAT 主板上印刷的字样中。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。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。

下载并配置 Amazon FreeRTOS

设置环境后，您可以下载 Amazon FreeRTOS。

下载 Amazon FreeRTOS

1. 浏览到 AWS IoT 控制台。
2. 在导航窗格中，选择 Software (软件)。
3. 在 Amazon FreeRTOS 设备软件下，选择配置下载。
4. 在软件配置下，找到连接到 AWS IoT - Infineon，然后选择下载。
5. 将下载的文件解压缩到 AmazonFreeRTOS 文件夹，然后记录文件夹的路径。

Note

Microsoft Windows 上的文件路径最大长度为 260 个字符。Amazon FreeRTOS 下载中的最长路径为 122 个字符。为了适应 Amazon FreeRTOS 项目中的文件，请确保 AmazonFreeRTOS 目录的路径长度少于 98 个字符。例如，C:\Users\Username\Dev\AmazonFreeRTOS 可以正常工作，但 C:\Users\Username\Documents\Development\Projects\AmazonFreeRTOS 会导致生成失败。

在本教程中，目录的路径称为 AmazonFreeRTOS。

配置项目

要运行演示，您必须将项目配置为使用 AWS IoT（需要您将主板注册为 AWS IoT 事物）。[将您的 MCU 主板注册到 AWS IoT \(p. 5\)](#) 是先决条件 (p. 4) 中的一个步骤。

配置 AWS IoT 终端节点

1. 浏览至 [AWS IoT 控制台](#)。
2. 在导航窗格中，选择 Settings。

您的 AWS IoT 终端节点将显示在终端节点中。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。

您的设备应具有 AWS IoT 事物名称。记下此名称。

4. 在您的 IDE 中，打开 `<BASE_FOLDER>\demos\common\include\aws_clientcredential.h` 并为以下 `#define` 常量指定值：

- `clientcredentialMQTT_BROKER_ENDPOINT ## AWS_IoT #####`
- `clientcredentialIOT_THING_NAME ### AWS_IoT #####`

配置 Wi-Fi 设置

1. 打开 `aws_clientcredential.h` 文件。

2. 为以下 `#define` 常量指定值：

- `clientcredentialWIFI_SSID Wi-Fi ### SSID`
- `clientcredentialWIFI_PASSWORD Wi-Fi #####`
- `clientcredentialWIFI_SECURITY Wi-Fi #####`

有效安全类型如下：

- `eWiFiSecurityOpen` (开放，不安全)
- `eWiFiSecurityWEP` (WEP 安全性)
- `eWiFiSecurityWPA` (WPA 安全性)
- `eWiFiSecurityWPA2` (WPA2 安全性)

配置您的 AWS IoT 凭证

Note

要配置 AWS IoT 凭证，您需要您在注册设备时从 AWS IoT 控制台下载的私有密钥和证书。在将设备注册为 AWS IoT 事物之后，可从 AWS IoT 控制台检索设备证书，但无法检索私有密钥。

Amazon FreeRTOS 是 C 语言项目，证书和私有密钥必须进行特别的格式设置才能添加到该项目中。您必须设置您的设备的证书和私有密钥的格式。

1. 在浏览器窗口中，打开 `<BASE_FOLDER>\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 `<BASE_FOLDER>\demos\common\include` 中。这将覆盖目录中的现有文件。

Note

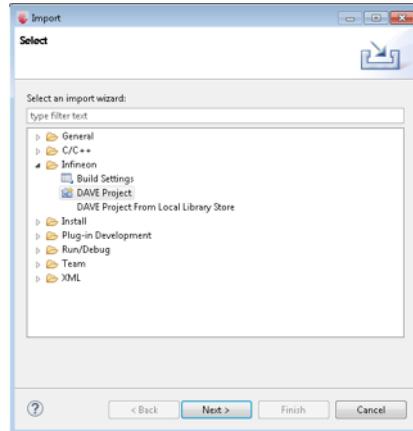
仅出于演示目的对证书和私有密钥进行了硬编码。生产级应用程序应将这些文件存储在安全位置。

生成并运行 Amazon FreeRTOS 演示项目

将 Amazon FreeRTOS 演示导入 DAVE

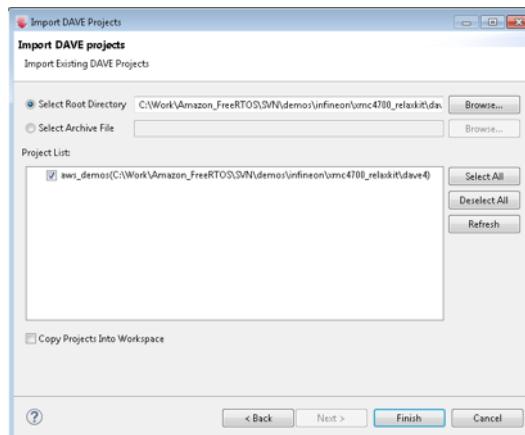
1. 启动 DAVE。

- 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)。



- 在 Import DAVE Projects (导入 DAVE 项目) 窗口中，选择 Select Root Directory (选择根目录)，选择 Browse (浏览)，然后选择 XMC4800 演示项目。

在您解压缩 Amazon FreeRTOS 下载文件的目录中，演示项目位于 `<BASE_FOLDER>/demos/infineon/xmc4800_iotkit/dave` 中。



确保 Copy Projects Into Workspace (将项目复制到工作区) 处于未选中状态。

- 选择 Finish。
- `aws_demos` 项目应导入到工作区中并且激活。
- 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 Amazon FreeRTOS 演示项目

在您配置项目之后，就可以在主板上运行演示项目。

- 使用 USB 线缆将您的 XMC4800 IoT Connectivity Kit 连接到计算机。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”。
- 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。
- 登录到 AWS IoT 控制台。

4. 在导航窗格中，选择测试以打开 MQTT 客户端。
5. 在订阅主题中，输入 `freertos/demos/echo`，然后选择订阅主题。
6. 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)。
7. 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)。
8. 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，来自步骤 4-5 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 freertos/demos/echo
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----
```

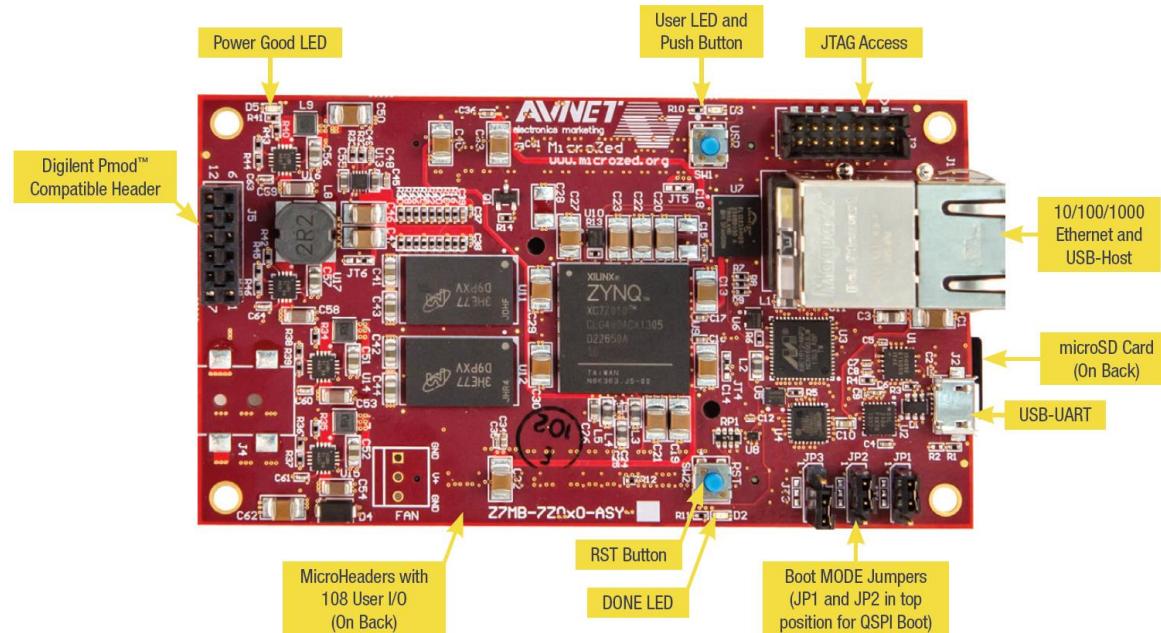
Xilinx Avnet MicroZed 工业 IoT 工具包入门

在开始之前，请参阅[先决条件 \(p. 4\)](#)。

如果您没有 Xilinx Avnet MicroZed Industrial IoT Kit，请访问 AWS 合作伙伴设备目录来从我们的[合作伙伴](#)购买一个。

设置 MicroZed 硬件

在设置 MicroZed 硬件时，下图可能会很有用：



设置 MicroZed 主板

1. 将计算机连接到 MicroZed 主板上的 USB-UART 端口。
2. 将计算机连接到 MicroZed 主板上的 JTAG Access 端口。
3. 将路由器或连接 Internet 的以太网端口连接到 MicroZed 主板上的以太网和 USB-HOST 端口。

设置环境

要设置 MicroZed 工具包的 Amazon FreeRTOS 配置，您必须使用 Xilinx 开发工具包 (XSDK)。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. 转至[开发工具包独立 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，请输入您的用户名和密码，然后选择 Sign in (登录)。

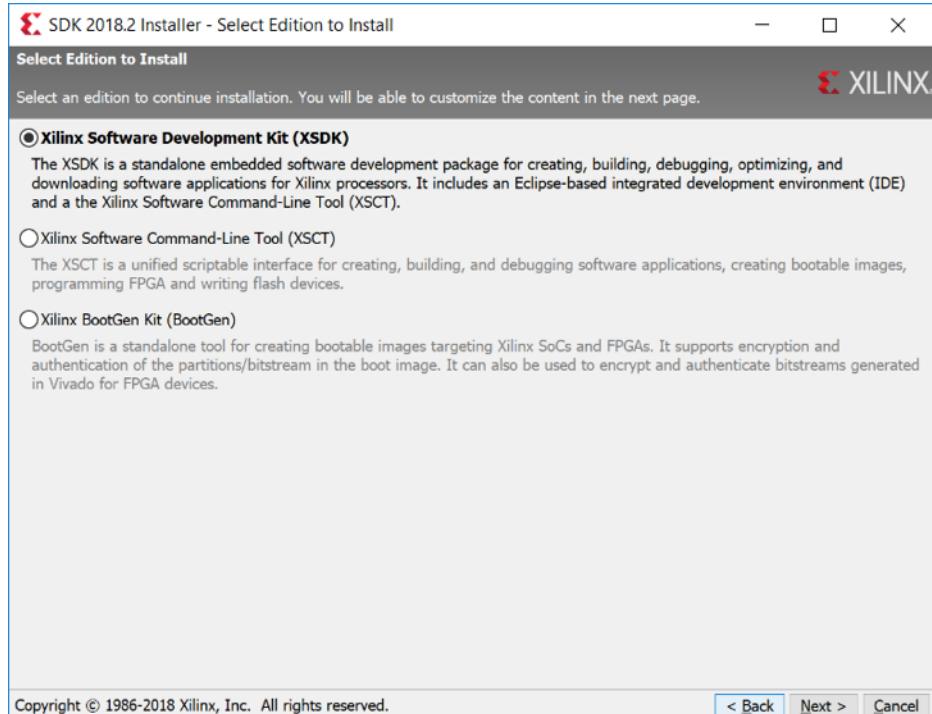
如果您没有账户，请选择 Create your account (创建账户)。注册后，您将收到一封电子邮件，其中包含用于激活您的 Xilinx 账户的链接。

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。

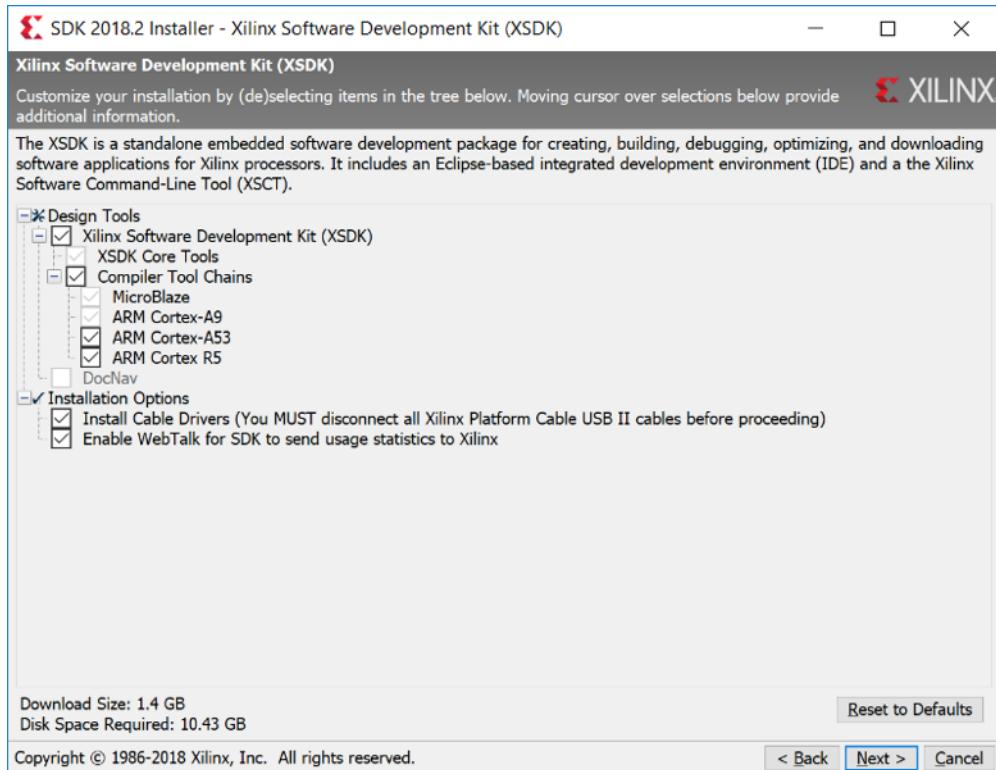
5. 保存 `xilinx_SDK_version_os` 文件。

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 MicroZed 的 USB-UART 连接，请手动安装 CP210x USB-to-UART Bridge VCP 驱动程序。有关说明，请参阅 [Silicon Labs CP210x USB-to-UART 安装指南](#)。

有关 XSDK 的更多信息，请参阅 Xilinx 网站上的 [Xilinx 开发工具包入门](#)。

下载并配置 Amazon FreeRTOS

设置环境后，您可以下载 Amazon FreeRTOS。

下载 Amazon FreeRTOS

1. 浏览到 AWS IoT 控制台。
2. 在导航窗格中，选择 Software (软件)。
3. 在 Amazon FreeRTOS 设备软件下，选择配置下载。
4. 在 Software Configurations (软件配置) 下，找到 Connect to AWS IoT - Xilinx，然后选择 Download (下载)。
5. 将下载的文件解压缩到 AmazonFreeRTOS 文件夹，然后记录文件夹的路径。

Note

Microsoft Windows 上的文件路径最大长度为 260 个字符。Amazon FreeRTOS 下载中的最长路径为 122 个字符。为了适应 Amazon FreeRTOS 项目中的文件，请确保 AmazonFreeRTOS 目录的路径长度少于 98 个字符。例如，C:\Users\Username\Dev\AmazonFreeRTOS 可以正常工作，但 C:\Users\Username\Documents\Development\Projects\AmazonFreeRTOS 会导致生成失败。

在本教程中，目录的路径称为 AmazonFreeRTOS。

配置项目

要运行演示，您必须将项目配置为使用 AWS IoT。要将项目配置为使用 AWS IoT，必须将主板注册为 AWS IoT 事物。这是[先决条件 \(p. 4\)](#)中的一个步骤。

配置 AWS IoT 终端节点

1. 浏览至 [AWS IoT 控制台](#)。
2. 在导航窗格中，选择 Settings。

您的 AWS IoT 终端节点显示在 Endpoint (终端节点) 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。记下设备的 AWS IoT 事物名称。
4. 利用您拥有的 AWS IoT 终端节点和 AWS IoT 事物名称，在 IDE 中打开 `<BASE_FOLDER>\demos\common\include\aws_clientcredential.h`，并为以下 #define 常量指定值：
 - `clientcredentialMQTT_BROKER_ENDPOINT ## AWS_IOT #####`
 - `clientcredentialIOT_THING_NAME ##### AWS_IOT #####`

配置您的 AWS IoT 凭证

要配置 AWS IoT 凭证，您需要您在将设备注册为 AWS IoT 事物时从 AWS IoT 控制台下载的私有密钥和证书。在将设备注册为 AWS IoT 事物之后，可从 AWS IoT 控制台检索设备证书，但无法检索私有密钥。

Amazon FreeRTOS 是 C 语言项目，证书和私有密钥必须进行特别的格式设置才能添加到该项目中。您需要设置您的设备的证书和私有密钥的格式。

1. 在浏览器窗口中，打开 `<BASE_FOLDER>\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 `<BASE_FOLDER>\demos\common\include` 中。这将覆盖目录中的现有文件。

Note

仅应出于演示的目的来将证书和私有密钥进行硬编码。生产级应用程序应将这些文件存储在安全位置。

生成并运行 Amazon FreeRTOS 演示项目

现在您已配置项目，可以在主板上构建和运行演示项目。

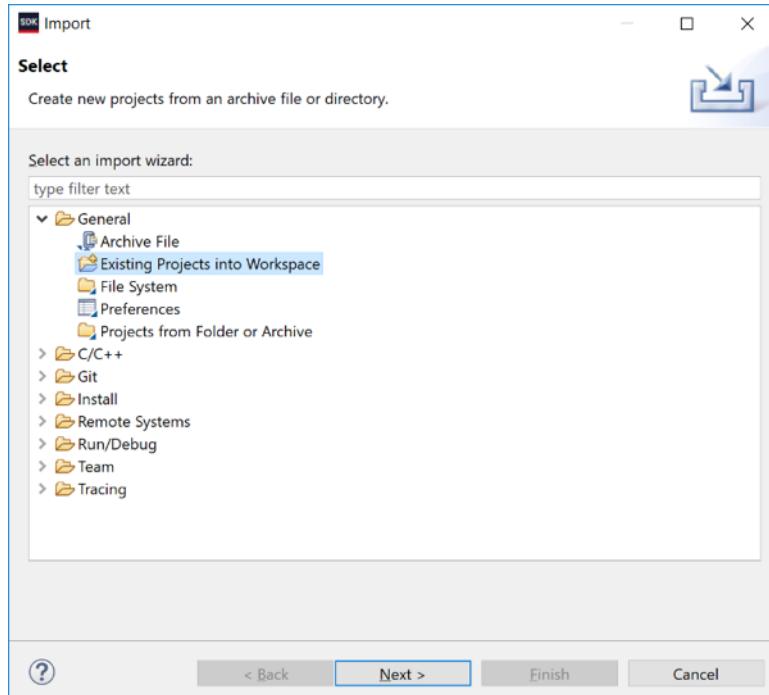
在运行演示项目之前，在 AWS IoT 控制台中使用 MQTT 客户端订阅演示的 MQTT 主题。

订阅 MQTT 主题

1. 登录到 AWS IoT 控制台。
2. 在导航窗格中，选择测试以打开 MQTT 客户端。
3. 在订阅主题中，输入 `freertos/demos/echo`，然后选择订阅主题。

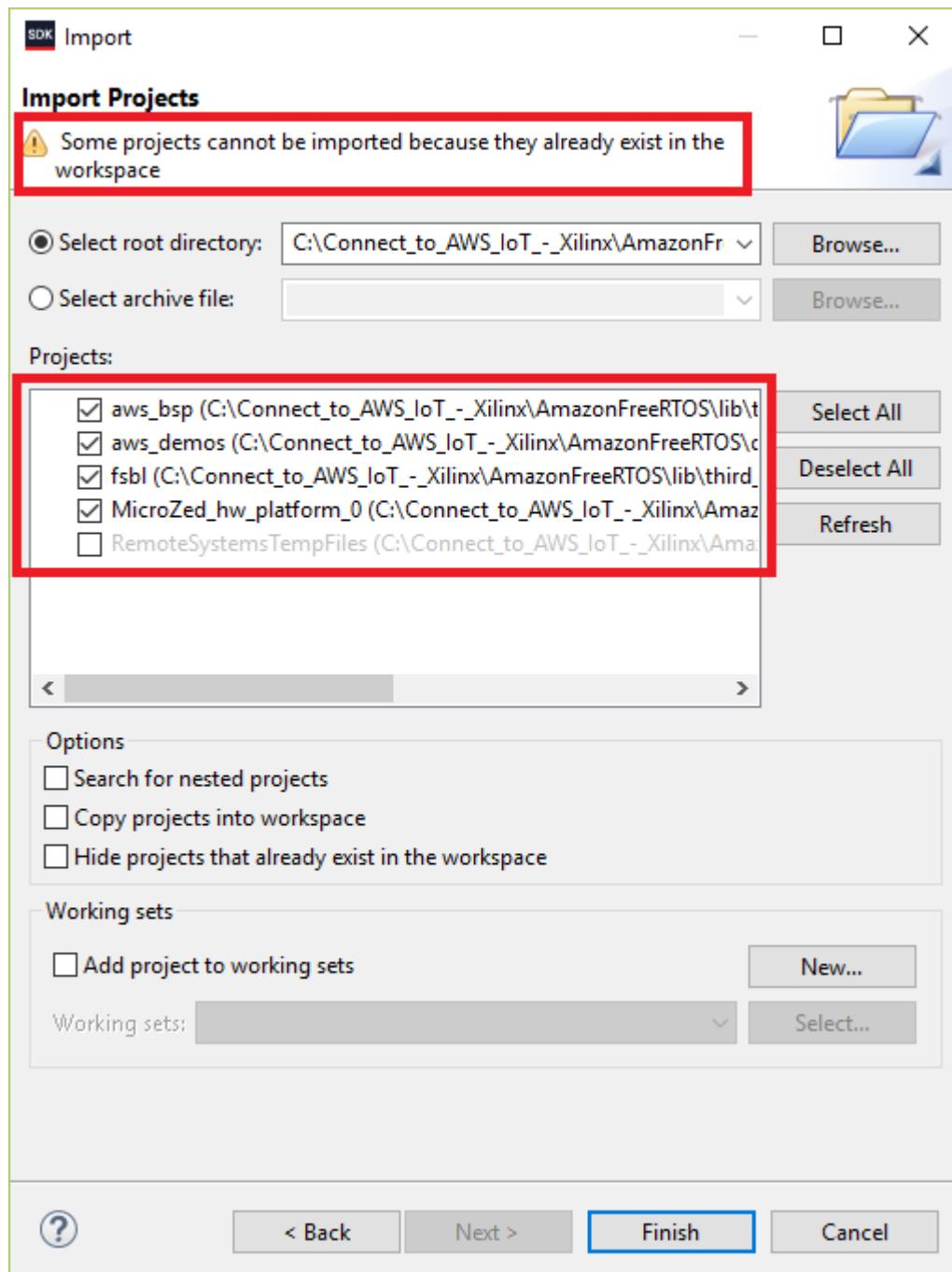
在 XSDK IDE 中打开 Amazon FreeRTOS 演示

1. 启动 XSDK IDE，并将工作区目录设置为 `<BASE_FOLDER>\demos\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。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。

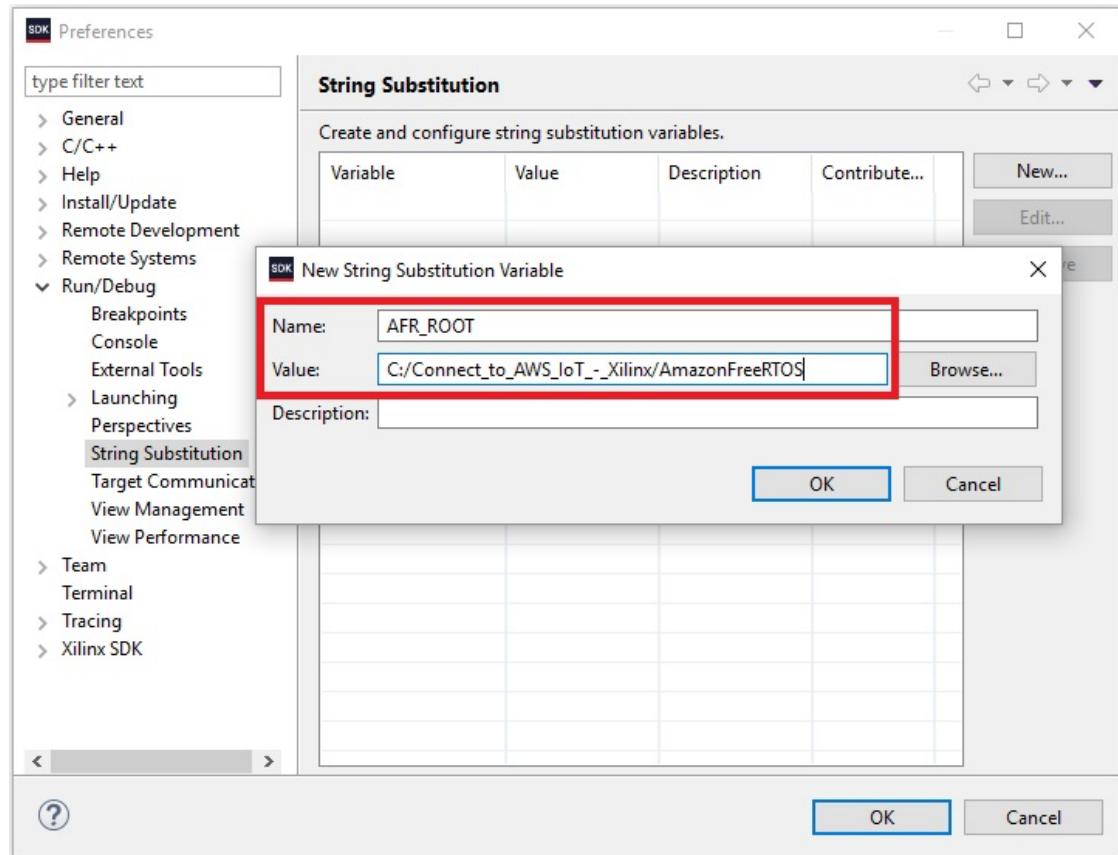


Note

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。

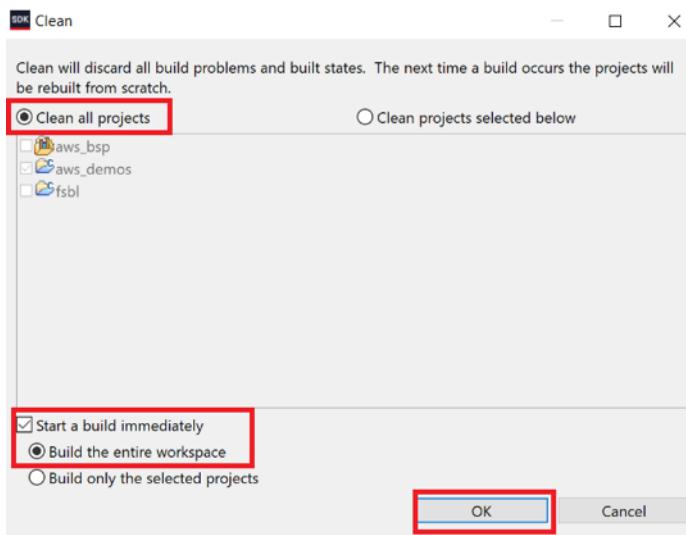
6. 选定所有项目后，选择 Finish (完成)。XSDK IDE 将打开在 MicroZed 主板上构建和运行 aws_demos 项目所需的所有项目。
7. 从文件菜单中，选择 Window (窗口)，然后选择 Preferences (首选项)。

8. 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)。
9. 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 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)。



生成 Amazon 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。

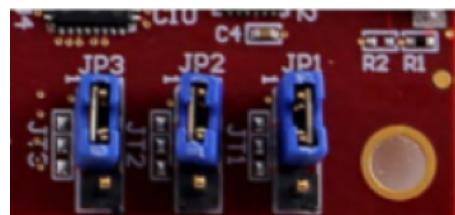


Note

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)。

JTAG 调试

1. 将您的 MicroZed 主板的启动模式跳线设置为 JTAG 启动模式：

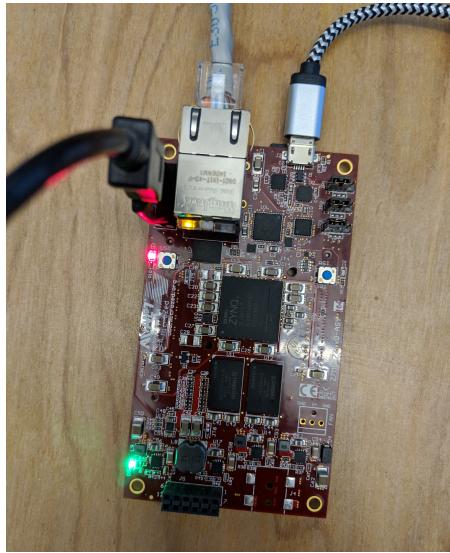


2. 将您的 MicroSD 卡插入位于 USB-UART 端口正下方的 MicroSD 卡槽。

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：



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)。

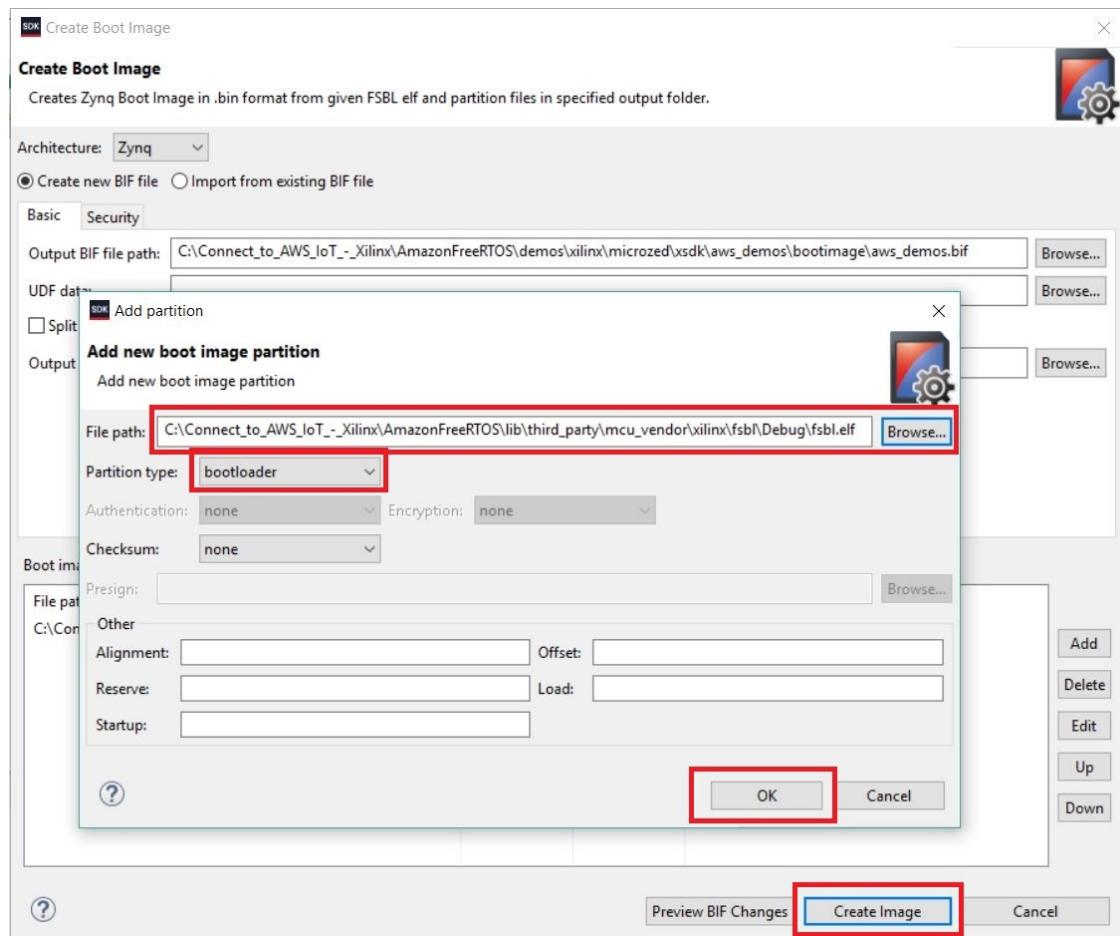
Note

当您首次运行应用程序时，将生成新的证书-密钥对。对于后续运行，您可以先在 main.c 文件中注释掉 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。

您可以选择从 MicroSD 卡或 QSPI 闪存启动 MicroZed 主板以运行 Amazon FreeRTOS 演示项目。有关说明，请参阅[为 Amazon FreeRTOS 演示项目生成启动映像 \(p. 47\)](#)和[运行 Amazon FreeRTOS 演示项目 \(p. 48\)](#)。

为 Amazon 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 **<BASE_FOLDER>\demos\xilinx\microzed\xsdk\aws_demos\bootimage\aws_demos.bif** 中)。
4. 选择 Add。
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 **<BASE_FOLDER>\lib\third_party\mcu_vendor\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 demos\xilinx\microzed\xsdk\aws_demos\bootimage\BOOT.bin 中)。

运行 Amazon FreeRTOS 演示项目

要运行 Amazon FreeRTOS 演示项目，您可以从 MicroSD 卡或 QSPI 闪存启动 MicroZed 主板。

在设置 MicroZed 主板以运行 Amazon FreeRTOS 演示项目时，请参考[设置 MicroZed 硬件 \(p. 39\)](#)中的演示图。确保已将 MicroZed 主板连接到计算机。

从 MicroSD 卡启动 Amazon FreeRTOS 项目

格式化 Xilinx Avnet MicroZed 工业 IoT 工具包附带的 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：

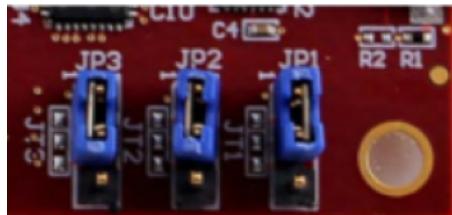
SD Card



- 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 Amazon FreeRTOS 演示项目

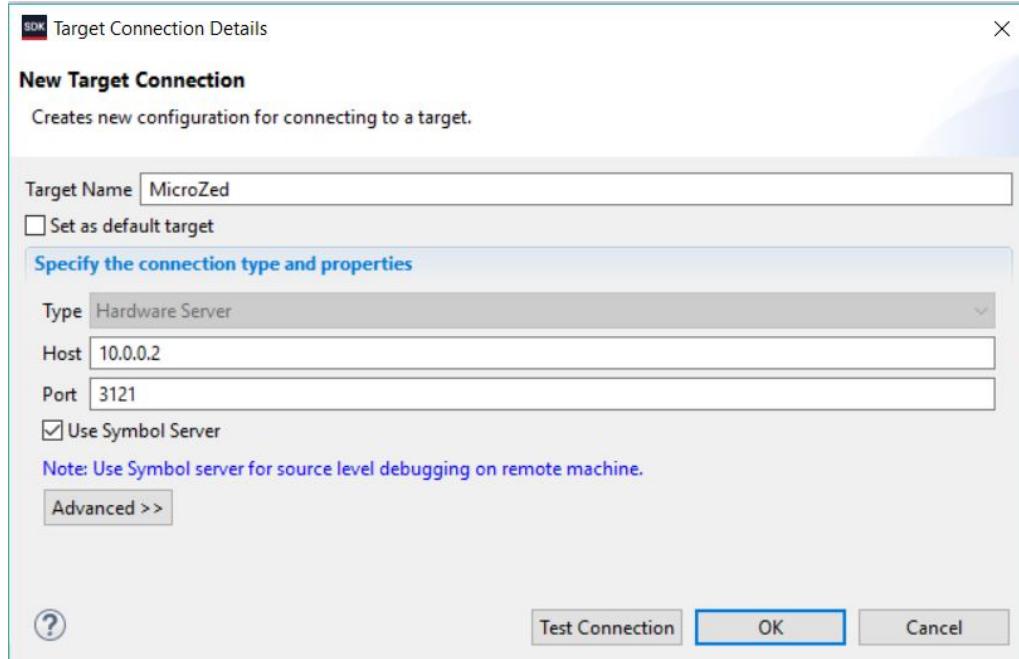
- 将您的 MicroZed 主板的启动模式跳线设置为 JTAG 启动模式：



- 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。
- 在 XSDK IDE 中，从菜单中选择 Xilinx，然后选择 Program Flash (对闪存编程)。
- 在 Program Flash Memory (对闪存编程) 中，应自动填充硬件平台。对于 Connection (连接)，选择 MicroZed 硬件服务器以将主板连接到主机。

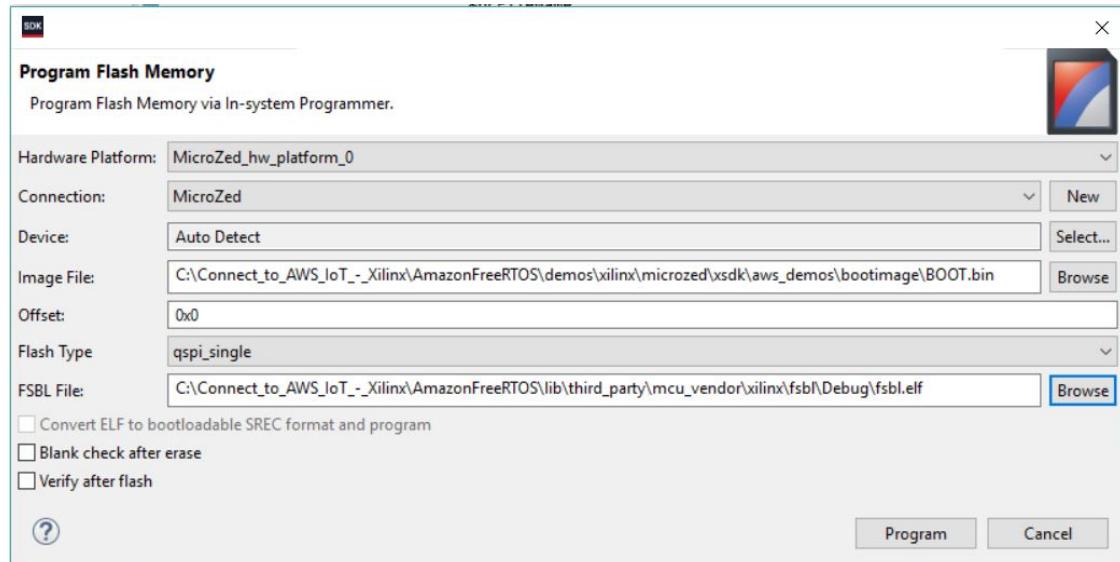
Note

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。

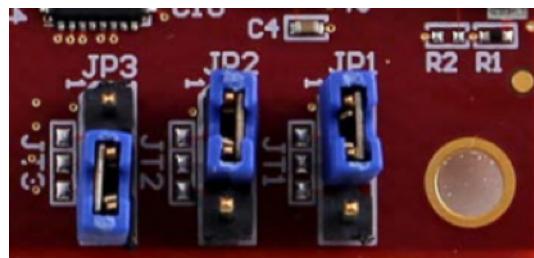


- 在 Image File (映像文件) 中，输入 BOOT.bin 映像文件的目录路径。选择 Browse (浏览) 以改为浏览文件。

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. 将您的 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。

问题排查

一般故障排除技巧

- 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，如生成 Amazon FreeRTOS 演示项目 (p. 45) 中所述。

Note

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。

Renesas Starter Kit+ for RX65N-2MB 入门

在开始之前，请参阅[先决条件 \(p. 4\)](#)。

如果您没有 Renesas RSK+ for RX65N-2MB，请访问 AWS 合作伙伴设备目录并从我们的[合作伙伴](#)购买一个。

设置 Renesas 硬件

设置 RSK+ for RX65N-2MB

1. 将计算机连接到 RSK+ for RX65N-2MB 上的 USB-UART 端口。
2. 将路由器或连接 Internet 的以太网端口连接到 RSK+ for RX65N-2MB 上的以太网端口。

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 RSK+ for RX65N-2MB 上的“E1/E2 Lite”端口。
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. 将中心正 +5V 电源适配器连接到 RSK+ for RX65N-2MB 上的 PWR 接头。
4. 在调试器连接到主机和 RSK+ for RX65N-2MB 后，E2 Lite 调试器驱动程序将开始安装。

请注意，安装驱动程序需要管理员权限。

设置环境

要为 RSK+ for RX65N-2MB 设置 Amazon FreeRTOS 配置，请使用 Renesas e²studio IDE 和 CC-RX 编译器。

Note

仅 Windows 7、8 和 10 操作系统支持 Renesas e²studio 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，请输入您的用户名和密码，然后选择 Login (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 V2.08 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](#)。

下载并配置 Amazon FreeRTOS

设置环境后，您可以下载 Amazon FreeRTOS。

下载 Amazon FreeRTOS

下载 Amazon FreeRTOS

1. 浏览到 AWS IoT 控制台。
2. 在导航窗格中，选择 Software (软件)。
3. 在 Amazon FreeRTOS Device Software (Amazon FreeRTOS 设备软件) 下，选择 Configure download (配置下载)。
4. 在软件配置下，找到连接到 AWS IoT - Renesas，然后选择下载。
5. 将下载的文件解压缩到 AmazonFreeRTOS 文件夹，然后记录文件夹的路径。

Note

Microsoft Windows 上的文件路径最大长度为 260 个字符。Amazon FreeRTOS 下载中的最长路径为 122 个字符。为了适应 Amazon FreeRTOS 项目中的文件，请确保 AmazonFreeRTOS 目录的路径长度少于 98 个字符。例如，C:\Users\Username\Dev\AmazonFreeRTOS 可以正常工作，但 C:\Users\Username\Documents\Development\Projects\AmazonFreeRTOS 会导致生成失败。

在本教程中，目录的路径称为 AmazonFreeRTOS。

配置项目

要运行演示，您必须将项目配置为使用 AWS IoT。要将项目配置为使用 AWS IoT，必须将主板注册为 AWS IoT 事物。这是[先决条件 \(p. 4\)](#)中的一个步骤。

配置 AWS IoT 终端节点

1. 浏览至 [AWS IoT 控制台](#)。
2. 在导航窗格中，选择 Settings。

您的 AWS IoT 终端节点显示在 Endpoint (终端节点) 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。记下设备的 AWS IoT 事物名称。
4. 利用您拥有的 AWS IoT 终端节点和 AWS IoT 事物名称，在 IDE 中打开 <**BASE_FOLDER**>\demos\common\include\aws_clientcredential.h，并为以下 #define 常量指定值：
 - clientcredentialMQTT_BROKER_ENDPOINT ## AWS_IOT #####
 - clientcredentialIOT_THING_NAME ##### AWS_IOT #####

配置您的 AWS IoT 凭证

要配置 AWS IoT 凭证，您需要您在将设备注册为 AWS IoT 事物时从 AWS IoT 控制台下载的私有密钥和证书。在将设备注册为 AWS IoT 事物之后，可从 AWS IoT 控制台检索设备证书，但无法检索私有密钥。

Amazon FreeRTOS 是 C 语言项目，证书和私有密钥必须进行特别的格式设置才能添加到该项目中。您需要设置您的设备的证书和私有密钥的格式。

1. 在浏览器窗口中，打开 `<BASE_FOLDER>\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 `<BASE_FOLDER>\demos\common\include` 中。这将覆盖目录中的现有文件。

Note

仅应出于演示的目的来将证书和私有密钥进行硬编码。生产级应用程序应将这些文件存储在安全位置。

生成并运行 Amazon 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。

在运行演示项目之前，在 AWS IoT 控制台中使用 MQTT 客户端订阅演示的 MQTT 主题。

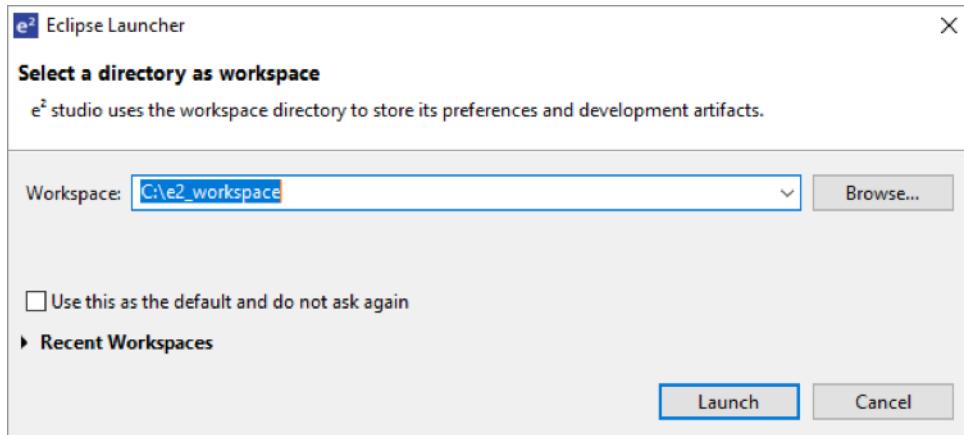
订阅 MQTT 主题

1. 登录到 AWS IoT 控制台。
2. 在导航窗格中，选择测试以打开 MQTT 客户端。
3. 在订阅主题中，输入 `freertos/demos/echo`，然后选择订阅主题。

在 e²studio 中构建 Amazon FreeRTOS 演示

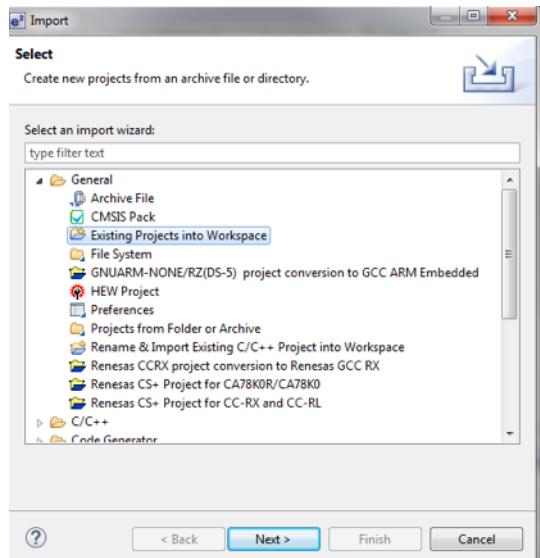
在 e²studio 中导入和构建演示

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)。

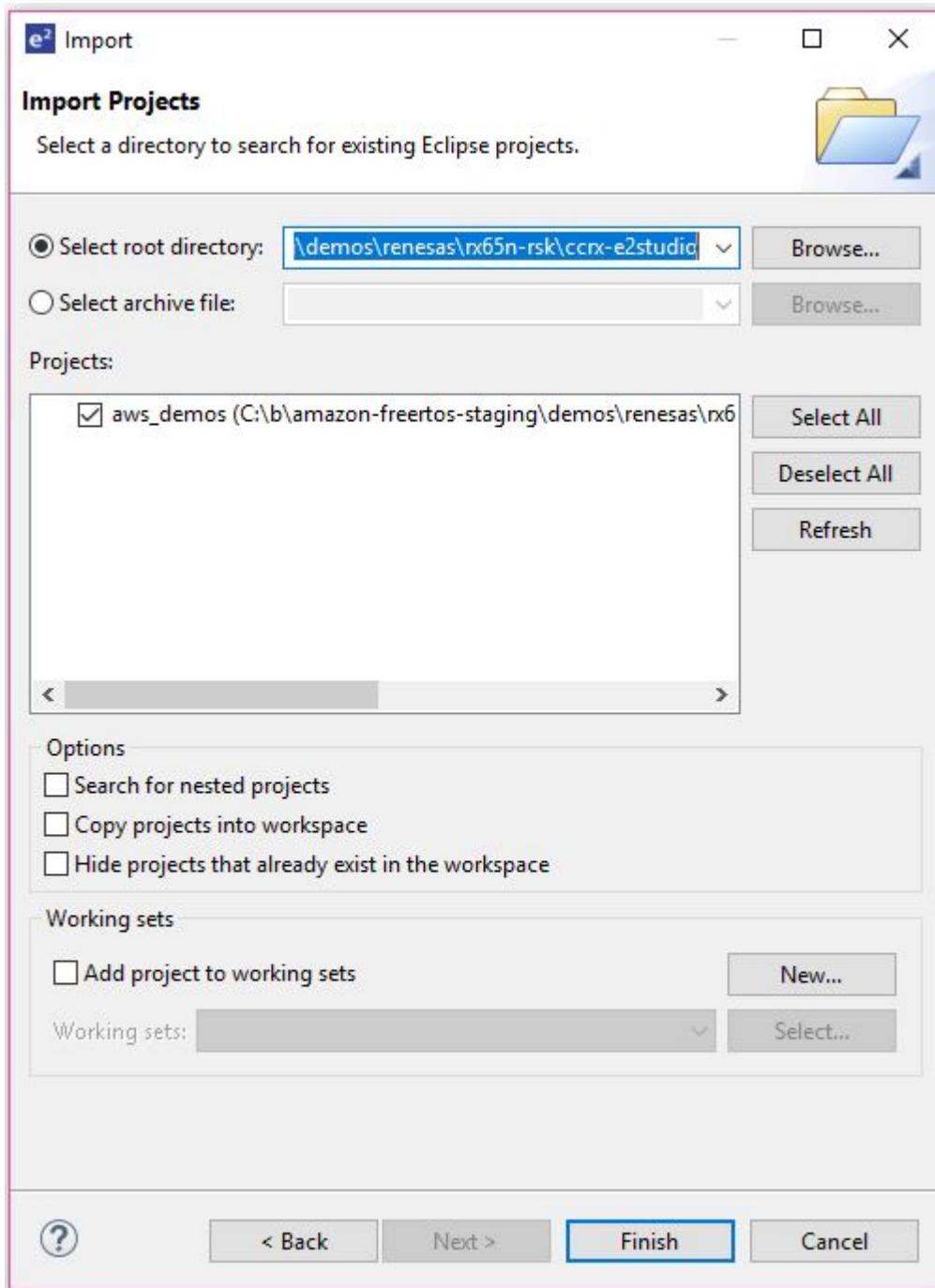


3. 当您首次打开 e²studio 时，Toolchain Registry (工具链注册表) 窗口会打开。选择 Renesas Toolchains (Renesas 工具链)，并确认已选择 **CC-RX v2.07.00**。选择 Register (注册)，然后选择 OK (确定)。

4. 如果您是首次打开 e²studio，将显示 Code Generator Registration (代码生成器注册) 窗口。选择 OK。
5. 将显示 Code Generator COM component register (代码生成器 COM 组件注册) 窗口。在 Please restart e²studio to use Code Generator (请重启 e²studio 以使用代码生成器) 下，选择 OK (确定)。
6. 将显示 Restart e²studio (重启 e²studio) 窗口。选择 OK。
7. e²studio 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 e²studio 欢迎屏幕上，选择 Go to the e²studio 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 <BASE_FOLDER>\demos\renesas\rx65n-rsk\ccrx-e2studio，然后选择 Finish (完成)。



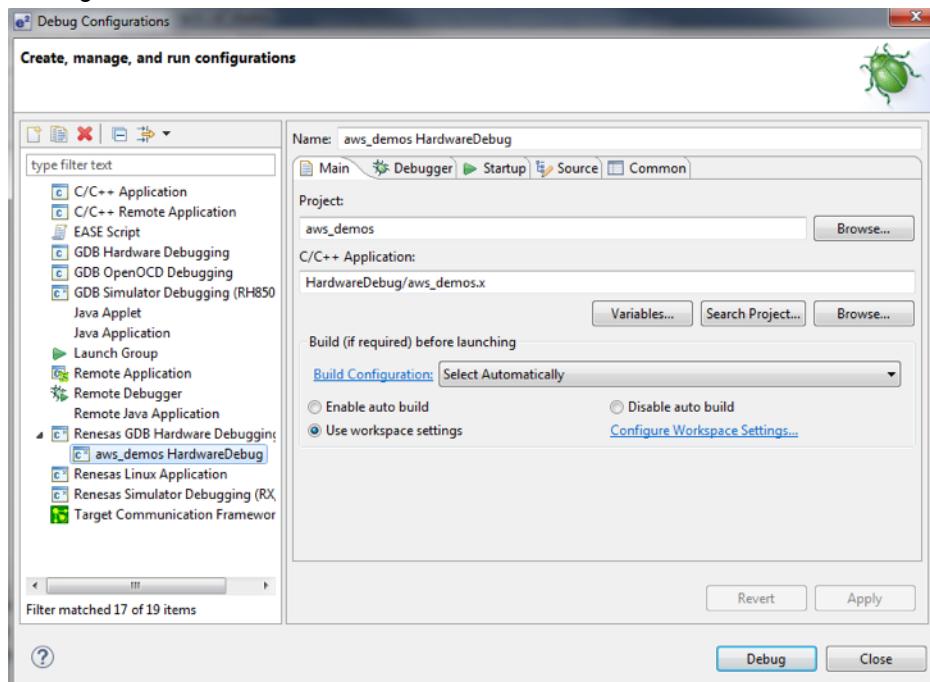
12. 从 Project (项目) 菜单中，选择 Project (项目)、Build All (全部生成)。

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](#)。

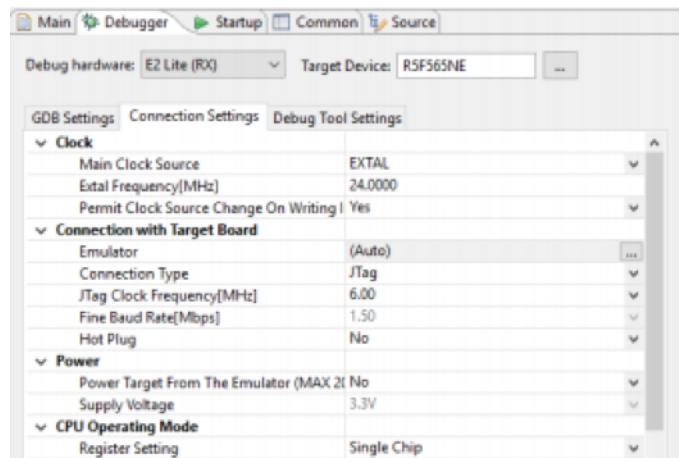
运行 Amazon FreeRTOS 项目

在 e²studio 中运行项目

1. 确认已将 E2 Lite 调试器模块连接到 RSK+ for RX65N-2MB
2. 从顶部菜单中，选择 Run (运行)、Debug Configuration (调试配置)。
3. 展开 Renesas GDB Hardware Debugging (Renesas 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)。选择是。

E2 Lite 调试器上绿色的“ACT”LED 会亮起。

- 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。

在 AWS IoT 控制台的 MQTT 客户端中，您会看到设备发送的 MQTT 消息。

有关 Renesas 发布的最新项目，请参阅 [GitHub](#) 上的 amazon-freertos 存储库的 `renesas-rx` 分支。

MediaTek MT7697Hx 开发工具包入门

在开始之前，请参阅 [先决条件 \(p. 4\)](#)。

如果您没有 MediaTek MT7697Hx Development Kit，请访问 AWS 合作伙伴设备目录来从我们的[合作伙伴](#)购买一个。

设置环境

在您设置环境之前，请将计算机连接到 MediaTek MT7697Hx 开发工具包的 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、生成并运行 Amazon 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

- 转到 [Keil MDK Getting Started \(Keil MDK 入门\)](#) 页面，然后选择 Download MDK-Core (下载 MDK-Core)。
- 输入并提交信息以注册到 Keil。
- 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。
- 打开 Keil MDK 安装程序并按照步骤完成操作。确保您已安装 MediaTek 设备包 (MT76x7 系列)。

设置串行连接

要建立与 MediaTek MT7697Hx 开发工具包的串行连接，您必须安装 Arm Mbed Windows 串行端口驱动程序。您可从 [Mbed](#) 下载驱动程序。在 Windows serial driver (Windows 串行驱动程序) 页面上，下载并安装 MediaTek MT7697Hx 开发工具包的驱动程序。

安装驱动程序后，COM 端口显示在 Windows 设备管理器上。如需调试，您可以使用终端实用工具（例如 HyperTerminal 或 TeraTerm）打开与端口的会话。

下载并配置 Amazon FreeRTOS

设置环境后，您可以下载 Amazon FreeRTOS。

下载 Amazon FreeRTOS

下载 Amazon FreeRTOS

1. 浏览到 AWS IoT 控制台。
2. 在导航窗格中，选择 Software (软件)。
3. 在 Amazon FreeRTOS 设备软件下，选择配置下载。
4. 在软件配置下，找到连接到 AWS IoT - MediaTek，然后选择下载。
5. 将下载的文件解压缩到文件夹，然后记录文件夹路径。解压缩的文件夹名为 AmazonFreeRTOS。

Note

Microsoft Windows 上的文件路径最大长度为 260 个字符。Amazon FreeRTOS 下载中的最长路径为 122 个字符。为了适应 Amazon FreeRTOS 项目中的文件，请确保 AmazonFreeRTOS 目录的路径长度少于 98 个字符。例如，C:\Users\Username\Dev\AmazonFreeRTOS 可以正常工作，但 C:\Users\Username\Documents\Development\Projects\AmazonFreeRTOS 会导致生成失败。

在本教程中，目录的路径称为 AmazonFreeRTOS。

配置项目

要运行演示，您必须将项目配置为使用 AWS IoT（需要您将主板注册为 AWS IoT 事物）。[将您的 MCU 主板注册到 AWS IoT \(p. 5\)](#) 是先决条件 (p. 4) 中的一个步骤。

配置 AWS IoT 终端节点

1. 浏览至 [AWS IoT 控制台](#)。
2. 在导航窗格中，选择 Settings。

您的 AWS IoT 终端节点将显示在终端节点中。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。

您的设备应具有 AWS IoT 事物名称。记下此名称。

4. 在您的 IDE 中，打开 <BASE FOLDER>\demos\common\include\aws_clientcredential.h 并为以下 #define 常量指定值：
 - clientcredentialMQTT_BROKER_ENDPOINT ## AWS_IoT #####
 - clientcredentialIOT_THING_NAME #### AWS_IoT #####

配置 Wi-Fi 设置

1. 打开 aws_clientcredential.h 文件。
2. 为以下 #define 常量指定值：

- clientcredentialWIFI_SSID *Wi-Fi* ### *SSID*
- clientcredentialWIFI_PASSWORD *Wi-Fi* #####
- clientcredentialWIFI_SECURITY *Wi-Fi* #####

有效安全类型如下：

- eWiFiSecurityOpen (开放，不安全)
- eWiFiSecurityWEP (WEP 安全性)
- eWiFiSecurityWPA (WPA 安全性)

- eWiFiSecurityWPA2 (WPA2 安全性)

配置您的 AWS IoT 凭证

Note

要配置 AWS IoT 凭证，您需要您在注册设备时从 AWS IoT 控制台下载的私有密钥和证书。在将设备注册为 AWS IoT 事物之后，可从 AWS IoT 控制台检索设备证书，但无法检索私有密钥。

Amazon FreeRTOS 是 C 语言项目，证书和私有密钥必须进行特别的格式设置才能添加到该项目中。您必须设置您的设备的证书和私有密钥的格式。

1. 在浏览器窗口中，打开 `<BASE_FOLDER>\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 `<BASE_FOLDER>\demos\common\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。

使用 Keil MDK 生成并运行 Amazon FreeRTOS 演示项目

在 Keil μVision 中生成 Amazon FreeRTOS 演示项目

1. 从开始菜单，打开 Keil μVision 5。
2. 打开 `<BASE_FOLDER>/demos/mediatek/mt7697hx-dev-kit/uvision/aws_demo.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 `<BASE_FOLDER>/demos/mediatek/mt7697hx-dev-kit/uvision/out/Objects/aws_demo.axf` 的演示可执行文件。

运行 Amazon FreeRTOS 演示项目

1. 将 MediaTek MT7697Hx 开发工具包设置为 PROGRAM 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 MT7697Hx MediaTek 开发工具包上选择 RESET 按钮。

在 Keil μVision 中调试 Amazon FreeRTOS 项目

目前，您先必须编辑 Keil μVision 中包含的 MediaTek 包，然后才能调试 MediaTek 与 Keil μVision 的 Amazon FreeRTOS 演示项目。

编辑 MediaTek 包用于调试 Amazon 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\MediaTek\MTx\4.6.1\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 和 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 `<BASE_FOLDER>/demos/mediatek/mt7697hx-dev-kit/common/application_code/main.c`，并将宏 MTK_DEBUGGER 设置为 1。
5. 在 μVision 中重新生成演示项目。
6. 将 MediaTek MT7697Hx 开发工具包设置为 PROGRAM 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 μVision 菜单，选择 Debug (调试)，然后选择 Start/Stop Debug Session (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。μVision 将演示刷入主板，运行演示，然后在 `main()` 函数开始前停止。
8. 从菜单中，选择 Debug (调试)，然后选择 Stop (停止) 以停止会话。程序计数器在以下行停止：

```
{ volatile int wait_ice = 1 ; while ( wait_ice ) ; }
```

9. 在 Call Stack + Locals (调用堆栈 + 本地) 窗口上，将 `wait_ice` 的值更改为 0。
10. 在项目源代码中设置断点，然后运行代码。

开始使用 FreeRTOS Windows 仿真器

在开始之前，请参阅[先决条件 \(p. 4\)](#)。

Amazon FreeRTOS 以 zip 文件格式发布，包含您所指定平台的 Amazon 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。

设置环境

1. 安装最新版的 [WinPCap](#)。

2. 安装 Microsoft Visual Studio Community 2017。
3. 确保您有活动的有线以太网连接。

下载并配置 Amazon FreeRTOS

设置环境后，您可以下载 Amazon FreeRTOS。

下载 Amazon FreeRTOS

1. 在 AWS IoT 控制台中，浏览到 [Amazon FreeRTOS 页面](#)。
2. 在导航窗格中，选择 Software (软件)。
3. 在 Amazon FreeRTOS 设备软件下，选择配置下载。
4. 选择下载 FreeRTOS 软件。
5. 在软件配置列表中，找到 Windows 仿真器的连接到 AWS IoT- Windows 预定义配置，然后选择下载。
6. 将下载的文件解压缩到 AmazonFreeRTOS 文件夹，然后记录文件夹的路径。

Note

Microsoft Windows 上的文件路径最大长度为 260 个字符。Amazon FreeRTOS 下载中的最长路径为 122 个字符。为了适应 Amazon FreeRTOS 项目中的文件，请确保 AmazonFreeRTOS 目录的路径长度少于 98 个字符。例如，C:\Users\Username\Dev\AmazonFreeRTOS 可以正常工作，但 C:\Users\Username\Documents\Development\Projects\AmazonFreeRTOS 会导致生成失败。

在本教程中，目录的路径称为 AmazonFreeRTOS。

配置项目

配置网络接口

1. 在 Visual Studio 中运行项目。程序会枚举您的网络接口。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 Attempting to open interface number 1.
```

在调试器的输出中，您可能会看到 Cannot find or open the PDB file (无法找到或打开 PDB 文件)。您可以忽略这些消息。

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。

2. 打开 `<BASE_FOLDER>\demos\pc\windows\common\config_files\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。

要运行演示，您必须将项目配置为使用 AWS IoT。要将项目配置为使用 AWS IoT，必须将主板注册为 AWS IoT 事物。这是[先决条件 \(p. 4\)](#)中的一个步骤。

配置 AWS IoT 终端节点

1. 浏览至 [AWS IoT 控制台](#)。
2. 在导航窗格中，选择 Settings。

您的 AWS IoT 终端节点显示在 Endpoint (终端节点) 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。记下设备的 AWS IoT 事物名称。
4. 利用您拥有的 AWS IoT 终端节点和 AWS IoT 事物名称，在 IDE 中打开 `<BASE_FOLDER>\demos\common\include\aws_clientcredential.h`，并为以下 `#define` 常量指定值：
 - `clientcredentialMQTT_BROKER_ENDPOINT ## AWS_IOT #####`
 - `clientcredentialIOT_THING_NAME ##### AWS_IOT #####`

配置您的 AWS IoT 凭证

要配置 AWS IoT 凭证，您需要您在将设备注册为 AWS IoT 事物时从 AWS IoT 控制台下载的私有密钥和证书。在将设备注册为 AWS IoT 事物之后，可从 AWS IoT 控制台检索设备证书，但无法检索私有密钥。

Amazon FreeRTOS 是 C 语言项目，证书和私有密钥必须进行特别的格式设置才能添加到该项目中。您需要设置您的设备的证书和私有密钥的格式。

1. 在浏览器窗口中，打开 `<BASE_FOLDER>\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 `<BASE_FOLDER>\demos\common\include` 中。这将覆盖目录中的现有文件。

Note

仅应出于演示的目的来将证书和私有密钥进行硬编码。生产级应用程序应将这些文件存储在安全位置。

生成并运行 Amazon FreeRTOS 演示项目

将 Amazon FreeRTOS 演示加载到 Visual Studio 中

1. 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 `<BASE_FOLDER>\demos\pc\windows\visual_studio\aws_demos.sln`，然后选择 Open (打开)。
2. 从 Build (生成) 菜单，选择 Build Solution (生成解决方案)，确保解决方案已生成且没有错误或警告。

运行 Amazon FreeRTOS 演示

1. 重新生成 Visual Studio 项目以接受在标头文件中所做的更改。
2. 登录 [AWS IoT 控制台](#)。
3. 在导航窗格中，选择测试以打开 MQTT 客户端。
4. 在订阅主题中，输入 `freertos/demos/echo`，然后选择订阅主题。
5. 从 Visual Studio 中的 Debug (调试) 菜单，选择 Start Debugging (开始调试)。

在 [AWS IoT 控制台](#) 中，MQTT 客户端显示从 FreeRTOS Windows 仿真器收到的消息。

Nordic nRF52840-DK 入门

Amazon FreeRTOS 对 Nordic nRF52840-DK 的支持为公共测试版。BLE 演示可能会发生变化。

在开始之前，请参阅[先决条件 \(p. 4\)](#)。

如果您没有 Nordic nRF52840-DK，请访问 AWS 合作伙伴设备目录来从我们的[合作伙伴](#)购买一个。

要运行 Amazon FreeRTOS BLE 演示，您还需要具有蓝牙和 Wi-Fi 功能的 iOS 或 Android 移动设备。

设置 Nordic 硬件

将主机连接到标记了 J2 的 USB 端口（位于 Nordic nRF52840 主板上的纽扣电池座的正上方）。

有关设置 Nordic nRF52840-DK 的更多信息，请参阅[nRF52840 开发工具包用户指南](#)。

设置环境

下载并安装 Segger Embedded Studio

Amazon FreeRTOS 支持将 Segger Embedded Studio 作为 Nordic nRF52840-DK 的开发环境。

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。

下载适用于 Amazon FreeRTOS 蓝牙设备的移动开发工具包

要跨 BLE 运行 Amazon FreeRTOS 演示项目，您需要在移动设备上运行 Amazon FreeRTOS BLE 移动开发工具包演示应用程序。

设置 Amazon FreeRTOS BLE 移动开发工具包演示应用程序

1. 按照[适用于 Amazon FreeRTOS 蓝牙设备的移动开发工具包](#)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[Amazon FreeRTOS BLE 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 \(p. 7\)](#)中所述设置属性。
4. 从顶部菜单中，依次选择 Tools (工具)、Terminal Emulator (终端仿真器) 和 Connect **port** (连接 <端口>) (115200,N,8,1)。

Note

您还可以建立与所选终端工具的串行连接，例如 PuTTy、Tera Term 或 GNU Screen。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 \(p. 7\)](#)中所述。

下载并配置 Amazon FreeRTOS

设置硬件和环境后，您可以下载 Amazon FreeRTOS。

下载 Amazon FreeRTOS

要下载适用于 Nordic nRF52840-DK 的 Amazon FreeRTOS，请转至 [Amazon FreeRTOS GitHub 页面](#) 并克隆存储库。Amazon FreeRTOS BLE 库仍为公共测试版，因此，您需要切换分支以访问 Nordic nRF52840-DK 主板的代码。查看名为 `feature/ble-beta` 的分支。

Note

Microsoft Windows 上的文件路径最大长度为 260 个字符。Amazon FreeRTOS 下载中的最长路径为 122 个字符。为了适应 Amazon FreeRTOS 项目中的文件，请确保 `AmazonFreeRTOS` 目录的路径长度少于 98 个字符。例如，`C:\Users\Username\Dev\AmazonFreeRTOS` 可以正常工作，但 `C:\Users\Username\Documents\Development\Projects\AmazonFreeRTOS` 会导致生成失败。

在本教程中，目录的路径称为 `AmazonFreeRTOS`。

配置项目

要运行演示，您必须将项目配置为使用 AWS IoT。要将项目配置为使用 AWS IoT，必须将主板注册为 AWS IoT 事物。这是[先决条件 \(p. 4\)](#)中的一个步骤。

配置 AWS IoT 终端节点

1. 浏览至 [AWS IoT 控制台](#)。
2. 在导航窗格中，选择 Settings。

您的 AWS IoT 终端节点显示在 Endpoint (终端节点) 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。记下设备的 AWS IoT 事物名称。
4. 利用您拥有的 AWS IoT 终端节点和 AWS IoT 事物名称，在 IDE 中打开 `<BASE_FOLDER>\demos\common\include\aws_clientcredential.h`，并为以下 `#define` 常量指定值：
 - `clientcredentialMQTT_BROKER_ENDPOINT ## AWS_IOT #####`
 - `clientcredentialIOT_THING_NAME ##### AWS_IOT #####`

启用演示

1. 确保 BLE GATT 演示已启用。转至 `<BASE_FOLDER>\demos\nordic\nrf52840-dk\common\config_files\aws_ble_config.h`，并确保将 `bleconfigENABLE_GATT_DEMO` 设置为 1。

2. 打开 `<BASE_FOLDER>\demos\common\demo_runner\aws_demo_runner.c`，在演示声明中取消注释 `extern void vStartMQTTLIEchoDemo(void);`。在 `DEMO_RUNNER_RunDemos` 定义中，取消注释 `vStartMQTTLIEchoDemo();`。

生成并运行 Amazon FreeRTOS 演示项目

在下载 Amazon FreeRTOS 并配置演示项目后，可以在主板上构建和运行演示项目。

从 Segger Embedded Studio 生成并运行 Amazon FreeRTOS BLE 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 `<BASE_FOLDER>\demos\nordic\nrf52840-dk\ses\aws_demos_ble.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 Project Explorer (项目浏览器) 中右键单击 `aws_ble_demos` 演示项目，然后选择 Build (生成)。

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。选择 `Activate Your Free License` (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)。
5. 按照 [MQTT over BLE 演示应用程序](#) 中的说明操作，将 Amazon FreeRTOS BLE 移动开发工具包演示应用程序作为移动 MQTT 代理完成演示。

Amazon FreeRTOS 开发人员指南

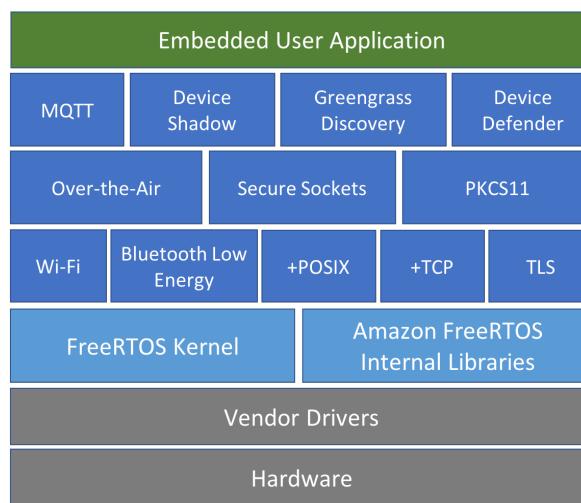
本部分包含了编写用于 Amazon FreeRTOS 的嵌入式应用程序时所需的信息。

主题

- [Amazon FreeRTOS 架构 \(p. 66\)](#)
- [FreeRTOS 内核基础知识 \(p. 66\)](#)
- [Amazon FreeRTOS 库 \(p. 70\)](#)
- [Amazon FreeRTOS 无线更新 \(p. 118\)](#)
- [Amazon FreeRTOS 控制台 \(p. 161\)](#)
- [Amazon FreeRTOS 问题排查 \(p. 163\)](#)

Amazon FreeRTOS 架构

Amazon FreeRTOS 设计用于嵌入式微控制器。它通常作为单个已编译映像，与设备应用程序所需的所有组件一起，刷入到设备。此映像中结合了嵌入式开发人员针对该应用程序编写的功能、Amazon 提供的软件库、FreeRTOS 内核，以及适用于硬件平台的驱动程序和板卡支持程序包 (BSP)。不论使用的是何种微处理器，对于 FreeRTOS 内核和所有 Amazon FreeRTOS 软件库，嵌入式应用程序开发人员均可以采用相同的标准化接口。



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。它是构建嵌入式微控制器应用程序的理想之选。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。

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）。已阻止的任务不需要耗时的

定期维护。“直接到任务”通知可实现快速的任务信号发送，几乎没有 RAM 开销。它们可用于大多数任务间信号发送以及“中断到任务”信号发送场景。

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。

FreeRTOS 内核计划程序

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。

内存管理

本部分提供了有关内核内存分配和应用程序内存管理的信息。

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()`。

应用程序内存管理

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。

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 API 允许指定阻止时间。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。

在许多常见的设计场景中，其他 FreeRTOS 基元（如“直接到任务”通知以及流和消息缓冲区）可作为队列的轻型替代方案。

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。

“直接到任务”通知

通过任务通知，任务可以与其他任务进行交互，并与中断服务例程 (ISR) 同步，且无需单独的通信对象（如信号灯）。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。要启用流缓冲区功能，请将 `<BASE_DIR>/libs/FreeRTOS/stream_buffer.c` 源文件包括在您的项目中。

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。

流缓冲区实施采用的是“直接到任务”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，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。要启用消息缓冲区功能，请将 `<BASE_DIR>/libs/FreeRTOS/stream_buffer.c` 源文件包括在您的项目中。

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。

消息缓冲区实施采用的是“直接到任务”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，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。

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。

Amazon FreeRTOS 库

Amazon FreeRTOS 库为 FreeRTOS 内核及其内部库提供其他功能。您可以使用 Amazon FreeRTOS 库在嵌入式应用程序中实现联网和安全性。Amazon FreeRTOS 库还使您的应用程序能够与 AWS IoT 服务进行交互。

您可以从 [Amazon FreeRTOS 控制台](#) 下载为符合 Amazon FreeRTOS 要求的平台配置的 Amazon FreeRTOS 版本。有关合格平台的列表，请参阅 [Amazon FreeRTOS 合作伙伴](#) 网站。GitHub 上也提供了 Amazon FreeRTOS。

Amazon FreeRTOS 移植库

以下移植库包含在可从 Amazon FreeRTOS 控制台下载的 Amazon FreeRTOS 的配置中。这些库与平台相关。其内容因硬件平台而异。

Amazon FreeRTOS 移植库

library

参考

例

用

Amazon

FreeRTOS

低

(BLE)

耗

蓝

牙

(BLE)

库 ,

您

的

微

控

制

器

可

通

过

网

关

设

备

与

AWS

IoT

MQTT

代

理

进

行

通

信

。有

关

更

多

信

息

，

请

参

Library

参
考

阅

Amazon
FreeRTOS

低
功
耗

蓝
牙

库

(测
试

版) (p. 84)。

Note

Amazon
FreeRTOS

BLE

库

为

公

共

测

试

版。

Library
参
考

Amazon
FreeRTOS
AWS
IoT
无线
(OTA)
代理
库将您的
Amazon
FreeRTOS
设备连接到
AWS
IoT
OTA
代理。

有关更多信息，请参阅
[Amazon
FreeRTOS
无线
\(OTA\)
代理
库 \(p. 104\)](#)。

Library
参
考

FreeRTOS

+POSIX

API

使

用

FreeRTOS

+POSIX

库

将

与

POSIX

兼

容

的

应

用

程

序

移

植

到

Amazon

FreeRTOS

生

态

系

统。

有关更多信息，请参阅

FreeRTOS

+POSIX。

Library
参
考

箱
接
插
/息|
請
參
閱

Amazon
FreeRTOS

安
全
套
接
字
库 (p. 108)。

Library
参
考

FreeRTOS

+TCP

是 |

参

考

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。

Library

参考

例

用

Amazon

FreeRTOS

Wi-

Fi

库 ,

您

可

以

与

微

控

制

器

的

低

级

别

无

线

堆

栈

进

行

交

互。

Library
参考

PKCS#11

PKCS

#11

库

是公

有密

钥加

密标

准#11

的参

考实

施，可

支持预

配置和

TLS

客

户端

身

份验

证。

有关更多信
息，请参阅

Amazon
FreeRTOS

公

有密

钥加

密

Library
参考
标准
(PKCS)
#11
库 (p. 106)。

TLS
关
更多信
息，请参
阅
Amazon
FreeRTOS
传
输
层
安
全
性
(TLS) (p. 114)。

Amazon FreeRTOS 应用程序库

您可以选择在 Amazon FreeRTOS 配置中包含以下独立应用程序库以便与 AWS IoT 进行交互。

Amazon FreeRTOS 应用程序库

Library
参考
Greengrass
Amazon
FreeRTOS
AWS
IoT
Greengrass
库
将您的
Amazon
FreeRTOS
设备连接到
AWS

Library

参
考

IoT

Greengrass。

有关
更多信
息，请
参阅

Amazon
FreeRTOS
AWS
IoT
Greengrass
Discovery
库 (p. 97)。

Library
参
考

Amazon
FreeRTOS

MQTT

奔
勃

Amazon
FreeRTOS

设
备

MQTT

提
供

API

用
于

发
布

(旧
版)

和

MQTT

API

阅
读

MQTT

主
(测)

题
(测)

的
(测)

客
户

端。MQTT

是

设备

用
来

与

AWS

IoT

进
行

交
互

的

协
议。

有
关

传
统

Amazon

FreeRTOS

MQTT

库

的
更
多

Library

参
考

信
息
,

请
参
阅

Amazon
FreeRTOS
MQTT
库
(传
统) (p. 101)。

有
关
新

Amazon
FreeRTOS
MQTT
库
(公
共
测
试
版)

的
更
多
信
息
,

请
参

阅Amazon
FreeRTOS
MQTT
库
(测
试
版) (p. 99)。

Library
参
考

Device

Shadow

AWS

IoT

Device

Shadow

库 ,

您

的

Amazon

FreeRTOS

设

备可

以

与

AWS

IoT

设

备影

子进

行交

互。

有关更多信息 ,
请参阅

Amazon

FreeRTOS

AWS

IoT

Device

Shadow

库 (p. 112)。

Library
参
考

Amazon
FreeRTOS
AWS
IoT
Device
Defender
库
将
您
的
Amazon
FreeRTOS
设备
连接
到
AWS
IoT
Device
Defender。

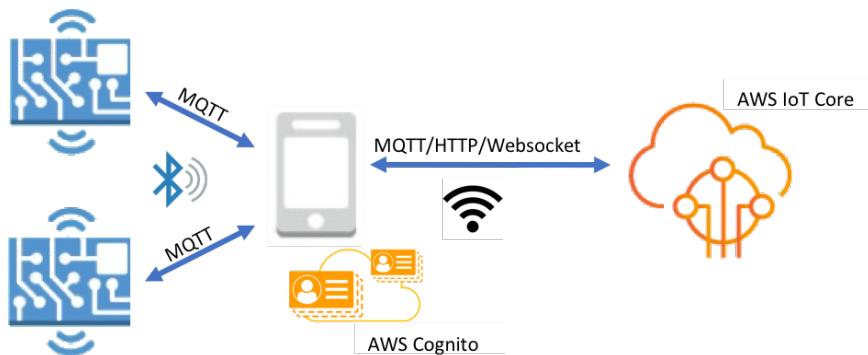
有关
更多信
息 ,
请参阅
Amazon
FreeRTOS
AWS
IoT
Device
Defender
库 (p. 94)。

Amazon FreeRTOS 低功耗蓝牙库 (测试版)

概述

低功耗蓝牙 (BLE) 库是面向 Amazon FreeRTOS 的公共测试版，可能会发生变化。

Amazon FreeRTOS 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。利用 Amazon FreeRTOS BLE 库，您的微控制器可以安全地与 AWS IoT MQTT 代理进行通信。

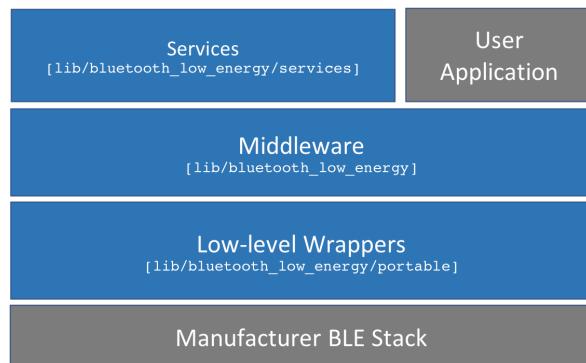


通过适用于 Amazon FreeRTOS 蓝牙设备的移动开发工具包，您可以编写本机移动应用程序以通过 BLE 与微控制器上的嵌入式应用程序进行通信。有关移动开发工具包的更多信息，请参阅[适用于 Amazon FreeRTOS 蓝牙设备的移动开发工具包 \(p. 93\)](#)。

除了支持 MQTT 之外，Amazon FreeRTOS BLE 库还包含用于配置 Wi-Fi 网络的服务。Amazon FreeRTOS BLE 库还包括一些中间件和低级别 API 以便更直接地控制 BLE 堆栈。Amazon FreeRTOS BLE 库的源文件位于 [AmazonFreeRTOS/lib/bluetooth_low_energy](#) 中。

Amazon FreeRTOS BLE 架构

Amazon FreeRTOS BLE 库包含三个层：服务、中间件和低级别包装程序。



服务

Amazon FreeRTOS BLE 服务层包含三个通用属性 (GATT) 服务，这些服务使用了中间件 API：设备信息、Wi-Fi 预配置以及通过 BLE 的 MQTT 通信。有关更多信息，请参阅[服务 \(p. 86\)](#)。

中间件

Amazon FreeRTOS BLE 中间件是来自低级别 API 的抽象层。中间件 API 构成了一个面向 BLE 堆栈的更方便用户使用的界面。有关更多信息，请参阅[中间件 \(p. 86\)](#)。

低级别包装程序

低级别 Amazon FreeRTOS BLE 包装程序是来自制造商的 BLE 堆栈的抽象层。低级别包装程序提供了一组可直接控制硬件的通用 API。低级别 API 优化了 RAM 的使用，但功能有限。要使用 Amazon FreeRTOS BLE 服务，您可以与 BLE 服务 API 进行交互，这将需要比低级别 API 更多的资源。

依赖项和要求

仅 MQTT over BLE 和 Wi-Fi 预配置服务具有库依赖项。

GATT 服务	依赖关系
MQTT over BLE	Amazon FreeRTOS MQTT 库 (测试版) (p. 99)
Wi-Fi 预配置	Amazon FreeRTOS Wi-Fi 库 (p. 114)

要与 AWS IoT MQTT 代理进行通信，您必须有一个 AWS 账户，并且您必须将设备注册为 AWS IoT 事物。有关设置的更多信息，请参阅 [AWS IoT 开发人员指南](#)。

Amazon FreeRTOS BLE 使用 Amazon Cognito 在您的移动设备上进行用户身份验证。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 Identity 必须附加了适当的策略。有关更多信息，请参阅 [Amazon Cognito 开发人员指南](#)。

功能

服务

设备信息

设备信息服务可收集有关微控制器的信息，其中包括：

- 您的设备正在使用的 Amazon FreeRTOS 的版本。
- 为其注册的账户的 AWS IoT 终端节点。
- BLE 最大传输单元 (MTU)。

Wi-Fi 预配置

利用 Wi-Fi 预配置服务，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。

MQTT over BLE

MQTT over BLE 服务将微控制器连接到支持蓝牙的移动设备，以使用 AWS 移动开发工具包间接连接到 AWS IoT 云。微控制器充当 MQTT 客户端，移动设备充当 MQTT 代理，AWS IoT 云充当 MQTT 服务器。

中间件

利用中间件 API，您可以跨多个层将多个回调注册到单个事件。

灵活的回调订阅

假设您的 BLE 硬件断开连接，并且 MQTT over BLE 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。BLE 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。

源和标头文件

下面的树形图显示了所需的源和标头文件以及这些文件在 Amazon FreeRTOS 目录结构中的位置。项目还必须构建相关库的源文件。

```
|  
+ - lib  
  + - bluetooth_low_energy  
    | + - aws_ble_event_manager.c  
    | + - aws_ble_gap.c [Middleware GAP]  
    | + - aws_ble_gatt.c [Middleware GATT]  
    | + - portable [Wrappers, wrapping APIs in lib/include/bluetooth_low_energy]  
      + - services  
        + - device_information [Service providing device info to the phone APP]  
          | + - aws_ble_device_information.c  
        + - mqtt_ble [Used to do MQTT over BLE]  
          | + - aws_mqtt_proxy.c  
        + - wifi_provisioning [WIFI provisioning service over BLE]  
          + - aws_ble_wifi_provisioning.c  
+ - include  
  + - bluetooth_low_energy [Wrapping APIs in lib/include/bluetooth_low_energy]  
    | + - bt_hal_avsrc_profile.h  
  + # bt_hal_gatt_client.h  
  + # bt_hal_gatt_server.h  
  + # bt_hal_gatt_types.h  
  + # bt_hal_manager_adapter_ble.h  
  + # bt_hal_manager_adapter_classic.h  
  + # bt_hal_manager.h  
  + # bt_hal_manager_types.h  
  + - private [For internal library use only!]  
    | + - aws_ble_internals.h  
    | + - aws_ble_config_defaults.h  
    | + - aws_ble_event_manager.h  
  + - aws_ble.h  
  + - aws_ble_device_information.h  
  + - aws_ble_services_init.h  
  + - aws_ble_wifi_provisioning.h
```

Amazon FreeRTOS BLE 库配置文件

使用 Amazon FreeRTOS MQTT over BLE 服务的应用程序必须提供一个定义了配置参数的 `aws_ble_config.h` 标头文件。未定义的配置参数将采用 `lib\include\private\aws_ble_config_defaults.h` 中指定的默认值。

优化

在优化主板性能时，请注意以下事项：

- 低级别 API 占用的 RAM 更少，但它提供的功能有限。
- 您可以将 `aws_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。
- 您可以删除未使用的服务以节省 RAM。
- 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，Amazon FreeRTOS BLE 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。

要使用较低的 LE 安全级别，请将 `eBTpropertySecureConnectionOnly` 设置为 FALSE，方式是使用属性 `eBTpropertySecureConnectionOnly` 调用 API `pxSetDeviceProperty`。

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 BLE 堆栈交互，则您只需初始化中间件即可。

中间件

中间件负责初始化堆栈的较低层。

初始化中间件

1. 在调用 BLE 中间件 API 之前，您必须初始化所有 BLE 硬件驱动程序。
2. 启用 BLE。

```
const BTInterface_t * pxIface = BTGetBluetoothInterface();
xStatus = pxIface->pxEnable( 0 );
```

3. 要初始化 BLE，请使用一组所需的属性（例如，安全连接模式、设备名称和 MTU 大小）调用 `BLE_Init`。

```
xStatus = BLE_Init( &xServerUUID, xDeviceProperties, MAX_PROPERTIES );
```

低级别 API

如果您不想使用 Amazon FreeRTOS BLE GATT 服务，则可绕过中间件，并直接与低级别 API 交互以节省资源。

初始化低级别 API

1. 驱动程序初始化不是 BLE 低级别 API 的一部分。在调用 API 之前，您必须初始化所有 BLE 硬件驱动程序。
2. BLE 低级别 API 提供了对 BLE 堆栈的启用/禁用调用以优化能力和资源。在调用 API 之前，您必须启用 BLE。

```
const BTInterface_t * pxIface = BTGetBluetoothInterface();
xStatus = pxIface->pxEnable( 0 );
```

3. 蓝牙管理器包含 BLE 和蓝牙经典功能的通用 API。常见管理器的回调必须是第二个初始化的。

```
xStatus = xBTInterface.pxBTInterface->pxBtManagerInit( &xBTManagerCb );
```

4. BLE 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 = ( BTLeAdapter_t * ) xBTInterface.pxBTInterface-
>pxGetLeAdapter();
xStatus = xBTInterface.pxBTLeAdapterInterface->pxBleAdapterInit( &xBTLeAdapterCb );
```

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 BLE 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

有关完全 API 参考，请参阅 [低功耗蓝牙 \(BLE\) API 参考](#)。

示例用法

广告

1. 设置广告参数。

```
BLEAdvertisementParams_t xAdvParams =  
{  
    .bIncludeTxPower      = true,  
    .bIncludeName         = true,  
    .bSetScanRsp          = true,  
    .ulAppearance          = 0,  
    .ulMinInterval        = 0x20,  
    .ulMaxInterval        = 0x40,  
    .usManufacturerLen    = 0,  
    .pcManufacturerData   = NULL,  
    .pxUUID1              = &xDeviceInfoSvcUUID,  
    .pxUUID2              = NULL  
};  
  
if( xStatus == eBTStatusSuccess )  
{  
    ( void ) BLE_SetAdvData( BTAdvInd, &xAdvParams, vSetAdvCallback );  
}
```

2. 启动广告。

```
void vSetAdvCallback ( BTStatus_t xStatus )  
{  
    if( xStatus == eBTStatusSuccess )  
    {  
        ( void ) BLE_StartAdv( vStartAdvCallback );  
    }  
}
```

添加新服务

1. 为新服务分配内存。

```
xStatus = BLE_CreateService( &pxGattDemoService, gattDemoNUM_CHARS,  
    gattDemoNUM_CHAR_DESCRS, xNumDescrsPerChar, gattDemoNUM_INCLUDED_SERVICES );
```

2. 创建服务。

```
pxGattDemoService->xAttributeData.xUuid = xServiceUUID;  
  
pxGattDemoService->pxDescriptors[ egattDemoCharCounterCCFGDESCR ].xAttributeData.xUuid  
    = xClientCharCfgUUID;  
pxGattDemoService->pxDescriptors[ egattDemoCharCounterCCFGDESCR ].xAttributeData.pucData = NULL;  
pxGattDemoService->pxDescriptors[ egattDemoCharCounterCCFGDESCR ].xAttributeData.xSize  
    = 0;  
pxGattDemoService->pxDescriptors[ egattDemoCharCounterCCFGDESCR ].xPermissions =  
    ( eBTPermReadEncryptedMitm | eBTPermWriteEncryptedMitm );  
pxGattDemoService->pxDescriptors[ egattDemoCharCounterCCFGDESCR ].pxAttributeEventCallback =  
    vEnableNotification;  
  
xCharUUID.uu.uu16 = gattDemoCHAR_COUNTER_UUID;  
pxGattDemoService->pxCharacteristics[ egattDemoCharCounter ].xAttributeData.xUuid =  
    xCharUUID;  
pxGattDemoService->pxCharacteristics[ egattDemoCharCounter ].xAttributeData.pucData =  
    NULL;  
pxGattDemoService->pxCharacteristics[ egattDemoCharCounter ].xAttributeData.xSize = 0;  
pxGattDemoService->pxCharacteristics[ egattDemoCharCounter ].xPermissions =  
    ( eBTPermRead );  
pxGattDemoService->pxCharacteristics[ egattDemoCharCounter ].xProperties =  
    ( eBTPropRead | eBTPropNotify );  
pxGattDemoService->pxCharacteristics[ egattDemoCharCounter ].pxAttributeEventCallback =  
    vReadCounter;  
pxGattDemoService->pxCharacteristics[ egattDemoCharCounter ].xNbDescriptors = 1;  
pxGattDemoService->pxCharacteristics[ egattDemoCharCounter ].pxDescriptors[ 0 ] =  
    &pxGattDemoService->pxDescriptors[ egattDemoCharCounterCCFGDESCR ];  
  
xCharUUID.uu.uu16 = gattDemoCHAR_CONTROL_UUID;  
pxGattDemoService->pxCharacteristics[ egattDemoCharControl ].xAttributeData.xUuid =  
    xCharUUID;  
pxGattDemoService->pxCharacteristics[ egattDemoCharControl ].xAttributeData.pucData =  
    NULL;  
pxGattDemoService->pxCharacteristics[ egattDemoCharControl ].xAttributeData.xSize = 0;  
pxGattDemoService->pxCharacteristics[ egattDemoCharControl ].xPermissions =  
    ( eBTPermReadEncryptedMitm | eBTPermWriteEncryptedMitm );  
pxGattDemoService->pxCharacteristics[ egattDemoCharControl ].xProperties =  
    ( eBTPropRead | eBTPropWrite );  
pxGattDemoService->pxCharacteristics[ egattDemoCharControl ].pxAttributeEventCallback =  
    vWriteCommand;  
pxGattDemoService->pxCharacteristics[ egattDemoCharControl ].xNbDescriptors = 0;  
pxGattDemoService->pxCharacteristics[ egattDemoCharControl ].pxDescriptors = NULL;  
  
pxGattDemoService->xServiceType = eBTServiceTypePrimary;  
pxGattDemoService->ucInstId = 0;  
  
xStatus = BLE_AddService( pxGattDemoService );
```

3. 启动服务。

```
xStatus = BLE_StartService( pxGattDemoService, vServiceStartedCb );
```

4. 订阅服务所需的任何事件。在此示例中，我们订阅连接事件。

```
xCallback.pxConnectionCb = vConnectionCallback;
```

```
BLE_RegisterEventCb( eBLEConnection, xCallback );
```

有关所有 Amazon FreeRTOS BLE 演示应用程序，请参阅[低功耗蓝牙演示应用程序](#)。

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 Amazon FreeRTOS 移植到新目标，您必须为 Wi-Fi 预配置服务和 BLE 功能实施一些 API。

Wi-Fi 预配置 API

要使用 Wi-Fi 预配置服务，您必须实施以下 API：

- `WIFI_NetworkGet`
- `WIFI_NetworkDelete`
- `WIFI_NetworkAdd`

BLE API

要使用 Amazon FreeRTOS BLE 中间件，您必须实施一些 API。

GAP for Bluetooth Classic 和 GAP for BLE 的通用 API

- `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

特定于 GAP for BLE 的 API

- 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

适用于 Amazon FreeRTOS 蓝牙设备的移动开发工具包

低功耗蓝牙 (BLE) 库是面向 Amazon FreeRTOS 的公共测试版，可能会发生变化。

您可以使用适用于 Amazon FreeRTOS 蓝牙设备的移动开发工具包来创建通过 BLE 与微控制器交互的移动应用程序。移动开发工具包也可以与 AWS 服务通信，使用 Amazon Cognito 进行用户身份验证。

适用于 Amazon FreeRTOS 蓝牙设备的 Android 开发工具包

使用适用于 Amazon FreeRTOS 蓝牙设备的 Android 开发工具包来生成通过 BLE 与微控制器交互的 Android 移动应用程序。该开发工具包在 [GitHub](#) 上提供。

安装 Android 开发工具包

1. 从 [GitHub](#) 中下载开发工具包。
2. 打开 Android Studio 并将 amazon-freertos-ble-android-sdk/amazonfreertosdk/ 目录导入到您的应用程序项目。
3. 在应用程序的 gradle 文件中，添加以下依赖项：

```
dependencies {  
    implementation project(":amazonfreertosdk")  
}
```

4. 在应用程序的 AndroidManifest.xml 文件中，添加以下权限：

```
<uses-permission android:name="android.permission.BLUETOOTH"/>  
    <!-- initiate device discovery and manipulate bluetooth settings -->  
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>  
    <!-- allow scan BLE -->  
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />  
  
    <!-- AWS Mobile SDK -->  
<uses-permission android:name="android.permission.INTERNET" />  
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

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 \(p. 165\)](#)和[Amazon FreeRTOS BLE 移动开发工具包演示应用程序 \(p. 167\)](#)。

适用于 Amazon FreeRTOS 蓝牙设备的 iOS 开发工具包

使用适用于 Amazon FreeRTOS 蓝牙设备的 iOS 开发工具包来生成通过 BLE 与微控制器交互的 iOS 移动应用程序。该开发工具包在 [GitHub](#) 上提供。

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` 安装 CocoaPods。

2. 使用 CocoaPods 安装开发工具包：

```
$ pod 'AmazonFreeRTOS', :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 \(p. 165\)](#)和[Amazon FreeRTOS BLE 移动开发工具包演示应用程序 \(p. 167\)](#)。

Amazon FreeRTOS AWS IoT Device Defender 库

概述

AWS IoT Device Defender 是一项 AWS IoT 服务，您可以借助此服务审核设备的配置，监控连接的设备以检测异常行为，并降低安全风险。有了此服务，您可以在整个 AWS IoT 设备机群中实施一致的 IoT 配置，并能够在设备遭到破坏时快速响应。

Amazon FreeRTOS 提供了库，允许基于 Amazon FreeRTOS 的设备使用 AWS IoT Device Defender。可以使用 [Amazon FreeRTOS 控制台](#)，将 Device Defender 库添加到软件配置中，从而下载 Amazon FreeRTOS Device Defender 库。也可以克隆 Amazon FreeRTOS GitHub 存储库，然后在 `lib` 目录中查找库。

Amazon FreeRTOS AWS IoT Device Defender 库的源文件位于 `AmazonFreeRTOS/lib/defender` 中。

源和标头文件

```
Amazon FreeRTOS
|
+ - lib
  |
  + - defender
    |
    + # aws_defender.c
    + # aws_defender_states.dot
    + # aws_defender_states.png
    + # draw_states.py
    + # portable
      |
      + # freertos
        |
        + # aws_defender_cpu.c
        + # aws_defender_tcp_conn.c
        + # aws_defender_uptime.c
      + # stub
        |
        + # aws_defender_cpu.c
        + # aws_defender_tcp_conn.c
        + # aws_defender_uptime.c
      + # makefile
    + # template
      |
      + # aws_defender_cpu.c
      + # aws_defender_tcp_conn.c
      + # aws_defender_uptime.c
      + # makefile
    + # unit_test
      |
      + # aws_defender_cpu.c
```

```
| | | + # aws_defender_tcp_conn.c
| | | + # aws_defender_uptime.c
| | + # unix
| | | + # aws_defender_cpu.c
| | | + # aws_defender_tcp_conn.c
| | | + # aws_defender_uptime.c
| | | + # makefile
| + # report
| | + # aws_defender_report.c
| | + # aws_defender_report_cpu.c
| | + # aws_defender_report_header.c
| | + # aws_defender_report_tcp_conn.c
| | + # aws_defender_report_uptime.c
+ - include
+ - aws_defender.h
+ - private
+ - aws_defender_cpu.h
+ - aws_defender_internals.h
+ - aws_defender_report_cpu.h
+ - aws_defender_report.h
+ - aws_defender_report_header.h
+ - aws_defender_report_tcp_conn.h
+ - aws_defender_report_types.h
+ - aws_defender_report_uptime.h
+ - aws_defender_report_utils.h
+ - aws_defender_tcp_conn.h
+ - aws_defender_uptime.h
```

开发人员支持

Amazon FreeRTOS Device Defender API 错误代码

eDefenderErrSuccess

操作成功。

eDefenderErrFailedToCreateTask

操作无法启动。

eDefenderErrAlreadyStarted

操作正在进行中。

eDefenderErrNotStarted

Device Defender 代理尚未启动。

eDefenderErrOther

出现未指定的错误。

Amazon FreeRTOS Device Defender API

本部分包含了有关 Device Defender API 的信息。

DEFENDER_MetricsInit

指定设备将发送到 AWS IoT Device Defender 的 Device Defender 指标。

```
DefenderErr_t DEFENDER_MetricsInit(DefenderMetric_t * pxMetricsList);
```

参数

metrics_list

Device Defender 指标列表。有效值为：

- DEFENDER_tcp_connections – 跟踪 TCP 连接的数量。

返回值

返回 DefenderErr_t 枚举中的一个。有关更多信息，请参阅[Amazon FreeRTOS Device Defender API 错误代码 \(p. 95\)](#)。

DEFENDER_ReportPeriodSet

设置报告周期时间间隔（以秒为单位）。Device Defender 按时间间隔提供指标报告。如果设备处于唤醒状态，且时间间隔已过，则设备将报告指标。

```
DefenderErr_t DEFENDER_ReportPeriodSet(int32_t LPeriodSec);
```

参数

period_sec

在此时间（以秒计）后将报告发送到 AWS IoT Device Defender。

返回值

返回 DefenderErr_t 枚举中的一个。有关更多信息，请参阅[Amazon FreeRTOS Device Defender API 错误代码 \(p. 95\)](#)。

DEFENDER_Start

启动 Device Defender 代理。

```
DefenderErr_t DEFENDER_Start(void);
```

返回值

返回 DefenderErr_t 枚举中的一个。有关更多信息，请参阅[Amazon FreeRTOS Device Defender API 错误代码 \(p. 95\)](#)。

DEFENDER_Stop

停止 Device Defender 代理。

```
DefenderErr_t DEFENDER_Stop(void);
```

返回值

返回 DefenderErr_t 枚举中的一个。有关更多信息，请参阅[Amazon FreeRTOS Device Defender API 错误代码 \(p. 95\)](#)。

DEFENDER_ReportStatusGet

获取最后一个 Device Defender 报告的状态。有效的状态代码值为：

eDefenderRepSuccess

最后一个报告已成功发送并已确认。

eDefenderRepInit

Device Defender 已启动，但尚未发送报告。

eDefenderRepRejected

最后一个报告已拒绝。

eDefenderRepNoAck

最后一个报告未确认。

eDefenderRepNotSent

最后一个报告未发送，可能存在连接问题。

```
DefenderReportStatus_t DEFENDER_ReportStatusGet(void);
```

示例用法

在嵌入式应用程序中使用 Device Defender

下面的代码演示了如何在嵌入式应用程序中配置并启动 Device Defender 代理：

```
void MyDefenderInit(void)
{
    // Specify metrics to send to Device Defender
    defender_metric_t metrics_list[] = {
        DEFENDER_tcp_connections
    };
    ( void ) DEFENDER_MetricsInit( metrics_list );

    // Set the reporting interval
    // You can use a shorter period to trigger the violation faster, however
    // the Device Defender service is not guaranteed to accept reports faster
    // than every 300 seconds (5 minutes) per device.
    int report_period_sec = 300;
    ( void ) DEFENDER_ReportPeriodSet( report_period_sec );

    // Start the Device Defender agent
    DEFENDER_Start();
}
```

Amazon FreeRTOS AWS IoT Greengrass Discovery 库

概述

微控制器使用 AWS IoT Greengrass Discovery 库来发现网络上的 Greengrass 核心。通过使用 AWS IoT Greengrass Discovery API，设备可以在找到核心的终端节点之后将消息发送到 Greengrass 核心。

Amazon FreeRTOS AWS IoT Greengrass 库的源文件位于 [AmazonFreeRTOS/lib/greengrass](#) 中。

依赖项和要求

要使用 Greengrass Discovery 库，您必须在 AWS IoT 中创建事物，包括证书和策略。有关更多信息，请参阅 [AWS IoT 入门](#)。必须为 AmazonFreeRTOS\demos\common\include\aws_client_credentials.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。

`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 [AWS IoT Greengrass 入门](#)。

尽管 Greengrass 连接不需要 MQTT 库，我们仍强烈建议您安装它。该库可用于在发现 Greengrass 核心后与其进行通信。

源和标头文件

```
Amazon FreeRTOS
|
+ - lib
  + - greengrass
    |  + # aws_greengrass_discovery.c
    |  + # aws_helper_secure_connect.c
  + - include
    + - aws_greengrass_discovery.h
    + - private
      + - aws_ggd_config_defaults.h
```

API 参考

有关完全 API 参考，请参阅 [Greengrass API 参考](#)。

示例用法

Greengrass 工作流

MCU 设备向 AWS IoT 请求包含 Greengrass 核心连接参数的 JSON 文件，以启动发现过程。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 `lib\include\private\aws_ggd_config_defaults.h` 中定义。您可以在 `lib\include\` 中覆盖这些设置中的任意一个。

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。

对于更多自定义选项，如动态分配证书，必须调用以下 API：

`GGD_JSONRequestStart`

向 AWS IoT 发出 HTTP GET 请求，以启动发现请求来查找 Greengrass 核心。`GD_SecureConnect_Send` 用于将请求发送给 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。

Amazon FreeRTOS MQTT 库 (测试版)

新的 MQTT 库是面向 Amazon FreeRTOS 的公共测试版，可能会发生变化。

概述

您可以使用 Amazon FreeRTOS MQTT 库来创建发布和订阅 MQTT 主题的应用程序，就像网络上的 MQTT 客户端。Amazon FreeRTOS MQTT 库实施 MQTT 3.1.1 标准以便与 [AWS IoT MQTT 服务器](#) 兼容。此库还可与其他 MQTT 服务器兼容。

Amazon FreeRTOS MQTT 库的源文件位于 [AmazonFreeRTOS/lib/mqtt](#) 中。

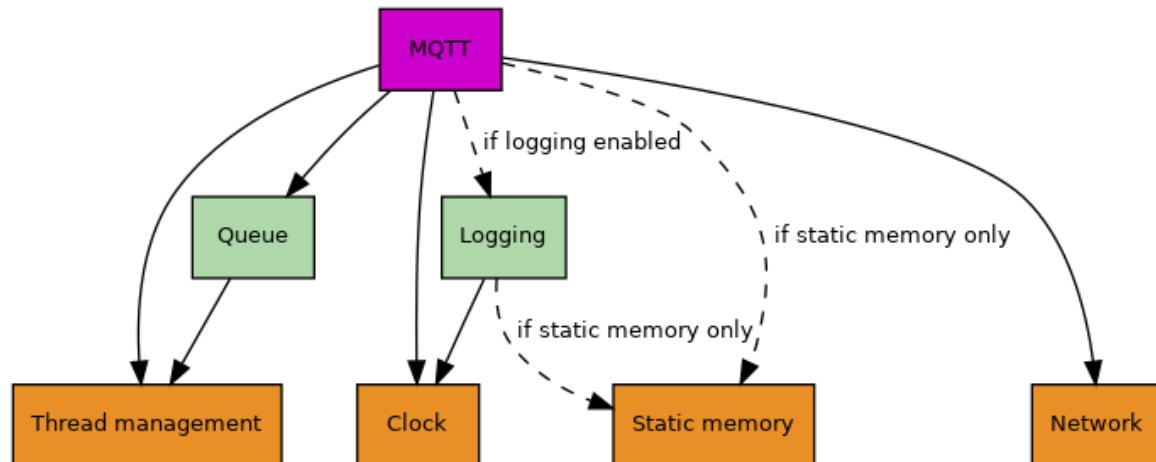
此处说明的 Amazon FreeRTOS MQTT 库为公共测试版。有关传统 Amazon FreeRTOS MQTT 库的更多信息，请参阅 [Amazon FreeRTOS MQTT 库 \(传统 \) \(p. 101\)](#)。

依赖项和要求

Amazon FreeRTOS MQTT 库具有以下依赖项：

- 用于维护数据结构（这些结构管理正在进行的 MQTT 操作）的队列库
- 日志记录库，如果配置参数 AWS_IOT_MQTT_LOG_LEVEL 未设置为 AWS_IOT_LOG_NONE
- 平台层，提供了一个操作系统接口，面向线程管理、时钟函数、联网和其他平台级功能
- C 标准库标头

下图演示了这些依赖项。



功能

Amazon FreeRTOS MQTT 库具有以下功能：

- 默认情况下，该库具有完全异步的 MQTT API。您可以选择以同步方式将此库与 AwsIotMqtt_Wait 函数结合使用。
- 此库是线程感知的，可并行化以实现高吞吐量。
- 此库具有可扩展的性能和占用空间。使用配置设置来根据系统资源定制库。

配置

Amazon FreeRTOS MQTT 库的配置设置定义为 C 预处理程序常量。在名为 AWS_IOT_CONFIG_FILE 的文件中或使用编译器选项（例如 gcc 中的 -D）将配置设置设为 #define 常量。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。如果未定义配置设置，则 MQTT 库使用默认值。

有关配置 Amazon FreeRTOS MQTT 库的更多信息，请参阅 [MQTT API 参考（测试版）](#)。

API 参考

有关完全 API 参考，请参阅 [MQTT API 参考（测试版）](#)。

示例用法

`aws_iot_demo_mqtt.c`

有关 Amazon FreeRTOS MQTT 库的示例用法，请参阅 `aws_iot_demo_mqtt.c` 中定义的 MQTT 演示应用程序。

MQTT 演示说明了 MQTT 的订阅-发布工作流程。在订阅多个主题筛选条件后，应用程序会将数据突增发布到各种主题名称。当每条消息到达时，演示会将确认消息发送回 MQTT 服务器。

要运行 MQTT 演示，您需要配置以下参数：

全局演示配置参数

这些配置参数适用于所有演示。

AWS_IOT_DEMO_SECURED_CONNECTION

确定演示在默认情况下是否将 TLS 安全连接用于远程主机。

AWS_IOT_DEMO_SERVER

要使用的默认远程主机。

AWS_IOT_DEMO_PORT

要使用的默认远程端口。

AWS_IOT_DEMO_ROOT_CA

要使用的默认可信服务器根证书的路径。

AWS_IOT_DEMO_CLIENT_CERT

要使用的默认客户端证书的路径。

AWS_IOT_DEMO_PRIVATE_KEY

要使用的默认客户端证书私有密钥的路径。

MQTT 演示配置参数

这些配置参数适用于 MQTT 演示。

AWS_IOT_DEMO_MQTT_PUBLISH_BURST_SIZE

每次突增时要发布的消息的数目。

AWS_IOT_DEMO_MQTT_PUBLISH_BURST_COUNT

本演示中的发布突增数。

Amazon FreeRTOS MQTT 库 (传统)

概述

Amazon FreeRTOS 包括一个开源 MQTT 客户端库，您可以使用它来创建发布和订阅 MQTT 主题的应用程序，就像网络上的 MQTT 客户端。

Amazon FreeRTOS MQTT 库的源文件位于 [AmazonFreeRTOS/lib/mqtt](#) 中。

新 Amazon FreeRTOS MQTT 库为公共测试版。有关更多信息，请参阅[Amazon FreeRTOS MQTT 库 \(测试版 \)](#) (p. 99)。

FreeRTOS MQTT 代理

Amazon FreeRTOS 还包括一个名为“FreeRTOS MQTT 代理”的开源守护程序，可用于管理 MQTT 库。MQTT 代理提供了一个简单接口以通过底层 MQTT 库连接、发布和订阅 MQTT 主题。

MQTT 代理运行在一个单独的 FreeRTOS 任务中，并按照 MQTT 协议规范中的规定，自动发送常规的保持活动消息。所有 MQTT API 均可阻止并采用超时参数，即 API 等待相应操作完成所花费的最长时间值。如果操作在给定时间内未完成，API 会返回超时错误代码。

依赖项和要求

Amazon FreeRTOS MQTT 库使用 [Amazon FreeRTOS 安全套接字库 \(p. 108\)](#) 和 Amazon FreeRTOS 缓冲池库。如果 MQTT 代理连接到安全 MQTT 代理，则库也将使用 [Amazon FreeRTOS 传输层安全性 \(TLS\) \(p. 114\)](#)。

功能

回调

您可以指定可选的回调，每当 MQTT 代理 (Agent) 与代理 (Broker) 断开连接或者收到代理 (Broker) 的发布消息时，即调用此回调。收到的发布消息存储在从中央缓冲池获取的缓冲区内。此消息将传递给回调。此回调在 MQTT 任务的上下文中运行，因此必须快速。如果需要更长的处理时间，则必须从回调返回 pdTRUE，以获得缓冲区的所有权。随后，在完成操作后，必须调用 `FreeRTOS_Agent_ReturnBuffer` 将缓冲区归还给缓冲池。

订阅管理

您可以通过订阅管理，为每个订阅筛选条件注册一个回调。在订阅时提供此回调。每当与主题筛选条件匹配的主题收到发布消息时，即调用此回调。缓冲区所有权的工作方式与通用回调情况中所述相同。

MQTT 任务唤醒

每当用户调用 API 执行任何操作，或者收到代理的发布消息时，便会唤醒 MQTT 任务唤醒功能。如果平台能够将连接套接字上接收的数据通知主机 MCU，则在该平台上有可能实现基于发布消息接收的异步唤醒。如果平台不具备此功能，则需要 MQTT 任务持续轮询连接套接字上接收的数据。为确保接收发布消息与调用回调之间的延迟时间最短，`mqttconfigMQTT_TASK_MAX_BLOCK_TICKS` 宏会控制 MQTT 任务保持“被阻止”状态的最长时间。对于无法将连接套接字上接收的数据通知主机 MCU 的平台，该值必须很小。

源和标头文件

```
Amazon FreeRTOS
  |
  + - lib
    |
    + - mqtt
      |   + - aws_mqtt_lib.c           [Required to use the MQTT library and the
MQTT agent]
      |   + - aws_mqtt_agent.c         [Required to use the MQTT agent]
      |
      + - include
        |
        + - private                  [For internal library use only!]
          |   + - aws_doubly_linked_list.h
          |   +- aws_mqtt_agent_config_defaults.h
          |   + - aws_mqtt_buffer.h
          |   + - aws_mqtt_config_defaults.h
          |
          + - aws_mqtt_agent.h         [Include to use the MQTT agent API]
          + - aws_mqtt_lib.h          [Include to use the MQTT library API]
```

主要配置

在 MQTT 连接请求期间，可以指定以下标志：

- `mqttconfigKEEP_ALIVE_ACTUAL_INTERVAL_TICKS` : 保持活动消息的发送频率。
- `mqttconfigENABLE_SUBSCRIPTION_MANAGEMENT` : 启用订阅管理。
- `mqttconfigMAX_BROKERS` : 同时运行的 MQTT 客户端的最大数量。
- `mqttconfigMQTT_TASK_STACK_DEPTH` : 任务堆栈深度。
- `mqttconfigMQTT_TASK_PRIORITY` : MQTT 任务的优先级。
- `mqttconfigRX_BUFFER_SIZE` : 用于接收数据的缓冲区的长度。
- `mqttagentURL_IS_IP_ADDRESS` : 如果提供的 URL 是 IP 地址，则在 `xFlags` 中设置此位。
- `mqttagentREQUIRE_TLS` : 在 `xFlags` 设置此位以使用 TLS。
- `mqttagentUSE_AWS_IOT_ALPN_443` : 在 `xFlags` 中设置此位，以使用 AWS IoT 对 TLS 端口 443 上的 MQTT 的支持。

有关 ALPN 的更多信息，请参阅《AWS IoT 开发人员指南》中的 [AWS IoT 协议](#)，以及 AWS 上的物联网博客上的 [MQTT 及端口 443 上的 TLS 客户端身份验证：为什么有用及其工作原理](#) 博客帖子。

优化

及时处理接收的数据包

实施 MQTT 代理的任务大部分时间都处于阻止状态（因此，不使用任何 CPU 周期），等待事件处理。通过在从网络接收到 MQTT 数据包后取消阻止代理任务来最大化 MQTT 吞吐量。如果这样做了，就会尽早处理接收到的数据包。如果没有这样做，接收到的数据包将不会得到处理，直到 MQTT 代理出于其他原因不再处于阻止状态。

通过执行由名为 `SOCKETS_SetSockOpt()` 的 MQTT 代理安装的回调，并将 `lOptionName` 参数设置为 `SOCKETS_SO_WAKEUP_CALLBACK` 来使 MQTT 代理不再处于阻止状态。这里需要安全套接字文档的链接。如果您使用的是 FreeRTOS+TCP TCP/IP 堆栈，则将在正确的时间执行回调，前提是在 `FreeRTOSIPConfig.h`（它是 TCP/IP 堆栈的配置文件）中将 `ipconfigSOCKET_HAS_USER_WAKE_CALLBACK` 设置为 1。如果您未使用 FreeRTOS+TCP TCP/IP 堆栈，则安全套接字可确保此功能包含为使用中的堆栈实施的安全套接字抽象层中。

如果 TCP/IP 堆栈无法在接收到数据时取消阻止 MQTT 代理，则通过 `mqttconfigMQTT_TASK_MAX_BLOCK_TICKS` 常量设置正在接收的数据包和正在处理的数据包之间的最大时间。

最小化 RAM 消耗

以下配置常量会直接影响 MQTT 代理所需的 RAM 量：

- `mqttconfigMQTT_TASK_STACK_DEPTH`
- `mqttconfigMAX_BROKERS`
- `mqttconfigMAX_PARALLEL_OPS`
- `mqttconfigRX_BUFFER_SIZE`

您应将这些常量设置为可能的最小值。

要求和使用限制

使用 `xTaskCreateStatic()` API 函数创建 MQTT 代理任务 - 以便在编译时静态分配任务的堆栈和控制块。这将确保 MQTT 代理可用于不允许动态内存分配的应用程序，但意味着依赖于在 `FreeRTOSConfig.h` 中将 `configSUPPORT_STATIC_ALLOCATION` 设置为 1。

MQTT 代理使用 FreeRTOS 定向到任务通知功能。调用 MQTT 代理 API 函数可能会更改调用任务的通知值和状态。

MQTT 数据包存储在缓冲池模块提供的缓冲区中。强烈建议确保池中的缓冲区数量至少为任何时候都在进行的 MQTT 事务数的两倍。

开发人员支持

mqttconfigASSERT

mqttconfigASSERT() 与 FreeRTOS configASSERT() 宏等效，并且二者的使用方式完全相同。如果您需要断言 MQTT 代理中的语句，请定义 mqttconfigASSERT()。如果您不需要断言 MQTT 代理中的语句，请将 mqttconfigASSERT() 保持未定义状态。如果您定义 mqttconfigASSERT() 来调用 FreeRTOS configASSERT()（如下所示），则 MQTT 代理将仅包含断言语句（如果定义了 FreeRTOS configASSERT()）。

```
#define mqttconfigASSERT( x ) configASSERT( x )
```

mqttconfigENABLE_DEBUG_LOGS

将 mqttconfigENABLE_DEBUG_LOGS 设置为 1 可通过调用 vLoggingPrintf() 来打印调试日志。

初始化

在尝试 MQTT 通信前，必须初始化 MQTT 代理与其依赖库，如下所示。在建立网络连接后初始化库。

```
BaseType_t SYSTEM_Init() { BaseType_t xResult = pdPASS; /* The bufferpool libraries provides the buffers use to store MQTT packets.*/
    xResult = BUFFERPOOL_Init();
    if( xResult == pdPASS ) { /* Create the MQTT agent task. */
        xResult = MQTT_AGENT_Init();
        if( xResult == pdPASS ) { /* Initialize the secure sockets abstraction layer.*/
            xResult = SOCKETS_Init();
        }
    }
    return xResult;
}
```

API 参考

有关完全 API 参考，请参阅 [MQTT 库 API 参考（旧版）](#) 和 [MQTT 代理 API 参考（旧版）](#)。

移植

必须将 MQTT 代理调用的安全套接字抽象层移植到特定的架构。有关更多信息，请参阅 [Amazon FreeRTOS 移植指南](#)。

Amazon FreeRTOS 无线 (OTA) 代理库

概述

可以使用 OTA 代理来管理 Amazon FreeRTOS 设备固件更新的通知、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，可以使用 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。

Amazon FreeRTOS OTA 代理库的源文件位于 [AmazonFreeRTOS/lib/ota](#) 中。

有关将无线更新用于 Amazon FreeRTOS 的更多信息，请参阅[Amazon FreeRTOS 无线更新 \(p. 118\)](#)。

功能

以下是完整的 OTA 代理接口：

`OTA_AgentInit`

初始化 OTA 代理。发起人提供消息收发协议上下文、可选的回调和超时。

`OTA_AgentShutdown`

使用完 OTA 代理后清理资源。

`OTA_GetAgentState`

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。

源和标头文件

```
Amazon FreeRTOS
|
+ - lib
  + - ota
    + - aws_ota_agent.c
    + - aws_ota_cbor.c
    + - portable
      + - README.md
      + - vendor
      + - board
        + - aws_ota_pal.c
  + - include
    + - aws_ota_agent.h
    + - private
      + - aws_ota_agent_internal.h
      + - aws_ota_cbor.h
      + - aws_ota_cbor_internal.h
      + - aws_ota_pal.h
      + - aws_ota_types.h
```

API 参考

有关完全 API 参考，请参阅 [OTA 代理 API 参考](#)。

示例用法

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. 连接到 AWS IoT MQTT 代理。有关更多信息，请参阅 [Amazon FreeRTOS MQTT 库（传统）\(p. 101\)](#)。
 2. 调用 `OTA_AgentInit` 初始化 OTA 代理。应用程序可以定义一个自定义 OTA 回调函数，或通过指定 `NULL` 回调函数指针来使用默认回调。还必须提供初始化超时。
- 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。超时定义了等待初始化完成所花费的时间。
3. 如果在代理准备就绪之前 `OTA_AgentInit` 超时，则可以调用 `OTA_GetAgentState` 以确认代理已初始化并按预期运行。
 4. OTA 更新完成后，Amazon FreeRTOS 调用具有以下事件之一的作业完成回调：`accepted`、`rejected` 或 `self test`。
 5. 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。
 6. 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 功能移植到平台的信息，请参阅 [OTA 可移植抽象层](#)。

Amazon FreeRTOS 公有密钥加密标准 (PKCS) #11 库概述

公有密钥加密标准 #11 (PKCS#11) 是一个加密 API，用于提取密钥存储、加密对象的 get/set 属性以及会话语义。请参阅 Amazon FreeRTOS 源代码存储库中的 `pkcs11.h` (获取自 OASIS，标准正文)。在 Amazon FreeRTOS 参考实施中，PKCS#11 API 调用由 TLS 帮助程序接口生成，目的是为了在 `SOCKETS_Connect` 期间执行 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配置工作流程生成，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（预配置和 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。

Amazon FreeRTOS PKCS#11 库的源文件位于 [AmazonFreeRTOS/lib/secure_sockets/portable](#) 中。

功能

下面列出了使用的部分 PKCS#11。以下列表按照为实现预配置、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。

预配置 API

- `C_GetFunctionList`
- `C_Initialize`
- `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`

客户端身份验证

- `C_Initialize`
- `C_GetSlotList`
- `C_OpenSession`
- `C_FindObjectsInit`

- C_FindObjects
- C_FindObjectsFinal
- C_GetAttributeValue
- C_FindObjectsInit
- C_FindObjects
- C_FindObjectsFinal
- C_GetAttributeValue
- C_GenerateRandom
- C_SignInit
- C_Sign
- C_DigestInit
- C_DigestUpdate
- C_DigestFinal

清除

- C_CloseSession
- C_Finalize

非对称加密支持

Amazon FreeRTOS PKCS#11 参考实施支持 2048 位 RSA (仅限签名) 以及具有 NIST P-256 曲线的 ECDSA。下文将介绍如何创建基于 P-256 客户端证书的 AWS IoT 事物。

请确保使用的是以下版本 (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
```

编写以下步骤时 , 假定您已使用 aws configure 命令配置了 AWS CLI。

创建基于 P-256 客户端证书的 AWS IoT 事物

1. 运行 aws iot create-thing --thing-name dcgecc , 以创建 AWS IoT 事物。
2. 运行 openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -pkeyopt ec_param_enc:named_curve -outform PEM -out dcgecc.key , 以使用 OpenSSL 创建 P-256 密钥。
3. 运行 openssl req -new -nodes -days 365 -key dcgecc.key -out dcgecc.req , 创建证书注册请求 , 采用第 2 步中创建的密钥进行签名。
4. 运行 aws iot create-certificate-from-csr --certificate-signing-request file://dcgecc.req --set-as-active --certificate-pem-outfile dcgecc.crt , 向 AWS IoT 提交证书注册请求。
5. 运行 aws iot attach-thing-principal --thing-name dcgecc --principal "arn:aws:iot:us-east-1:123456789012:cert/86e41339a6d1bbc67abf31faf455092cdebf8f21ffbc67c4d238d1326c7de" 将证书 (由 ARN 输出通过上一个命令引用) 附加到事物。
6. 运行 aws iot create-policy --policy-name FullControl --policy-document file://policy.json , 以创建策略。 (此策略过于宽松。只应当用作开发目的。)

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 Amazon 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. 运行 `aws iot attach-principal-policy --policy-name FullControl --principal "arn:aws:iot:us-east-1:785484208847:cert/86e41339a6d1bbc67abf31faf455092cdebf8f21ffbc67c4d238d1326c7de..."` 将委托人（证书）和策略附加到事物。

接下来，请遵照本指南 [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。

Amazon FreeRTOS 安全套接字库

概述

您可以使用 Amazon FreeRTOS 安全套接字库来创建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。

Amazon 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 Amazon FreeRTOS 安全套接字库和 Berkeley 套接字接口之间的差异的信息，请参阅 [安全套接字 API 参考](#) 中的 `SOCKETS_SetSockOpt`。

Amazon FreeRTOS 安全套接字库的源文件位于 [AmazonFreeRTOS/lib/secure_sockets/portable](#) 中。

Note

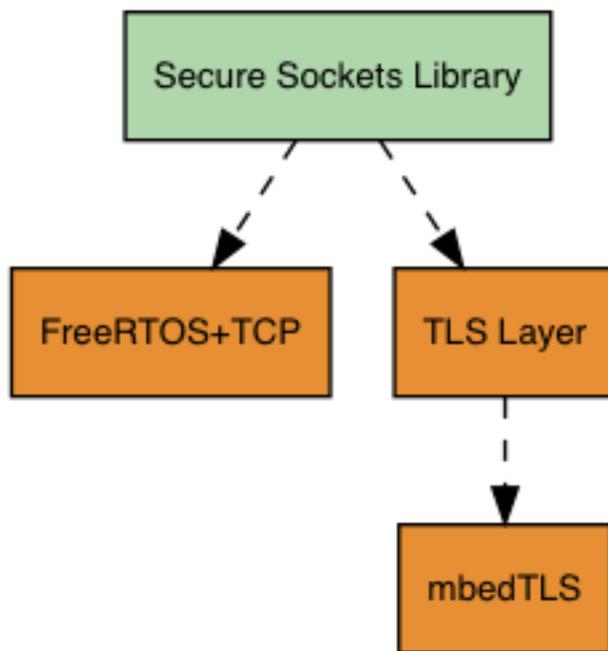
目前，Amazon FreeRTOS 安全套接字仅支持客户端 API。

依赖项和要求

Amazon FreeRTOS 安全套接字库依赖于 TCP/IP 堆栈和 TLS 实现。Amazon 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/I 的自定义实现，以及带 `mbedtls` 的 Amazon FreeRTOS TLS 层
- `FreeRTOS+TCP` 以及带 `mbedtls` 的 Amazon FreeRTOS TLS 层

以下依赖项图显示了 Amazon FreeRTOS 安全套接字库附带的参考实施。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 Amazon FreeRTOS TLS 层的更多信息，请参阅[Amazon FreeRTOS 传输层安全性 \(TLS\) \(p. 114\)](#)。



功能

Amazon 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
- 用于发送和接收数据的线程安全的 API
- 易于启用的 TLS

占用空间

代码大小 (使用 GCC for ARM Cortex-M 生成的示例)

文件名称	大小 (已针对速度进行优化)	大小 (已针对速度和大小进行优化)	
安全套接字库	因端口而异	因端口而异	
例如，对于 TI CC3220SF： lib/secure_sockets/portable/ti/cc3220_launchpad/ aws_secure_sockets.c	5.0 K	4.3 K	

源和标头文件

```
Amazon FreeRTOS
|
+ - lib
  + - include
    |   + - aws_secure_sockets.h
    |   + - private
```

```
|      + - aws_secure_sockets_config_defaults.h
+ - secure_sockets
  + - portable
    + - ...
      + - aws_secure_sockets.c
```

问题排查

错误代码

Amazon 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 Amazon FreeRTOS 安全套接字 API 返回错误代码，则 [Amazon FreeRTOS MQTT 库（传统）\(p. 101\)](#)（依赖于 Amazon FreeRTOS 安全套接字库）返回错误代码 `AWS_IOT_MQTT_SEND_ERROR`。

开发人员支持

Amazon 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

Amazon FreeRTOS 安全套接字库仅支持 TCP 套接字。不支持 UDP 套接字。

Amazon FreeRTOS 安全套接字库仅支持客户端 API。不支持服务器 API，包括 `Bind`、`Accept` 和 `Listen`。

初始化

要使用 Amazon 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

有关完全 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.\n
Change this to the certificate of your choice. */
static const char ctlsECHO_SERVER_CERTIFICATE_PEM[ ] =
"-----BEGIN CERTIFICATE-----\n"
"MIIBtjCCAVugAwIBAgITBmyf1XSXNmY/Owua2eiedgPySjAKBggqhkjOPQDAjA5\n"
"MQswCQYDVQQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwFwYDVQDExBBbWF6b24g\n"
"Um9vdCBDSQAzMB4XDTE1MDUyNjAwMDAwMFoXTDQwMDUyNjAwMDAwMFowOTELMakG\n"
"A1UEBhMCVVMxDzANBgnVBAoTBkFTYXpvbjEZMBcGA1UEAxMQOW1hem9uIFJvb3Qg\n"
"Q0EgMzBZMBMGByqGSM49AgECCqGSM49AwEHA0IABCmXp8ZBf8ANm+gBG1bG81Kl\n"
"ui2yEujSLtf6ycXYqm0fc4E705hr0XwzpcV0ho6AF2hiRVd9RFgdszf1ZwjrZt6j\n"
"QjBAMA8GA1UdEwEB/wQFMAMBAf8wDgYDVROPAQH/BAQDAgGGB0GA1UdDgQWBBSr\n"
"ttvXBp43rDCGB5Fwx5zEGbF4wDACKBggqhkjOPQDAGNJADBAiEA4IWSoxe3jfkr\n"
"BqWTrBqYagFy+uGh0PsceGCMQ5nFuMQCIQCcAu/x1Jyz1vnrxir4tiz+OpAUfem\n"
"YyRIHN8wfVoOw==\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_SNDFTIMEO, &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, &ampxEchoServerAddress, sizeof( xEchoServerAddress ) ) ==
        0 )
    {
        /* Send the string to the socket. */
        xTransmitted = SOCKETS_Send( xSocket,                                     /* The socket
receiving. */                                         ( void * )"some message",          /* The data being
sent. */                                         111

```

```
        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

Amazon 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：[Amazon FreeRTOS 公有密钥加密标准 \(PKCS\) #11 库](#) (p. 106)
- 这些区域有：[Amazon FreeRTOS 传输层安全性 \(TLS\)](#) (p. 114)

有关移植的更多信息，请参阅 [Amazon FreeRTOS 移植指南](#) 中的 [Amazon FreeRTOS Qualification Program 开发人员指南](#)。

Amazon FreeRTOS AWS IoT Device Shadow 库

概述

Amazon FreeRTOS 设备影子 API 定义了创建、更新和删除设备影子的函数。有关 Amazon FreeRTOS 设备影子的更多信息，请参阅[设备影子](#)。设备影子通过 MQTT 协议进行访问。FreeRTOS 设备影子 API 与 MQTT API 配合使用，处理有关使用 MQTT 协议的详细信息。

Amazon FreeRTOS AWS IoT 设备影子库的源文件位于 [AmazonFreeRTOS/lib/shadow](#) 中。

依赖项和要求

要将 AWS IoT Device Shadows 与 Amazon FreeRTOS 结合使用，您需要在 AWS IoT 中创建事物，包括证书和策略。有关更多信息，请参阅[AWS IoT 入门](#)。必须为 `AmazonFreeRTOS/demos/common/include/aws_client_credentials.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  
    网络所使用的 Wi-Fi 安全类型。  
keyCLIENT_CERTIFICATE_PEM  
    与 IoT 事物关联的证书 PEM。  
keyCLIENT_PRIVATE_KEY_PEM  
    与 IoT 事物关联的私有密钥 PEM。有关更多信息，请参阅配置项目 \(p. 10\)。
```

请确保设备上已安装 Amazon FreeRTOS MQTT 库。有关更多信息，请参阅 [Amazon FreeRTOS MQTT 库（传统）\(p. 101\)](#)。请确保 MQTT 缓冲区足够大，能够包含影子 JSON 文件。设备影子文档的最大大小为 8 KB。设备影子 API 的所有默认设置均可在 lib\include\private\aws_shadow_config_defaults.h 文件中进行设置。您可以在 demos/<platform>/common/config_files/aws_shadow_config.h 文件中修改这些设置中的任意一个。

您必须具有已向 AWS IoT 注册的 IoT 事物，包括证书和允许访问设备影子的策略。有关更多信息，请参阅 [AWS IoT 入门](#)。

源和标头文件

```
Amazon FreeRTOS
+ - lib
  |
  + - shadow
    |   + - aws_shadow.c
    |   + - aws_shadow_json.c
    + - include
      + - aws_shadow.h
      + - private
        + - aws_shadow_config_defaults.h
        + - aws_shadow_json.h
```

API 参考

有关完全 API 参考，请参阅 [Device Shadow API 参考](#)。

示例用法

1. 使用 SHADOW_ClientCreate API 创建影子客户端。对于大部分应用程序，唯一可填充的字段为 xCreateParams.xMQTTClientType = eDedicatedMQTTClient。
2. 通过调用 SHADOW_ClientConnect API，并传递 SHADOW_ClientCreate 返回的客户端句柄，来建立 MQTT 连接。
3. 调用 SHADOW_RegisterCallbacks API 来配置回调，以更新、获取和删除影子。

建立连接后，可以通过以下 API 使用设备影子：

`SHADOW_Delete`

删除设备影子。

`SHADOW_Get`

获取当前的设备影子。

`SHADOW_Update`

更新设备影子。

Note

使用完设备影子后，调用 `SHADOW_ClientDisconnect` 断开影子客户端的连接，释放系统资源。

Amazon FreeRTOS 传输层安全性 (TLS)

Amazon FreeRTOS 传输层安全性 (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。无需对安全套接字接口做任何更改，即可换出 TLS 接口。请参阅 Amazon FreeRTOS 源代码存储库中的 `aws_tls.h`。

TLS 接口是可选的，因为可以选择从安全套接字直接连接到加密库。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。

Amazon FreeRTOS Wi-Fi 库

概述

Amazon 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 Amazon 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。

Amazon FreeRTOS Wi-Fi 库的源文件位于 [AmazonFreeRTOS/lib/wifi/portable](#) 中。

依赖项和要求

Amazon FreeRTOS Wi-Fi 库需要 [FreeRTOS+TCP](#) 内核。

功能

Wi-Fi 库包括以下功能：

- 对 WEP、WPA 和 WPA2 身份验证的支持
- 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`。

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 WPA2 安全类型。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`

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 \(p. 117\)](#)和[API 参考 \(p. 117\)](#)。

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。

占用空间

代码大小（使用 GCC for ARM Cortex-M 生成的示例）

文件名称	大小（使用 -O1 优化）	大小（使用 Os 优化）	
Wi-Fi 库（已启用所有选项）	因端口而异	因端口而异	

文件名称	大小 (使用 -O1 优化)	大小 (使用 Os 优化)	
例如，对于 TI CC3220SF： lib/ wifi/portable/ ti/ cc3220_launchpad/ aws_wifi.c	3.7 K	3.0 K	

源和标头文件

```
Amazon FreeRTOS
|
+ - lib
    + - include
        |   + - aws_wifi.h           [Include to use the AFR WIFI API]
        + - wifi
            + - portable
                + ...
                    + - aws_wifi.c      [Port-specific folder structure]
                                         [Required to use the AFR WIFI API]
```

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 \(p. 117\)](#)。

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 [demos/vendor/board/common/application_code/main.c](#) 文件作为初始化模板时，请执行以下操作：

- 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 `WFINetworkParams_t` 结构传递到 `WIFI_ConnectAP()` 以将主板连接到可用的 Wi-Fi 网络。有关 `WFINetworkParams_t` 结构的更多信息，请参阅示例用法 (p. 117) 和 API 参考 (p. 117)。

API 参考

有关完全 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"  
  
INetworkParams_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  
}
```

扫描附近 AP

```
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

aws_wifi.c 实现需要实施 aws_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。

Amazon FreeRTOS 无线更新

通过无线 (OTA) 更新，您可以将文件部署到机群中的一个或多个设备。尽管 OTA 更新设计用于更新设备固件，但也可以用来将任意文件发送到已注册 AWS IoT 的一个或多个设备。在通过无线发送文件时，最佳实践是对文件进行数字签名，以便接收文件的设备可以验证文件在传输途中未被篡改。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 \(p. 152\)](#) 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 Amazon FreeRTOS 任务，以订阅 AWS IoT 作业的通知主题并侦听更新消息。当有可用更新时，OTA 代理将请求发布到 AWS IoT 流主题，并使用 MQTT 协议

议接收文件数据块。然后，代理将数据块重组为文件，并检查所下载文件的数字签名。如果文件有效，代理会安装固件更新。如果不使用 Amazon FreeRTOS OTA 更新演示应用程序，则必须将 [Amazon FreeRTOS 无线 \(OTA\) 代理库 \(p. 104\)](#) 集成到您自己的应用程序中，以获取固件更新功能。

Amazon FreeRTOS 无线更新使以下操作成为可能：

- 在部署之前对固件进行数字签名和加密。
- 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。

无线更新先决条件

要使用无线更新，需要执行以下操作：

- 创建 Amazon S3 存储桶以存储更新 (p. 119).
- 创建 OTA 更新服务角色 (p. 119).
- 创建 OTA 用户策略 (p. 120).
- 创建代码签名证书 (p. 122).
- 如果您使用的是 Code Signing for AWS IoT，请[授予 Code Signing for AWS IoT 访问权 \(p. 126\)](#)。
- 下载 [Amazon FreeRTOS 及 OTA 库 \(p. 127\)](#)。如果您使用的不是 Amazon FreeRTOS，可以提供自己的 OTA 代理实施。

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。如果使用 Code Signing for AWS IoT，则用于创建代码签名作业的命令将占用源存储桶（未签名的固件映像所在的位置），以及目标存储桶（已签名的固件映像写入的位置）。可以为源和目标指定相同的存储桶。文件名将更改为 GUID，因而原始文件不会被覆盖。

创建 Amazon S3 存储桶

1. 转至 <https://console.aws.amazon.com/s3/>。
2. 选择 Create bucket (创建存储桶)。
3. 键入存储桶名称，然后选择 Next (下一步)。
4. 选中版本控制以在同一存储桶中保留所有版本，然后选择下一步。
5. 选择 Next (下一步) 接受默认权限。
6. 选择 Create bucket (创建存储桶)。

有关 Amazon S3 的更多信息，请参阅 [Amazon Simple Storage Service Console User Guide](#)。

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 <https://console.aws.amazon.com/iam/>。

2. 从导航窗格中，选择 Roles。
3. 选择 Create role (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：标签。
8. 选择 Next: Review。
9. 键入角色名称和描述，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](#)。

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 Search (搜索) 框中，输入 **AmazonFreeRTOSOTAUpdate**。在托管策略列表中，选中 AmazonFreeRTOSOTAUpdate，然后选择 Attach policy (附加策略)。

将所需权限添加到 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。将 `<example-bucket>` 替换为您的存储桶的名称。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:GetObjectVersion",  
                "s3:GetObject"  
            ],  
            "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。

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

您必须授予 IAM 用户执行无线更新的权限。IAM 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 AWS Certificate Manager 中的证书。
- 访问 AWS IoT 流服务。
- 访问 Amazon FreeRTOS OTA 更新。

- 访问 AWS IoT 作业。
- 访问 IAM。
- 访问 Code Signing for AWS IoT。
- 列出 Amazon FreeRTOS 硬件平台。

要授予 IAM 用户所需的权限，可创建 OTA 用户策略，然后将其附加到 IAM 用户。有关更多信息，请参阅[IAM 策略](#)。

创建 OTA 用户策略

1. 打开 <https://console.aws.amazon.com/iam/> 控制台。
2. 在导航窗格中，选择 Users。
3. 从列表中选择 IAM 用户。
4. 选择 Add permissions。
5. 选择直接附加现有策略。
6. 选择 Create policy。
7. 选择 JSON 选项卡，然后将以下策略文档复制并粘贴到策略编辑器中：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:listBucket",  
                "s3>ListAllMyBuckets",  
                "s3>CreateBucket",  
                "s3:PutBucketVersioning",  
                "s3:GetBucketLocation",  
                "s3:GetObjectVersion",  
                "acm:ImportCertificate",  
                "acm>ListCertificates",  
                "iot:*",  
                "iam>ListRoles",  
                "freertos>ListHardwarePlatforms",  
                "freertos:DescribeHardwarePlatform"  
            ],  
            "Resource": "*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:GetObject",  
                "s3:PutObject"  
            ],  
            "Resource": "arn:aws:s3:::<example-bucket>/*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": "iam:PassRole",  
            "Resource": "arn:aws:iam::<your-account-id>:role/<role-name>"  
        }  
    ]  
}
```

将 `<example-bucket>` 替换为 Amazon S3 存储桶的名称，该存储桶即存储 OTA 更新固件映像的位置。将 `<your-account-id>` 替换为您的 AWS 账户 ID。您可以在控制台右上角找到 AWS 账户 ID。在输入账户 ID 时，请删除任何短划线 (-)。将 `<role-name>` 替换为刚创建的 IAM 服务角色的名称。

8. 选择查看策略。
9. 为新的 OTA 用户策略输入名称，然后选择 Create policy (创建策略)。

将 OTA 用户策略附加到 IAM 用户

1. 在 IAM 控制台的导航窗格中，选择 Users (用户)，然后选择您的用户。
2. 选择 Add permissions。
3. 选择直接附加现有策略。
4. 搜索刚创建的 OTA 用户策略，然后选中旁边的复选框。
5. 选择 Next: Review。
6. 选择 Add permissions。

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，可以创建自签名的证书和私有密钥。对于生产环境，请通过知名证书颁发机构 (CA) 购买证书。

不同平台需要不同类型的代码签名证书。以下部分将介绍如何为不同的符合 Amazon FreeRTOS 条件的平台创建代码签名证书。

主题

- 为 Texas Instruments CC3200SF-LAUNCHXL 创建代码签名证书 (p. 122)
- 为 Microchip Curiosity PIC32MZEF 创建代码签名证书 (p. 124)
- 为 Espressif ESP32 创建代码签名证书 (p. 124)
- 为 Amazon FreeRTOS Windows 模拟器创建代码签名证书 (p. 125)
- 为自定义硬件创建代码签名证书 (p. 126)

为 Texas Instruments CC3200SF-LAUNCHXL 创建代码签名证书

对于固件代码签名，SimpleLink Wi-Fi CC3220SF Wireless Microcontroller Launchpad 开发工具包支持两种证书链：

- 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)

通过操场证书链，您可以尝试带自签名代码签名证书的 OTA 更新。

安装 [SimpleLink CC3220 SDK 版本 2.10.00.04](#)。默认情况下，所需的文件位于以下位置：

C:\ti\simplelink_cc32xx_sdk_2_10_00_04\tools\cc32xx_tools\certificate-playground (Windows)

/Applications/Ti/simplelink_cc32xx_sdk_2_10_00_04/tools/cc32xx_tools/certificate-playground (macOS)

SimpleLink CC3220 SDK 中的证书为 DER 格式。要创建自签名的代码签名证书，必须将证书转换为 PEM 格式。

按照以下步骤创建代码签名证书，该证书与 Texas Instruments 操场证书层次结构相链接，并符合 AWS Certificate Manager 和 Code Signing for AWS IoT 标准。

创建自签名的代码签名证书

1. 在工作目录中，使用以下文本创建名为 cert_config 的文件。将 `test_signer@amazon.com` 替换为您的电子邮件地址。

```
[ req ]  
prompt = no  
distinguished_name = my_dn  
  
[ my_dn ]  
commonName = test_signer@amazon.com  
  
[ my_exts ]  
keyUsage = digitalSignature  
extendedKeyUsage = codeSigning
```

2. 创建私有密钥和证书签名请求 (CSR) :

```
openssl req -config cert_config -extensions my_exts -nodes -days 365 -newkey rsa:2048 -keyout tisigner.key -out tisigner.csr
```

3. 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_2_10_00_04\tools\cc32xx_tools\certificate-playground\dummy-root-ca-cert-key (Windows)

/Applications/Ti/simplelink_cc32xx_sdk_2_10_00_04/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
```

4. 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_2_10_00_04\tools\cc32xx_tools\certificate-playground\dummy-root-ca-cert (Windows)

/Applications/Ti/simplelink_cc32xx_sdk_2_10_00_04/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
```

5. 使用 Texas Instruments 根 CA 对 CSR 进行签名：

```
openssl x509 -extfile cert_config -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
```

6. 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。

7. 将代码签名证书、私有密钥和证书链导入 AWS Certificate Manager 中：

```
aws acm import-certificate --certificate file://tisigner.crt.pem --private-key file://tisigner.key --certificate-chain file://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，假定您计划使用 Code Signing for AWS IoT 来签署固件映像。虽然我们推荐使用 Code Signing for AWS IoT，不过您也可以手动签署固件映像。

为 Microchip Curiosity PIC32MZEF 创建代码签名证书

Microchip Curiosity PIC32MZEF 支持带 ECDSA 代码签名证书的自签名 SHA256。

1. 在工作目录中，使用以下文本创建名为 cert_config 的文件。将 *test_signer@amazon.com* 替换为您的电子邮件地址：

```
[ 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 -extensions my_exts -nodes -days 365 -key ecdsasigner.key -out ecdsasigner.crt
```

4. 将代码签名证书、私有密钥和证书链导入 AWS Certificate Manager 中：

```
aws acm import-certificate --certificate file://ecdsasigner.crt --private-key file://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，假定您计划使用 Code Signing for AWS IoT 来签署固件映像。虽然我们推荐使用 Code Signing for AWS IoT，不过您也可以手动签署固件映像。

为 Espressif ESP32 创建代码签名证书

Espressif ESP32 主板支持带 ECDSA 代码签名证书的自签名 SHA256。

1. 在工作目录中，使用以下文本创建名为 cert_config 的文件。将 *test_signer@amazon.com* 替换为您的电子邮件地址：

```
[ 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 -extensions my_exts -nodes -days 365 -key  
ecdsasigner.key -out ecdsasigner.crt
```

4. 将代码签名证书、私有密钥和证书链导入 AWS Certificate Manager 中：

```
aws acm import-certificate --certificate file://ecdsasigner.crt --private-key  
file://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，假定您计划使用 Code Signing for AWS IoT 来签署固件映像。虽然我们推荐使用 Code Signing for AWS IoT，不过您也可以手动签署固件映像。

为 Amazon FreeRTOS Windows 模拟器创建代码签名证书

Amazon 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，可使用以下步骤创建一个：

1. 在工作目录中，使用以下文本创建名为 cert_config 的文件。将 `test_signer@amazon.com` 替换为您的电子邮件地址：

```
[ 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 -extensions my_exts -nodes -days 365 -key ecdsasigner.key -out ecdsasigner.crt
```

4. 将代码签名证书、私有密钥和证书链导入 AWS Certificate Manager 中：

```
aws acm import-certificate --certificate file://ecdsasigner.crt --private-key file://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，假定您计划使用 Code Signing for AWS IoT 来签署固件映像。虽然我们推荐使用 Code Signing for AWS IoT，不过您也可以手动签署固件映像。

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。

在创建代码签名证书后，将其导入 ACM 中：

```
aws acm import-certificate --certificate file://code-sign.crt --private-key file://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和私有密钥的内容复制并粘贴到 `aws_ota_codesigner_certificate.h` 文件中并进行格式化，该文件属于您稍后下载的 Amazon FreeRTOS 代码的一部分。

授予 Code Signing for AWS IoT 访问权

在生产环境中，应当对固件更新进行数字化签名，以确保更新的真实性和完整性。可以手动签署更新，也可以使用 Code Signing for AWS IoT 对代码进行签名。要使用 Code Signing for Amazon FreeRTOS，必须为 IAM 用户账户授予 Code Signing for Amazon FreeRTOS 访问权。

授予 IAM 用户账户 Code Signing for AWS IoT 权限

1. 登录 <https://console.aws.amazon.com/iam/>。
2. 在导航窗格中，选择 Policies。
3. 选择 Create Policy。
4. 在 JSON 选项卡上，将以下 JSON 文档复制并粘贴到策略编辑器中。此策略允许 IAM 用户访问所有代码签名操作。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "signer:*"  
            ],  
            "Resource": [  
                "arn:aws:acm:region:account-id:certificate/certificate-arn"  
            ]  
        }  
    ]  
}
```

```
    " * "
}
]
}
```

5. 选择查看策略。
6. 输入策略名称和描述，然后选择 Create policy (创建策略)。
7. 在导航窗格中，选择 Users。
8. 选择 IAM 用户账户。
9. 在权限选项卡中，请选择添加权限。
10. 选择 Attach existing policies directly (直接附加现有策略)，然后选中刚创建的代码签名策略旁边的复选框。
11. 选择 Next: Review。
12. 选择 Add permissions。

下载 Amazon FreeRTOS 及 OTA 库

可按照本部分中的步骤下载代码并构建演示应用程序。

为 Texas Instruments CC3200SF-LAUNCHXL 下载和构建 Amazon FreeRTOS

下载 Amazon FreeRTOS 和 OTA 演示代码

1. 浏览至 AWS IoT 控制台，从导航窗格中选择 Software (软件)。
2. 在 Amazon FreeRTOS 设备软件下，选择配置下载。
3. 从软件配置列表中，选择 Connect to AWS IoT - TI (连接到 AWS IoT – TI)。选择配置名称，而不是 Download (下载) 链接。
4. 在 Libraries (库) 下，选择 Add another library (添加另一个库)，然后选择 OTA Updates (OTA 更新)。
5. 在 Demo Projects (演示项目) 下，选择 OTA Updates (OTA 更新)。
6. 在 Name required (需要名称) 下，输入 **Connect-to-IoT-OTA-TI**，然后选择 Create and download (创建和下载)。

将包含 Amazon FreeRTOS 和 OTA 演示代码的 zip 文件保存到计算机上。

构建演示应用程序

1. 将 .zip 文件解压缩。
2. 按照 [Amazon FreeRTOS 入门 \(p. 4\)](#) 中的说明进行操作，将 aws_demos 项目导入 Code Composer Studio，为主板配置 AWS IoT 终端节点、Wi-Fi SSID 和密码，以及私有密钥和证书。
3. 在 Code Composer Studio 中打开项目，然后打开 demos/common/demo_runner/aws_demo_runner.c。找到 DEMO_RUNNER_RunDemos 函数，并确保除 vStartOTAUpdateDemoTask 之外的所有函数调用均已注释。
4. 构建解决方案，并确保其构建没有错误。
5. 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
6. 在主板上运行项目，确保它可以连接到 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 0 [Tmr Svc] Starting Wi-Fi Module ...
1 0 [Tmr Svc] Simple Link task created
Device came up in Station mode
2 142 [Tmr Svc] Wi-Fi module initialized.
3 142 [Tmr Svc] Starting key provisioning...
4 142 [Tmr Svc] Write root certificate...
5 243 [Tmr Svc] Write device private key...
6 340 [Tmr Svc] Write device certificate...
7 433 [Tmr Svc] Key provisioning done...
[WLAN EVENT] STA Connected to the AP: Mobile , BSSID: 24:de:c6:5d:32:a4
[NETAPP EVENT] IP acquired by the device

Device has connected to Mobile
Device IP Address is 192.168.111.12

8 2666 [Tmr Svc] Wi-Fi connected to AP Mobile.
9 2666 [Tmr Svc] IP Address acquired 192.168.111.12
10 2667 [OTA] OTA demo version 0.9.2
11 2667 [OTA] Creating MQTT Client...
12 2667 [OTA] Connecting to broker...
13 3512 [OTA] Connected to broker.
14 3715 [OTA Task] [prvSubscribeToJobNotificationTopics] OK: $aws/things/OtaGA/jobs/$next/
get/accepted
15 4018 [OTA Task] [prvSubscribeToJobNotificationTopics] OK: $aws/things/OtaGA/jobs/notify-
next
16 4027 [OTA Task] [prvPAL_GetPlatformImageState] xFileInfo.Flags = 0250
17 4027 [OTA Task] [prvPAL_GetPlatformImageState] eOTA_PAL_ImageState_Valid
18 4034 [OTA Task] [OTA_CheckForUpdate] Request #0
19 4248 [OTA] [OTA_AgentInit] Ready.
20 4249 [OTA Task] [prvParseJSONbyModel] Extracted parameter [ clientToken: 0:OtaGA ]
21 4249 [OTA Task] [prvParseJSONbyModel] parameter not present: execution
22 4249 [OTA Task] [prvParseJSONbyModel] parameter not present: jobID
23 4249 [OTA Task] [prvParseJSONbyModel] parameter not present: jobDocument
24 4249 [OTA Task] [prvParseJSONbyModel] parameter not present: afr_ota
25 4250 [OTA Task] [prvParseJSONbyModel] parameter not present: streamname
26 4250 [OTA Task] [prvParseJSONbyModel] parameter not present: files
27 4250 [OTA Task] [prvParseJSONbyModel] parameter not present: filepath
28 4250 [OTA Task] [prvParseJSONbyModel] parameter not present: filesize
29 4250 [OTA Task] [prvParseJSONbyModel] parameter not present: fileid
30 4250 [OTA Task] [prvParseJSONbyModel] parameter not present: certfile
31 4251 [OTA Task] [prvParseJSONbyModel] parameter not present: sig-sha1-rsa
32 4251 [OTA Task] [prvParseJobDoc] Ignoring job without ID.
33 4251 [OTA Task] [prvOTA_Close] Context->0x2001b2c4
34 5248 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
35 6248 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
36 7248 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
37 8248 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
38 9248 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
```

为 Microchip Curiosity PIC32MZEF 下载和构建 Amazon FreeRTOS

下载 Amazon FreeRTOS OTA 演示代码

1. 浏览至 AWS IoT 控制台，从导航窗格中选择 Software (软件)。
2. 在 Amazon FreeRTOS 设备软件下，选择配置下载。
3. 从软件配置列表中，选择 Connect to AWS IoT - Microchip (连接到 AWS IoT – Microchip)。选择配置名称，而不是 Download (下载) 链接。
4. 在 Libraries (库) 下，选择 Add another library (添加另一个库)，然后选择 OTA Updates (OTA 更新)。
5. 在 Demo projects (演示项目) 下，选择 OTA Update (OTA 更新)。

6. 在 Name required (需要名称) 下 , 为自定义的 Amazon FreeRTOS 软件配置输入名称。
7. 选择 Create and download (创建和下载)。

构建 OTA 更新演示应用程序

1. 将刚下载的 .zip 文件解压缩。
2. 按照 [Amazon FreeRTOS 入门 \(p. 4\)](#) 中的说明进行操作 , 将 aws_demos 项目导入 MPLAB X IDE , 为主板配置 AWS IoT 终端节点、Wi-Fi SSID 和密码 , 以及私有密钥和证书。
3. 打开 aws_demos/lib/aws/ota/aws_ota_codesigner_certificate.h。
4. 将代码签名证书的内容粘贴到 static const char signingcredentialsIGNING_CERTIFICATE_PEM 变量中。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\"MIIBXTCCAQOgAwIBAgIJAM4DeybzCtTwKMAoGCCqGSM49BAMCMCExHzAdBgNVBAMM\n\"FnRlc3Rf621nbmVyQGFtYXpbvi5jb20wHhcNMTcxMTAzMTkxODM1WhcNMTgxMTAz\\n\"\n\"MTkxODM2WjAhMR8wHQYDVQBZZ0ZXN0X3NpZ251ckBhbWF6b24uY29tMFkwEwYH\\n\"\n\"KoZIzj0CAQYIKoZIzj0DAQcDQgAERavZfvwL1X+E4dIF7dbkVMUu4IrJ1CasFkc8\\n\"\n\"gZxPzn683H40XMK1tDZPEwr9ng78w+QYQg7ygnr2stz8yhh06MkMCiwcWYDVR0P\\n\"\n\"BAQDAgeAMBMDGA1UdJQQMMAoGCCsGAQUFBwMDMAoGCCqGSM49BAMCA0gAMEUCIFOR\\n\"\n\"r5cb7rEUNtWoVg05MacrgOABfSoVYvBOK9fp63WAqt5h3BaS123coKSGg84twlq\\n\"\n\"TkO/pV/xEmyZmZdV+HxV/OM=\\n\"\n\"-----END CERTIFICATE-----\\n\";
```

5. 安装 [Python 3](#) 或更高版本。
6. 运行 pip install pyopenssl 以安装 pyOpenSSL。
7. 复制路径 \demos\common\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。
8. 在 MPLAB X IDE 中打开项目 , 然后打开 demos/common/demo_runner/aws_demo_runner.c。找到 DEMO_RUNNER_RunDemos 函数 , 并确保除 vStartOTAUpdateDemoTask 之外的所有函数调用均已注释。
9. 构建解决方案 , 并确保其构建没有错误。
10. 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
11. 从主板上拔掉调试器 , 在主板上运行项目 , 确保它可以连接到 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。确保选择了 ICD4 选项卡。您应当看到如下输出。

```
Bootloader version 00.09.00\n[prvBOOT_Init] Watchdog timer initialized.\n[prvBOOT_Init] Crypto initialized.\n\n[prvValidateImage] Validating image at Bank : 0\n[prvValidateImage] No application image or magic code present at: 0xbd000000\n[prvBOOT_ValidateImages] Validation failed for image at 0xbd000000\n\n[prvValidateImage] Validating image at Bank : 1\n[prvValidateImage] No application image or magic code present at: 0xbd100000\n[prvBOOT_ValidateImages] Validation failed for image at 0xbd100000
```

```
[prvBOOT_ValidateImages] Booting default image.

>0 36246 [IP-task] vDHCPPProcess: offer ac140a0eip
1 36297 [IP-task] vDHCPPProcess: 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] vDHCPPProcess: offer c0a804beip
3 7230 [IP-task] vDHCPPProcess: 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] MOTT 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 #
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] MOTT 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
```

以上输出显示 Microchip Curiosity PIC32MZEF 能够连接到 AWS IoT，并订阅 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。

为 Espressif ESP32 下载和构建 Amazon FreeRTOS

- 从 [GitHub](#) 下载 Amazon FreeRTOS 源。在 IDE 中创建一个项目，其中包含所有所需的源和库。
- 按照 [Espressif 入门](#) 中的说明，设置所需的基于 GCC 的工具链。
- 在文本编辑器中打开 `demos/common/demo_runner/aws_demo_runner.c`。找到 `DEMO_RUNNER_RunDemos` 函数，并确保除 `vStartOTAUpdateDemoTask` 之外的所有函数调用均已注释。
- 在 `demos/espressif/esp32_devkitc_esp_wrover_kit/make/` 目录中运行 `make`，构建演示项目。可以按照 [Espressif 入门](#) 中的说明，运行 `make flash monitor`，以刷入演示程序并验证其输出。
- 运行 OTA 更新演示之前：
 - 请确保 `vStartOTAUpdateDemoTask` 是 `demos/common/demo_runner/aws_demo_runner.c` 中 `DEMO_RUNNER_RunDemos()` 函数调用的唯一函数。
 - 请确保 SHA-256/ECDSA 代码签名证书已复制到 `demos/common/include/aws_ota_codesigner_certificate.h` 中。

为自定义硬件平台下载和构建 Amazon FreeRTOS

从 [GitHub](#) 下载 Amazon FreeRTOS 源。在 IDE 中创建一个项目，其中包含所有所需的源和库。

构建并运行项目，确保它可以连接到 AWS IoT。

有关将 Amazon FreeRTOS 移植到新平台的更多信息，请参阅 [Amazon FreeRTOS 移植指南 \(p. 186\)](#)。

OTA 教程

本部分包含了如何使用 OTA 更新在运行 Amazon FreeRTOS 的设备上更新固件的教程。不过，您可以使用 OTA 更新将文件发送到与 AWS IoT 相连的任何设备。

可以使用 AWS IoT 控制台或 AWS CLI 来创建 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。如果要自动执行 OTA 更新作业、使用大量设备，或者使用不符合 Amazon FreeRTOS

条件的设备，则 AWS CLI 将很有用。有关 Amazon FreeRTOS 的合格设备的更多信息，请参阅 [Amazon 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。

如果要使用 Code Signing for AWS IoT，则将未签名的固件映像上传到 Amazon S3 存储桶。

5. 创建 OTA 更新。

设备上的 Amazon 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 Amazon FreeRTOS，则必须实现一个 OTA 代理运行于设备上。

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 Amazon 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 [Texas Instruments CC3200SF-LAUNCHXL 上安装初始版本的固件 \(p. 133\)](#)
- 在 [Microchip Curiosity PIC32MZEF 上安装初始版本的固件 \(p. 135\)](#)
- 在 [Espressif ESP32 上安装初始版本的固件 \(p. 138\)](#)
- [Windows 模拟器上的初始固件 \(p. 140\)](#)
- [在自定义板卡上安装初始版本的固件 \(p. 140\)](#)

在 Texas Instruments CC3200SF-LAUNCHXL 上安装初始版本的固件

编写以下步骤时，假定您已经按照[为 Texas Instruments CC3200SF-LAUNCHXL 下载和构建 Amazon FreeRTOS \(p. 127\)](#) 中的说明，构建了 aws_demos 项目。

1. 在 Texas Instruments CC3200SF-LAUNCHXL 上，将 SOP 跳线放在中间一组针脚（位置 = 1）上，然后重置板卡。
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 Start Image Creator (启动映像创建器)。
4. 选择 New Project (新项目)。
5. 在 Start new project (启动新项目) 页面上，输入项目名称。对于 Device Type (设备类型)，选择 CC3220SF。对于 Device Mode (设备模式)，选择 Develop (开发)。选择 Create Project (创建项目)。
6. 断开终端模拟器。在 Uniflash 应用程序窗口的右侧，选择 Connect (连接)。
7. 在 Files (文件) 下，选择 User Files (用户文件)。
8. 在 File (文件) 选择器窗格中，选择 Add File (添加文件) 图标 。
9. 浏览至 /Applications/Ti/simplelink_cc32xx_sdk_2_10_00_04/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)。
10. 在 File (文件) 选择器窗格中，选择 Add File (添加文件) 图标 。

11. 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)。
12. 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 `AmazonFreeRTOS/demos/ti/cc3200_launchpad/ccs/Debug` 目录中。选择 Open。
 - 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 Write (写入)。
13. 在左侧窗格的 Files (文件) 下，选择 Service Pack。
14. 在 Service Pack File Name (Service Pack 文件名称) 下，选择 Browse (浏览)，浏览至 `simplelink_cc32x_sdk_2_10_00_04/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)。
15. 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)。
16. 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。
17. 在 Source File (源文件) 下，选择 Browse (浏览)，选择 `simplelink_cc32xx_sdk_2_10_00_04/tools/cc32xx_tools/certificate-playground\certcatalogPlayGround20160911.lst`，然后选择 Open (打开)。
18. 在 Signature Source File (签名源文件) 下，选择 Browse (浏览)，选择 `simplelink_cc32xx_sdk_2_10_00_04/tools/cc32xx_tools/certificate-playground\certcatalogPlayGround20160911.lst.signed.bin`，然后选择 Open (打开)。
19.  选择  按钮保存项目。
20.  选择  按钮。
21. 选择 Program Image (Create and Program) (编程映像 (创建和编程))。
22. 编程过程完成之后，将 SOP 跳线放在第一组针脚 (位置 = 0) 上，重置板卡，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。

终端应当显示类似以下内容的输出：

```
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 Microchip Curiosity PIC32MZEF 上安装初始版本的固件

编写以下步骤时，假定您已经按照为 Microchip Curiosity PIC32MZEF 下载和构建 Amazon FreeRTOS (p. 128) 中的说明，构建了 aws_demos 项目。

将演示应用程序刻录到板卡

1. 重新构建 aws_demos 项目并确保编译过程没有错误。

2.



在工具栏上，选择 。

3. 编程过程完成后，断开 ICD 4 调试器并重置板卡。重新连接终端模拟器，以确保输出与使用 MPLAB X IDE 调试演示代码时是相同的。

终端应当显示类似以下内容的输出：

```
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] vDHCPPProcess: offer ac140a0eip
1 36297 [IP-task] vDHCPPProcess: 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
```

以下过程将创建一个统一的十六进制文件或出厂映像，其中包含一个参考启动加载程序和带加密签名的应用程序。启动加载程序在启动时验证应用程序的加密签名，并支持 OTA 更新。

构建和刷入出厂映像

1. 确保已从 [Source Forge](#) 安装了 SRecord 工具。验证包含 srec_cat 和 srec_info 程序的目录位于系统路径中。
2. 更新出厂映像的 OTA 序列号和应用程序版本。
3. 构建 aws_demos 项目。
4. 运行 factory_image_generator.py 脚本生成出厂映像。

```
factory_image_generator.py -b mplab.production.bin -p MCHP-Curiosity-PIC32MZEF -k
private_key.pem -x aws_bootloader.X.production.hex
```

此命令采用以下参数：

- mplab.production.bin：应用程序二进制。
- MCHP-Curiosity-PIC32MZEF：平台名称。
- private_key.pem：代码签名私有密钥。
- aws_bootloader.X.production.hex：启动加载程序十六进制文件。

在构建 aws_demos 项目时，生成应用程序二进制映像和启动加载程序十六进制文件是构建过程的一部分。demos/microchip/ 目录下的每个项目均包含 dist/pic32mz_ef_curiosity/production/ 目录，其中便包括了这些文件。生成的统一的十六进制文件名为 mplab.production.unified.factory.hex。

5. 使用 MPLab IPE 工具将生成的十六进制文件编程到设备中。
6. 您可以在上传映像时查看板卡的 UART 输出，以检查出厂映像是否正常。如果一切设置正确，您应当看到映像成功启动：

```
[prvValidateImage] Validating image at Bank : 0
[prvValidateImage] Valid magic code at: 0xbd000000
[prvValidateImage] Valid image flags: 0xfc at: 0xbd000000
[prvValidateImage] Addresses are valid.
[prvValidateImage] Crypto signature is valid.
[...]
[prvBOOT_ValidateImages] Booting image with sequence number 1 at 0xbd000000
```

7. 如果证书的配置不正确，或如果 OTA 映像未正确签署，则在芯片的启动加载程序擦除无效更新之前，可能会看到类似以下内容的消息。检查代码签名证书是否一致，并仔细查看之前的步骤。

```
[prvValidateImage] Validating image at Bank : 0
[prvValidateImage] Valid magic code at: 0xbd000000
[prvValidateImage] Valid image flags: 0xfc at: 0xbd000000
[prvValidateImage] Addresses are valid.
[prvValidateImage] Crypto signature is not valid.
[prvBOOT_ValidateImages] Validation failed for image at 0xbd000000
[BOOT_FLASH_EraseBank] Bank erased at : 0xbd000000
```

在 Espressif ESP32 上安装初始版本的固件

编写本指南时，假定您已经执行了 [Espressif ESP32-DevKitC 和 ESP-WROVER-KIT 入门](#) 和 [无线更新先决条件](#) 中的步骤。在尝试 OTA 更新之前，可能需要运行 [Amazon FreeRTOS 入门](#) 中所述的 MQTT 演示项目，以确保板卡和工具链设置正确。

将初始出厂映像刷入板卡

1. 在 `demos/common/demo_runner/aws_demo_runner.c` 中，在 `DEMO_RUNNER_RunDemos` 函数中注释除 `vStartOTAUpdateDemoTask` 之外的所有函数调用。
2. 使用所选的 OTA 更新演示，按照 [ESP32 入门](#) 中概述的相同步骤，构建并刷入映像。如果之前已构建并刷入了项目，则可能需要先运行 `make clean`。运行 `make flash monitor` 之后，应当看到类似以下内容的输出：某些消息的顺序可能有所不同，因为演示应用程序一次会运行多项任务：

```
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] vDHCPPProcess: 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] vDHCPPProcess: 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 #
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->0x3ffbba8
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
[ ... ]
```

3. ESP32 板卡此时正在侦听 OTA 更新。ESP-IDF 监视器通过 make flash monitor 命令启动。可以按 Ctrl+J 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）来侦听板卡的串行输出。示例中可能包括这些串行输出。请注意，连接到板卡的串行端口可能会导致板卡重新启动。

Windows 模拟器上的初始固件

在使用 Windows 模拟器时，无需刷入初始版本的固件。Windows 模拟器是 aws_demos 应用程序的一部分，其中也包含了固件。

在自定义板卡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 Amazon FreeRTOS 源代码结构的更多信息，请参阅[浏览演示应用程序 \(p. 164\)](#)。

请确保 Amazon 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板卡上，并确保其正常运行。

更新固件版本

Amazon 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. 打开 demos/common/include/aws_application_version.h 并递增 APP_VERSION_BUILD 命牌值。
3. 如果使用的是 Microchip Curiosity PIC32MZEF，则在 \demos\common\ota\bootloader\utility\user-config\ota-descriptor.config 中递增 OTA 序列号。对于生成的每个新的 OTA 映像，都应当递增 OTA 序列号。
4. 重新构建项目。

必须将固件更新复制到 Amazon S3 存储桶中，该存储桶的创建方法如[创建 Amazon S3 存储桶以存储更新 \(p. 119\)](#)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：

- Texas Instruments CC3200SF-LAUNCHXL : demos\ti\cc3220_launchpad\ccs\debug\aws_demos.bin
- Microchip Curiosity PIC32MZEF : demos\microchip\curiosity_pic32mzef\mplab\dist\pic32mz_ef_curiosity\production\mplab.production.ota.bin
- Espressif ESP32 : demos/espressif/esp32_devkitc_esp_wrover_kit/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，选择 Manage (管理)，然后选择 Jobs (作业)。
2. 选择 Create。
3. 在 Create an Amazon FreeRTOS Over-the-Air (OTA) update job (创建 Amazon FreeRTOS 无线 (OTA) 更新作业) 下，选择 Create OTA update job (创建 OTA 更新作业)。
4. 可以将一个 OTA 更新部署到单个设备或一组设备。在 Select devices to update (选择要更新的设备) 下，选择 Select (选择)。要更新单个设备，可选择 Things (事物) 选项卡。要更新一组设备，可选择 Thing Groups (事物组) 选项卡。
5. 如果要更新单个设备，选中与设备关联的 IoT 事物旁边的复选框。如果要更新一组设备，选中与设备关联的事物组旁边的复选框。选择 Next (下一步)。
6. 在 Select and sign your firmware image (选择并签署固件映像) 下，选择 Sign a new firmware image for me (为我签署一个新的固件映像)。
7. 在 Code signing profile (代码签名配置文件) 下，选择 Create (创建)。
8. 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 Amazon FreeRTOS 条件的硬件平台。如果使用的是不符合条件的平台，则必须使用 CLI 创建 OTA 更新。有关更多信息，请参阅[使用 AWS CLI 创建 OTA 更新 \(p. 142\)](#)。

- b. 在 Code signing certificate (代码签名证书) 下 , 选择 Select (选择) , 以选择之前导入的证书 , 或选择 Import (导入) 以导入新证书。
- c. 在 Pathname of code signing certificate on device (设备上代码签署证书的路径名) 下 , 输入代码签名证书在设备上的完全限定路径名称。这可能因平台而异。

Note

在 Microchip Curiosity PIC32MZEF 上运行时 , 将首先按照名称在证书存储中搜索代码签名证书。如果未找到 , 则使用内置的证书。

Important

在 Texas Instruments CC3220SF-LAUNCHXL 上 , 如果代码签名证书位于该平台文件系统的根目录下 , 则文件名前面不要添加前导斜线字符 (/)。否则 , OTA 更新将在身份验证过程中失败 , 且产生 file not found 错误。

9. 在 Select your firmware image in S3 or upload it (在 S3 中选择固件映像或上传) 下 , 选择 Select (选择)。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 Curiosity PIC32MZEF 演示项目将产生两个二进制映像 , 其默认名称为 mplab.production.bin 和 mplab.production.ota.bin。如果要上传映像用于 OTA 更新 , 请使用第二个文件。

10. 在 Pathname of firmware image on device (设备上固件映像的路径名) 下 , 输入要将固件映像复制到设备上所在位置的完全限定路径名称。此位置也因平台而异。

Important

在 Texas Instruments CC3220SF-LAUNCHXL 上 , 由于安全限制原因 , 固件映像路径名必须为 /sys/mcuflashimg.bin。

11. 在 IAM role for OTA update job (OTA 更新作业的 IAM 角色) 下 , 选择可访问 S3 存储桶并具有以下策略的角色 :

- AWSIoTThingsRegistration
- AmazonFreeRTOSOTAUpdate

12. 选择 Next (下一步)。

13. 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))。

14. 为 OTA 更新作业输入 ID 和描述 , 然后选择 Create (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 , 输入要将固件映像复制到设备上所在位置的完全限定路径名称。在不同的平台上可能会有所不同。
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 (设备上代码签署证书的路径名) 下，输入代码签名证书在设备上的完全限定路径名称。对于不同的平台可能会有所不同。
3. 在 Pathname of firmware image on device (设备上固件映像的路径名) 下，输入要将固件映像复制到设备上所在位置的完全限定路径名称。在不同的平台上可能会有所不同。
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 Select your firmware image in Amazon S3 (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：

- 您的姓名。
- 您的 IP 地址。
- 您的电子邮件地址。
- 您的位置。
- 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)。

作业将出现在 AWS IoT 控制台中，且其状态为 IN PROGRESS (正在进行)。

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。

有关如何使用控制台创建 OTA 更新的完整演练，请参阅 [OTA 演示应用程序 \(p. 178\)](#)。

使用 AWS CLI 创建 OTA 更新

要使用 AWS CLI 创建 OTA 更新，可执行以下步骤：

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

在使用 CLI 执行 OTA 更新时，可以使用 Code Signing for AWS IoT 或自己签署固件更新。

使用 Code Signing for Amazon FreeRTOS 签署固件映像

要使用 Code Signing for AWS IoT 签署固件映像，必须安装[代码签名工具](#)。下载工具，并阅读自述文件了解安装说明。有关 Code Signing for AWS IoT 的更多信息，请参阅[Code Signing for AWS IoT](#)。

安装并配置代码签名工具后，将未签名的固件映像复制到 Amazon S3 存储桶中，并使用以下 CLI 命令启动代码签名作业。`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>` 和 `<your-destination-bucket-name>` 可以是同一 Amazon S3 存储桶。

以下文本描述了 `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。找到该版本的方法：浏览至 Amazon S3 控制台，选择存储桶，然后在页面顶部的 `Versions`（版本）旁，选择 `Show`（显示）。

destination

已签名的固件在 S3 存储桶中的目的地。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。

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。可以通过浏览至 Amazon S3 控制台并选择存储桶，找到生成的文件名。如果没有看到带有 GUID 文件名的文件，可刷新浏览器。

命令将显示作业 ARN 和作业 ID。稍后会需要这些值。有关 Code Signing for AWS IoT 的更多信息，请参阅 [Code Signing for 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

OTA 更新服务通过 MQTT 消息发送更新。要执行此操作，必须创建包含已签名的固件更新的流。创建 JSON 文件 (stream.json) 标识已签名的固件映像。JSON 文件应当包含以下内容：

```
[  
  {  
    "fileId":<your_file_id>,  
    "s3Location":{  
      "bucket":<your_bucket_name>,  
      "key":<your_s3_object_key>  
    }  
  }  
]
```

以下列表描述了 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。如果使用的是 Code Signing for AWS IoT，则文件名是 Code Signing for AWS IoT 生成的 GUID。

使用 `create-stream` 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>
```

以下列表描述了 `create-stream` 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）。如果使用 Code Signing for AWS IoT，则此密钥为 GUID。

下面是一个 stream.json 文件示例：

```
[  
  {  
    "fileId":123,  
    "s3Location":{  
      "bucket": "codesign-ota-bucket",  
      "key": "48c67f3c-63bb-4f92-a98a-4ee0fbc2bef6"  
    }  
  }  
]
```

role-arn

允许访问 Amazon S3 存储桶的 IAM 角色

要查找已签名的固件映像的 Amazon S3 对象密钥，可使用 aws signer describe-signing-job --job-id <my-job-id> 命令，其中 my-job-id 是 create-signing-job CLI 命令所显示的作业 ID。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

可使用 create-ota-update CLI 命令创建 OTA 更新作业：

```
aws iot create-ota-update --ota-update-id "<my_ota_update>" --target-selection SNAPSHOT  
--description "<a cli ota update>" --files file://<ota.json> --targets arn:aws:iot:<your-  
aws-region>:<your-aws-account>:thing/<your-thing-name> --role-arn arn:aws:iam::<your-aws-  
account>:role/<your-ota-service-role>
```

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：

- 您的姓名
- 您的 IP 地址
- 您的电子邮件地址
- 您的位置

- 银行详细信息
- 医疗信息

`ota-update-id`

任意 OTA 更新 ID。

`target-selection`

有效值为：

- `SNAPSHOT`：作业在将更新部署到所选 IoT 事物或组之后终止。
- `CONTINUOUS`：作业继续将更新部署到已添加到所选组的设备。

`description`

OTA 更新的文本描述。

`files`

JSON 文件的一个或多个引用，包含有关 OTA 更新的数据。JSON 文件可以包含以下属性：

- `fileName`：完全限定的固件映像文件名。对于 Texas Instruments CC3200SF-LAUNCHXL，必须为 `"/sys/mcuflashimg.bin"`。对于 Microchip，必须为 `"mplab.production.bin"`
- `fileLocation`：包含有关固件更新的信息。
 - `stream`：包含固件更新的流。
 - `streamId`：在 `create-stream` CLI 命令中指定的流 ID。
 - `fileId`：在 JSON 文件中指定的传递到 `create-stream` 的文件 ID。
 - `s3Location`：固件更新在 Amazon S3 中的位置。
 - `bucket`：包含固件更新的 Amazon S3 存储桶。
 - `key`：固件更新密钥。
 - `version`：固件更新版本。
- `codeSigning`：包含有关代码签名作业的信息。
 - `awsSignerJobId`：签署人的作业 ID，由 `start-signing-job` 命令生成。
 - `startSigningJobParamater`：启动代码签名作业所需的信息。
 - `signingProfileParameter`：创建签名作业配置文件所需的信息。
 - `certificateArn`：证书的 ACM ARN，用于创建代码签名作业。
 - `platformId`：正在使用的硬件平台的 ID。
 - `certificatePathOnDevice`：证书在设备上的路径。
 - `signingProfileName`：签名配置文件名称。如果不存在此名称的配置文件，则必须为 `signingProfileParameter` 提供值。如果存在具有指定名称的配置文件，并且为 `signingProfileParameter` 提供了值，则提供的值必须与用于签名配置文件的值完全匹配。
 - `destination`：已签名的项目所放置的位置。
 - `s3Destination`：已签名的项目所放置的 Amazon S3 存储桶。
 - `bucket`：Amazon S3 存储桶。
 - `prefix`：代码签名项目的前缀。默认情况下，该名称为 `signedImage/`。此属性将在文件夹下创建一个名为 `signedImage` 的文件夹。
 - `customCodeSigning`：包含有关自定义签名的信息。
 - `signature`：包含自定义签名。
 - `inlineDocument`：自定义签名。
 - `certificateChain`：包含自定义签名的证书链。
 - `certificateName`：代码签署证书在设备上的路径名。
 - `inlineDocument`：证书链。
 - `hashAlgorithm`：用于创建签名的哈希算法。

- `signatureAlgorithm`：用于代码签名的签名算法。
- `attributes`：任意键/值对。

`targets`

一个或多个 IoT 事物 ARN，用于指定要通过 OTA 更新进行更新的设备。

`role-arn`

服务角色的 ARN。

以下示例显示了如何将 JSON 文件传递到 `create-ota-update` 命令，该命令采用的是 Code Signing for AWS IoT：

```
[  
  {  
    "fileName": "firmware.bin",  
    "fileLocation": {  
      "stream": {  
        "streamId": "004",  
        "fileId": 123  
      }  
    },  
    "codeSigning": {  
      "awsSignerJobId": "48c67f3c-63bb-4f92-a98a-4ee0fbc2bef6"  
    }  
  }  
]
```

以下示例显示了如何将 JSON 文件传递到 `create-ota-update` CLI 命令，该命令使用内联文件提供自定义代码签名材料：

```
[  
  {  
    "fileName": "firmware.bin",  
    "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>"  
      }  
    }  
  }  
]
```

以下示例显示了如何将 JSON 文件传递到 `create-ota-update` CLI 命令，该命令允许 Amazon FreeRTOS OTA 启动代码签名作业并创建代码签名配置文件和流：

```
[
```

```
[  
 {  
   "fileName": "<your_firmware_path_on_device>",  
   "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>",  
         "platformId": "<your_platform_id>",  
         "certificatePathOnDevice": "<certificate_path>"  
       },  
       "destination": {  
         "s3Destination": {  
           "bucket": "<your_destination_bucket>"  
         }  
       }  
     }  
   }  
 }  
 ]
```

以下示例显示了如何将 JSON 文件传递到 create-ota-update CLI 命令，该命令创建了 OTA 更新，OTA 更新使用现有配置文件启动代码签名作业并使用指定的流：

```
[  
 {  
   "fileName": "<your_firmware_path_on_device>",  
   "fileVersion": "1",  
   "fileLocation": {  
     "s3Location": {  
       "bucket": "<your_bucket_name>",  
       "key": "<your_object_key>",  
       "version": "<your_S3_object_version>"  
     }  
   },  
   "codeSigning":{  
     "startSigningJobParameter":{  
       "signingProfileName": "<your_unique_profile_name>",  
       "destination": {  
         "s3Destination": {  
           "bucket": "<our_destination_bucket>"  
         }  
       }  
     }  
   }  
 }  
 ]
```

以下示例显示了如何将 JSON 文件传递到 create-ota-update CLI 命令，该命令允许 Amazon FreeRTOS OTA 使用现有代码签名作业 ID 创建流：

```
[  
 {  
   "fileName": "<your_firmware_path_on_device>",  
   "fileVersion": "1"  
   "codeSigning":{  
 }
```

```
    "awsSignerJobId": "<your_signer_job_id>"  
  }  
]  
]
```

以下示例显示了如何将 JSON 文件传递到 create-ota-update CLI 命令，该命令创建了 OTA 更新。更新根据指定的 S3 对象创建流，并使用自定义代码签名：

```
[  
  {  
    "fileName": "<your_firmware_path_on_device>",  
    "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>"  
      }  
    }  
  }  
]
```

可以使用 get-ota-update CLI 命令来获取 OTA 更新的状态：

```
aws iot get-ota-update --ota-update-id <your-ota-update-id>
```

此命令返回以下值之一：

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。

列出 OTA 更新

可以使用 list-ota-updates CLI 命令获取所有 OTA 更新的列表：

```
aws iot list-ota-updates
```

list-ota-updates 命令的输出如下所示：

```
{
    "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

可以使用 get-ota-update CLI 命令获取有关特定 OTA 更新的信息：

```
aws iot get-ota-update --ota-update-id <my-ota-update-id>
```

get-ota-update 命令的输出如下所示：

```
{
    "otaUpdateInfo": {
        "otaUpdateId": "myotaupdate1",
        "otaUpdateArn": "arn:aws:iot:us-west-2:123456789012:otaupdate/my_ota_update",
        "creationDate": 1522444438.424,
        "lastModifiedDate": 1522444440.681,
        "description": "a test OTA update",
        "targets": [
            "arn:aws:iot:us-west-2:123456789012:thing/myDevice"
        ],
        "targetSelection": "SNAPSHOT",
        "otaUpdateFiles": [
            {
                "fileName": "app.bin",
                "fileLocation": {
                    "stream": {
                        "streamId": "003",
                        " fileId": 123
                    }
                }
            }
        ],
        "codeSigning": {
            "awsSignerJobId": "592932bb-24a1-4f91-8ddd-66145352ad19",
            ...
        }
    }
}
```

```
        "customCodeSigning": {}  
    }  
}  
],  
"otaUpdateStatus": "CREATE_COMPLETE",  
"awsIotJobId": "f76da3c0_10eb_41df_9029_ba7abc20f609",  
"awsIotJobArn": "arn:aws:iot:us-  
west-2:123456789012:job:f76da3c0_10eb_41df_9029_ba7abc20f609"  
}  
}
```

删除与 OTA 相关的数据

目前还不能使用 AWS IoT 控制台来删除流或 OTA 更新。可以使用 AWS CLI 来删除流、OTA 更新以及在 OTA 更新过程中创建的 IoT 作业。

删除 OTA 流

在创建 OTA 更新时，您或 AWS IoT 控制台将创建流以将固件分割为多个数据块，以便能通过 MQTT 发送。可以使用 `delete-stream` CLI 命令删除该流。例如：

```
aws iot delete-stream --stream-id <your_stream_id>
```

删除 OTA 更新

在创建 OTA 更新时，将创建以下事物：

- OTA 更新作业数据库中的一个条目。
- 执行更新的 AWS IoT 作业。
- 每个要更新的设备的 AWS IoT 作业执行。

`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 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 作业。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

在创建 OTA 更新时，Amazon FreeRTOS 会创建 AWS IoT 作业。也会为处理作业的每个设备创建作业执行。可以使用 `delete-job` CLI 命令删除作业及关联的作业执行：

```
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。

作业的删除可能需要几分钟，具体取决于为作业创建的作业执行数量以及其他因素。在删除作业过程中，作业的状态将显示为 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。此命令将删除单个设备的作业执行。例如：

```
aws iot delete-job-execution --job-id <your-job-id> --thing-name
    <your-thing-name> --execution-number
    <your-job-execution-number> --no-force
```

如同 `delete-job` CLI 命令一样，可以将 `--force` 参数传递到 `delete-job-execution` 以强制删除正在执行的作业执行。有关更多信息，请参阅 [DeleteJobExecution API](#)。

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能够恢复到已知状态。

有关使用 OTA 更新应用程序的更多信息，请参阅 [OTA 演示应用程序 \(p. 178\)](#)。

OTA Update Manager 服务

OTA Update Manager 服务可用于：

- 创建 OTA 更新。
- 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。
- 要发送到目标设备的文件列表。
- 允许访问 AWS IoT 作业服务的 IAM 角色。
- 用户定义的名称值对列表 (可选)。

OTA 更新设计用于更新设备固件，但也可以用来将所需的任意文件发送到已注册 AWS IoT 的一个或多个设备。在通过无线发送文件时，最佳实践是对文件进行数字签名，以便接收文件的设备可以验证文件在传输途中未经篡改。可以使用 [Code Signing for Amazon FreeRTOS](#) 来签署文件，也可以使用自己的代码签名工具。

在对文件进行数字签名后，可以使用 Amazon 流服务创建流。该服务将文件分割为多个数据块，以便通过 MQTT 发送到设备。

在创建 OTA 更新时，OTA Manager 服务将创建 AWS IoT 作业以通知设备有可用的更新。Amazon 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，代理将对更新进行流式处理以通过 MQTT，并将文件存储在本地。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 Amazon 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

OTA 代理旨在简化为将 OTA 更新功能添加到产品中所必须编写的代码。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 完成事件消息（后者为可选操作）。

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。要熟悉如何配置 AWS 账户的 AWS IoT 事物、凭证、代码签名证书、预配置设备和 OTA 更新作业，请参阅 [Amazon FreeRTOS 先决条件](#)。

MQTT 连接管理

OTA 代理使用 MQTT 协议完成与 AWS IoT 服务的所有通信，但它并不管理 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在内。

简单的 OTA 演示

下面是一个简单的 OTA 演示节选，显示了代理是如何连接到 MQTT 代理并初始化 OTA 代理的。在该示例中，我们对演示进行配置，使用默认的 OTA 完成回调，并每隔一秒简单地打印输出某些统计数据。为简洁起见，我们省略了演示的某些细节。

有关使用 AWS IoT MQTT 代理的工作示例，请参阅 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。

```
/* Create the MQTT Client. */
if( MQTT_AGENT_Create( &( xMQTT_h ) ) == eMQTTAgentSuccess )
{
    for ( ; ; )
    {
        memset( &xConnParm, 0, sizeof( xConnParm ) );

        /* ... Set MQTT connection parameters here per your application needs ... */

        configPRINTF( ( "Connecting to %s\r\n", clientcredentialMQTT_BROKER_ENDPOINT ) );
        if( MQTT_AGENT_Connect( xMQTT_h, &( xConnParm ), myappMAX_AWS_CONNECT_WAIT_IN_TICKS ) == eMQTTAgentSuccess )
        {
            configPRINTF( ( "Connected to broker.\r\n" ) );

            /* Initialize the OTA Agent with the default completion callback handler. */
            OTA_AgentInit( xMQTT_h, ( const uint8_t * )( clientcredentialIOT_THING_NAME ), NULL,
/* NULL uses the default
callback handler. */ ( TickType_t ) ~0 );

            while( ( eState = OTA_GetAgentState() ) != eOTA_AgentState_NotReady )
            {
                /* Wait forever for OTA traffic but allow other tasks to run

```

```
        and output statistics only once per second. */

    vTaskDelay( myappONE_SECOND_DELAY_IN_TICKS );
    configPRINTF( ( "State: %s Received: %u Queued: %u Processed: %u
Dropped: %u\r\n",
                    pcStateStr[eState],
                    OTA_GetPacketsReceived(),
                    OTA_GetPacketsQueued(),
                    OTA_GetPacketsProcessed(),
                    OTA_GetPacketsDropped() ) );
}

/* ... Handle MQTT disconnect per your application needs ... */
}
else
{
    configPRINTF( ( "ERROR: MQTT_AGENT_Connect() Failed.\r\n" ) );
}
/* After failure to connect or a disconnect, wait an arbitrary one second before
retry. */
vTaskDelay( myappONE_SECOND_DELAY_IN_TICKS );
}
else
{
    configPRINTF( ( "Failed to create MQTT client.\r\n" ) );
}
```

以下是该演示应用程序的主要流程：

- 创建 MQTT 代理上下文。
- 连接到 AWS IoT 终端节点。
- 初始化 OTA 代理。
- 循环执行 OTA 更新作业并每秒钟输出一次统计数据。
- 如果代理停止，请等待一秒后尝试重新连接。

使用自定义回调处理 OTA 完成事件

以上示例使用了内置的回调处理程序来处理 OTA 完成事件，方法是将 OTA_AgentInit API 的第三个参数指定为 NULL。如果要对完成事件实施自定义处理，则必须将回调处理程序的函数地址传递到 OTA_AgentInit API。在 OTA 过程中，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。

```
/**
 * @brief OTA Job callback events.
 *
 * After an OTA update image is received and authenticated, the agent calls the user
 * callback (set with the OTA_AgentInit API) with the value eOTA_JobEvent_Activate 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 eOTA_JobEvent_StartTest, 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 eOTA_JobEvent_Fail instead to allow the user to log the failure and take
 * any action deemed appropriate by the user code.
 */
typedef enum {
    eOTA_JobEvent_Activate, /*! OTA receive is authenticated and ready to activate. */
    eOTA_JobEvent_Fail,     /*! OTA receive failed. Unable to use this update. */
}
```

```
    eOTA_JobEvent_StartTest /*! OTA job is now in self test, perform user tests. */  
} OTA_JobEvent_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：

eOTA_JobEvent_Activate 事件

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（如有必要）。

eOTA_JobEvent_Fail 事件

如果回调处理程序收到此事件，则更新失败。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。

eOTA_JobEvent_StartTest 事件

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 代理将把 eOTA_JobEvent_StartTest 事件发送给已准备好进行测试的回调函数。开发人员可以选择添加任何必要的测试，以确定设备固件在更新后可以正常使用。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(eOTA_ImageState_Accepted) 函数，将该固件提交为新的永久映像。

如果设备没有特殊的硬件或机制需要进行测试，则可以使用默认的回调处理程序。一旦收到 eOTA_JobEvent_Activate 事件，默认处理程序将立即重置设备。

OTA 安全性

以下是 OTA 安全性的三个方面：

连接安全性

OTA Update Manager 依赖于现有安全机制，如 AWS IoT 使用的 TLS 双向身份验证。OTA 更新流量经由 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 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。Amazon FreeRTOS OTA Update Manager 使用 Code Signing for AWS IoT 自动签署固件。有关更多信息，请参阅 [Code Signing for AWS IoT](#)。在固件抵达设备时，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，必须有权调用 CreateDeployment、CreateJob 和 CreateStream API。此外，在 Amazon S3 存储桶策略或 ACL 中，必须赋予 AWS IoT 服务委托人读取权限，以便在流式处理过程中可以访问存储在 Amazon S3 中的固件更新。

Code Signing for AWS IoT

对于 AWS IoT 支持的任何设备，AWS IoT 控制台将使用 [Code Signing for AWS IoT](#) 自动签署固件映像。

Code Signing for AWS IoT 使用您导入 ACM 的证书和私有密钥。您可以使用自签名证书进行测试，但我们建议您从知名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 密钥使用和扩展密钥使用扩展。密钥使用扩展设置为 Digital Signature，扩展密钥使用扩展设置为 Code Signing。有关签署代码映像的更多信息，请参阅 [Code Signing for AWS IoT 开发人员指南](#) 和 [Code Signing for AWS IoT API 参考](#)。

Note

可以从 <https://tools.signer.aws.a2z.com/awssigner-tools-v2.zip> 下载 Code Signing for AWS IoT 开发工具包。

OTA 故障排除

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

- [为 OTA 更新设置 Cloudwatch 日志 \(p. 156\)](#)
- [使用 AWS CloudTrail 记录 AWS IoT OTA API 调用 \(p. 159\)](#)
- [使用 Texas Instruments CC3220SF Launchpad 排查 OTA 更新问题 \(p. 161\)](#)

为 OTA 更新设置 Cloudwatch 日志

OTA 更新服务支持 Amazon CloudWatch 日志记录。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 CloudWatch Logs 的更多信息，请参阅 [Cloudwatch 日志](#)。

要启用日志记录，必须创建 IAM 角色并配置 OTA 更新登录。

Note

在启用 OTA 更新日志记录之前，请务必了解 CloudWatch Logs 访问权限。拥有 CloudWatch Logs 访问权限的用户可以查看您的调试信息。有关信息，请参阅 [Amazon CloudWatch Logs 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 Settings (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 Set role (设置角色) 下，选择 Create new (新建) 以创建日志记录 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 Update。

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。
- 创建 AWS IoT 作业。
- AWS IoT 作业已完成。

- 创建流。

可以在 [CloudWatch 控制台](#) 中查看日志。

在 CloudWatch Logs 中查看 OTA 更新

1. 从导航窗格中，选择 Logs (日志)。
2. 在 Log Groups (日志组) 中，选择 AWSIoTLogsV2。

OTA 更新日志可以包含以下属性：

accountId

在其中生成日志的 AWS 账户 ID。

actionType

生成日志的操作。可以设置为以下值之一：

- CreateOTAUpdate：创建 OTA 更新。
- DeleteOTAUpdate：删除 OTA 更新。
- StartCodeSigning：启动代码签名作业。
- CreateAWSJob：创建 AWS IoT 作业。
- CreateStream：创建流。
- GetStream：流的请求已发送到 AWS IoT 流服务。
- DescribeStream：流相关信息的请求已发送到 AWS IoT 流服务。

awsJobId

生成日志的 AWS IoT 作业 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

有关生成日志操作的其他信息。

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。

status

生成日志操作的状态。有效值为：

- 成功
- 失败

streamId

生成日志的 AWS IoT 流 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": "875157236366",  
    "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 作业时生成的日志示例：

```
{  
    "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."
}
```

以下是设备向流服务请求流时生成的日志示例：

```
{
  "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 AWS CloudTrail 记录 AWS IoT OTA API 调用

Amazon FreeRTOS 已与 CloudTrail 集成，后者是一种服务，可捕获所有 AWS IoT OTA API 调用，并将日志文件传输至指定的 Amazon S3 存储桶。CloudTrail 从您的代码捕获针对 AWS IoT OTA API 的 API 调用。通过使用 CloudTrail 收集的信息，可以确定向 AWS IoT OTA 发出的请求、发出请求的源 IP 地址、请求的发起人以及发出时间等。

要了解有关 CloudTrail 的更多信息，包括如何对其进行配置和启用，请参阅 [AWS CloudTrail User Guide](#)

CloudTrail 中的 Amazon FreeRTOS 信息

在 AWS 账户中启用 CloudTrail 日志记录后，将在 CloudTrail 日志文件中跟踪对 AWS IoT OTA 操作发出的大多数 API 调用，这些调用将与其他 AWS 服务记录一起写入日志文件。CloudTrail 根据时间段和文件大小来确定何时创建并写入新文件。

Note

CloudTrail 不会记录 AWS IoT OTA 数据层面的操作（设备端）。使用 CloudWatch 可监控这些操作。

CloudTrail 将记录 AWS IoT OTA 控制层面的操作。例如，对 CreateOTAUpdate、GetOTAUpdate 和 CreateStream 部分的调用将在 CloudTrail 日志文件中生成条目。

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身份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 userIdentity 元素](#)。AWS OTA IoT 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，将使用 Amazon S3 服务器端加密 (SSE) 对日志文件进行加密。

如果您需要获得日志文件传输的通知，则可以将 CloudTrail 配置为在传输新日志文件时发布 Amazon SNS 通知。有关更多信息，请参阅[CloudTrail 配置 Amazon SNS 通知](#)。

还可以将多个 AWS 区域和多个 AWS 账户中的 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[接收多个区域中的 CloudTrail 日志文件](#)和[从多个账户中接收 CloudTrail 日志文件](#)。

了解 Amazon 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，该条目演示了调用 CreateOTAUpdate 操作的日志。

```
{  
  "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",  
  "version": "1.05",  
  "region": "us-east-1",  
  "sourceIPAddress": "127.0.0.1",  
  "userAgent": "aws-sdk-node/2.395.0",  
  "awsRegion": "us-east-1",  
  "versionId": "1.05",  
  "eventSource": "iot:ota",  
  "eventName": "CreateOTAUpdate",  
  "awsRequestID": "12345678901234567890123456789012",  
  "resourceARN": "arn:aws:iot:us-east-1:123456789012:ota-update/12345678901234567890123456789012",  
  "resourceType": "ota-update",  
  "awsResponseURL": "https://iot.us-east-1.amazonaws.com/ota-update/12345678901234567890123456789012",  
  "status": "Success",  
  "error": null,  
  "response": null, "request": null}
```

```
"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 Texas Instruments CC3220SF Launchpad 排查 OTA 更新问题

CC3220SF Launchpad 平台提供了软件篡改检测机制，该机制采用安全警报计数器，每当完整性遭到破坏计数器便会递增。当安全警报计数器达到预先确定的阈值（默认值为 15），且主机收到异步事件 SL_ERROR_DEVICE_LOCKED_SECURITY_ALERT 时，设备将锁定。锁定的设备随后将只具备有限的可访问性。要恢复设备，可以对设备重新编程，或使用“恢复出厂设置”流程来恢复为出厂映像。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。有关更多信息，请参阅 [Texas Instruments SimpleLink CC3120、CC3220 Wi-Fi Internet-on-a-chip 解决方案内置安全功能应用程序报告](#)。

Amazon FreeRTOS 控制台

可以使用 [Amazon FreeRTOS 控制台](#) 管理软件配置，并为设备下载 Amazon FreeRTOS 软件。Amazon FreeRTOS 软件在各种平台上进行了资格预审。包括所需的硬件驱动程序、库和项目示例，以帮助您快速入门。您可以选择预定义配置，或创建自定义配置。

请确保您已完成了 Amazon FreeRTOS 入门先决条件。

预定义的 Amazon FreeRTOS 配置

预定义配置针对以下经过资格预审的平台而定义：

- TI CC3220SF-LAUNCHXL
- STM32 IoT Discovery Kit
- NXP LPC54018 IoT Module
- Microchip Curiosity PIC32MZEF
- Infineon XMC4800 IoT 连接工具包
- Renesas RSK+ for RX65N-2MB
- MediaTek MT7697Hx 开发工具包
- Xilinx Avnet MicroZed 工业 IoT 初学者工具包
- FreeRTOS Windows 模拟器

有了预定义配置，您可以借助支持的使用案例迅速上手，而无需考虑需要使用哪个库。要使用预定义配置，可浏览至 [Amazon FreeRTOS 控制台](#)，找到要使用的配置，然后选择 Download (下载)。

如果要更改配置的 Amazon FreeRTOS 版本、硬件平台或库，也可以自定义预定义配置。自定义预定义配置将创建新的自定义配置，并不会覆盖 Amazon FreeRTOS 控制台中的预定义配置。

根据预定义配置创建自定义配置

1. 浏览至 [Amazon FreeRTOS 控制台](#)。
2. 在导航窗格中，选择 Software (软件)。
3. 在 Amazon FreeRTOS Device Software (Amazon FreeRTOS 设备软件) 下，选择 Configure download (配置下载)。
4. 选择要自定义的预定义配置旁边的省略号，然后选择 Customize (自定义)。
5. 在 Configure Amazon FreeRTOS Software (配置 Amazon FreeRTOS 软件) 页面上，选择 Amazon FreeRTOS 版本、硬件平台和库，然后为新配置指定名称和描述。
6. 在页面底部，选择 Create and download (创建和下载) 以创建和下载自定义配置。

自定义 Amazon FreeRTOS 配置

您可以通过自定义配置指定硬件平台、集成开发平台 (IDE)、编译器以及必需的 RTOS 库。这样一来，可以在设备上留出更多的空间用于应用程序代码。

创建自定义配置

1. 浏览至 [Amazon FreeRTOS 控制台](#)，然后选择 Create new (新建)。
2. 选择要使用的 Amazon FreeRTOS 版本。默认情况下使用最新版本。
3. 在 New Software Configuration (新建软件配置) 页面上，选择 Select a hardware platform (选择硬件平台)，然后从经过资格预审的平台中选择一个。
4. 选择要使用的 IDE 和编译器。
5. 对于所需的 Amazon FreeRTOS 库，选择 Add Library (添加库)。如果选择的库需要另一个库，将为您添加此库。如果要选择更多库，可选择 Add another library (添加另一个库)。
6. 在 Demo Projects (演示项目) 部分中，启用演示项目之一。将在项目文件中启用演示。
7. 在 Name required (需要名称) 中，为自定义配置输入名称。

Note

请勿在自定义配置名称中使用任何个人身份信息。

8. 在 Description (描述) 中，为自定义配置输入说明。
9. 在页面底部，选择 Create and download (创建和下载) 以创建和下载自定义配置。

编辑自定义配置

1. 浏览至 [Amazon FreeRTOS 控制台](#)。
2. 在导航窗格中，选择 Software (软件)。
3. 在 Amazon FreeRTOS Device Software (Amazon FreeRTOS 设备软件) 下，选择 Configure download (配置下载)。
4. 选择要编辑的配置旁边的省略号，然后选择 Edit (编辑)。
5. 在 Configure Amazon FreeRTOS Software (配置 Amazon FreeRTOS 软件) 页面上，可以更改配置的 Amazon FreeRTOS 版本、硬件平台、库和描述。
6. 在页面底部，选择 Save and download (保存并下载) 以保存并下载配置。

删除自定义配置

1. 浏览至 [Amazon FreeRTOS 控制台](#)。
2. 在导航窗格中，选择 Software (软件)。
3. 在 Amazon FreeRTOS Device Software (Amazon FreeRTOS 设备软件) 下，选择 Configure download (配置下载)。
4. 选择要删除的配置旁边的省略号，然后选择 Delete (删除)。

快速连接工作流程

Amazon FreeRTOS 控制台还包括适用于具有预定义配置的所有主板的快速连接工作流程选项。快速连接工作流程帮助您为 AWS IoT 和 AWS IoT Greengrass 配置并运行 Amazon FreeRTOS 演示应用程序。要开始使用，请选择 Predefined configurations (预定义配置) 选项卡，找到您的主板，然后选择 Quick connect (快速连接) 并按照快速连接工作流程步骤进行操作。

Amazon FreeRTOS 问题排查

Amazon FreeRTOS 支持 Amazon CloudWatch 和 AWS CloudTrail 日志记录服务，帮助排查 Amazon FreeRTOS 无线更新的问题。有关 OTA 更新问题排查的更多信息，请参阅 [OTA 问题排查](#)。

Amazon FreeRTOS 演示项目

在您对 Amazon FreeRTOS 有了初步了解之后，本节包含的资源非常有用。如果您尚未掌握基本知识，建议您首先阅读 [Amazon FreeRTOS 入门 \(p. 4\)](#)。

主题

- [浏览演示应用程序 \(p. 164\)](#)
- [低功耗蓝牙演示应用程序（测试版）\(p. 165\)](#)
- [安全套接字 Echo 客户端演示 \(p. 175\)](#)
- [设备影子演示应用程序 \(p. 176\)](#)
- [Greengrass Discovery 示例应用程序 \(p. 177\)](#)
- [OTA 演示应用程序 \(p. 178\)](#)
- [Microchip Curiosity PIC32MZEF 的演示启动加载程序 \(p. 181\)](#)

浏览演示应用程序

本节包含有关如何组织目录和文件的信息以及演示的配置文件。

目录和文件组织结构

主 Amazon FreeRTOS 目录中有两个子文件夹：

- `demos`

包含可在 Amazon FreeRTOS 设备上运行以演示 Amazon FreeRTOS 功能的示例代码。对于选定的每个目标平台，都有一个子目录。这些子目录包含演示使用的代码，但并非所有演示都可以独立运行。如果您使用的是 [Amazon FreeRTOS 控制台](#)，则在 `demos` 下只有您选择的目标平台的子目录。

`AmazonFreeRTOS\demos\common\demo_runner\aws_demo_runner.c` 中的函数 `DEMO_RUNNER_RunDemos()` 包含调用各个示例的代码。默认情况下，仅调用 `vStartMQTTEchoDemo()` 函数。根据您在下载代码时选择的配置，或者是从 GitHub 获取代码，将注释掉或完全忽略其他示例运行程序函数。虽然您可以更改此处选择的演示，但请注意，并非所有的示例组合都能协同工作。根据组合，由于内存约束，软件可能无法在选定目标上执行。可由 Amazon FreeRTOS 执行的所有示例显示在 `demos` 下的公用目录中。

- `lib`

`lib` 目录包含 Amazon 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。如果您需要更改这些库之一，请确保它遵循在 `libs/include` 目录中定义的库接口。

配置文件

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 `demos/<vendor>/<platform>/common/config_files` 中找到配置文件。

低功耗蓝牙演示应用程序 (测试版)

低功耗蓝牙 (BLE) 库是面向 Amazon FreeRTOS 的公共测试版，可能会发生变化。

概述

Amazon FreeRTOS BLE 包括三个演示应用程序：

[MQTT over BLE \(p. 170\) 演示](#)

此应用程序演示如何使用 MQTT over BLE 服务。

[Wi-Fi 预配置 \(p. 172\) 演示](#)

此应用程序演示如何使用 Wi-Fi 预配置服务。

[通用属性服务器 \(p. 174\) 演示](#)

此应用程序演示如何使用 Amazon FreeRTOS BLE 中间件 API 来创建一个简单的 GATT 服务器。

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 Amazon FreeRTOS 蓝牙设备的 iOS 开发工具包 \(p. 93\)](#) 或 [适用于 Amazon FreeRTOS 蓝牙设备的 Android 开发工具包 \(p. 93\)](#)。

[为 Amazon FreeRTOS BLE 设置 AWS IoT 和 Amazon Cognito](#)

要跨 MQTT 将设备连接到 AWS IoT，您需要设置 AWS IoT 和 Amazon Cognito。

设置 AWS IoT

1. 在 <https://aws.amazon.com> 上设置 AWS 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。由于移动开发工具包使用 Amazon Cognito 进行设备身份验证，您无需为使用 BLE 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。记录 AWS IoT REST API 终端节点。

有关设置的更多信息，请参阅 [AWS IoT 入门](#)。

创建 Amazon Cognito 用户池

1. 打开 Amazon Cognito 控制台，并选择 Manage User Pools (管理用户池)。
2. 选择 Create a user pool。

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，并选择管理身份池。
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 Create Pool。
5. 展开查看详细信息，记下两个 IAM 角色名称。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，用于访问 Amazon Cognito。
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 策略并将其附加到经过身份验证的身份

1. 打开 IAM 控制台，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。

将您的 AWS IoT 和 Amazon Cognito 信息记录下来。您需要终端节点 ID 来对移动应用程序进行 AWS 云的身份验证。

为 BLE 设置您的 Amazon FreeRTOS 环境

要设置环境，您需要在微控制器上下载 Amazon FreeRTOS 以及 [Amazon FreeRTOS 低功耗蓝牙库（测试版）\(p. 84\)](#)，并下载和配置适用于您移动设备上 Amazon FreeRTOS 蓝牙设备的移动开发工具包。

通过 Amazon FreeRTOS BLE 设置您的微控制器环境

1. 下载 Amazon FreeRTOS [GitHub](#) 存储库的 `feature/ble-beta` 分支。
2. 在微控制器上设置 Amazon FreeRTOS。

有关在符合 Amazon FreeRTOS 标准的微控制器上开始使用 Amazon FreeRTOS 的信息，请参阅[开始使用 Amazon FreeRTOS](#) 中有关您主板的指南。

Note

您可在任何启用了 BLE 并具有 Amazon FreeRTOS 和移植的 Amazon FreeRTOS BLE 库的微控制器上运行演示。目前，Amazon FreeRTOS [MQTT over BLE \(p. 170\)](#) 演示项目已经完全移植到以下启用 BLE 的设备：

- STMicroelectronics STM32L4 Discovery Kit IoT Node，具有 STBTLE-1S BLE 模块
- Espressif ESP32-DevKitC 和 ESP-WROVER-KIT
- Nordic nRF52840-DK

常见组件

该 Amazon FreeRTOS 演示应用程序有两个常见组件：

- 网络管理器
- BLE 移动开发工具包演示应用程序

网络管理器

网络管理器管理微控制器的网络连接。它位于您的 Amazon FreeRTOS 目录中的 `\demos\common\network_manager\aws_iot_network_manager.c`。如果为 Wi-Fi 和 BLE 均启用了网络管理器，则默认情况下通过 BLE 启动演示。如果 BLE 连接中断，并且您的主板启用了 Wi-Fi，则网络管理器切换到可用的 Wi-Fi 连接来防止您从网络断开连接。

要通过网络管理器启用某种网络连接类型，请将该网络连接类型添加到 `demos/vendor/board/common/config_files/aws_iot_network_config.h` 中的 `configENABLED_NETWORKS` 参数。例如，如果您同时启用了 BLE 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 `lib\include\aws_iot_network.h` 中以 `#define AWSIOT_NETWORK_TYPE` 开头的行。

Amazon FreeRTOS BLE 移动开发工具包演示应用程序

Amazon FreeRTOS BLE 移动开发工具包演示应用程序位于 `amazon-freertos-ble-android-sdk/app` 下 [Amazon FreeRTOS 蓝牙设备的 Android 开发工具包](#) 和 `amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo` 下 [Amazon FreeRTOS 蓝牙设备的 iOS 开发工具包](#) 中。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。

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. 确认您已安装 [适用于 Amazon FreeRTOS 蓝牙设备的 iOS 开发工具包 \(p. 93\)](#)。
2. 从 `amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/` 发布以下命令：

```
$ sudo pod install
```

3. 打开具有 Xcode 的项目，将签名开发人员账户更改为您的账户。
4. 打开 `amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/AmazonFreeRTOSDemo/Amazon/AmazonConstants.swift` 并重新定义以下变量：
 - `region`：您的 AWS 区域。
 - `iotPolicyName`：您的 AWS IoT 策略名称。
 - `mqttCustomTopic`：您要发布到的 MQTT 主题。
5. 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`：您的 AWS 区域。

在 `CognitoUserPool` 下，重新定义以下变量：

- `PoolId`：您的 Amazon Cognito 用户池 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`：您的 AWS 区域。

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 Amazon FreeRTOS 蓝牙设备的 Android 开发工具包 \(p. 93\)](#)。
2. 打开 `amazon-freertos-ble-android-sdk/app/src/main/java/com/amazon/aws/freertosandroid/AuthenticatorActivity.java` 并重新定义以下变量：
 - `AWS_IOT_POLICY_NAME`：您的 AWS IoT 策略名称。
 - `AWS_IOT_REGION`：您的 AWS 区域。
 - `COGNITO_POOL_ID`：您的 Amazon Cognito 身份池 ID。
 - `COGNITO_REGION`：您的 AWS 区域。
3. 打开 `amazon-freertos-ble-android-sdk/app/src/main/java/com/amazon/aws/freertosandroid/MainActivity.java` 并重新定义以下变量：
 - `BLE_DEVICE_MAC_ADDR`：设备的 MAC 地址。
 - `BLE_DEVICE_NAME`：您的设备名称。
 - `MTU`：微控制器与移动设备之间所需的 MTU。
4. 打开 `amazon-freertos-ble-android-sdk/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`：您的 AWS 区域。

在 `CognitoUserPool` 下，重新定义以下变量：

- `PoolId`：您的 Amazon Cognito 用户池 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`：您的 AWS 区域。

通过 BLE 发现微控制器并建立安全连接

1. 在您的微控制器上运行 BLE 演示项目。
2. 在移动设备上运行 BLE 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
```

3. 确认您的微控制器显示在 BLE 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 Amazon FreeRTOS 的设备以及设备信息服务 (\lib\bluetooth_low_energy\services\device_information) 显示在列表中。

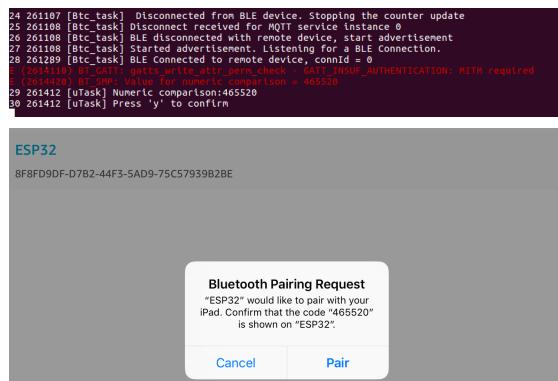
4. 从设备列表中选择您的微控制器。应用程序与主板建立连接，在已连接设备的旁边会显示一根绿色线条。



您可以通过将线条拖动到左侧，从微控制器上断开连接。



5. 系统可能会提示您将微控制器与移动设备配对。



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

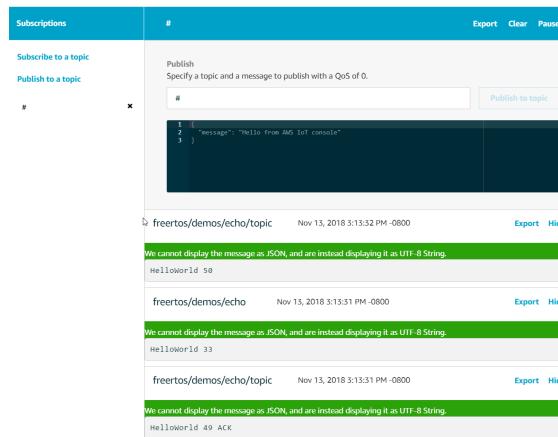
BLE 移动开发工具包演示应用程序使用 Amazon Cognito 进行用户身份验证。确保您已设置 Amazon Cognito 用户和身份池，并且将 IAM 策略附加到了通过身份验证的身份。

MQTT over BLE

在 MQTT over BLE 演示中，您的微控制器通过 MQTT 代理将消息发布到 AWS 云。

订阅演示 MQTT 主题

1. 登录到 AWS IoT 控制台。
2. 在导航窗格中，选择测试以打开 MQTT 客户端。
3. 在订阅主题中，输入 `freertos/demos/echo`，然后选择订阅主题。



您可以通过 BLE 或 Wi-Fi 连接运行 MQTT 演示。网络管理器 (p. 167) 的配置确定所用的连接类型。

如果您使用 BLE 将微控制器与移动设备配对，则通过您移动设备上的 BLE 移动开发工具包演示应用程序来路由 MQTT 消息。

如果您使用 Wi-Fi，演示与 MQTT Hello World 演示项目一样位于 `demos/common/mqtt/aws_hello_world.c` 中。该演示用于大多数开始使用 Amazon FreeRTOS 演示项目。

启用演示

如果您已按照设备入门指南中的说明启用了 BLE 演示，则可以跳过这些说明。

1. 启用 Wi-Fi 预配置服务。打开 `demos/vendor/board/common/config_files/aws_ble_config.h` 并将 `#define bleconfigENABLE_WIFI_PROVISIONING` 设置为 1。

Note

默认情况下禁用 Wi-Fi 预配置服务。

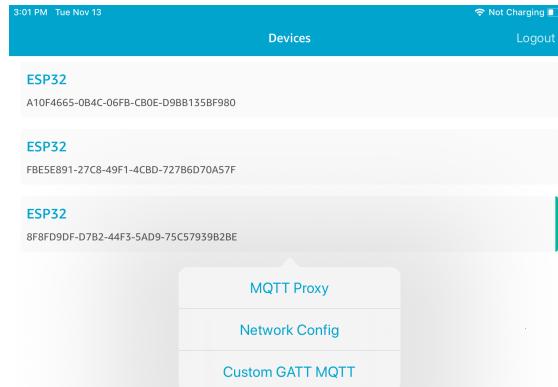
2. 打开 `demos\common\demo_runner\aws_demo_runner.c`，在演示声明中取消注释 `extern void vStartMQTTBLEEchoDemo(void);`。在 `DEMO_RUNNER_RunDemos` 定义中，取消注释 `vStartMQTTBLEEchoDemo();`。

运行演示

如果仅为 Wi-Fi 配置了网络管理器，则只需在主板上生成并运行演示项目。

如果为 BLE 配置了网络管理器，请执行以下操作：

1. 在微控制器上生成并运行演示项目。
2. 确保您已使用 [Amazon FreeRTOS BLE 移动开发工具包演示应用程序 \(p. 167\)](#) 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. 点击 Enable MQTT proxy (启用 MQTT 代理) 以启用 MQTT 代理。滑块应变为绿色。



启用 MQTT 代理之后，MQTT 消息显示在 `freertos/demos/echo` 主题上，并且数据输出到 UART 终端。

```
3 1993 [pthread] Received: HelloWorld 3
4 1994 [pthread] [INFO] [MQTT][1994] MQTT PUBLISH operation queued.
5 1995 [pthread] [INFO] [MQTT][1995] Waiting for operation PUBLISH to complete.
6 2002 [pthread] [INFO] [MQTT][2002] MQTT operation PUBLISH complete with result SUCCESS.
7 2002 [pthread] Sent: HelloWorld 3 ACK
8 2087 [pthread] [INFO] [MQTT][2087] MQTT PUBLISH operation queued.
9 2087 [pthread] [INFO] [MQTT][2087] Waiting for operation PUBLISH to complete.
10 2095 [pthread] [INFO] [MQTT][2095] MQTT operation PUBLISH complete with result SUCCESS.
1 2095 [pthread] Sent: HelloWorld 4
2 2101 [pthread] Received: HelloWorld 4
3 2102 [pthread] [INFO] [MQTT][2102] MQTT PUBLISH operation queued.
4 2102 [pthread] [INFO] [MQTT][2102] Waiting for operation PUBLISH to complete.
5 2110 [pthread] [INFO] [MQTT][2110] MQTT operation PUBLISH complete with result SUCCESS.
6 2110 [pthread] Sent: HelloWorld 4 ACK
```

Wi-Fi 预配置

Wi-Fi 预配置是一项 Amazon FreeRTOS BLE 服务，让您可以安全地将 Wi-Fi 网络凭证通过 BLE 从移动设备发送到微控制器。Wi-Fi 预配置服务的源代码位于 `lib/bluetooth_low_energy/services/wifi_provisioning`。

Note

Espressif ESP32-DevKitC 上当前支持 Wi-Fi 预配置演示。
Android 版本的演示移动应用程序当前不支持 Wi-Fi 预配置。

启用演示

1. 启用 Wi-Fi 预配置服务。打开 `demos/vendor/board/common/config_files/aws_ble_config.h` 并将 `#define bleconfigENABLE_WIFI_PROVISIONING` 设置为 1。

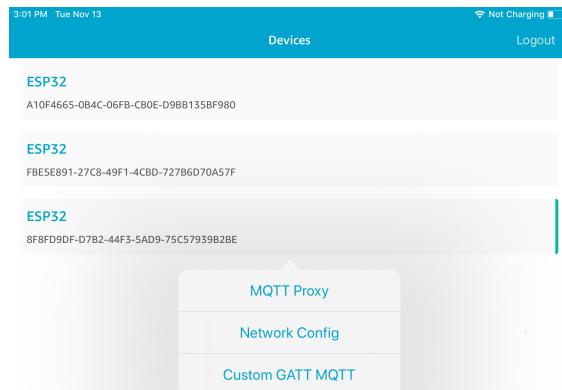
Note

默认情况下禁用 Wi-Fi 预配置服务。

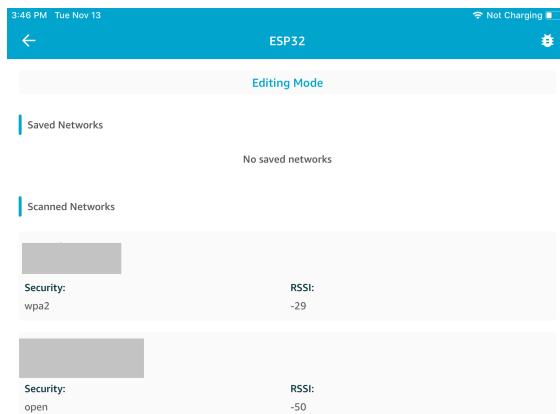
2. 配置 [网络管理器 \(p. 167\)](#) 以启用 BLE 和 Wi-Fi。

运行演示

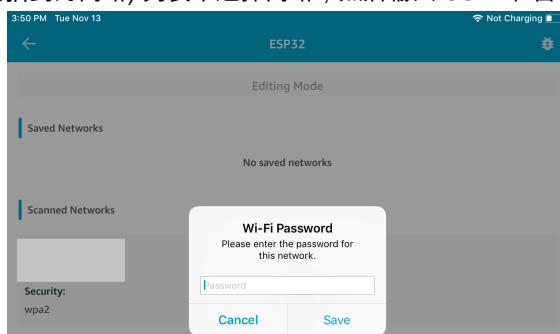
1. 在微控制器上生成并运行演示项目。
2. 确保您已使用 [Amazon FreeRTOS BLE 移动开发工具包演示应用程序 \(p. 167\)](#) 将微控制器与移动设备配对。
3. 从演示移动应用程序的 Devices (设备) 列表中，选择您的微控制器，然后选择 Network Config (网络配置) 以打开网络配置设置。



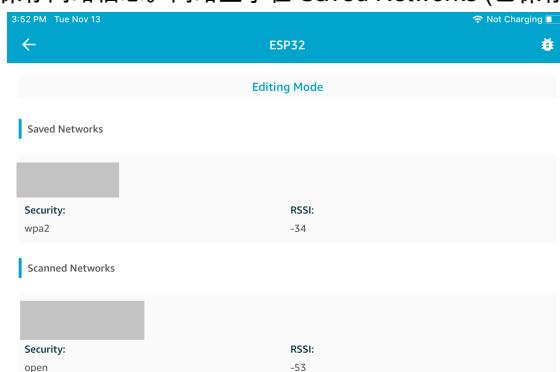
4. 在为主板选择 Network Config (网络配置) 之后，微控制器会将附近的网络列表发送到移动设备。可用 Wi-Fi 网络显示在 Scanned Networks (扫描到的网络) 下的列表中。



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）。

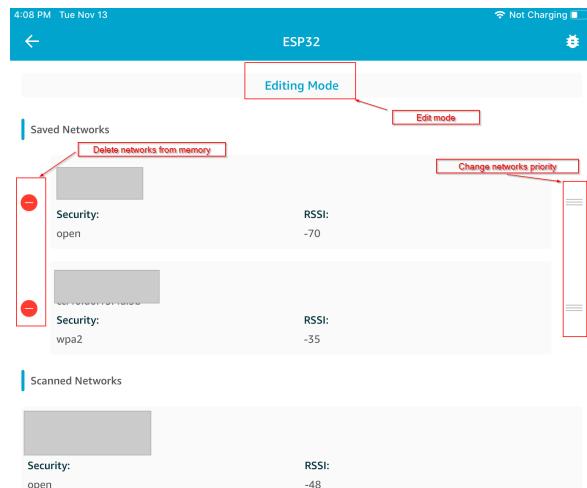


该微控制器连接到网络并保存网络信息。网络显示在 Saved Networks (已保存网络) 下。



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 [Amazon FreeRTOS BLE 移动开发工具包演示应用程序 \(p. 167\)](#)。

使用 BLE 移动开发工具包，您可以为连接到微控制器上 GATT 服务器的移动设备创建自己的 GATT 客户端，并与演示移动应用程序并行运行。

启用演示

1. 启用 BLE GATT 演示。在 `demos\vendor\board\common\config_files\aws_ble_config.h` 中，添加 `#define bleconfigENABLE_GATT_DEMO (1)` 到列表以定义语句。

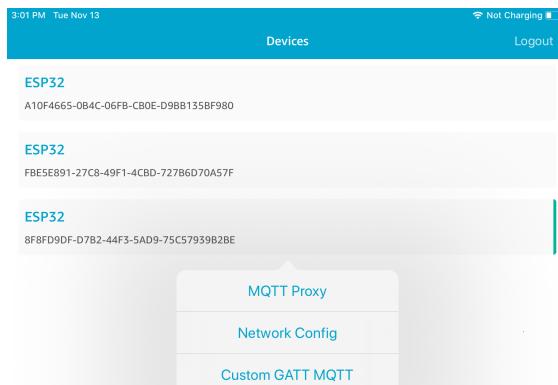
Note

默认情况下禁用 BLE GATT 演示。

2. 打开 `demos\common\demo_runner\aws_demo_runner.c`，在演示声明中取消注释 `extern void vStartMQTTBLEEchoDemo(void);`。在 `DEMO_RUNNER_RunDemos` 定义中，取消注释 `vStartMQTTBLEEchoDemo();`。

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 [Amazon FreeRTOS BLE 移动开发工具包演示应用程序 \(p. 167\)](#) 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. 点击 Enable MQTT proxy (启用 MQTT 代理) 以启用 MQTT 代理。滑块应变为绿色。



5. 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。
6. 选择 Start Counter (启动计数器) 以开始发布数据到 freertos/demos/echo MQTT 主题。

在启用 MQTT 代理之后，Hello World 和递增计数消息显示在 freertos/demos/echo 主题上。

安全套接字 Echo 客户端演示

以下示例使用单个 RTOS 堆栈。可在以下位置找到此示例的源代码：[demos/common/tcp/aws_tcp_echo_client_single_task.c](#)。

在开始之前，请确认您已将 Amazon FreeRTOS 下载到微控制器，并且已构建并运行 Amazon FreeRTOS 演示项目。您可以从 [GitHub](#) 下载 Amazon FreeRTOS。有关设置符合 Amazon FreeRTOS 要求的主板的更多信息，请参阅 [Amazon FreeRTOS 入门](#)。

运行演示

1. 按照 [Amazon FreeRTOS Qualification Program 开发人员指南](#) 的“TLS 服务器设置”部分中的说明操作来设置 TLS Echo 服务器。

在步骤 6 结束时，TLS Echo 服务器应已运行并在端口 9000 上侦听。您无需完成步骤 7、8 和 9。

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. 在 demos/common/demo_runneraws_demo_runner.c 中，将演示函数切换为 vStartTCPEchoClientTasks_SingleTasks()：

```
//extern void vStartMQTTEchoDemo( void );
extern void *vStartTCPEchoClientTasks_SingleTasks*( void );

/**
 * @brief Runs demos in the system.
 */
void DEMO_RUNNER_RunDemos( void )
{
    //vStartMQTTEchoDemo();
    vStartTCPEchoClientTasks_SingleTasks();
}
```

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。

设备影子演示应用程序

设备影子示例演示了如何以编程方式更新并响应设备影子中的变化。此情景中的设备是灯泡，其颜色可以设置为红色或绿色。设备影子示例应用程序位于 AmazonFreeRTOS/demos/common/shadow 目录中。此示例创建三个任务：

- 调用 prvShadowMainTask 的主演示任务。
- 调用 prvUpdateTask 的设备更新任务。
- 调用 prvShadowUpdateTasks 的多个影子更新任务。

prvShadowMainTask 初始化设备影子客户端并启动与 AWS IoT 的 MQTT 连接。然后，它将创建设备更新任务。最后，它创建影子更新任务，然后终止。在 AmazonFreeRTOS/demos/common/shadow/aws_shadow_lightbulb_on_off.c 中定义的 democonfigSHADOW_DEMO_NUM_TASKS 常量控制创建的影子更新任务数。

prvShadowUpdateTasks 生成初始事物影子文档并使用文档更新设备影子。然后，它会进入无限循环，定义更新事物影子的所需状态，请求灯泡改变其颜色（从红色到绿色到红色）。

prvUpdateTask 响应设备影子所需状态的更改。当所需的状态发生更改时，此任务将更新报告的设备影子状态，体现新的所需状态。

1. 将以下策略添加到您的设备证书：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:Connect",
            "Resource": "arn:aws:iot:us-west-2:123456789012:client/<yourClientId>"
        },
    ]
}
```

```
{  
    "Effect": "Allow",  
    "Action": "iot:Subscribe",  
    "Resource": "arn:aws:iot:us-west-2:123456789012:topicfilter/$aws/things/  
thingName/shadow/*"  
},  
{  
    "Effect": "Allow",  
    "Action": "iot:Receive",  
    "Resource":  
    "arn:aws:iot:us-west-2:123456789012:topic/$aws/things/thingName/shadow/*"  
},  
{  
    "Effect": "Allow",  
    "Action": "iot:Publish",  
    "Resource":  
    "arn:aws:iot:us-west-2:123456789012:topic/$aws/things/thingName/shadow/*"  
}  
]  
}
```

- 取消对 `aws_demo_runner.c` 中的 `vStartShadowDemoTasks` 声明的注释并调用。此函数创建运行 `prvShadowMainTask` 函数的任务。

您可以使用 AWS IoT 控制台来查看设备的影子并确认所需的状态，定期报告状态更改。

- 在 AWS IoT 控制台中，从左侧导航窗格中选择管理。
- 在管理下，选择事物，然后选择要查看其影子的事物。
- 在事物详细信息页面上，从左侧导航窗格中，选择影子以显示事物影子。

有关设备与影子如何交互的更多信息，请参阅[设备影子数据流](#)。

Greengrass Discovery 示例应用程序

在运行 FreeRTOS Greengrass Discovery 演示之前，您必须创建 Greengrass 组，然后添加 Greengrass 核心。有关更多信息，请参阅[AWS Greengrass 入门](#)。

在具有运行 Greengrass 软件的核心之后，为您的 Amazon FreeRTOS 设备创建 AWS IoT 事物、证书和策略。有关更多信息，请参阅[将您的 MCU 主板注册到 AWS IoT \(p. 5\)](#)。

为您的 Amazon FreeRTOS 设备创建 IoT 事物之后，按照说明设置环境并在支持的设备之一上生成 Amazon FreeRTOS：

Note

使用[将您的 MCU 主板注册到 AWS IoT \(p. 5\)](#)说明，但不要下载与 AWS IoT-XX 配置预定义的连接之一，而是下载与 AWS IoT Greengrass - XX 配置的连接之一。按照“配置项目”中的步骤操作。为您的设备生成了 Amazon FreeRTOS 之后，返回此主题。

此时，您已下载 Amazon FreeRTOS 软件，将其导入到 IDE 中，并生成了项目且没有错误。项目已配置为运行 Greengrass Connectivity 演示。在 AWS IoT 控制台中，选择测试，然后添加订阅到 `freertos/demos/ggd`。该演示发布一系列消息到 Greengrass 核心。消息还会发布到 AWS IoT，此时由 AWS IoT MQTT 客户端接收消息。

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
```

OTA 演示应用程序

Amazon FreeRTOS 中包括一个演示应用程序，演示了 OTA 库的使用。OTA 演示应用程序位于 `demos\common\ota` 子目录中。

在您创建 OTA 更新之前，请阅读 [Amazon FreeRTOS 无线更新 \(p. 118\)](#) 并完成其中列出的所有先决条件。

OTA 演示应用程序：

1. 初始化 FreeRTOS 网络堆栈和 MQTT 缓冲池。(请参阅 `main.c.`)
2. 创建任务来执行 OTA 库。(请参阅 `aws_ota_update_demo.c` 中的 `vOTAUpdateDemoTask.`)
3. 使用 `MQTT_AGENT_Create` 创建 MQTT 客户端。
4. 使用 `MQTT_AGENT_Connect` 连接到 AWS IoT MQTT 代理。
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。

您可以使用 AWS IoT 控制台或 AWS CLI 来创建 OTA 更新任务。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。

```
313 267848 [OTA] [OTA] Queued: 1 Processed: 1 Dropped: 0
314 268733 [OTA Task] [OTA] Set job doc parameter [ jobId:
fe18c7ec_8c31_4438_b0b9_ad55acd95610 ]
315 268734 [OTA Task] [OTA] Set job doc parameter [ streamname: 327 ]
316 268734 [OTA Task] [OTA] Set job doc parameter [ filepath: /sys/mcuflashimg.bin ]
317 268734 [OTA Task] [OTA] Set job doc parameter [ filesize: 130388 ]
318 268735 [OTA Task] [OTA] Set job doc parameter [ fileid: 126 ]
319 268735 [OTA Task] [OTA] Set job doc parameter [ attr: 0 ]
320 268735 [OTA Task] [OTA] Set job doc parameter [ certfile: tisigner.crt.der ]
321 268737 [OTA Task] [OTA] Set job doc parameter [ sig-sha1-rsa:
Q56qxHRq3Lxv6KkorvilVs4AyGJbWsJd ]
322 268737 [OTA Task] [OTA] Job was accepted. Attempting to start transfer.
323 268737 [OTA Task] Sending command to MQTT task.
324 268737 [MQTT] Received message 50000 from queue.
325 268848 [OTA] [OTA] Queued: 2 Processed: 1 Dropped: 0
326 269039 [MQTT] MQTT Subscribe was accepted. Subscribed.
327 269039 [MQTT] Notifying task.
328 269040 [OTA Task] Command sent to MQTT task passed.
329 269041 [OTA Task] [OTA] Subscribed to topic: $aws/things/TI-LaunchPad/streams/327

330 269848 [OTA] [OTA] Queued: 2 Processed: 1 Dropped: 0
... // Output removed for brevity
346 284909 [OTA Task] [OTA] file token: 74594452
.. // Output removed for brevity
363 301327 [OTA Task] [OTA] file ready for access.
364 301327 [OTA Task] [OTA] Returned buffer to MQTT Client.
365 301328 [OTA Task] Sending command to MQTT task.
366 301328 [MQTT] Received message 60000 from queue.
367 301328 [MQTT] Notifying task.
368 301329 [OTA Task] Command sent to MQTT task passed.
369 301329 [OTA Task] [OTA] Published file request to $aws/bin/things/TI-LaunchPad/
streams/327/get
370 301632 [OTA Task] [OTA] Received file block 0, size 1024
371 301647 [OTA Task] [OTA] Remaining: 127
```

```
... // Output removed for brevity
508 304622 [OTA Task] Sending command to MQTT task.
509 304622 [MQTT] Received message 70000 from queue.
510 304622 [MQTT] Notifying task.
511 304623 [OTA Task] Command sent to MQTT task passed.
512 304623 [OTA Task] [OTA] Published file request to $aws/bin/things/TI-LaunchPad/
streams/327/get
513 304860 [OTA] [OTA] Queued: 47 Processed: 47 Dropped: 83
514 304926 [OTA Task] [OTA] Received file block 4, size 1024
515 304941 [OTA Task] [OTA] Remaining: 82
... // Output removed for brevity
797 315047 [MQTT] MQTT Publish was successful.
798 315048 [MQTT] Notifying task.
799 315048 [OTA Task] Command sent to MQTT task passed.
800 315049 [OTA Task] [OTA] Published 'IN_PROGRESS' status to $aws/things/TI-LaunchPad/
jobs/fe18c7ec_8c31_4438_b0b9_ad55acd9561801 315049 [OTA Task] Sending command to MQTT task.
802 315049 [MQTT] Received message d0000 from queue.
803 315150 [MQTT] MQTT Unsubscribe was successful.
804 315150 [MQTT] Notifying task.
805 315151 [OTA Task] Command sent to MQTT task passed.
806 315152 [OTA Task] [OTA] Un-subscribed from topic: $aws/things/TI-LaunchPad(streams/327

807 315172 [OTA Task] Sending command to MQTT task.
808 315172 [MQTT] Received message e0000 from queue.
809 315273 [MQTT] MQTT Unsubscribe was successful.
810 315273 [MQTT] Notifying task.
811 315274 [OTA Task] Command sent to MQTT task passed.
812 315274 [OTA Task] [OTA] Un-subscribed from topic: $aws/things/TI-LaunchPad(streams/327

813 315275 [OTA Task] [OTA] Resetting MCU to activate new image.
0 0 [Tmr Svc] Starting Wi-Fi Module ...
1 0 [Tmr Svc] Simple Link task created

Device came up in Station mode

2 137 [Tmr Svc] Wi-Fi module initialized.
3 137 [Tmr Svc] Starting key provisioning...
4 137 [Tmr Svc] Write root certificate...
5 243 [Tmr Svc] Write device private key...
6 339 [Tmr Svc] Write device certificate...
7 436 [Tmr Svc] Key provisioning done...
Device disconnected from the AP on an ERROR!!!

[WLAN EVENT] STA Connected to the AP: Guest , BSSID: 44:48:c1:ba:b2:c3

[NETAPP EVENT] IP acquired by the device

Device has connected to Guest

Device IP Address is 192.168.3.72

8 1443 [Tmr Svc] Wi-Fi connected to AP Guest.
9 1444 [Tmr Svc] IP Address acquired 192.168.3.72
10 1444 [OTA] OTA demo version 0.9.1
11 1445 [OTA] Creating MQTT Client...
12 1445 [OTA] Connecting to broker...
13 1445 [OTA] Sending command to MQTT task.
14 1445 [MQTT] Received message 10000 from queue.
15 2910 [MQTT] MQTT Connect was accepted. Connection established.
16 2910 [MQTT] Notifying task.
17 2911 [OTA] Command sent to MQTT task passed.
18 2912 [OTA] Connected to broker.
19 2913 [OTA Task] Sending command to MQTT task.
20 2913 [MQTT] Received message 20000 from queue.
```

```
21 3014 [MQTT] MQTT Subscribe was accepted. Subscribed.
22 3014 [MQTT] Notifying task.
23 3015 [OTA Task] Command sent to MQTT task passed.
24 3015 [OTA Task] [OTA] Subscribed to topic: $aws/things/TI-LaunchPad/jobs/$next/get/
accepted

25 3028 [OTA Task] Sending command to MQTT task.
26 3028 [MQTT] Received message 30000 from queue.
27 3129 [MQTT] MQTT Subscribe was accepted. Subscribed.
28 3129 [MQTT] Notifying task.
29 3130 [OTA Task] Command sent to MQTT task passed.
30 3138 [OTA Task] [OTA] Subscribed to topic: $aws/things/TI-LaunchPad/jobs/notify-next

31 3138 [OTA Task] [OTA] Check For Update #0
32 3138 [OTA Task] Sending command to MQTT task.
33 3138 [MQTT] Received message 40000 from queue.
34 3241 [MQTT] MQTT Publish was successful.
35 3241 [MQTT] Notifying task.
36 3243 [OTA Task] Command sent to MQTT task passed.
37 3245 [OTA Task] [OTA] Set job doc parameter [ clientToken: 0:TI-LaunchPad ]
38 3245 [OTA Task] [OTA] Set job doc parameter [ jobId:
fe18c7ec_8c31_4438_b0b9_ad55acd95610 ]
39 3245 [OTA Task] [OTA] Identified job doc parameter [ self_test ]
40 3246 [OTA Task] [OTA] Set job doc parameter [ updatedBy: 589827 ]
41 3246 [OTA Task] [OTA] Set job doc parameter [ streamname: 327 ]
42 3246 [OTA Task] [OTA] Set job doc parameter [ filepath: /sys/mcuflashimg.bin ]
43 3247 [OTA Task] [OTA] Set job doc parameter [ filesize: 130388 ]
44 3247 [OTA Task] [OTA] Set job doc parameter [ fileid: 126 ]
45 3247 [OTA Task] [OTA] Set job doc parameter [ attr: 0 ]
46 3247 [OTA Task] [OTA] Set job doc parameter [ certfile: tisigner.crt.der ]
47 3249 [OTA Task] [OTA] Set job doc parameter [ sig-sha1-rsa:
Q56qxHRq3Lxv6KkorvilVs4AyGJbWsJd ]
48 3249 [OTA Task] [OTA] Job is ready for self test.
49 3250 [OTA Task] Sending command to MQTT task.
51 3351 [MQTT] MQTT Publish was successful.
52 3352 [MQTT] Notifying task.
53 3352 [OTA Task] Command sent to MQTT task passed.
54 3353 [OTA Task] [OTA] Published 'IN_PROGRESS' status to $aws/things/TI-LaunchPad/jobs/
fe18c7ec_8c31_4438_b0b9_ad55acd95610/u55 3353 [OTA Task] Sending command to MQTT task.
56 3353 [MQTT] Received message 60000 from queue.
57 3455 [MQTT] MQTT Unsubscribe was successful.
58 3455 [MQTT] Notifying task.
59 3456 [OTA Task] Command sent to MQTT task passed.
60 3456 [OTA Task] [OTA] Un-subscribed from topic: $aws/things/TI-LaunchPadstreams/327

61 3456 [OTA Task] [OTA] Accepted final image. Commit.
62 3578 [OTA Task] Sending command to MQTT task.
63 3578 [MQTT] Received message 70000 from queue.
64 3779 [MQTT] MQTT Publish was successful.
65 3780 [MQTT] Notifying task.
66 3780 [OTA Task] Command sent to MQTT task passed.
67 3781 [OTA Task] [OTA] Published 'SUCCEEDED' status to $aws/things/TI-LaunchPad/jobs/
fe18c7ec_8c31_4438_b0b9_ad55acd95610/upd68 3781 [OTA Task] [OTA] Returned buffer to MQTT
Client.
69 4251 [OTA] [OTA] Queued: 1 Processed: 1 Dropped: 0
70 4381 [OTA Task] [OTA] Missing job parameter: execution
71 4382 [OTA Task] [OTA] Missing job parameter: jobId
72 4382 [OTA Task] [OTA] Missing job parameter: jobDocument
73 4382 [OTA Task] [OTA] Missing job parameter: ts_ota
74 4382 [OTA Task] [OTA] Missing job parameter: files
75 4382 [OTA Task] [OTA] Missing job parameter: streamname
76 4382 [OTA Task] [OTA] Missing job parameter: certfile
77 4382 [OTA Task] [OTA] Missing job parameter: filepath
78 4383 [OTA Task] [OTA] Missing job parameter: filesize
79 4383 [OTA Task] [OTA] Missing job parameter: sig-sha1-rsa
80 4383 [OTA Task] [OTA] Missing job parameter: fileid
```

```
81 4383 [OTA Task] [OTA] Missing job parameter: attr
82 4383 [OTA Task] [OTA] Returned buffer to MQTT Client.
83 5251 [OTA] [OTA] Queued: 2 Processed: 2 Dropped: 0
```

Microchip Curiosity PIC32MZEF 的演示启动加载程序

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 Amazon FreeRTOS 的无线 (OTA) 固件更新。

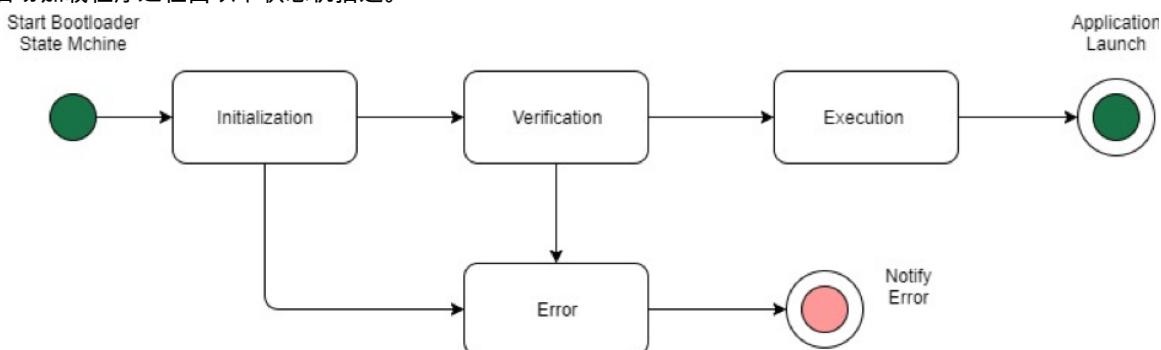
固件验证包括验证无线接收的新固件的真实性和完整性。在启动之前，启动加载程序验证应用程序的加密签名。该演示在 SHA256 之上使用了椭圆曲线数字签名算法 (ECDSA)。提供的实用程序可用于生成签名的应用程序，该应用程序可刷入设备。

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。

启动加载程序状态

启动加载程序进程由以下状态机描述。



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 均显示为 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

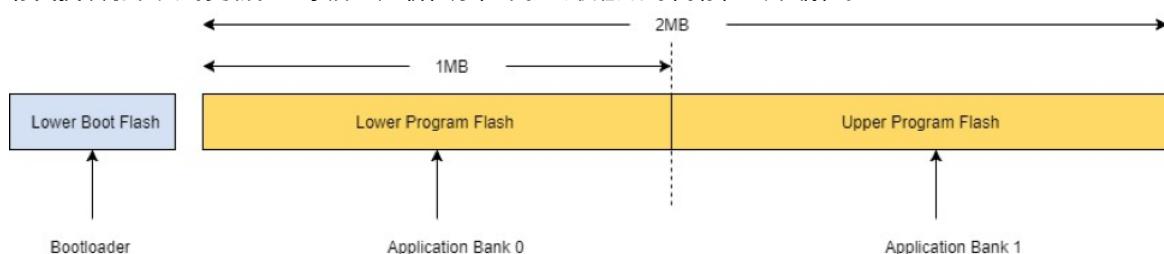
如果启动默认映像的配置选项已启用，则启动加载程序将从默认执行地址启动应用程序。除了在调试期间，否则应禁用此选项。

启动加载程序出错状态

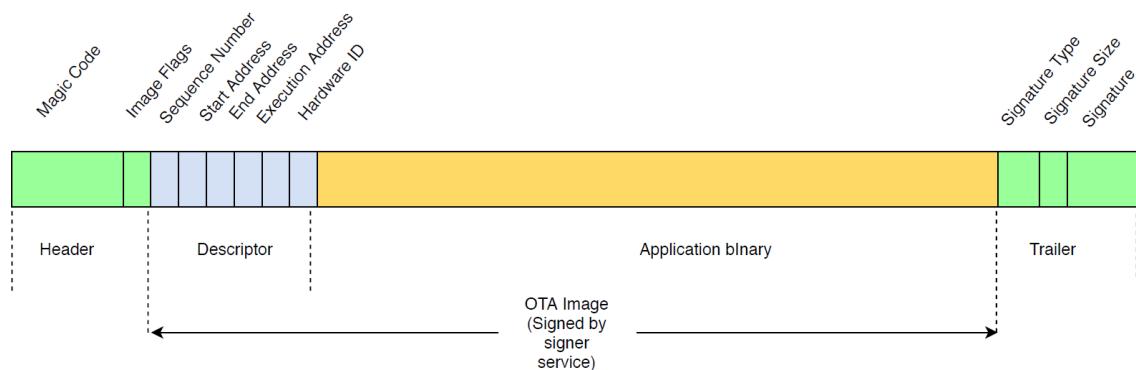
启动加载程序处于出错状态，设备上没有有效的映像。启动加载程序必须通知用户。默认实施发送日志消息到控制台，并无限期快速闪烁主板上的 LED。

闪存设备

Microchip Curiosity PIC32MZEF 平台包含 2 MB 的内部程序缓存，划分为两个库。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



图中显示了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)
Trailer	292 字节

映像标头

设备上存在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。启动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，此时启动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。
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程序生成，该实用程序使用配置文件 (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

描述符字段	大小 (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 `aws_boot_config.h` 中提供。一些选项仅提供用于调试目的。`aws_boot_config.h` 位于 `/demos/microchip/curiosity_pic32_bl/config_files/` 中。

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。

生成启动加载程序

在 Amazon FreeRTOS 源代码存储库的 `demos\microchip\curiosity_pic32mzef\mplab` 下，演示启动加载程序包括在 `aws_demos` 项目中作为可加载的对象。生成 `aws_demos` 项目时，它先生成启动加载程序，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 `/demos/common/ota/bootloader/utility/` 中。

启动加载程序生成前步骤

此生成前步骤执行名为 `codesigner_cert_utility.py` 的实用程序脚本，该脚本从代码签名证书中提取公有密钥，并生成包含采用 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。

Amazon FreeRTOS 移植指南

本移植指南将引导您修改 Amazon FreeRTOS 软件包，以在不符合 Amazon FreeRTOS 要求的主板上使用。Amazon FreeRTOS 旨在让您仅选择主板或应用程序所需的那些库。MQTT、Shadow 和 Greengrass 库设计与大多数设备原样兼容，因此这些库没有移植指南。

有关移植 FreeRTOS 内核的信息，请参阅 [FreeRTOS Kernel Porting Guide](#)。

主题

- [启动加载程序 \(p. 186\)](#)
- [日志记录 \(p. 186\)](#)
- [连接 \(p. 187\)](#)
- [安全性 \(p. 188\)](#)
- [将自定义库与 Amazon FreeRTOS 配合使用 \(p. 189\)](#)
- [OTA 可移植抽象层 \(p. 189\)](#)

启动加载程序

启动加载程序必须支持双库，并包括用于在映像标头中检查 CRC 和应用程序版本的逻辑。在 CRC 有效的情况下，启动加载程序根据标头中的应用程序版本，引导最新的映像。如果 CRC 检查失败，则启动加载程序应清零标头，作为对以后重启过程的优化。

由于 OTA v1 代理执行加密签名验证，我们建议 v1 启动加载程序不链接到加密代码，以使其尽可能小。您必须提供符合规范的启动加载程序。

日志记录

Amazon FreeRTOS 提供了线程安全的日志记录任务，可通过调用 `configPRINTF` 函数来使用。`configPRINTF` 设计为行为类似于 `printf`。要移植 `configPRINTF`，请初始化您的通信外围设备，然后定义 `configPRINT_STRING` 宏，这样它会获取输入字符串并在您首选的输出上显示。

日志记录配置

应该为您主板的日志记录实施定义 `configPRINT_STRING`。当前示例使用 UART 串行终端，但也可使用其他接口。

```
#define configPRINT_STRING( x )
```

使用 `configLOGGING_MAX_MESSAGE_LENGTH` 设置要输出的最大字节数。长度超过此值的消息将截断。

```
#define configLOGGING_MAX_MESSAGE_LENGTH
```

当 `configLOGGING_INCLUDE_TIME_AND_TASK_NAME` 设置为 1 时，所有输出的消息将在前面附加额外的调试信息（消息编号、FreeRTOS 滴答计数和任务名称）。

```
#define configLOGGING_INCLUDE_TIME_AND_TASK_NAME      1
```

vLoggingPrintf 是 FreeRTOS 线程安全的 printf 调用的名称。使用 AmazonFreeRTOS 日志记录无需更改此值。

```
#define configPRINTF( x ) vLoggingPrintf x
```

连接

您必须先配置连接外围设备。您可以使用 Wi-Fi、蓝牙、以太网或其他连接介质。目前，只为主板端口定义了 Wi-Fi 管理 API，但是，如果您使用以太网，FreeRTOS TCP/IP software 提供了很好的起点。

Wi-Fi 管理

Wi-Fi 管理库支持遵循 802.11 (a/b/n) 协议的网络连接。如果您的硬件不支持 Wi-Fi，则无需移植此库。

lib/wifi/portable/<vendor>/<platform>/aws_wifi.c 文件中列出了必须移植的函数。您可在 lib/include/aws_wifi.h 中查找各个公共接口的详细说明。

以下函数必须移植：

```
WiFiReturnCode_t WIFI_On( void );
WiFiReturnCode_t WIFI_Off( void );
WiFiReturnCode_t WIFI_ConnectAP( const WiFiNetworkParams_t * const pxNetworkParams );
WiFiReturnCode_t WIFI_Disconnect( void );
WiFiReturnCode_t WIFI_Reset( void );
WiFiReturnCode_t WIFI_Scan( WiFiScanResult_t * pxBuffer, uint8_t uxNumNetworks );
```

套接字

支持您的主板与网络中其他节点之间的 TCP/IP 网络通信的套接字库。套接字 API 基于 Berkeley 套接字接口，不过还包括安全通信选项。目前只支持客户端 API。我们建议您在添加 TLS 功能之前，先移植 TCP/IP 功能。

适用于 MQTT、Shadow 和 Greengrass 的库均调用这些套接字层。成功移植套接字层会使得构建在套接字上的协议正常工作。

Berkeley 套接字实施的主要差异

安全性

套接字接口必须配置为使用 TLS 进行安全通信。SetSockOpt 命令有一组非标准选项，必须实施这些选项以用于 AmazonFreeRTOS 示例。

```
SOCKETS_SO_REQUIRE_TLS
SOCKETS_SO_SERVER_NAME_INDICATION
SOCKETS_SO_TRUSTED_SERVER_CERTIFICATE
```

有关这些非标准选项的信息，请参阅[安全套接字文档 \(p. 108\)](#)。有关移植 TLS 和加密操作的信息，请参阅[TLS \(p. 114\)](#) 和[公有密钥加密标准 #11 \(p. 106\)](#) 部分。

错误代码

SOCKETS 库返回 API 的错误代码（而不是设置全局错误编号）。返回的所有错误代码必须为负值。

lib/secure_sockets/portable/<vendor>/<platform>/aws_secure_sockets.c 中列出了必须移植的公共接口。

在 lib/include/aws_secure_sockets.h 中可找到各个公共接口的详细说明。

如果您使用基于 mbed TLS 的 TLS，则可以实施网络发送和网络接收函数，这些函数可以注册到 TLS 层来发送和接收明文或加密缓冲，从而减少重构工作。

安全性

Amazon FreeRTOS 有两个库可以协同工作，提供平台安全：TLS 和 PKCS#11。Amazon FreeRTOS 提供了在 mbed TLS（一种第三方 TLS 库）上构建的软件安全解决方案。TLS API 使用 mbed TLS 加密和验证网络流量。PKCS#11 提供了一个标准接口来处理加密材料，并通过完全使用硬件的实施来取代软件加密操作。

TLS

如果您选择使用基于 mbed TLS 的实施，则在实施了 PKCS#11 的情况下，可以按原样使用 aws_tls.c。

lib/include/aws_tls.h 中列出了此库的公共接口以及各 TLS 接口的详细说明。TLS 库的 Amazon FreeRTOS 实施在 lib/tls/aws_tls.c 中。如果您决定使用自己的 TLS 支持，则可以实施 TLS 公共接口并将它们插入套接字公共接口中，或者使用您自己的 TLS 接口直接移植套接字库。

mbedtls_hardware_poll 函数为确定性随机位生成器提供随机性。出于安全考虑，不应向两个主板提供相同的随机性，并且主板不能重复提供相同的随机值，即使已重置主板。位于 demos/<vendor>/<platform>/common/application_code/<vendor code>/aws_entropy_hardware_poll.c 的使用 ebed TLS 移植中可找到此函数的实现示例。

使用 mbed TLS 之外的 TLS 库

如果您在将其他 TLS 库移植到 Amazon FreeRTOS，请确保在移植中实施了兼容的 TLS 密码套件。有关更多信息，请参阅[与 AWS IoT 兼容的密码套件](#)。以下密码套件与 AWS IoT Greengrass 设备兼容：

- TLS_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA（不推荐）
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA（不推荐）
- TLS_RSA_WITH_AES_128_CBC_SHA（不推荐）
- TLS_RSA_WITH_AES_256_CBC_SHA（不推荐）

由于对 SHA1 的攻击，建议您为 Amazon FreeRTOS 连接使用 SHA256 或 SHA384。

PKCS #11

Amazon FreeRTOS 为加密操作和密钥存储实施 PKCS#11 标准。PKCS#11 的标头文件是行业标准。要移植 PKCS#11，您必须实施函数来对非易失性存储（NVM）读取和写入凭证。

您需要实施的函数在 lib/third_party/pkcs11/pkcs11f.h 中列出。公共接口的实现位于：lib/pkcs11/portable/vendor/board/pkcs11.c。

在 Amazon FreeRTOS 中支持 TLS 客户端身份验证至少需要下列函数：

- C_GetFunctionList
- C_Initialize
- C_GetSlotList
- C_OpenSession
- C_FindObjectsInit
- C_FindObjects
- C_FindObjectsFinal
- C_GetAttributeValue
- C_FindObjectsInit
- C_FindObjects
- C_FindObjectsFinal
- C_GetAttributeValue
- C_SignInit
- C_Sign
- C_CloseSession
- C_Finalize

有关一般移植指南，请参阅开放标准 [PKCS #11 加密令牌接口基本规范](#)。

要让密钥和证书在重启循环中保留，还必须实施两个额外的非 PKCS #11 标准函数：

`prvSaveFile`

将客户端（设备）的私有密钥和证书写入内存。如果您的 NVM 容易在写入周期中损坏，您可能希望使用其他变量来记录是否已经初始化设备私有密钥和设备证书。

`prvReadFile`

将设备私有密钥或者设备证书从 NVM 检索并存储到 RAM 中，供 TLS 库使用。

将自定义库与 Amazon FreeRTOS 配合使用

所有 Amazon FreeRTOS 库可以使用自定义开发的库替换。所有自定义库必须符合所要替换的 Amazon FreeRTOS 库的 API。

OTA 可移植抽象层

Amazon FreeRTOS 定义了 OTA 可移植抽象层 (PAL)，以确保 OTA 库可用于各种各样的硬件。以下列出了 OTA PAL 接口。

`prvAbort`

中止 OTA 更新。

`prvCreateFileForRx`

创建新文件来存储收到数据块。

`prvCloseFile`

关闭指定的文件。如果文件标记为安全，这可能会对文件进行身份验证。

prvCheckFileSignature

验证指定文件的签名。对于内置签名验证强制实施的设备文件系统，这可能是多余的，因此作为无操作来实施。

prvWriteBlock

按照指定偏移量将数据块写入指定文件。在成功时返回写入的字节数，否则返回负数错误代码。

prvActivateNewImage

激活新的固件映像。对于某些移植，此函数可能不会返回。

prvSetImageState

执行平台所需的任何操作来接受或拒绝最新的固件映像（或包）。参阅平台实施以确定平台上发生了什么情况。

prvReadAndAssumeCertificate

从文件系统读取指定的签署人证书并将其返回给调用方。在部分平台上这是可选的。

Amazon FreeRTOS 资格审查计划

Amazon FreeRTOS 资格审查计划目前是 AWS 设备资格审查计划的一部分。有关 AWS 设备资格审查计划的更多信息，请访问 [AWS 合作伙伴网络网站](#)。

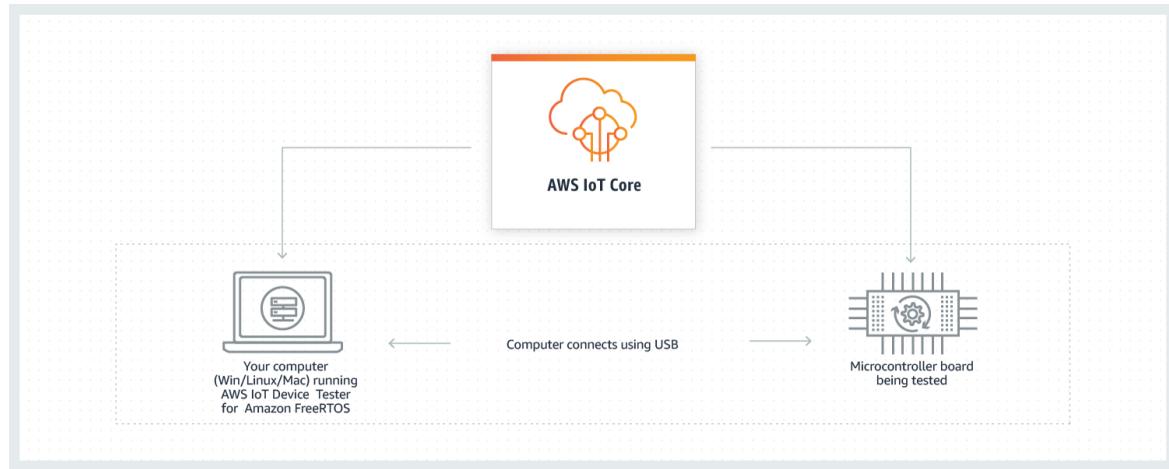
适用于 Amazon FreeRTOS 的 AWS IoT Device Tester 用户指南

借助 AWS IoT Device Tester，您可以测试 Amazon FreeRTOS 操作系统是否能在您的本地设备上正常运行，及是否能与 AWS IoT 云进行通信。AWS IoT Device Tester 检查 Amazon FreeRTOS 库的移植层界面在微控制器主板设备驱动程序上是否正常工作。此外，它会执行 AWS IoT 核心的端到端测试（例如，测试主板是否能够正确发送或接收 MQTT 消息并进行处理）。适用于 Amazon FreeRTOS 的 AWS IoT Device Tester 使用在 [Amazon FreeRTOS GitHub 存储库](#) 中发布的测试用例。AWS IoT Device Tester 包含 Test Manager 命令行工具和一组测试用例。

Test Manager 运行在与待测试设备连接的主机（Windows、Mac 或 Linux）上。Test Manager 执行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。测试用例包含测试逻辑并设置测试所需的资源。

测试用例是刷入主板中的应用程序二进制映像的一部分。应用程序二进制映像包括 Amazon FreeRTOS、半导体供应商移植的 Amazon FreeRTOS 界面以及主板设备驱动程序。测试用例作为嵌入式应用程序运行，可验证移植的 Amazon FreeRTOS 界面在设备驱动程序上是否正常工作。

下图显示了测试基础设施设置：



先决条件

此部分介绍通过 AWS IoT Device Tester 测试微控制器的先决条件。

下载 Amazon FreeRTOS

您可以从 [GitHub](#) 下载想要测试的 Amazon FreeRTOS 的版本。如果您使用 Windows，则必须保证文件路径较短。例如，为避免 Windows 对长文件路径的限制，请克隆到 C:\AFreeRTOS 而非 C:\Users\username\programs\projects\AmazonFreeRTOS\。

下载适用于 Amazon FreeRTOS 的 AWS IoT Device Tester。

Amazon FreeRTOS 的每个版本都有对应的 AWS IoT Device Tester 版本用于执行资格测试。下载 [AWS IoT Device Tester](#) 的相应版本。

将 AWS IoT Device Tester 提取到文件系统中您具有读取和写入权限的位置。由于路径长度限制，在 Microsoft Windows 上，将 AWS IoT Device Tester 提取到根目录，例如 C:\ 或 D:\。

创建和配置 AWS 账户

如果您没有 AWS 账户，请按照 [AWS 网页](#) 中的说明创建一个。选择创建 AWS 账户，然后按照提示操作。

在 AWS 账户中创建 IAM 用户

创建 AWS 账户时，将为您创建一个可以访问账户中所有资源的根用户。最佳实践是创建另一个用户用于日常任务。要创建 IAM 用户，请按照[在您的 AWS 账户中创建 IAM 用户](#)中的说明操作。有关根用户的更多信息，请参阅 [AWS 账户根用户](#)。

创建 IAM 策略并附加到 AWS 账户

IAM 策略授予 IAM 用户对 AWS 资源的访问权限。

创建 IAM 策略

1. 浏览至 [IAM 控制台](#)。
2. 在导航窗格中选择策略，然后选择创建策略。
3. 选择 JSON 选项卡，然后将位于[权限策略模板 \(p. 208\)???](#) ([p. 208](#))中的策略模板复制并粘贴到编辑器窗口。
4. 选择查看策略。
5. 在名称中，输入策略的名称。在描述中，输入可选描述。选择 Create Policy。

创建 IAM 策略后，必须将其附加到 IAM 用户。

将 IAM 策略附加到 IAM 用户

1. 浏览至 [IAM 控制台](#)。
2. 在导航窗格中，选择 Users。查找并选择 IAM 用户。
3. 选择添加权限，然后选择直接附加现有策略。查找并选择您的 IAM 策略，选择下一步：审核，然后选择添加权限。

安装 AWS 命令行界面 (CLI)

您需要使用 CLI 来执行一些操作，如果尚未安装 CLI，请按照[安装 AWS CLI](#) 中的说明操作。

首次进行测试，以确认您的微控制器主板是否符合要求

移植 Amazon FreeRTOS 界面时，您可以使用 AWS IoT Device Tester 进行测试。在移植了针对主板设备驱动程序的 Amazon FreeRTOS 界面之后，您可以使用 AWS IoT Device Tester 在微控制器主板上运行资格测试。

添加库移植层

要为 Amazon FreeRTOS 设备库 (TCP/IP、WiFi 等) 添加与 MCU 架构兼容的库移植层，您必须：

1. 在运行 AWS IoT Device Tester 测试之前实施 configPRINT_STRING() 方法。AWS IoT Device Tester 调用 configPRINT_STRING() 宏，以人类可读的 ASCII 字符串输出测试结果。
2. 移植驱动程序以实施 Amazon FreeRTOS 库的界面。有关更多信息，请参阅 [Amazon FreeRTOS 资格开发人员指南](#)。

配置您的 AWS 凭证

您必须在 `<devicetester_extract_location>/devicetester_afreertos_[win/mac/linux]/configs/config.json` 中配置 AWS 凭证。您可以采用以下两种方法之一来指定凭证：

- 环境变量
- 凭证文件

通过环境变量配置 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。AWS IoT Device Tester 可使用 `AWS_ACCESS_KEY_ID` 和 `AWS_SECRET_ACCESS_KEY` 环境变量来存储您的 AWS 凭证。设置环境变量的方式取决于运行的操作系统。

在 Windows 上设置环境变量

1. 从 Windows 10 桌面，打开高级用户任务菜单。要打开菜单，请将鼠标光标移动到屏幕的左下角（开始菜单图标）并右键单击。
2. 从高级用户任务菜单选择系统，然后选择高级系统设置。

Note

在 Windows 10 中，您可能需要向下滚动到相关的设置，然后选择系统信息。在系统中，选择高级系统设置。

在系统属性中，选择高级选项卡，然后选择环境变量按钮。

3. 在用户变量`<user-name>`下方，选择新建以创建环境变量。为每个环境变量输入名称和值。

在 macOS、Linux 或 UNIX 上设置环境变量

- 在任一文本编辑器中打开 `~/.bash_profile`，并添加以下行：

```
export AWS_ACCESS_KEY_ID="<your-aws-access-key-id>"  
export AWS_SECRET_ACCESS_KEY="<your-aws-secret-key>"
```

将尖括号中的项目 (< & >) 替换为 AWS 访问密钥 ID 和 AWS 私有密钥。

在设置环境变量后，关闭命令行窗口或终端，然后重新将其打开，以便新值生效。

要使用环境变量配置您的 AWS 凭证，请设置 `AWS_ACCESS_KEY_ID` 和 `AWS_SECRET_ACCESS_KEY`。在 `config.json` 中，对于 `method`，指定 `environment`：

```
{  
    "awsRegion": "us-west-2",  
    "auth": {  
        "method": "environment"  
    }  
}
```

使用凭证文件配置 AWS 凭证

创建一个包含凭证的凭证文件。AWS IoT Device Tester 使用的凭证文件与 AWS CLI 相同。有关更多信息，请参阅[。您还必须指定命名配置文件](#)。有关更多信息，请参阅[配置和凭证文件](#)。下方是一个示例 JSON 代码段，展示了如何在 config.json 文件中使用凭证文件指定 AWS 凭证：

```
{  
    "awsRegion": "us-west-2",  
    "auth": {  
        "method": "file",  
        "credentials": {  
            "profile": "default"  
        }  
    }  
}
```

配置 AWS 区域

AWS IoT Device Tester 在您的 AWS 账户中创建云资源。您可以通过在 config.json 文件中设置 awsRegion 参数来指定用于测试的 AWS 区域。默认情况下，AWS IoT Device Tester 使用 us-west-2 region。

在 AWS IoT Device Tester 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具有相同规格的设备。您可以配置 Device Tester 来测试某个池中的单个设备或多个设备。为了加快资格测试过程，AWS IoT Device Tester 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。

配置 AWS IoT Device Tester 用于单个设备测试

您可以通过在配置文件夹中编辑 device.json 文件模板来定义设备池。以下示例 device.json 文件用于创建具有一个设备的设备池：

```
[  
    {  
        "id": "<pool-id>",  
        "sku": "<sku>",  
        "features": [  
            {  
                "name": "WIFI",  
                "value": "Yes | No"  
            },  
            {  
                "name": "OTA",  
                "value": "Yes | No"  
            },  
            {  
                "name": "TCP/IP",  
                "value": "On-chip | Offloaded | No"  
            },  
            {  
                "name": "TLS",  
                "value": "On-chip | Offloaded | No"  
            }  
        ],  
        "devices": [  
            {  
                "id": "<device-id>",  
                "connectivity": {  
                    "protocol": "uart",  
                    "serialPort": "<serial-port>"  
                }  
            }  
        ]  
    }  
]
```

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

以下列表描述了在 `device.json` 文件中使用的属性：

`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

如果您希望在 AWS 合作伙伴设备目录中列出您的主板，您在此处指定的 SKU 必须与在列出过程中使用的 SKU 相匹配。

`features`

包含设备支持的功能的数组。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`：指示您的主板是否支持 TCP/IP 堆栈，以及是板载 (MCU) 支持还是分载到另一个模块。
- `WIFI`：指示您的主板是否具有 Wi-Fi 功能。
- `TLS`：指示您的主板是否支持 TLS，如果支持，是板载 (MCU) 支持还是分载到另一个模块。
- `OTA`：指示您的主板是否支持无线 (OTA) 更新功能。

`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。

`identifiers`

选填项。任意名称/值对的数组。您可以在下一部分所述的生成和刷入命令中使用这些值。

配置 AWS IoT Device Tester 用于多个设备测试

您可以通过编辑 `configs` 文件夹中 `device.json` 模板的 `devices` 部分来添加多个设备。有关 `device.json` 文件的结构和内容的更多信息，请参阅[配置 AWS IoT Device Tester 用于单个设备测试 \(p. 195\)](#)。

Note

同一个池中的所有设备必须具有相同的技术规格和 SKU。

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": "OTA",
      "value": "Yes | No"
    },
    {
      "name": "TCP/IP",
      "value": "On-chip | Offloaded | No"
    },
    {
      "name": "TLS",
      "value": "On-chip | Offloaded | No"
    }
  ],
  "devices": [
    {
      "id": "<device-id1>",
      "connectivity": {
        "protocol": "uart",
        "serialPort": "<computer_serial_port_1>"
      },
      "identifiers": [
        {
          "name": "serialNo",
          "value": "<serialNo-value>"
        }
      ]
    },
    {
      "id": "<device-id2>",
      "connectivity": {
        "protocol": "uart",
        "serialPort": "<computer_serial_port_2>"
      },
      "identifiers": [
        {
          "name": "serialNo",
          "value": "<serialNo-value>"
        }
      ]
    }
  ]
}]
```

配置生成、刷入和测试设置

要让 AWS IoT Device Tester 在您的主板上自动生成和刷入测试用例，您必须使用您的生成和刷入工具的命令行界面来配置 AWS IoT Device Tester。生成和刷入设置在位于配置文件夹的 `userdata.json` 模板文件中配置。

为测试一个设备配置设置

`userdata.json` 必须具有以下结构：

```
{  
  "sourcePath": "<path-to-afr-source-code>",  
  "buildTool": {  
    "name": "<your-build-tool-name>",  
    "version": "<your-build-tool-version>",  
    "command": [  
      "<your-build-script>"  
    ]  
  },  
}
```

```
"flashTool":{  
    "name":"<your-flash-tool-name>",  
    "version":"<your-flash-tool-version>",  
    "command": [  
        "<your-flash-script>"  
    ]  
},  
"clientWifiConfig":{  
    "wifiSSID":"<your-wifi-ssid>",  
    " wifiPassword": "<your-wifi-password>",  
    " wifiSecurityType": "< wifi-security-type>"  
},  
"testWifiConfig":{  
    "wifiSSID": "<your-wifi-ssid>",  
    " wifiPassword": "<your-wifi-password>",  
    " wifiSecurityType": "< wifi-security-type>"  
},  
"otaConfiguration":{  
    "otaFirmwareFilePath": "<path-to-the-device-binary>",  
    "deviceFirmwareFileName": "<your-device-firmware-name>.bin",  
    "awsSignerPlatform": "AmazonFreeRTOS-Default",  
    "awsSignerCertificateArn": "arn:aws:acm:us-east-1:1000000001:certificate/e416379d-f3d6-46f3-868e-8721075ff076",  
    "awsUntrustedSignerCertificateArn": "arn:aws:acm:us-east-1:1000000001:certificate/0c81e2c6-f85e-46b1-9ed1-2c404309b210",  
    "awsSignerCertificateFileName": "ecdsa-sha256-signer.crt.pem",  
    "compileCodesignerCertificate": true  
}  
}
```

下面列出了在 `userdata.json` 文件中使用的属性：

sourcePath

移植的 Amazon FreeRTOS 源代码的根目录的路径。AWS IoT Device Tester 将值存储在 `{testData.sourcePath}` 变量中。

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。

Note

如果您使用的是 IDE，则必须向 IDE 提供命令行以在无管控模式下运行。

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。

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。此属性配置第一个接入点的 Wi-Fi 设置。客户端 Wi-Fi 设置在 `$AFR_HOME/tests/common/include/aws_clientcredential.h` 中配置。使用在 `aws_clientcredential.h` 中找到的值设置以下宏。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。

- `wifi_ssid`：Wi-Fi SSID，使用 C 字符串。
- `wifi_password`：Wi-Fi 密码，使用 C 字符串。
- `wifiSecurityType`：使用的 Wi-Fi 安全性的类型。

testWifiConfig

测试 Wi-Fi 配置。Wi-Fi 库测试要求主板连接到两个接入点。此属性配置第二个接入点。测试 Wi-Fi 设置在 `$AFR_HOME/tests/common/wifi/aws_test_wifi.c` 中配置。使用在 `aws_test_wifi.c` 中找到的值设置以下宏。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。

- testwifiWIFI_SSID : Wi-Fi SSID，使用 C 字符串。
 - testwifiWIFI_PASSWORD : Wi-Fi 密码，使用 C 字符串。
 - testwifiWIFI_SECURITY : 使用的 Wi-Fi 安全性的类型。下列值之一：
 - eWiFiSecurityOpen
 - eWiFiSecurityWEP
 - eWiFiSecurityWPA
 - eWiFiSecurityWPA2
- otaConfiguration
- OTA 配置。
- otaFirmwareFilePath
- 在生成之后，所创建 OTA 映像的完整路径。
- deviceFirmwareFileName
- MCU 设备上将用于下载 OTA 固件的完整文件路径。有些设备不使用此字段，但您仍必须提供一个值。
- awsSignerPlatform
- AWS Code Signer 在创建 OTA 更新任务时使用的签名算法。目前，此字段的可能值为 AmazonFreeRTOS-TI-CC3220SF 和 AmazonFreeRTOS-Default。
- awsSignerCertificateArn
- 上传到 AWS Certificate Manager (ACM) 的可信证书的 Amazon 资源名称 (ARN)。有关创建可信证书的更多信息，请参阅[创建代码签名证书](#)。
- awsUntrustedSignerCertificateArn
- 上传到 ACM 但设备不应信任的代码签名证书的 Amazon 资源名称 (ARN)。这用于测试无效证书测试用例。
- compileCodesignerCertificate
- 如果代码签署人签名验证证书未预配置或刷入，从而必须编译到项目中，则此项设置为 true。AWS IoT Device Tester 从 ACM 提取可信证书并将其编译到 aws_codesigner_certificate.h 中。

为测试多个设备配置设置

生成、刷入和测试设置在 `userdata.json` 文件中进行。以下 JSON 示例显示您如何配置 AWS IoT Device Tester 来测试多个设备：

```
{  
    "sourcePath": "<absolute-path-to/amazon-freertos>",  
    "buildTool": {  
        "name": "<your-build-tool-name>",  
        "version": "<your-build-tool-version>",  
        "command": [  
            "<absolute-path-to/build-parallel-script> {{testData.sourcePath}}"  
        ]  
    },  
    "flashTool": {  
        "name": "<your-flash-tool-name>",  
        "version": "<your-flash-tool-version>",  
        "command": [  
            "<absolute-path-to/flash-parallel-script> {{testData.sourcePath}}  
            {{device.connectivity.serialPort}}"  
        ]  
    }  
}
```

```
        },
        "clientWifiConfig": {
            "wifiSSID": "<ssid>",
            "wifiPassword": "<password>",
            "wifiSecurityType": "eWiFiSecurityOpen | eWiFiSecurityWEP | eWiFiSecurityWPA | eWiFiSecurityWPA2"
        },
        "testWifiConfig": {
            "wifiSSID": "<ssid>",
            "wifiPassword": "<password>",
            "wifiSecurityType": "eWiFiSecurityOpen | eWiFiSecurityWEP | eWiFiSecurityWPA | eWiFiSecurityWPA2"
        },
        "otaConfiguration": {
            "otaFirmwareFilePath": "{{ testData.sourcePath }}/<relative-path-to/ota-image-generated-in-build-process>",
            "deviceFirmwareFileName": "<absolute-path-to/ota-image-on-device>",
            "awsSignerPlatform": "AmazonFreeRTOS-Default | AmazonFreeRTOS-TI-CC3220SF",
            "awsSignerCertificateArn": "arn:aws:acm:<region>:<account-id>:certificate:<certificate-id>",
            "awsUntrustedSignerCertificateArn": "arn:aws:acm:<region>:<account-id>:certificate:<certificate-id>",
            "awsSignerCertificateFileName": "<awsSignerCertificate-file-name>",
            "compileCodesignerCertificate": true | false
        }
    }
}
```

下面列出了在 `userdata.json` 中使用的属性：

sourcePath

移植的 Amazon FreeRTOS 源代码的根目录的路径。AWS IoT Device Tester 将值存储在 `{{ testData.sourcePath }}` 变量中。

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。在生成命令中对源代码路径的所有引用必须使用 AWS IoT Device Tester 变量 `{{ testdata.sourcePath }}` 进行替换。

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。在刷入命令中对源代码路径的所有引用必须使用 AWS IoT Device Tester 变量 `{{ testdata.sourcePath }}` 进行替换。

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。此属性配置第一个接入点的 Wi-Fi 设置。客户端 Wi-Fi 设置在 `$AFR_HOME/tests/common/include/aws_clientcredential.h` 中配置。使用在 `aws_clientcredential.h` 中找到的值设置以下宏。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。

- `wifi_ssid` : Wi-Fi SSID，使用 C 字符串。
- `wifi_password` : Wi-Fi 密码，使用 C 字符串。
- `wifiSecurityType` : 使用的 Wi-Fi 安全性的类型。

testWifiConfig

测试 Wi-Fi 配置。Wi-Fi 库测试要求主板连接到两个接入点。此属性配置第二个接入点。测试 Wi-Fi 设置在 `$AFR_HOME/tests/common/wifi/aws_test_wifi.c` 中配置。使用在 `aws_test_wifi.c` 中找到的值设置以下宏。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。

Note

如果您的主板不支持 Wi-Fi，您仍须在 `device.json` 文件中包含 `clientWifiConfig` 和 `testWifiConfig` 部分，但您可以忽略这两个属性的值。

- testwifiWIFI_SSID : Wi-Fi SSID，使用 C 字符串。
- testwifiWIFI_PASSWORD : Wi-Fi 密码，使用 C 字符串。
- testwifiWIFI_SECURITY : 使用的 Wi-Fi 安全性的类型。下列值之一：
 - eWiFiSecurityOpen
 - eWiFiSecurityWEP
 - eWiFiSecurityWPA
 - eWiFiSecurityWPA2

otaConfiguration

OTA 配置。

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 固件所在位置的完整路径。有些设备不使用此字段，但您仍必须提供一个值。

awsSignerPlatform

AWS Code Signer 在创建 OTA 更新任务时使用的签名算法。目前，此字段的可能值为 AmazonFreeRTOS-TI-CC3220SF 和 AmazonFreeRTOS-Default。

awsSignerCertificateArn

上传到 AWS Certificate Manager (ACM) 的可信证书的 Amazon 资源名称 (ARN)。有关创建可信证书的更多信息，请参阅[创建代码签名证书](#)。

awsUntrustedSignerCertificateArn

上传到 ACM 的代码签名证书的 ARN。

compileCodesignerCertificate

如果代码签署人签名验证证书未预配置或刷入，从而必须编译到项目中，则此项设置为 true。AWS IoT Device Tester 从 ACM 提取可信证书并将其编译到 aws_codesigner_certificate.h 中。

AWS IoT Device Tester 变量

用于生成代码和刷入设备的命令可能需要连接或者有关设备的其他信息才能成功运行。AWS IoT Device Tester 允许您使用 [JsonPath](#)，在刷入和生成命令中引用设备信息。使用简单 JsonPath 表达式，您可以按照 device.json 文件中的指定内容提取所需的信息。

路径变量

AWS IoT Device Tester 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}}

扩展到源代码路径的变量。

{{device.connectivity.serialPort}}

扩展到串行端口的变量。

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

扩展到您设备的序列号的变量。

AWS IoT Device Tester 变量和并发测试

要允许针对不同测试组并行生成源代码，AWS IoT Device Tester 需要将源代码复制到 AWS IoT Device Tester 所提取文件夹内的结果文件夹。您的生成或刷入命令中的源代码路径必须使用 `testdata.sourcePath` 变量来引用。AWS IoT Device Tester 使用所复制源代码的临时路径来替换此变量。

文件路径和 Windows 操作系统

如果您在 Windows 上运行 AWS IoT Device Tester，请在 AWS IoT Device Tester 配置文件的文件路径中使用正斜杠 (/)。例如，`userdata.json` 中的 `sourcePath` 应表示为 `c:/<dir1>/<dir2>`。

运行 Amazon FreeRTOS 资格套件

您可以使用 AWS IoT Device Tester 可执行文件与 AWS IoT Device Tester 交互。以下命令行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。

```
devicetester_[linux | mac | win_x86-64] 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 AWS IoT Device Tester，请使用正斜杠 (/) 指定 `userdata.json` 的路径。

使用以下命令运行特定测试组：

```
devicetester_[linux | mac | win_x86-64] run-suite --suite-id AFQ_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。

AWS IoT Device Tester 命令行选项

suite-id

选填项。指定要运行的测试套件。

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。

AWS IoT Device Tester 命令

AWS IoT Device Tester 命令支持以下操作：

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。

结果和日志

此部分介绍如何查看和解释测试结果及日志。

查看结果

AWS IoT Device Tester 执行资格测试套件之后，它会在每次运行资格测试套件时生成两个报告：`awsiotdevicetester_report.xml` 和 `AFO_Report.xml`。这些报告可在 `<devicetester-extract-location>/results/<execution-id>` 中找到。

`awsiotdevicetester_report.xml` 是您提交给 AWS 的资格测试报告，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：

- AWS IoT Device Tester 版本。
- 所测试的 Amazon FreeRTOS 版本。
- 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。
- 根据设备功能（例如，FullWiFi、FullMQTT 等）所测试的测试用例结果，按照库细分。

`AFO_Report.xml` 是标准 `junit.xml` 格式的报告，您可以集成到现有 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。
- 根据设备功能（例如，FullWiFi、FullMQTT 等）所测试的测试用例结果，按照库细分。

解释 AWS IoT Device Tester 结果

`awsiotdevicetester_report.xml` 或 `AFO_Report.xml` 中的报告部分列出了已经运行的测试以及测试结果。

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

```
<testsuites name="AFO 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

AWS IoT Device Tester 无法执行的测试用例数。

disabled

此属性未使用，可以忽略。

如果没有测试用例故障或错误，您的设备满足运行 Amazon FreeRTOS 的技术要求并可以与 AWS IoT 服务集成。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明。

如果出现测试用例失败或错误，您可以通过检查 `<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，但具有称为 `skipped` 的额外属性，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>
```

<testcase> 标签中使用的属性

name

测试用例的名称。

attempts

AWS IoT Device Tester 执行测试用例的次数。

测试用例失败或出现错误时，`<failure>` 或 `<error>` 标签将添加到 `<testcase>` 标签，包括用于故障排除的额外信息。例如：

```
<testcase classname="mcu.Full_MQTT" name="AFQP_MQTT_Connect_HappyCase" attempts="1">
  <failure type="Failure">Reason for the test case failure</failure>
  <error>Reason for the test case execution error</error>
</testcase>
```

查看日志

您可以在 `<devicetester-extract-location>/results/<execution-id>/logs` 中找到 AWS IoT Device Tester 从测试执行生成的日志。它会生成两组日志：

test_manager.log

包含 AWS IoT Device Tester 的 Test Manager 组件所生成的日志。例如，其中包括日志相关配置、测试顺序和报告生成。

`<test_group_name>.log` (for example, `Full_MQTT.log`)

测试组的日志，包括来自所测试设备的日志。

用于重新确定资格的测试

随着 AWS IoT Device Tester 资格测试的新版本发布，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 AWS IoT Device Tester 来测试微控制器主板。对于后续资格，请确保您具有 Amazon FreeRTOS 和 AWS IoT Device Tester 的最新版本并重新运行资格测试。

问题排查

建议采用以下工作流程对测试 Amazon FreeRTOS 设备进行问题排查：

1. 阅读控制台输出。
2. 查看 `AFO_Report.xml` 文件。
3. 查看位于 `/results/<uuid>/logs` 下的日志文件。
4. 调查以下问题领域之一：
 - 设备配置
 - 设备接口
 - 设备工具
 - Amazon FreeRTOS 源代码

排查设备配置问题

使用 AWS IoT Device Tester 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。

如果仍有问题，请参阅以下调试过程。

在哪里查找问题？

首先查看 `/results/<uuid>` 目录中的 `AFO_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，请查看各个测试组下的 `/results/<uuid>/<test-case-id>.log`。

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。AWS IoT Device Tester 执行 JSON 验证并输入调试信息。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修复错误，但是如果仍不能定位错误，可以在 IDE 中、使用文本编辑器（例如 Atom 或 Sublime）或者通过 JSONLint 等在线工具手动执行验证。

缺少必需参数错误

由于将新功能添加到 AWS IoT Device Tester 中，可能会对配置文件进行更改。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/<uuid>/logs` 下的 `<test_case_id>.log` 文件明确列出了所有缺少的参数。AWS IoT Device Tester 验证您的 JSON 配置文件架构，确保使用了支持的最新版本。

无法启动测试错误

在测试启动期间，您可能遇到指示失败的错误。造成这种情况有多种可能的原因，请确保检查以下领域正确与否：

- 确保您包括在执行命令中的池名称实际存在。这从您的 device.json 文件直接引用。
- 确保您池中的设备具有正确的配置参数。

设备接口和端口

此部分包含有关 AWS IoT Device Tester 连接到设备所用设备接口的信息。

支持的平台

AWS IoT Device Tester 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.*
- Windows : COM*

要查看您的设备 ID，请执行以下操作：

- 对于 Linux/macOS，打开终端并运行 ls /dev/tty*。
- 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。在连接到计算机时，设备有两个端口是很常见的情况，一个是用于刷入设备的数据端口，另一个用于读取输出。在 device.json 文件中设置正确的端口非常关键。否则，刷入或者读取输出可能会失败。

在有多个端口时，请确保在您的 device.json 文件中使用设备的数据端口。例如，如果您插入 Espressif WROver 设备并且分配给它的两个端口为 /dev/ttyUSB0 和 /dev/ttyUSB1，请在 device.json 文件中使用 /dev/ttyUSB1。

对于 Windows，请按照相同的逻辑操作。

读取设备数据

AWS IoT Device Tester 使用单独的设备生成和刷入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

这些设置由 AWS IoT Device Tester 处理，在您这一端无需任何配置。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，可以使用 screen 命令完成此操作。在 Windows 上，您可以使用诸如 TeraTerm 等程序。

```
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

对于 TI 设备，建议您下载并安装 Code Composer Studio 7.3。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：

- Amazon FreeRTOS 无法生成或刷入到您的设备。
- Code Composer Studio 安装程序可能会冻结。
- 在生成或刷入过程中，控制台未显示任何日志输出。
- 即使以无管控模式调用，生成命令尝试以 GUI 模式启动。

Amazon FreeRTOS 源代码

以下部分讨论对 Amazon FreeRTOS 源代码的问题进行排查。

代码勘误表

每个 Amazon FreeRTOS 发行版绑定了一个位于 `/amazon-freertos/tests` 目录下的文档，其中包含该发行版的所有勘误信息。建议您通读此文档，然后再运行任何测试。

对由于类似以下原因而当前未通过测试的任何设备，勘误表文档中均包含一个条目：

- 硬件不支持的特定功能。
- 硬件支持功能，但 Amazon FreeRTOS 目前在设备上尚不支持它。
- 硬件支持功能，但底层软件堆栈不支持硬件（非 AFR）。

如果勘误表不包含特定于您设备的信息，请按照接下来部分中概述的信息，继续调试过程。

调试 Amazon FreeRTOS

出现源代码错误时，AWS IoT Device Tester 将调试输出写入到 `/results/<uuid>/logs` 目录中的 `<test-group-id>.log` 文件。搜索文件中的任何错误实例。错误将指向 Amazon FreeRTOS 源代码中的一个位置。然后，您可以使用日志中的行号和文件路径信息来引用导致错误的一段源代码。

日志记录

AWS IoT Device Tester 日志放在一个位置。从 AWS IoT Device Tester 根目录，可用文件包括：

- `./results/<uuid>`
- `AFO_Report.xml`
- `awsiotdevicetester_report.xml`
- `/logs/<test_group_id>.log`

需要查看的最重要日志是 `<test_group_id>.log` 和 `results.xml`。后者包含有关哪些测试失败的信息以及具体错误消息。然后，您可以使用前者来深入挖掘问题以更好地了解上下文。

控制台错误

AWS IoT Device Tester 运行时，会向控制台报告失败及简短消息。查看 `<test_group_id>.log` 以了解有关错误的更多信息。

日志错误

`<test-group-id>.log` 文件位于 `/results/<uuid>` 目录中。每个测试执行具有唯一的测试 ID，用于创建 `<uuid>` 目录。单个测试组日志位于 `<uuid>` 目录下。使用 AWS IoT 控制台查找失败的测试组，然后在 `/results/<uuid>` 目录中打开该组的日志文件。此文件中的信息包括完整的生成和刷入命令输出，以及测试执行输出和更详细的 AWS IoT Device Tester 控制台输出。

权限策略模板

下面是授予 AWS IoT Device Tester 所需权限的策略模板：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "iam:CreatePolicy",
        "iam:DetachRolePolicy",
        "iam:DeleteRolePolicy",
        "s3:CreateBucket",
        "iam:DeletePolicy",
        "iam>CreateRole",
        "iam>DeleteRole",
        "iam:AttachRolePolicy",
        "s3:DeleteBucket",
        "s3:PutBucketVersioning"
      ],
      "Resource": [
        "arn:aws:s3:::idt*",
        "arn:aws:s3:::afr-ota*",
        "arn:aws:iam::*:policy/idt*",
        "arn:aws:iam::*:role/idt*"
      ]
    },
    {
      "Sid": "VisualEditor1",
      "Effect": "Allow",
      "Action": [
        "iot>DeleteCertificate",
        "iot:AttachPolicy",
        "iot:DetachPolicy",
        "s3:DeleteObjectVersion",
        "iot:DeleteOTAUpdate",
        "s3:PutObject",
        "s3:GetObject",
        "iam:PassRole",
        "iot>DeleteStream",
        "iot:DeletePolicy",
        "iot:UpdateCertificate",
        "iot:GetOTAUpdate",
        "s3:DeleteObject"
      ]
    }
  ]
}
```

```
"iot:DescribeJobExecution",
"s3:GetObjectVersion"
],
"Resource": [
"arn:aws:iot:/*::thinggroup/idt*",
"arn:aws:iot:/*::policy/idt*",
"arn:aws:iot:/*::otaupdate/idt*",
"arn:aws:iot:/*::thing/idt*",
"arn:aws:iot:/*::cert/*",
"arn:aws:iot:/*::job/*",
"arn:aws:iot:/*::stream/*",
"arn:aws:iam::role/idt*",
"arn:aws:s3:::afr-ota/*",
"arn:aws:s3:::idt/*",
"arn:aws:iam::role/idt*"
]
},
{
"Sid": "VisualEditor2",
"Effect": "Allow",
"Action": [
"iot:DetachThingPrincipal",
"iot:AttachThingPrincipal",
"s3>ListBucketVersions",
"iot>CreatePolicy",
"iam>ListRoles",
"freertos>ListHardwarePlatforms",
"signer>DescribeSigningJob",
"s3>ListBucket",
"signer:*",
"iot>DescribeEndpoint",
"iot>CreateStream",
"signer>StartSigningJob",
"s3>ListAllMyBuckets",
"signer>ListSigningJobs",
"acm>GetCertificate",
"acm>ListCertificates",
"acm>ImportCertificate",
"freertos>DescribeHardwarePlatform",
"iot>CreateKeysAndCertificate",
"iot>CreateCertificateFromCsr",
"s3>GetBucketLocation",
"iot>GetRegistrationCode",
"iot>RegisterCACertificate",
"iot>RegisterCertificate",
"iot>UpdateCACertificate",
"iot>DeleteCACertificate",
"iot>DeleteCertificate",
"iot>UpdateCertificate"
],
"Resource": "*"
},
{
"Sid": "VisualEditor3",
"Effect": "Allow",
"Action": [
"s3>PutObject",
"s3>GetObject"
],
"Resource": [
"arn:aws:s3:::afr/*",
"arn:aws:s3:::idt/*"
]
},
{
"Sid": "VisualEditor4",
"Effect": "Allow",
"Action": [
"iot>DescribeJobExecution",
"iot>GetJobExecution",
"iot>GetJobRun",
"iot>GetJobRunResult"
],
"Resource": [
"arn:aws:iot:/*::job/*"
]
}
```

```
"Effect": "Allow",
"Action": [
    "iot:CreateOTAUpdate",
    "iot:CreateThing",
    "iot>DeleteThing"
],
"Resource": "*"
}]
```