
FreeRTOS

使用者指南



FreeRTOS: 使用者指南

Copyright © 2021 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

什麼是 FreeRTOS ?	1
下載 FreeRTOS 原始程式碼	1
FreeRTOS 版本控制	1
FreeRTOS 架構	1
FreeRTOS 符合資格的硬體平台	2
開發工作流程	3
其他資源	3
FreeRTOS 核心基礎	4
FreeRTOS 核心排程器	4
記憶體管理	4
核心記憶體配置	4
應用程式記憶體管理	5
任務間協調	5
Queues	5
旗號與 Mutex	5
直達任務通知	6
串流緩衝區	6
訊息緩衝區	7
軟體計時器	8
低電力支援	8
FreeRTOSConfig.h	8
FreeRTOS 主控台	9
預先定義 FreeRTOS 組態	9
自訂 FreeRTOS 組態	9
快速連線工作流程	10
標記組態	10
搭配 IAM 政策使用標籤	10
AWS IoT 適用於 Embedded 的 裝置 SDK C	13
FreeRTOS 入門	14
FreeRTOS 示範應用程式	14
首要步驟	14
設定 AWS 帳戶和許可	15
向 AWS IoT 註冊您的 MCU 電路板	15
下載 FreeRTOS	17
設定 FreeRTOS 示範	18
故障診斷	19
一般入門故障診斷提示	19
安裝終端機模擬器	20
使用 CMake 配 FreeRTOS	20
Prerequisites	21
使用第三方的程式碼編輯器與除錯工具來開發 FreeRTOS 應用程式	22
建築 FreeRTOS 配 CMake	22
開發人員模式金鑰佈建	26
Introduction	26
選項 #1：從 AWS IoT 匯入私有金鑰	26
選項 #2：產生內建私有金鑰	26
主機板特定的入門指南	28
Cypress CYW943907AEVAL1F 開發套件	29
Cypress CYW954907AEVAL1F 開發套件	31
Cypress CY8CKIT-064S0S2-4343W	34
Microchip ATECC608A 安全元素與 Windows 模擬器	38
Espressif ESP32-DevKitC 和 ESP-WROVER-KIT	42
Espressif ESP32-WROOM-32SE	56
Infineon XMC4800 IoT 連接套件	61

Infineon OPTIGA Trust X 和 XMC4800 IoT Connectivity Kit	65
Marvell MW32x AWS IoT 入門套件	69
MediaTek MT7697Hx 開發套件	84
Microchip Curiosity PIC32MZ EF	88
Nordic nRF52840-DK	91
Nuvoton NuMaker-IoT-M487	94
NXP LPC54018 IoT 模組	100
Renesas Starter Kit+ for RX65N-2MB	103
STMicroelectronics STM32L4 探索套件 IoT 節點	106
Texas Instruments CC3220SF-LAUNCHXL	108
Windows 裝置模擬器	111
Xilinx Avnet MicroZed 工業 IoT 套件	114
無線更新	120
標記 OTA 資源	120
OTA 更新先決條件	120
建立 Amazon S3 儲存貯體來存放您的更新	120
建立 OTA 更新服務角色	121
建立 OTA 使用者政策	122
建立程式碼簽署憑證	124
將存取權限授予適用於 AWS IoT 的程式碼簽署	130
使用 OTA 程式庫下載 FreeRTOS	130
使用 MQTT 進行 OTA 更新的先決條件	130
使用 HTTP 進行 OTA 更新的先決條件	132
OTA 教學	135
安裝初始韌體	136
更新您的韌體版本	144
建立 OTA 更新 (AWS IoT 主控台)	144
使用 AWS CLI 建立 OTA 更新	147
OTA Update Manager 服務	162
將 OTA 代理程式整合到您的應用程式	162
連線管理	163
簡單OTA演示	163
針對 OTA 完成事件使用自訂回撥	165
OTA 安全性	166
適用於 AWS IoT 的程式碼簽署	166
OTA 故障診斷	166
設定 CloudWatch Logs 以進行 OTA 更新	167
使用 AWS CloudTrail 記錄 AWS IoT OTA API 呼叫	170
Get CreateOTAUpdate failure details using the AWS CLI	172
使用 AWS CLI 取得 OTA 失敗代碼	173
故障診斷多個裝置的 OTA 更新	174
故障診斷 Texas Instruments CC3220SF Launchpad 的 OTA 更新	174
FreeRTOS 程式庫	175
FreeRTOS 移植程式庫	175
FreeRTOS 應用程式庫	176
FreeRTOS 常見的程式庫	177
設定 FreeRTOS 程式庫	177
常見的程式庫	178
不可部分完成操作	178
Linear Containers	178
日誌	179
Static Memory	179
Task Pool	179
低功耗藍牙	182
Overview	182
Architecture	182
相依性和要求	183

程式庫組態檔案	184
Optimization	184
使用限制	184
Initialization	185
API 參考	186
範例使用方式	186
Porting	189
藍牙裝置適用的 SDKs 行動FreeRTOS	191
通用 I/O	191
AWS IoT Device Defender	192
Introduction	192
AWS IoT Greengrass	192
Overview	192
相依性和要求	192
API 參考	193
範例使用方式	193
coreHTTP	194
Introduction	194
coreJSON	195
Introduction	195
記憶體用量	195
coreMQTT	195
Introduction	195
OTA 代理程式	196
Overview	196
Features	196
API 參考	197
範例使用方式	197
Porting	197
corePKCS11	198
Overview	198
Features	198
非對稱加密系統支援	199
Porting	200
Secure Sockets	200
Overview	200
相依性和要求	200
Features	201
Troubleshooting	201
開發人員支援	201
使用限制	202
Initialization	202
API 參考	202
範例使用方式	202
Porting	204
AWS IoT Device Shadow	204
Introduction	204
AWS IoT 任務	204
Introduction	204
Transport Layer Security	205
Wi-Fi	205
Overview	205
相依性和要求	205
Features	205
Configuration	207
Initialization	207
API 參考	207

範例使用方式	208
Porting	209
FreeRTOS 示範	210
執行 FreeRTOS 示範	210
設定示範	210
低功耗藍牙	210
Overview	210
Prerequisites	211
常見元件	213
透過低功耗藍牙執行的 MQTT	217
Wi-Fi 佈建	218
一般屬性伺服器	220
Microchip Curiosity PIC32MZEF 的開機載入器	221
開機載入器狀態	221
快閃裝置	222
應用程式映像結構	223
映像標頭	223
映像描述項	224
映像尾部	225
開機載入器組態	225
建置開機載入器	225
AWS IoT Device Defender	226
Introduction	226
Functionality	226
AWS IoT Greengrass	229
Amazon EC2	231
coreHTTP	232
coreHTTP 交互身份驗證	232
coreHTTP 上傳Amazon S3	236
coreHTTP S3 下載	237
AWS IoT 任務	239
Introduction	239
原始程式碼組織	239
設定 AWS IoT MQTT 中介裝置連接	239
Functionality	239
coreMQTT	241
Introduction	241
Functionality	241
以指數退避與抖動重試邏輯	244
連接到 MQTT 中介裝置	244
描述 MQTT 主題	245
發布到主題	246
接收傳入的訊息	246
處理傳入的 MQTT 發布封包	247
取消訂用主題	247
無線更新	248
空中示範組態	251
Texas Instruments CC3220SF-LAUNCHXL	251
Microchip Curiosity PIC32MZEF	253
Espressif ESP32	257
Renesas RX65N	257
AWS IoT Device Shadow	277
Introduction	277
Functionality	277
連接到 AWS IoT MQTT 中介裝置	282
刪除影子文件	282
訂用影子主題	282

傳送影子更新	283
處理影子差量訊息和影子更新訊息	283
Secure Sockets	288
使用 AWS IoT Device Tester for FreeRTOS	290
IDT for FreeRTOS 的支援版本	291
IDT for FreeRTOS 的最新版本	291
舊版 IDT	292
不支援的 IDT 版本	293
Prerequisites	295
下載 FreeRTOS	295
LTS 資格 (使用 LTS 程式庫的 FreeRTOS 資格)	296
下載 IDT for FreeRTOS	296
建立並設定 AWS 帳戶	296
AWS IoT Device Tester 受管政策	298
(選用) 安裝 AWS Command Line Interface	298
首次準備測試微型控制器主機板	298
新增程式庫移植層	298
設定 AWS 登入資料	298
在 IDT for FreeRTOS 中建立裝置集區	299
設定建置、刷新和測試設定	302
執行低功耗藍牙測試	309
Prerequisites	310
Raspberry Pi 設定	310
FreeRTOS 裝置設定	312
執行 BLE 測試	312
故障診斷 BLE 測試	312
執行 FreeRTOS 資格套件	313
IDT for FreeRTOS 命令	315
重新取得資格的測試	316
測試套件版本	316
了解結果和日誌	317
檢視結果	317
使用 IDT 來開發和執行您自己的測試套件	320
下載最新版本的 IDT for FreeRTOS	320
測試套件建立工作流程	320
教學課程：建置並執行範例 IDT 測試套件	320
教學課程：開發簡單的 IDT 測試套件	324
建立 IDT 測試套件組態檔案	330
設定 IDT 狀態機器	335
建立 IDT 測試案例可執行檔	350
使用 IDT 環境	354
設定測試執行器的設定	357
偵錯和執行自訂測試套件	364
檢視 IDT 測試結果和Logs	366
IDT 用量指標	370
Troubleshooting	373
故障診斷裝置組態	373
故障診斷逾時錯誤	379
行動功能和 AWS 費用	379
支援政策	379
AWS 中的安全	380
Identity and Access Management	380
Audience	380
使用身分來驗證	381
使用政策管理存取權	382
進一步了解	383
AWS 服務如何使用 IAM	383

身分型政策範例	385
疑難排解	387
合規驗證	389
彈性	389
基礎設施安全	390
	cccxci

什麼是 FreeRTOS ?

FreeRTOS 是適用於微控制器和小型微處理器的即時作業系統 (RTOS) , 在市場中位居領導地位 , 此產品與全球領先的晶片公司合作開發超過 15 年 , 現在每 175 秒就有使用者下載一次。FreeRTOS 透過 MIT 開放原始碼授權自由散佈 , 包括核心和越來越多的程式庫集 , 適用於所有產業領域。FreeRTOS 是強調可靠性和易用性所建置而成的系統。

FreeRTOS 包含用於連接、安全和遠端 (OTA) 更新的程式庫。FreeRTOS 也包含示範應用程式，顯示 FreeRTOS 合格電路板上的 [功能](#)。

FreeRTOS 是一個開放原始碼專案。您可以下載原始程式碼、提供變更或改進，或在 GitHub 網站 (<https://github.com/aws/amazon-freertos>) 我們是根據 MIT 開放原始碼授權發行 FreeRTOS 程式碼，因此您可以在商業和個人專案中使用。

我們也歡迎您對 FreeRTOS 文件貢獻心力 (FreeRTOS 使用者指南、FreeRTOS 移植指南和 FreeRTOS 資格指南)。文件的 Markdown 來源可在 <https://github.com/awsdocs/aws-freertos-docs> 取得。它是根據 Creative Commons (CC BY-ND) 授權發行。

下載 FreeRTOS 原始程式碼

您可以從 FreeRTOS 主控台FreeRTOS下載針對 [合格平台設定的 FreeRTOS 版本](#)。如需合格平台的清單，請參閱 [FreeRTOS 符合資格的硬體平台 \(p. 2\)](#)或是 [FreeRTOS 合作夥伴網站](#)。

您也可以從 FreeRTOS GitHub 複製或下載。請前往 [README.md](#) 檔案以獲得指示。

FreeRTOS 版本控制

FreeRTOS 核心和元件會個別發布並使用語意版本控制。整合的 FreeRTOS 版本會定期釋出。所有版本都會使用格式為 YYYYMM.NN 的日期型版本控制，其中：

- Y 代表年份。
- M 代表月份。
- N 代表指定月份內的發行訂單 (00 為第一版)。

例如，2021 年 6 月的第二版會是 202106.01。

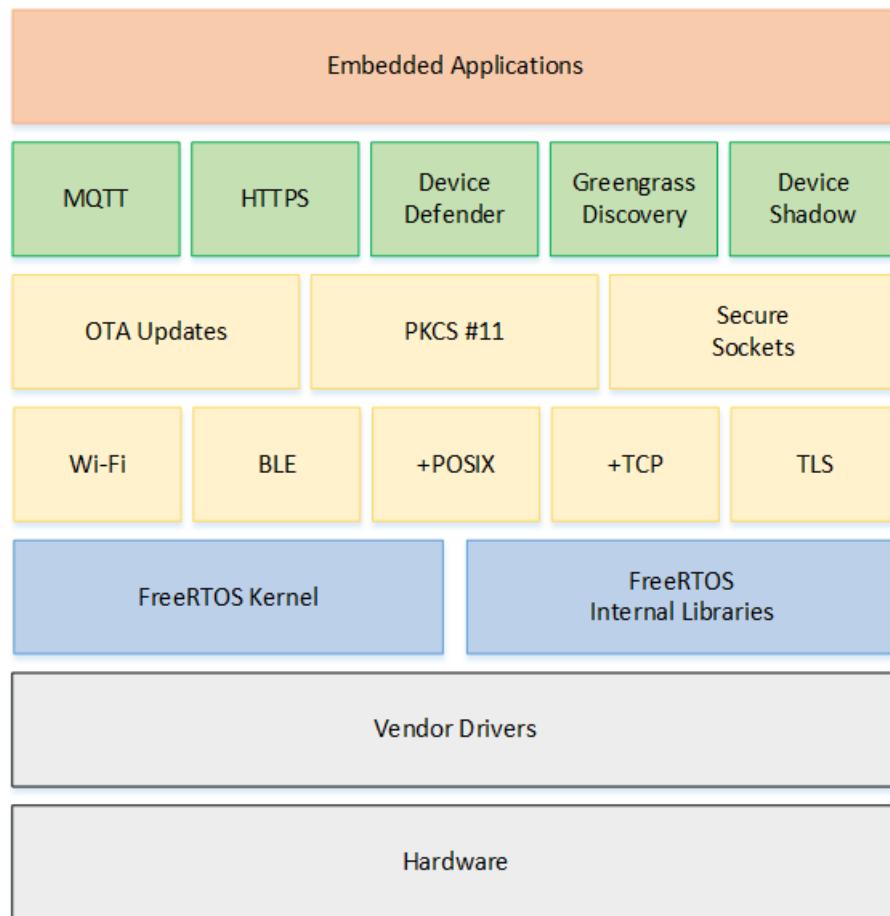
之前，FreeRTOS 會針對主要版本使用語意版本控制。雖然它已移至日期類型的版本控制 (FreeRTOS 1.4.8 更新至 FreeRTOS AWS 參考整合 201906.00)，但 FreeRTOS 核心和每個 FreeRTOS 程式庫仍然會保留語意版本控制。在語意版本控制中，版本編號本身 (X.Y.Z) 指出該版本是主要、次要或點版本。您可以使用程式庫的語意版本，來評估新版本對應用程式的範圍和影響。

LTS 版本的維護方式與其他版本類型不同。除了瑕疵解決之外，主要和次要版本也會經常使用新功能進行更新。LTS 版本只會更新以解決關鍵瑕疵和安全漏洞的變更。特定 LTS 版本在啟動後不會推出任何新功能。它們在發佈後維護至少三個日曆年，並為裝置製造商提供使用穩定基準的選項，而不是由主要和次要版本所代表的更動態基準。

FreeRTOS 架構

FreeRTOS 通常會做為單一編譯映像，與裝置應用程式需要的所有元件一起刷入裝置。此映像結合嵌入式開發人員編寫的應用程式功能、Amazon 所提供的軟體程式庫、FreeRTOS 核心，以及硬體平台的驅動程式與

主機板支援套件 (BSP)。嵌入式應用程式開發人員可以預期相同的標準化界面，包括 FreeRTOS 核心和所有 FreeRTOS 軟體程式庫，獨立於使用的各個微型控制器之外。



FreeRTOS 符合資格的硬體平台

以下硬體平台都符合 FreeRTOS 的資格：

- ATECC608AAWS零接觸配置套件 IoT
- Cypress CYW943907AEVAL1F 開發套件
- Cypress CYW954907AEVAL1F 開發套件
- Espressif ESP32-DevKitC
- Espressif ESP-WROVER-KIT
- 英飛龍XMC4800 IoT 連接套件
- Marvell MW320 AWS IoT 入門套件
- Marvell MW322 AWS IoT 入門套件
- 培養基Tek MT7697Hx 開發套件
- Microchip Curiosity PIC32MZEF 套件
- Nordic nRF52840-DK
- NuMaker-IoT-M487
- NXPLPC54018(NXP和LPC54018) IoT 模塊

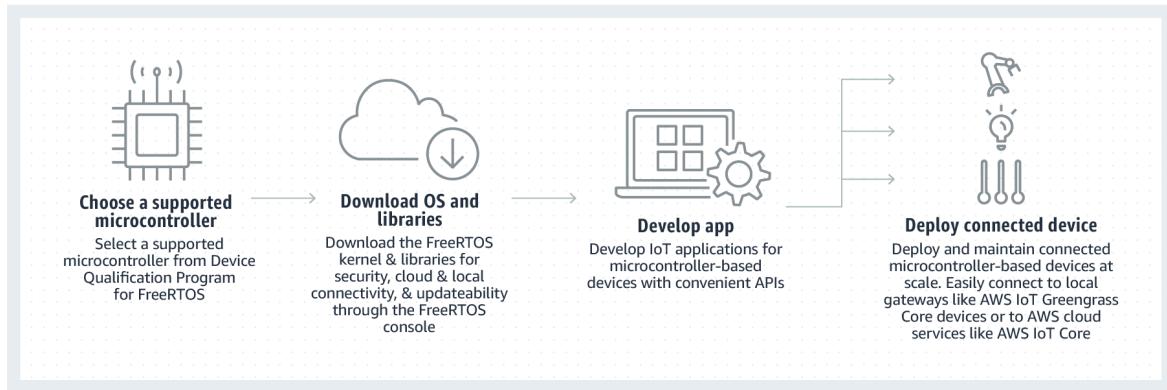
- OPTIGA Trust X 安全解決方案
- 運種的種今雖今運X65現運迄末 IoT 模塊
- STMicroelectronics STM32L4 發現套件 IoT 節點
- Texas Instruments CC3220SF-LAUNCHXL
- Microsoft Windows 7 或更新版本，含至少一個雙核心和有線乙太網路連線
- 西林克斯安富利 MicroZed 工業 IoT 試劑盒

合格的裝置也列在 [AWS Partner Device Catalog](#) 中。

如需有關讓新裝置符合資格的資訊，請參閱 [FreeRTOS 資格指南](#)。

開發工作流程

下載 FreeRTOS 來開始開發。您會將套件解壓縮並匯入 IDE。然後，您可以在您選取的硬體平台上開發應用程式，並使用適合您裝置的部署程序製造及部署這些裝置。部署的裝置可以連接至 AWS IoT 服務或 AWS IoT Greengrass，做為完整 IoT 方案的一部分。



其他資源

這些資源可能對您有所幫助。

- 如果您有關於 FreeRTOS 的任何疑問想要詢問 FreeRTOS 工程團隊，您可以在 [FreeRTOS 頁面 GitHub 上提出問題](#)。
- 有關 FreeRTOS 的技術問題，請造訪 [社群論壇 FreeRTOS](#)。
- 若要取得 AWS 的技術支援，請瀏覽 [AWS 支援中心](#)。
- 若要向 AWS 詢問有關 AWS 計費、帳戶服務、事件、濫用或其他問題，請瀏覽 [聯絡我們](#) 頁面。

FreeRTOS 核心基礎

核心是一種即時作業系統，支援許多架構。FreeRTOS 它非常適合用於建置內嵌微控制器應用程式。它提供的功能如下：

- 多工排程器。
- 多個記憶體配置選項 (包括建立完全靜態配置系統的能力)。
- 任務間的協調基本功能，包括任務通知、訊息佇列、多種旗號類型，以及串流及訊息緩衝區。

核心永遠不會在關鍵區段或中斷中執行非確定性操作，例如行走連結清單。FreeRTOS 核心包含高效率的軟體計時器實作，除非計時器需要服務，否則便不會使用任何 CPU 時間。FreeRTOS 封鎖的任務不需要耗費時間的定期服務。直達任務通知允許快速任務訊號，幾乎不會造成任何 RAM 額外負荷。它們可用於大多數的任務間及中斷至任務訊號案例。

核心小型、簡易且易於使用。FreeRTOS 典型 RTOS 核心二進位映像的範圍介於 4000 到 9000 位元組之間。

如需 FreeRTOS 核心的最新文件，請查看 [FreeRTOS.org](#)。FreeRTOS.org 提供一些有關使用 FreeRTOS 核心的詳細教學和指南，包括 [快速入門指南](#) 和更深入的 [掌握 FreeRTOS 即時核心](#)。

FreeRTOS 核心排程器

使用 RTOS 內嵌應用程式的結構可以視為一組獨立任務。每個任務都會在其自身的內容中執行，無須依存於其他任務。應用程式中在任何時間點上都只會有一個執行中的應用程式。即時 RTOS 排程器會決定每個任務執行的時機。每個任務都會取得一個自己的堆疊。當任務切換出去以讓另一個任務執行時，任務的執行內容會儲存到任務堆疊，使其可在相同任務於稍後切換回來繼續執行時進行還原。

為了提供具確定性的即時行為，FreeRTOS 任務排程器允許指派嚴格的優先順序給任務。RTOS 能確保可執行的最高優先順序任務獲得處理時間。具有相同優先順序的任務若同時執行，便需要共享處理時間。FreeRTOS 也會建立 idle 任務，該任務只會在其他任務未準備好執行時執行。

記憶體管理

本節提供有關核心記憶體配置和應用程式記憶體管理的相關資訊。

核心記憶體配置

每次建立任務、佇列或其他 RTOS 物件時，RTOS 核心都需要 RAM。RAM 可根據以下方式進行配置：

- 於編譯時間靜態配置。
- 由 RTOS API 物件建立函數從 RTOS 堆積動態配置。

動態建立 RTOS 物件時，基於多個原因，不一定適合使用標準 C 程式庫 `malloc()` 和 `free()` 函數：

- 它們可能無法用於內嵌系統。
- 它們會佔用寶貴的程式碼空間。

- 它們通常並非安全執行緒。
- 它們不具確定性。

基於這些原因，FreeRTOS 會將記憶體配置 API 保留在其可移植層。可攜式 layer 位於實作核心 RTOS 功能的來源檔案外部，因此您可以提供應用程式限定實作給您正在開發的即時系統。當 RTOS 核心需要 RAM 時，它會呼叫 `pvPortMalloc()` 而非 `malloc()`。釋放 RAM 時，RTOS 核心會呼叫 `vPortFree()` 而非 `free()`。

應用程式記憶體管理

當應用程式需要記憶體時，它們可以從 FreeRTOS 堆積配置它。FreeRTOS 提供數種堆積管理方案，其複雜度及功能各不相同。您也可以提供自己的堆積實作。

核心包含五個堆積實作：FreeRTOS

`heap_1`

為最簡易的實作。不允許釋放記憶體。

`heap_2`

允許釋放記憶體，但不會聯合相鄰的可用區塊。

`heap_3`

包裝標準 `malloc()` 及 `free()` 以確保執行緒的安全。

`heap_4`

聯合相鄰的可用區塊，避免分散。包含絕對地址置放選項。

`heap_5`

與 `heap_4` 相似。可延伸堆積，跨越多個不相鄰的記憶體區域。

任務間協調

本節包含有關 FreeRTOS 基本功能的資訊。

Queues

佇列是任務間通訊的主要形式。它們可用於在任務間及插斷與任務間傳送訊息。在大多數的情況下，它們會用來做為安全執行緒的先進先出 (FIFO) 緩衝區。新的資料會傳送到佇列後方。（資料也可以傳送到佇列前方。）訊息透過佇列透過複製傳送，這表示資料（可以是較大緩衝區的指標）本身會複製到佇列，而不只是存放資料的參考。

佇列 APIs 允許指定封鎖時間。當任務嘗試從空白佇列讀取時，任務會置放到封鎖狀態中，直到資料在佇列上可供使用，或是超過封鎖時間。處於封鎖狀態中的任務不會使用任何 CPU 時間，可讓其他任務執行。同樣的，當任務嘗試寫入填滿的佇列時，任務會置放到封鎖狀態中，直到佇列中有可用的空間，或是超過封鎖時間。若在相同佇列上有超過一個處於封鎖狀態的任務，則具有最高優先順序的任務會最先解除封鎖。

其他 FreeRTOS 基本功能（例如直達任務通知和串流及訊息緩衝區）可在許多常見設計案例中，提供除佇列之外的輕量替代項目。

旗號與 Mutex

核心提供二進位旗號、計數旗號，以及適用於互斥及同步的互斥。FreeRTOS

二進位旗號只能擁有兩個值。它們是實作同步處理 (任務間或任務與插斷間) 的好選擇。計數旗號可接收兩個以上的值。它們可讓許多任務共享資源或執行更複雜的同步處理操作。

Mutex 是二進位旗號，包含優先順序繼承機制。這表示若具有高優先順序的任務在嘗試取得由低優先順序任務保有的 mutex 時遭到封鎖，保有該字符串任務的優先順序會暫時提升至遭封鎖任務的優先順序。這項機制的目的是為了確保能盡可能縮短較高優先順序任務處於封鎖狀態中的時間，將已發生的優先順序反轉減至最少。

直達任務通知

任務通知可讓任務與其他任務互動，並和插斷服務常式 (ISR) 進行同步，而無須單獨的通訊物件 (例如旗號)。每個 RTOS 任務都具有一個 32 位元的通知值，用於存放通知的內容 (若有的話)。RTOS 任務通知是一種事件，會直接傳送到能解除鎖定接收端任務的任務，並選擇性更新接收端任務的通知值。

RTOS 任務通知可用來做為除二進位、計數旗號及佇列 (在某些情況下) 之外更快的輕量替代項目。相較於可用來執行相同功能的 FreeRTOS 功能，任務通知在速度和 RAM 使用量上都具有優勢。但是，只有在僅有一個任務能做為事件的收件人時，才能使用任務通知。

串流緩衝區

串流緩衝區可讓來自插斷服務常式的位元組串流傳遞至任務，或是從其中一個任務傳遞至另一個任務。位元組串流可為任意長度，並且不一定要有開始或結束。一次可以寫入任何數量的位元組，並且一次也能讀取任何數量的位元組。您可以透過在專案中加納入 `stream_buffer.c` 來源檔案，來啟用串流緩衝區功能。

串流緩衝區假設只有一個寫入緩衝區的任務或插斷 (寫入者)，並且只有一個從緩衝區讀取的任務或插斷 (讀取者)。寫入者和讀取者為不同的任務或插斷服務常式不會影響安全，但擁有多個寫入者或讀取者則不安全。

串流緩衝區實作會使用直達服務通知。因此，呼叫將呼叫端任務置放到封鎖狀態的串流緩衝區 API 可變更任務的通知狀態及值。

傳送資料

`xStreamBufferSend()` 是用來將資料傳送到任務中的串流緩衝區。`xStreamBufferSendFromISR()` 用於在插斷服務常式 (ISR) 中將資料傳送到串流緩衝區。

`xStreamBufferSend()` 允許指定封鎖時間。使用非零的封鎖時間呼叫 `xStreamBufferSend()` 以寫入串流緩衝區，並且緩衝區已填滿時，任務便會置放到封鎖狀態，直到有可用的空間或超過封鎖時間。

`sbSEND_COMPLETED()` 和 `sbSEND_COMPLETED_FROM_ISR()` 是當資料寫入串流緩衝區時，呼叫的巨集 (由 FreeRTOS API 交互呼叫)。它會取得已更新串流緩衝區的控點。這兩個巨集都會檢查串流緩衝區上是否有正在等待資料的遭封鎖任務；若有的話，便會將任務從封鎖狀態中移除。

您可以透過在 [FreeRTOSConfig.h \(p. 8\)](#) 中提供自己的 `sbSEND_COMPLETED()` 實作，來變更此預設行為。這在使用串流緩衝區來在多核心處理器的核心之間傳遞資料時很有用。在這種案例下，可實作 `sbSEND_COMPLETED()` 來在另一個 CPU 核心內產生插斷，然後插斷的服務常式便能使用 `xStreamBufferSendCompletedFromISR()` API 來檢查正在等待資料的任務，並視需要解除封鎖。

接收資料

`xStreamBufferReceive()` 用於從任務中的串流緩衝區讀取資料。`xStreamBufferReceiveFromISR()` 用於在插斷服務常式 (ISR) 中從串流緩衝區讀取資料。

`xStreamBufferReceive()` 允許指定封鎖時間。使用非零的封鎖時間呼叫 `xStreamBufferReceive()` 以從串流緩衝區讀取，並且緩衝區為空白時，任務便會置放到封鎖狀態，直到串流緩衝區中有指定數量的資料可用，或是超過封鎖時間。

解除封鎖任務前串流緩衝區中必須具備的資料數量稱為串流緩衝區的觸發層級。觸發層級為 10 的封鎖任務會在至少有 10 個位元組寫入緩衝區，或是超過任務的封鎖時間時解除封鎖。若在到達觸發層級前超過讀取

中任務的封鎖時間，則任務便會接收到任何寫入該緩衝區的資料。任務的觸發層級必須設為介於 1 和串流緩衝區大小之間的值。串流緩衝區的觸發層級會在呼叫 `xStreamBufferCreate()` 時設定。您可透過呼叫 `xStreamBufferSetTriggerLevel()` 來變更該層級。

`sbRECEIVE_COMPLETED()` 和 `sbRECEIVE_COMPLETED_FROM_ISR()` 是在從串流緩衝區讀取資料時呼叫的巨集（由 FreeRTOS API 交互呼叫）。巨集會檢查串流緩衝區上是否有正在等待可用空間的遭封鎖任務；若有的話，便會將任務從封鎖狀態中移除。您可以透過在 [FreeRTOSConfig.h \(p. 8\)](#) 中提供替代實作，來變更 `sbRECEIVE_COMPLETED()` 的預設行為。

訊息緩衝區

訊息緩衝區可讓來自插斷服務常式的可變長度離散訊息傳遞至任務，或是從其中一個任務傳遞至另一個任務。例如，長度為 10、20 和 123 位元組的訊息全部都可寫入相同的訊息緩衝區，並從從該緩衝區讀取。10 位元組的訊息只能以 10 位元組訊息的方式讀取，而無法以個別位元組進行讀取。訊息緩衝區是建置在串流緩衝區實作上。您可以透過在專案中加入 `stream_buffer.c` 原始檔案，來啟用訊息緩衝區功能。

訊息緩衝區假設只有一個寫入緩衝區的任務或插斷（寫入者），並且只有一個從緩衝區讀取的任務或插斷（讀取者）。寫入者和讀取者為不同的任務或插斷服務常式不會影響安全，但擁有多個寫入者或讀取者則不安全。

訊息緩衝區實作會使用直達任務通知。因此，呼叫將呼叫端任務置放到封鎖狀態的串流緩衝區 API 可變更任務的通知狀態及值。

為啟用訊息緩衝區來處理不同大小的訊息，每個訊息的長度都會在寫入訊息本身之前寫入訊息緩衝區。長度會存放在類型為 `size_t` 的變數中，通常在 32 位元組架構上的大小為 4 位元組。因此，寫入 10 位元組的訊息到訊息緩衝區，實際使用的緩衝區空間為 14 位元組。同樣的，寫入 100 位元組的訊息到訊息緩衝區，實際使用的緩衝區空間為 104 位元組。

傳送資料

`xMessageBufferSend()` 用於將資料從任務傳送至訊息緩衝區。`xMessageBufferSendFromISR()` 用於從插斷服務常式（ISR）將資料傳送到訊息緩衝區。

`xMessageBufferSend()` 允許指定封鎖時間。使用非零的封鎖時間呼叫 `xMessageBufferSend()` 以寫入訊息緩衝區，並且緩衝區已填滿時，任務便會置放到封鎖狀態，直到訊息緩衝區中有可用的空間或超過封鎖時間。

`sbSEND_COMPLETED()` 和 `sbSEND_COMPLETED_FROM_ISR()` 是當資料寫入串流緩衝區時，呼叫的巨集（由 FreeRTOS API 交互呼叫）。它會接收一個參數，該參數為已更新串流緩衝區的控點。這兩個巨集都會檢查串流緩衝區上是否有正在等待資料的遭封鎖任務；若有的話，它們便會將任務從封鎖狀態中移除。

您可以透過在 [FreeRTOSConfig.h \(p. 8\)](#) 中提供自己的 `sbSEND_COMPLETED()` 實作，來變更此預設行為。這在使用串流緩衝區來在多核心處理器的核心之間傳遞資料時很有用。在這種案例下，可實作 `sbSEND_COMPLETED()` 來在另一個 CPU 核心內產生插斷，然後插斷的服務常式便能使用 `xStreamBufferSendCompletedFromISR()` API 來檢查正在等待資料的任務，並視需要解除封鎖。

接收資料

`xMessageBufferReceive()` 是用來從任務中的訊息緩衝區讀取資料。`xMessageBufferReceiveFromISR()` 用於在插斷服務常式（ISR）中從訊息緩衝區讀取資料。`xMessageBufferReceive()` 允許指定封鎖時間。使用非零的封鎖時間呼叫 `xMessageBufferReceive()` 以從訊息緩衝區讀取，並且緩衝區為空白時，任務便會置放到封鎖狀態，直到有可用的資料或超過封鎖時間。

`sbRECEIVE_COMPLETED()` 和 `sbRECEIVE_COMPLETED_FROM_ISR()` 是在從串流緩衝區讀取資料時呼叫的巨集（由 FreeRTOS API 交互呼叫）。巨集會檢查串流緩衝區上是否有正在等待可用空間的遭封鎖任務；若有的話，便會將任務從封鎖狀態中移除。您可以透過在 [FreeRTOSConfig.h \(p. 8\)](#) 中提供替代實作，來變更 `sbRECEIVE_COMPLETED()` 的預設行為。

軟體計時器

軟體計時器允許在未來指定的時間執行函數。由計時器執行的函數稱為計時器的回撥函數。啟動的計時器和所執行回撥函數之間的時間稱為計時器的期間。核心提供高效率的軟體計時器實作，因為：FreeRTOS

- 它不會從插斷內容中執行計時器回撥函數。
- 除非計時器確實已過期，否則它便不會使用任何處理時間。
- 它不會將任何處理額外負荷新增到刻度插斷。
- 它不會在停用插斷時查核任何連結清單結構。

低電力支援

如同大多數嵌入式作業系統，FreeRTOS 核心使用硬體計時器來定期執行刻度中斷，其可用來測量時間。一般硬體計時器實作的省電受限於必須定期離開，然後再重新進入低電力狀態來處理刻度插斷。若刻度插斷的頻率過高，每一刻度進入及離開低電力狀態所使用的能源及時間，便會超過除最輕度省電模式之外所有模式潛在可節省的電力。

為了因應此限制，FreeRTOS 包含適用於低功耗應用程式的無刻度計時器模式。無刻度圖示模式會在時段停止定期刻度中斷（在無可執行的應用程式任務期間），然後在重新開始刻度中斷時，對 RTOS 刻度計數值進行修正。FreeRTOS 停止刻度插斷可讓微控制器保持在深度省電狀態，直到發生插斷或 RTOS 核心將任務轉換到準備就緒狀態時為止。

核心組態

您可以使用 FreeRTOS 標頭檔案設定特定主機板和應用程式的 FreeRTOSConfig.h 核心。每個建置在核心上的應用程式在其預處理器中都必須有一個 FreeRTOSConfig.h 標頭檔案包含路徑。FreeRTOSConfig.h 是應用程式特定，而且應該放置在應用程式下方，而非其中一個 FreeRTOS 核心來源碼資料夾。

FreeRTOS 示範和測試應用程式的 FreeRTOSConfig.h 檔案位於 `freertos/vendors/vendor/boards/board/aws_demos/config_files/FreeRTOSConfig.h` 和 `freertos/vendors/vendor/boards/board/aws_tests/config_files/FreeRTOSConfig.h`。

如需在 FreeRTOSConfig.h 中指定的可用組態參數清單，請查看 [FreeRTOS.org](https://www.FreeRTOS.org)。

FreeRTOS 主控台

在 [主控台FreeRTOS](#) 中，您可以設定和下載套件，其中包含為您的微控制器類型裝置撰寫應用程式時所需要的一切功能：

- 核心。FreeRTOS
- FreeRTOS 程式庫。
- 平台支援程式庫。
- 硬體驅動程式。

您可以使用預先定義的組態下載套件，或是透過選取您的硬體平台及應用程式所需的程式庫，來建立自己的組態。這些組態會儲存在 AWS 中，可隨時下載。

預先定義 FreeRTOS 組態

預先定義組態可讓您快速開始使用支援的使用案例，而無須思考必要的程式庫為何。若要使用預先定義組態，請瀏覽到 [FreeRTOS 主控台](#)，找到您希望使用的組態，然後選擇 Download (下載)。

若您希望變更 FreeRTOS 版本、硬體平台或是組態的程式庫，您也可以自訂預先定義組態。自訂預先定義組態會建立新的自訂組態，而不會覆寫 FreeRTOS 主控台中的預先定義組態。

從預先定義組態建立自訂組態

1. [瀏覽至 FreeRTOS 主控台](#)。
2. 在導覽窗格中，選擇 Software (軟體)。
3. 在 FreeRTOS Device Software (RTOS 裝置軟體) 下方，選擇 Configure download (設定下載)。
4. 選擇您希望自訂預先定義組態旁邊的省略符號 (...)，然後選擇 Customize (自訂)。
5. 在 Configure FreeRTOS Software (設定 RTOS 軟體) 頁面上，選擇 FreeRTOS 版本、硬體平台及程式庫，並給予新組態一個名稱及描述。
6. 在頁面底部，選擇 Create and download (建立和下載)。

自訂 FreeRTOS 組態

自訂組態可讓您指定您的硬體平台、整合式開發環境 (IDE)、編譯器和您所需要的 RTOS 程式庫。這可在您的裝置上節省更多應用程式程式碼的空間。

建立自訂組態

1. [瀏覽至 FreeRTOS 主控台](#)，然後選擇 Create new (建立新項目)。
2. 選取您希望使用的 FreeRTOS 版本。預設會使用最新版本。
3. 在 New Software Configuration (新增軟體組態) 頁面上，選擇 Select a hardware platform (選取硬體平台)，然後選擇其中一個預先符合資格的平台。
4. 選擇您希望使用的 IDE 及編譯器。
5. 對於所需的 FreeRTOS 程式庫，請選擇 Add Library (新增程式庫)。若您選擇的程式庫需要其他程式庫，便會為您一起新增。若您希望選擇多個程式庫，請選擇 Add another library (新增另一個程式庫)。

6. 在 Demo Projects (示範專案) 部分，啟用其中一個示範專案。這會在專案檔案中啟用示範。
7. 在 Name required (必要名稱) 中，輸入您自訂組態的名稱。

Note

請不要在您的自訂組態名稱中使用任何個人識別資訊。

8. 在 Description (描述) 部分，輸入對自訂組態的描述。
9. 在頁面底部，選擇 Create and download (建立和下載)。

編輯自訂組態

1. 瀏覽至 [FreeRTOS 主控台](#)。
2. 在導覽窗格中，選擇 Software (軟體)。
3. 在 FreeRTOS Device Software (RTOS 裝置軟體) 下方，選擇 Configure download (設定下載)。
4. 選擇您希望編輯組態旁邊的省略符號 (...), 然後選擇 Edit (編輯)。
5. 在 Configure FreeRTOS Software (設定 RTOS 軟體) 頁面上，您可以變更您組態的 FreeRTOS 版本、硬體平台、程式庫及描述。
6. 在頁面底部，選擇 Save and download (儲存和下載)。

刪除自訂組態

1. 瀏覽至 [FreeRTOS 主控台](#)。
2. 在導覽窗格中，選擇 Software (軟體)。
3. 在 FreeRTOS Device Software (RTOS 裝置軟體) 下方，選擇 Configure download (設定下載)。
4. 選擇您希望刪除組態旁邊的省略符號 (...), 然後選擇 Delete (刪除)。

快速連線工作流程

FreeRTOS 主控台還包括快速連線工作流程選項，適用於具有預先定義組態之所有電路板。此快速連線工作流程可協助您設定和執行適用於 AWS IoT 和 AWS IoT Greengrass 的 FreeRTOS 示範應用程式。若要開始使用，請選擇 Predefined configurations (預先定義的組態) 索引標籤、找到您的主機板、選擇 Quick connect (快速連線)，然後依照快速連線工作流程的步驟操作。

標記組態

您可以在建立或編輯主控台的組態時，將標籤套用至 FreeRTOS 組態。若要將標籤套用至組態，請前往主控台。在 Tags (標籤) 中輸入標籤的名稱和值。

您可以使用標籤來管理 IAM 政策組態的存取許可。如需相關資訊，請參閱「[搭配 IAM 政策使用標籤 \(p. 10\)](#)」。

如需使用標籤來管理 AWS IoT 資源的詳細資訊，請前往 [開發人員指南](#)中的使用標籤與 政策AWS IoT。

搭配 IAM 政策使用標籤

您可以使用 FreeRTOS 主控台，在您用於操作的 IAM 政策中套用以標籤為基礎的資源層級許可。這可讓您更有效地控制使用者可以建立、修改或使用哪些組態。如需使用 AWS IoT 的標記和 IAM 政策的詳細資訊，請前往 [開發人員指南](#)中的使用標籤搭配 IAM 政策AWS IoT。

在 IAM 政策定義中使用 Condition 元素 (也稱為 Condition 區塊) , 以及以下條件內容金鑰和值 , 來根據資源標籤控制使用者存取 (許可) :

- 使用 aws:ResourceTag/*tag-key: tag-value* 以允許或拒絕 FreeRTOS 組態上具有特定標籤的使用者動作。
- 在建立或修改 FreeRTOS 主控台組態時 , 使用 aws:RequestTag/*tag-key: tag-value* 以要求使用 (或不使用) 特定標籤。
- 在建立或修改 FreeRTOS 主控台組態時 , 使用 aws:TagKeys: [*tag-key, ...*] 以要求使用 (或不使用) 特定標籤金鑰組。

如需詳細資訊 , 請前往 [AWS Identity and Access Management 使用者指南](#) 中的使用標籤控制存取。如需 IAM 中 JSON 政策的元素、變數和評估邏輯之詳細語法、描述和範例 , 請參閱 [IAM JSON 政策參考](#)。

以下範例政策會套用兩個以標籤為基礎的限制。受到此政策限制的 IAM 使用者 :

- 無法將標籤 env=prod 提供給資源 (在範例中 , 請參閱此行 "aws:RequestTag/env" : "prod")
- 無法修改或存取包含現有標籤 env=prod 的資源 (在範例中 , 請參閱此行 "aws:ResourceTag/env" : "prod")。

```
{  
    "Version" : "2012-10-17",  
    "Statement" : [  
        {  
            "Effect" : "Deny",  
            "Action" : "freertos:*",  
            "Resource" : "*",  
            "Condition" : {  
                "StringEquals" : {  
                    "aws:RequestTag/env" : "prod"  
                }  
            }  
        },  
        {  
            "Effect" : "Deny",  
            "Action" : "freertos:*",  
            "Resource" : "*",  
            "Condition" : {  
                "StringEquals" : {  
                    "aws:ResourceTag/env" : "prod"  
                }  
            }  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "iot:*"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

您也可以透過將其包含在清單中 , 為特定標籤金鑰指定多個標籤值 , 如下所示 :

```
{  
    ...  
    "StringEquals" : {  
        "aws:ResourceTag/env" : [ "dev", "test" ]  
    }  
}
```

}

Note

如果您允許或拒絕使用者根據標籤存取資源，請務必考慮明確拒絕使用者將這些標籤新增至相同資源或從中移除的能力。否則，使用者可能透過修改標籤來避開您的限制，並取得資源的存取。

AWS IoT 適用於 Embedded 的 裝置 SDK C

Note

此軟體開發套件適合經驗豐富的嵌入式軟體開發人員使用。

(C-SDK) 是 MIT 開放原始碼授權下的 C 來源檔案集合，可用於嵌入式應用程式，以將 適用於 Embedded C 的 AWS IoT 裝置 SDK 裝置安全地連接到 IoT。AWS IoT Core 它包含 MQTT 使用者端、HTTP client、JSON Parser 和 AWS IoT Device Shadow、AWS IoT 任務和 AWS IoT Device Defender 程式庫。此 開發套件是以來源形式分發，連同應用程式程式碼、其他程式庫和您選擇的作業系統（ OS ）一起建置到使用者程式中。

如需機群 Provisioning，請使用 v4_beta_deprecated 的 適用於 Embedded C 的 AWS IoT 裝置 SDK 版本，網址為 https://github.com/aws/aws-iot-device-sdk-embedded-C/tree/v4_beta_deprecated 如需詳細資訊，請查看此分支中的 README。

適用於 Embedded C 的 AWS IoT 裝置 SDK 通常會針對需要最佳化 C 語言執行階段的資源限制裝置。您可以在任何作業系統上使用軟體開發套件，並將其裝載在任何處理器類型上（例如 MCUs 和 MPUs ）。不過，如果您的裝置記憶體和處理資源足以提供，我們建議您使用 [裝置AWS IoT SDKs 的較高順序之一](#)。

如需詳細資訊，請參閱下列內容：

- AWS IoT 上的 適用於 Embedded 的 裝置 SDK GitHub
- AWS IoT 適用於 Embedded 的 裝置 SDK C 讀我檔案
- AWS IoT 適用於 Embedded 的 裝置 SDK C 範例

FreeRTOS 入門

此 FreeRTOS 入門教學課程會示範如何在主機機器上下載和設定 FreeRTOS，然後在符合 [資格的微控制器主機板](#) 上編譯並執行簡單的示範應用程式。

在本教學課程中，我們假設您熟悉 AWS IoT 和 AWS IoT 主控台。如果還不熟悉，在繼續之前，我們建議您先完成 [AWS IoT 入門](#) 教學課程。

主題：

- [FreeRTOS 示範應用程式 \(p. 14\)](#)
- [首要步驟 \(p. 14\)](#)
- [故障診斷入門 \(p. 19\)](#)
- [使用 CMake 配 FreeRTOS \(p. 20\)](#)
- [開發人員模式金鑰佈建 \(p. 26\)](#)
- [主機板特定的入門指南 \(p. 28\)](#)

FreeRTOS 示範應用程式

本教學課程中的示範應用程式是 coreMQTT 檔案中所定義的 `/demos/coreMQTT/mqtt_demo-mutual_auth.c` 交互身份驗證示範。它使用 [coreMQTT 程式庫 \(p. 195\)](#) 連接到 AWS 雲端，然後定期發布訊息到由 MQTT 中介裝置 AWS IoT 代管的 MQTT 主題。

一次只能執行一個 FreeRTOS 示範應用程式。當您建置 FreeRTOS 示範專案時，在 `freertos/vendors/vendor/boards/board/aws_demos/config_files/aws_demo_config.h` 標頭檔案中啟用的第一個示範就是執行的應用程式。在本教學課程中，您不需要啟用或停用任何示範。交互身份驗證示範預設為可使用。coreMQTT

如需包含 FreeRTOS 的示範應用程式詳細資訊，請參閱 [FreeRTOS 示範 \(p. 210\)](#)。

首要步驟

若要開始搭配使用 FreeRTOS 與 AWS IoT，您需要一個 AWS 帳戶，這是對 AWS IoT 和 FreeRTOS 具有存取許可的 IAM 使用者。您也需要下載 FreeRTOS 並將您主機板的 FreeRTOS 示範專案設定為與 AWS IoT 搭配使用。以下章節將逐步引導您完成這些要求。

Note

如果您使用的是 Espressif ESP32-DevKitC、ESP-WROVER-KIT 或 ESP32-WROOM-32SE，請略過這些步驟並前往 [Espressif ESP32-DevKitC 和 ESP-WROVER-KIT 入門 \(p. 42\)](#)。

如果您使用的是 Nordic nRF52840-DK，請略過這些步驟並前往 [Nordic nRF52840-DK 入門 \(p. 91\)](#)。

1. 設定 AWS 帳戶和許可 (p. 15)

完成 [設定 AWS 帳戶和許可 \(p. 15\)](#) 中的指示後，您就可以遵循 主控台中的 [Quick Connect \(快速連接\) FreeRTOS 工作流程](#)，快速將主機板連接到 雲端。AWS 如果您依照 Quick Connect (快速連接) 工作流程，您不需要完成此清單中的其餘步驟。請注意，您目前無法在 FreeRTOS 主控台上取得以下主機板的 FreeRTOS 組態：

- Cypress CYW943907AEVAL1F 開發套件

- Cypress CYW954907AEVAL1F 開發套件
- 2. 向 AWS IoT 註冊您的 MCU 電路板 (p. 15)
- 3. 下載 FreeRTOS (p. 17)
- 4. 設定 FreeRTOS 示範 (p. 18)

設定 AWS 帳戶和許可

若要建立 AWS 帳戶，請參閱[建立或啟用 AWS 帳戶](#)。

若要新增 IAM 使用者至您的 AWS 帳戶，請參閱[IAM 使用者指南](#)。若要授予 IAM 使用者帳戶存取 AWS IoT 和 FreeRTOS 的許可，請將以下 IAM 政策連接到您的 IAM 使用者帳戶：

- AmazonFreeRTOSFullAccess
- AWSIoTFullAccess

將 AmazonFreeRTOSFullAccess 政策連接到您的 IAM 使用者

1. 瀏覽至[IAM 主控台](#)，並從導覽窗格中，選擇 Users (使用者)。
2. 在搜尋文字方塊中，輸入您的使用者名稱，然後從清單中選擇它。
3. 選擇 Add permissions (新增許可)。
4. 選擇 Attach existing policies directly (直接連接現有政策)。
5. 在搜索框中，輸入 **AmazonFreeRTOSFullAccess**，從列表中選擇，然後選擇 下一步：回顧。
6. 選擇 Add permissions (新增許可)。

將 AWSIoTFullAccess 政策連接到您的 IAM 使用者

1. 瀏覽至[IAM 主控台](#)，並從導覽窗格中，選擇 Users (使用者)。
2. 在搜尋文字方塊中，輸入您的使用者名稱，然後從清單中選擇它。
3. 選擇 Add permissions (新增許可)。
4. 選擇 Attach existing policies directly (直接連接現有政策)。
5. 在搜索框中，輸入 **AWSIoTFullAccess**，從列表中選擇，然後選擇 下一步：回顧。
6. 選擇 Add permissions (新增許可)。

如需 IAM 和使用者帳戶的詳細資訊，請參閱[IAM 使用者指南](#)。

如需政策的詳細資訊，請參閱[IAM 許可與政策](#)。

設置好 AWS 帳戶和權限，您可以繼續[向 AWS IoT 註冊您的 MCU 電路板 \(p. 15\)](#) 或 快速連接 工作流程 [FreeRTOS 控制檯](#)。

向 AWS IoT 註冊您的 MCU 電路板

您的開發板必須註冊 AWS IoT 才能與 AWS 雲端進行通訊。若要向 AWS IoT 註冊您的電路板，您需要以下項目：

AWS IoT 政策

AWS IoT 政策可授予裝置存取 AWS IoT 資源的許可。這是存放在 AWS 雲端。

AWS IoT 實物

AWS IoT 實物可讓您在 AWS IoT 中管理裝置。這是存放在 AWS 雲端。

私密金鑰和 X.509 憑證

私有金鑰和憑證可讓 AWS IoT 對您的裝置進行身份驗證。

如果您使用 快速連接 工作流程 [FreeRTOS 控制檯](#)、政策、AWS IoT 並且會為您創建密鑰和證書。如果您使用的是 Quick Connect (快速連接) 工作流程，您可以忽略以下程序。

若要手動註冊主機板，請依照以下的程序。

建立 AWS IoT 政策

1. 若要建立 IAM 政策，您需要知道您的 AWS 區域和 AWS 帳戶號碼。

若要尋找您的 AWS 帳戶號碼，請開啟 [AWS 管理主控台](#)，尋找並展開右上角您帳戶名稱下的功能表，然後選擇 My Account (我的帳戶)。您的帳戶 ID 會顯示在 Account Settings (帳戶設定) 下方。

若要尋找您 AWS 帳戶的 AWS 區域，請使用 AWS Command Line Interface。若要安裝 AWS CLI 安裝，請依照 [AWS Command Line Interface 使用者指南](#) 中的指示進行。安裝 AWS CLI 之後，開啟命令提示字元視窗，然後輸入下列命令：

```
aws iot describe-endpoint
```

輸出應如下所示：

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

在此範例中，區域為 us-west-2。

2. 瀏覽至 [AWS IoT 主控台](#)。
3. 在導航窗格中，選擇 安全，選擇 政策，然後選擇 創建。
4. 輸入可識別政策的名稱。
5. 在 Add statements (新增陳述式) 區段中，選擇 Advanced mode (進階模式)。將下列 JSON 複製並貼入政策編輯器視窗。Replace (取代) `aws-region` 和 `aws-account` 使用 AWS 區域和帳戶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:"*"  
        }  
    ]  
}
```

]
}

此政策可授予下列許可：

`iot:Connect`

 授予裝置能使用任何用戶端 ID 連線至 AWS IoT 訊息中介裝置的許可。

`iot:Publish`

 授予裝置能發佈任何 MQTT 主題之 MQTT 訊息的許可。

`iot:Subscribe`

 授予裝置能訂閱任何 MQTT 主題篩選條件的許可。

`iot:Receive`

 授予裝置能接收 AWS IoT 訊息中介裝置中任何 MQTT 主題訊息的許可。

6. 選擇 Create (建立)。

要創建 IoT 設備、私鑰和證書

1. 瀏覽至 [AWS IoT 主控台](#)。
2. 在導覽窗格中，選擇 Manage (管理)，然後選擇 Things (實物)。
3. 如果您沒有任何 IoT 在您的帳戶中登記的事項，您還沒有什麼東西 頁面顯示。如果您看到此頁面，請選擇 Register a thing (註冊實物)。否則，請選擇 Create (建立)。
4. 在 Creating AWS IoT things (建立 IoT 物件) 頁面上，選擇 Create a single thing (建立單一物件)。
5. 在 Add your device to the thing registry (將您的裝置新增至物件登錄檔) 頁面中，輸入物件的名稱，然後選擇 Next (下一步)。
6. 在 Add a certificate for your thing (新增物件的憑證) 頁面中，選擇 One-click certificate creation (按一下 建立憑證) 下方的 Create certificate (建立憑證)。
7. 選擇各個項目的 Download (下載) 連結，下載您的私有金鑰和憑證。
8. 選擇 Activate (啟用) 以啟用您的憑證。需先啟用憑證才可開始使用。
9. 選擇 Attach a policy (連接政策)，將政策連接到您的憑證，以授予裝置存取 AWS IoT 操作的許可。
10. 選擇您剛建立的政策，然後選擇 Register thing (註冊實物)。

當您主機板的註冊是使用 AWS IoT，您可以繼續 [下載 FreeRTOS \(p. 17\)](#)。

下載 FreeRTOS

您可以下載 FreeRTOS 從 FreeRTOS 控制檯或從 [FreeRTOS GitHub 存儲庫](#)。

Note

如果您遵循 快速連接 工作流程 [FreeRTOS 控制檯](#)，您可以忽略以下過程。

從 FreeRTOS 主控台下載 FreeRTOS

1. 登入 [FreeRTOS 主控台](#)。
2. 低於 預定義配置，查找 連接到 AWS IoT- **platform**，然後選擇 下載。
3. 解壓縮下載的檔案至一個目錄，並複製該目錄路徑。

Important

- 在本教學課程中，FreeRTOS 下載目錄的路徑會以 **freertos** 表示。

- *freertos* 路徑中的空格字元可能會導致建置失敗。當您複製或拷貝儲存庫時，請確定您建立的路徑不包含空格字元。
- Microsoft Windows 的檔案路徑長度上限為 260 個字元。FreeRTOS 下載目錄路徑過長可能會導致建置失敗。

Note

如果您開始使用CypressCYW954907AEVAL1F或CYW943907AEVAL1F開發工具包,您必須下載FreeRTOS 從 GitHub. 如需說明，請參閱 [README.md](#) 檔案。您目前無法從 FreeRTOS 主控台取得這些主機板的 FreeRTOS 組態。

在您下載 FreeRTOS 後，您可以繼續 [設定 FreeRTOS 示範 \(p. 18\)](#)。

設定 FreeRTOS 示範

您需要在 FreeRTOS 目錄中編輯一些組態檔案，才可以在主機板上編譯和執行任何示範。

如果您在 主控台中遵循 Quick Connect (快速連接) FreeRTOS 工作流程，請依照主控台上工作流程中的組態指示，並忽略這些程序。

設定 AWS IoT 端點

您必須將 AWS IoT 端點提供給 FreeRTOS，以便在您主機板上執行的應用程式將請求傳送至正確的端點。

1. [瀏覽至 AWS IoT 主控台](#)。
2. 在導覽窗格中選擇 Settings (設定)。

您的 AWS IoT 端點會顯示在 Endpoint (端點) 中。它看起來應該會像這樣：*1234567890123-ats.iot.us-east-1.amazonaws.com*。記下此端點。

3. 在導覽窗格中，選擇 Manage (管理)，然後選擇 Things (實物)。

您的裝置必須擁有 AWS IoT 實物名稱。記下此名稱。

4. Open `/demos/include/aws_clientcredential.h`.
5. 指定以下常數的值：

- `#define clientcredentialMQTT_BROKER_ENDPOINT "Your AWS IoT endpoint";`
- `#define clientcredentialIOT_THING_NAME "The AWS IoT thing name of your board"`

設定您的 Wi-Fi

如果您的主機板是透過 Wi-Fi 連線連接到網際網路，您必須提供 Wi-Fi 憑證給 FreeRTOS，以便連線至網路。如果您的主機板不支援 Wi-Fi，您可以略過這些步驟。

1. `demos/include/aws_clientcredential.h`.
2. 指定以下 `#define` 常數的值：
 - `#define clientcredentialWIFI_SSID "The SSID for your Wi-Fi network"`
 - `#define clientcredentialWIFI_PASSWORD "The password for your Wi-Fi network"`
 - `#define clientcredentialWIFI_SECURITY The security type of your Wi-Fi network`

有效安全類型為：

- `eWiFiSecurityOpen` (開放，不具安全性)

- eWiFiSecurityWEP (WEP 安全性)
- eWiFiSecurityWPA (WPA 安全性)
- eWiFiSecurityWPA2 (WPA2 安全性)

格式化您的 AWS IoT 登入資料

FreeRTOS 需要與註冊實物和其許可政策相關聯的 AWS IoT 憑證和私有金鑰，才能成功代表您的裝置與 AWS IoT 通訊。

Note

若要設定您的 AWS IoT 登入資料，您需要擁有註冊裝置時，從 AWS IoT 主控台下載的私有金鑰和憑證。在您將裝置註冊為 AWS IoT 實物後，您可以從 AWS IoT 主控台獲取裝置憑證，但無法取得私有金鑰。

FreeRTOS 是一項 C 語言專案，憑證和私有金鑰必須以特殊格式新增到專案。

1. 在瀏覽器視窗中，開啟 `tools/certificate_configuration/CertificateConfigurator.html`。
2. 在 Certificate PEM file (憑證 PEM 檔案) 下方，選擇您從 AWS IoT 主控台下載的 `ID-certificate.pem.crt`。
3. 在 Private Key PEM file (私有金鑰 PEM 檔案) 下方，選擇您從 AWS IoT 主控台下載的 `ID-private.pem.key`。
4. 選擇 Generate and save aws_clientcredential_keys.h，然後在 `demos/include` 中儲存檔案。這會覆寫資料夾中現有的檔案。

Note

將憑證和私有金鑰硬式編碼，僅作示範用途。生產層級應用程式必須將這些檔案存放在安全的位置。

在設定 FreeRTOS 後，您就可以繼續參考您主機板適用的入門指南來編譯並執行 FreeRTOS 示範。在入門教學課程中所用的示範應用程式是 coreMQTT 交互身份驗證示範，位於 `demos/mqtt/iot_demo_mqtt.c`。

完成 [首要步驟 \(p. 14\)](#) 後，您就可以設定平台的硬體和軟體開發環境，然後在您的主機板上編譯並執行示範。如需主機板特定的指示，請參閱 [主機板特定的入門指南 \(p. 28\)](#)。

故障診斷入門

下列主題可協助您對 FreeRTOS 入門所遇到的問題進行疑難排解：

主題

- [一般入門故障診斷提示 \(p. 19\)](#)
- [安裝終端機模擬器 \(p. 20\)](#)

如需主機板特定的故障診斷，請參閱您主機板適用的 [FreeRTOS 入門 \(p. 14\)](#) 指南。

一般入門故障診斷提示

執行 Hello World 示範專案後，AWS IoT 主控台中不會顯示任何訊息。我要怎麼做？

請嘗試以下做法：

- 開啟終端機視窗以檢視範例的記錄輸出。這可協助您判斷何處發生錯誤。
- 確認您的網路登入資料有效。

執行示範時，在我的終端機中顯示的對數會被截斷。如何增加其長度？

在您正在執行的示範的 configLOGGING_MAX_MESSAGE_LENGTH 檔案中，將 FreeRTOSconfig.h 的值增加到 255：

```
#define configLOGGING_MAX_MESSAGE_LENGTH      255
```

安裝終端機模擬器

終端機模擬器可以協助您診斷問題或驗證裝置程式碼是否正確執行。有各種適用於 Windows、macOS 和 Linux 的終端機模擬器。

您必須將電路板連接到電腦，然後再嘗試建立電路板與終端機模擬器的序列連線。

請使用以下設定來設定終端機模擬器：

終端機設定	數值
傳輸速率	115200
資料	8 位元
同位	無
停止	1 位元
流量控制	無

尋找您的開發板的序列埠

如果您不知道電路板的序列連接埠，則可以從命令列或終端機發出以下其中一個命令，來傳回所有連接至主機電腦之裝置的序列連接埠：

Windows

```
chgport
```

Linux

```
ls /dev/tty*
```

macOS

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

使用 CMake 配 FreeRTOS

您可以使用 CMake 生成項目構建文件 FreeRTOS 應用程序源代碼，以及用於構建和運行源代碼。

您還可以使用IDE編輯、調試、編譯、閃存和運行代碼 FreeRTOS-合格的設備。每個電路板特定的入門指南都包含了設定 IDE 特定平台的說明。如果您更喜歡在沒有IDE的情況下工作,可以使用其他第三方代碼編輯和調試工具來開發和調試您的代碼,然後使用 CMake 以構建和運行應用程序。

以下主板支持 CMake:

- Espressif ESP32-DevKitC
- Espressif ESP-WROVER-KIT
- 英飛龍XMC4800 IoT 連接套件
- Marvell MW320 AWS IoT 入門套件
- Marvell MW322 AWS IoT 入門套件
- Microchip Curiosity PIC32MZEF 套件
- Nordic nRF52840 DK Development kit
- STMicroelectronics STM32L4 視覺套件 IoT 節點
- Texas Instruments CC3220SF-LAUNCHXL
- Microsoft Windows 模擬器

有關使用的更多信息,請參閱以下主題 CMake 配 FreeRTOS.

主題

- [Prerequisites \(p. 21\)](#)
- [使用第三方的程式碼編輯器與除錯工具來開發 FreeRTOS 應用程式 \(p. 22\)](#)
- [建築 FreeRTOS 配 CMake \(p. 22\)](#)

Prerequisites

請確認您的主機機器符合下列先決條件，再繼續操作：

- 您的設備的編譯工具鏈必須支持機器的操作系統。CMake 支持所有版本的Windows, macOS、和Linux 不支援「適用於 Linux 的 Windows 子系統 (WSL)」。使用本機 CMake 在Windows計算機上。
- 您必須有 CMake 版本3.13或更高版本已安裝。

您可以下載 CMake 從 [CMake.org公司](#).

Note

如果您下載了的二進制分發 CMake,請確保添加 CMake 可執行到PATH環境變量中,然後使用 CMake 從命令行。

您還可以下載和安裝 CMake 使用包管理器,例如 [內花](#) 於 macOS, 和 [勺子](#) 或 [巧克力色](#) 在Windows上。

Note

的 CMake 許多Linux發行版的包管理器中提供的包版本已過時。如果您分發的軟件包管理器不提供最新版本的 CMake,您可以嘗試備用包管理器,例如 [linuxbrew](#) 或 [nix](#).

- 您必須有相容的原生建置系統。

CMake 可以將許多原生建置系統當作目標，包括 [GNU Make](#) 或 [Ninja](#)。Make和Ninja均可在Linux上安裝包管理器, macOS 和Windows。如果您在 Windows 上使用 Make，則可從 [Equation](#) 安裝獨立版本，或是安裝 Make 隨附的 [MinGW](#)。

Note

使可執行 MinGW 被稱為 `mingw32-make.exe`,而不是 `make.exe`.

我們建議您使用 Ninja，因為它比 Make 更快，且可為所有桌面作業系統提供原生支援。

使用第三方的程式碼編輯器與除錯工具來開發 FreeRTOS 應用程式

如需開發 FreeRTOS 的應用程式，您可以使用程式碼編輯器和除錯擴充功能，或是第三方的除錯工具。

舉例來說，如果您將 [Visual Studio 程式碼](#) 做為程式碼編輯器，就能將 [Cortex-Debug](#) VS 程式碼擴充功能做為除錯工具進行安裝。完成應用程序開發後，您可以調用 CMake 命令行工具，用於從 VS 代碼內構建項目。有關使用的更多信息 CMake 構建 FreeRTOS 應用程序，請參閱 [建築 FreeRTOS 配 CMake \(p. 22\)](#)。

若要進行除錯，您可提供具備除錯組態的 VS 程式碼，該組態類似如下：

```
"configurations": [
  {
    "name": "Cortex Debug",
    "cwd": "${workspaceRoot}",
    "executable": "./build/st/stm321475_discovery/aws_demos.elf",
    "request": "launch",
    "type": "cortex-debug",
    "serverType": "stutil"
  }
]
```

建築 FreeRTOS 配 CMake

根據預設，CMake 以您的主機作業系統作為目標系統。要將其用於交叉編譯，CMake 需要一個工具鏈文件，該文件用於指定要使用的編譯器。英寸 FreeRTOS，我們提供默認工具鏈文件 [freertos/tools/cmake/toolchains](#)。將此文件提供給 CMake 取決於您是否使用 CMake 命令行界面或 GUI。有關更多詳細資訊，請參閱下面的 [產生建置檔案 \(CMake 命令行工具\) \(p. 22\)](#) 指示。有關交叉編譯的更多信息 CMake，請參閱 [交叉編譯 在官員 CMake 維基](#)。

要構建 CMake-基於項目

1. 運行 CMake 為原生構建系統生成構建文件，如 Make 或 Ninja。

您可以使用 [CMake 命令列工具](#) 或 [CMake GUI](#)，為您的原生建置系統產生建置檔案。

如需有關產生 FreeRTOS 建置檔案的資訊，請參閱 [產生建置檔案 \(CMake 命令行工具\) \(p. 22\)](#) 和 [產生建置檔案 \(CMake GUI\) \(p. 24\)](#)。

2. 叫用原生建置系統將專案製成可執行檔。

如需有關製作 FreeRTOS 建置檔案的資訊，請參閱 [從產生的建置檔案來建置 FreeRTOS \(p. 26\)](#)。

產生建置檔案 (CMake 命令行工具)

您可以使用 CMake 命令行工具(命令)生成構建文件 FreeRTOS。若要產生建置檔案，您需要指定一個目標電路板、編譯器和原始程式碼的位置並建立目錄。

您可將下列選項用於 cmake：

- **-DVENDOR** – 指定目標電路板。
- **-DCOMPILER** – 指定編譯器。
- **-S** – 指定原始程式碼的位置。

- -B – 指定產生的建置檔案的位置。

Note

編譯器必須在系統的 PATH 變數中，否則您必須指定編譯器的位置。

例如，如果廠商是 Texas Instruments，電路板是 CC3220 Launchpad，而編譯器是 GCC for ARM，您可以發出以下命令，從目前的目錄將下列原始檔案建置到名為 *build-directory* 的目錄中：

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

Note

如果您正在使用 Windows，則必須指定本地構建系統，因為 CMake 默認情況下使用 Visual Studio。例如：

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

或者：

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

規則表達式 `${VENDOR}.*` 和 `${BOARD}.*` 用來搜尋相符的電路板，因此，您在 VENDOR 和 BOARD 選項中不需要使用廠商和電路板的全名。部分名稱假如只有單一相符項，亦可使用。例如，以下命令從相同的來源產生相同的建置檔案：

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

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

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

您可以使用 `CMAKE_TOOLCHAIN_FILE` 選項，如果您希望使用不在默認目錄中的工具鏈文件 `cmake/toolchains`。例如：

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

如果工具鏈文件沒有為您的編譯器使用絕對路徑，並且您沒有將編譯器添加到 PATH 環境變量，CMake 可能無法找到它。確保 CMake 查找您的工具鏈文件，您可以使用 `AFR_TOOLCHAIN_PATH` 選項。此選項搜索指定的工具鏈目錄路徑和工具鏈的子文件夾，路徑位於 bin。例如：

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

要啓用調試，請設置 `CMAKE_BUILD_TYPE` 至 `debug`。啓用這個選項後，CMake 將調試標記添加到編譯選項，並構建 FreeRTOS 帶調試符號。

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

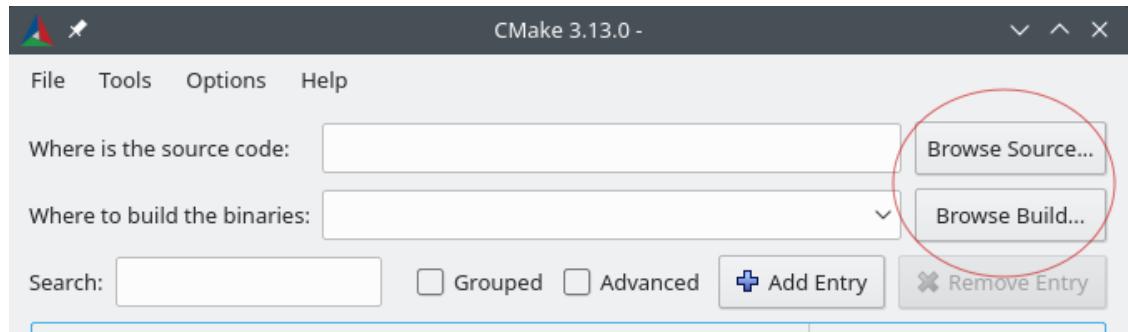
您也可以將 CMAKE_BUILD_TYPE 設為 release，以將最佳化旗標新增到編譯選項。

產生建置檔案 (CMake GUI)

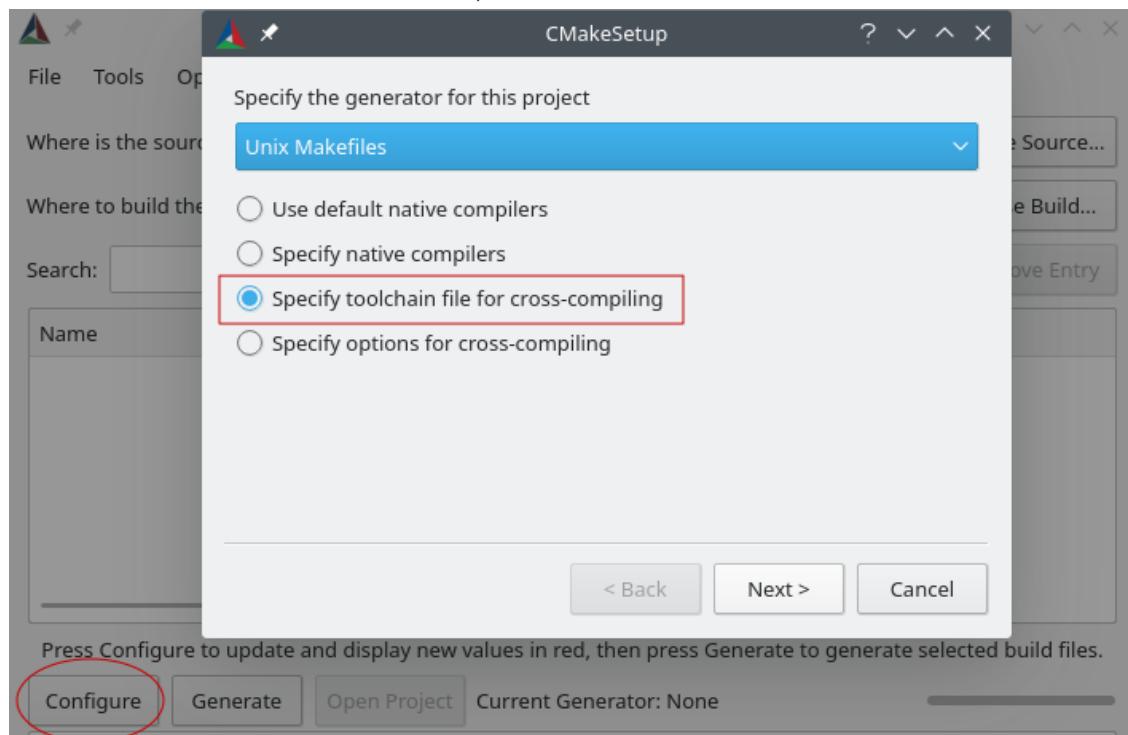
您可以使用 CMake 要生成的 GUI FreeRTOS 構建文件。

要使用生成構建文件 CMake 圖形用戶界面

1. 從命令列發出 cmake-gui，以啟動 GUI。
2. 選擇 Browse Source (瀏覽來源) 並指定來源輸入，然後選擇 Browse Build (瀏覽建置) 並指定建置輸出。



3. 選擇 Configure (設定)，然後在 Specify the build generator for this project (指定此專案的建置產生器) 下方，尋找和選擇你要用來建置所產生的建置檔案的建置系統。如果您沒有看到彈出式視窗，則您可能正在重複使用現有的建置目錄。在這種情況下，刪除 CMake 通過選擇 刪除緩存 從 文件 菜單。



4. 選擇 Specify toolchain file for cross-compiling (指定跨編譯的工具鏈檔案)，然後選擇 Next (下一步)。
5. 選擇工具鏈檔案 (例如，*freertos/tools/cmake/toolchains/arm-ti.cmake*)，然後選擇 Finish (完成)。

FreeRTOS 是預設組態是範本電路板，這不提供任何可攜式層目標。結果，將出現一個窗口，其中包含消息 Error in configuration process.

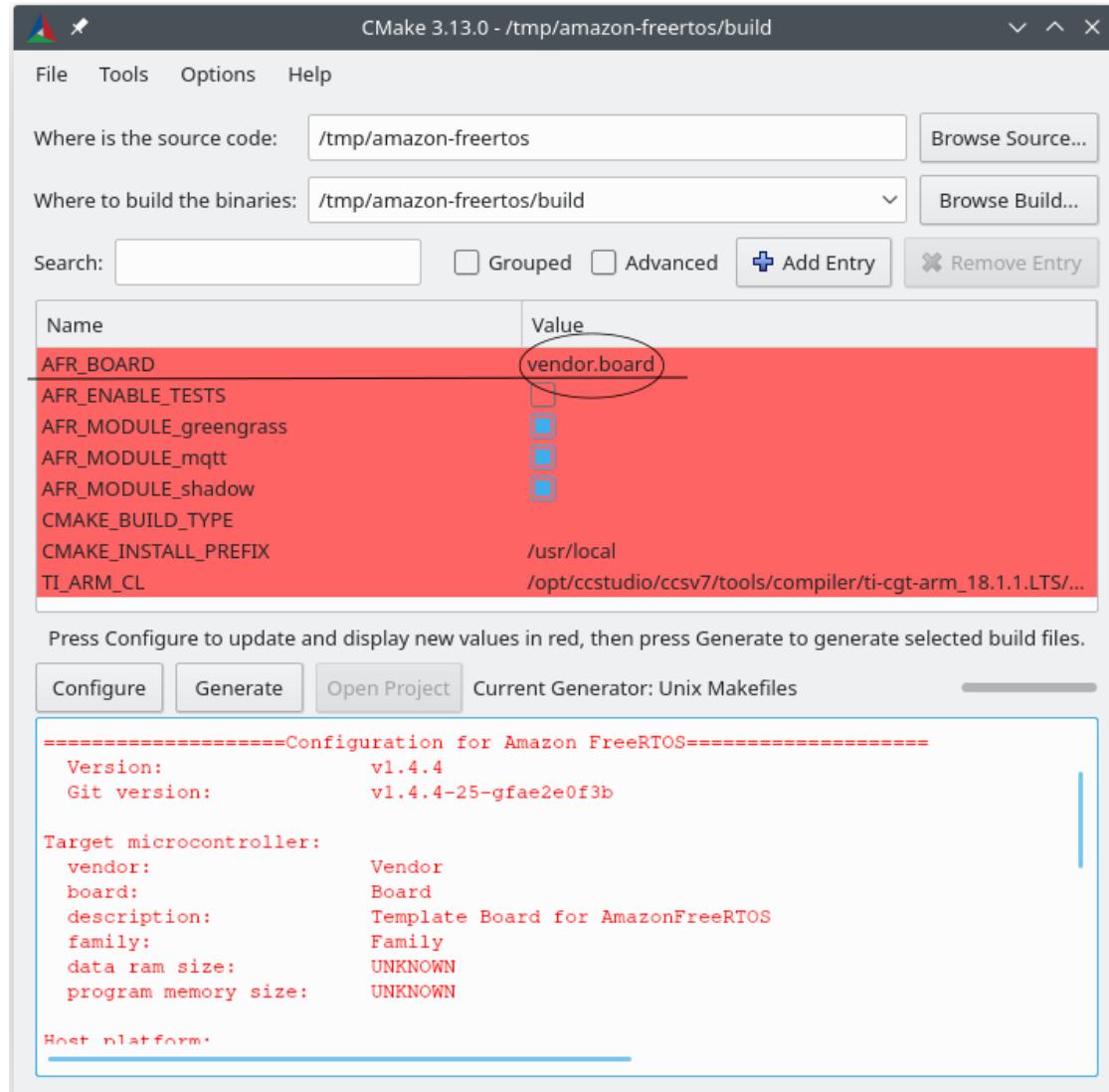
Note

如果您看到以下錯誤：

```
CMake Error at tools/cmake/toolchains/find_compiler.cmake:23 (message):  
Compiler not found, you can specify search path with AFR_TOOLCHAIN_PATH.
```

表示該編譯器不在您的 PATH 環境變數中。您可以設置 AFR_TOOLCHAIN_PATH 的變量，以告知 CMake 其中安裝了編譯器。如果您沒有看到 AFR_TOOLCHAIN_PATH 變數，請選擇 Add Entry (新增項目)。在彈出窗口中，名稱類型 **AFR_TOOLCHAIN_PATH**。低於 編譯器路徑 鍵入編譯器的路徑。例如，C:/toolchains/arm-none-eabi-gcc。

6. GUI 現在應該看起來像這樣：



選擇 AFR_BOARD，選擇您的電路板，然後再次選擇 Configure (設定)。

7. 選擇生成 CMake 生成構建系統文件(例如,makefiles或ninja文件),這些文件將顯示在您在第一步中指定的構建目錄中。請按照下一節的指示來產生二進位影像。

從產生的建置檔案來建置 FreeRTOS

使用原生建置系統進行建置

您可以從輸出二進位檔目錄中呼叫建置系統命令，以原生建置系統來建置 FreeRTOS。

例如，如果您的建置檔案輸出目錄是 `<build_dir>`，而且您使用 Make 作為原生建置系統，請執行下列命令：

```
cd <build_dir>
make -j4
```

建築使用 CMake

您還可以使用 CMake 用於構建的命令行工具 FreeRTOS. CMake 提供用於調用本地構建系統的抽象層。例如：

```
cmake --build build_dir
```

以下是一些其他常見的 CMake 命令行工具的構建模式：

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

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

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

有關 CMake 構建模式，請參閱 [C製作文檔](#)。

開發人員模式金鑰佈建

Introduction

本節討論在 IoT 裝置上取得信任的 X.509 client 憑證以進行實驗室測試的兩個選項。視裝置的功能而定，可能支援或不支援各種佈建相關作業，包括產生內建 ECDSA 金鑰、匯入私有金鑰及註冊 X.509 憑證。此外，不同的使用案例需要不同等級的金鑰保護，從內建快閃記憶體儲存到使用專用加密硬體。本節提供用在您裝置加密功能中的邏輯。

選項 #1：從 AWS IoT 匯入私有金鑰

基於實驗室測試目的，如果您的裝置允許匯入私有金鑰，請遵循 [設定 FreeRTOS 示範 \(p. 18\)](#) 中的說明。

選項 #2：產生內建私有金鑰

如果您的裝置有安全元素，或者您偏好自行產生裝置金鑰對和憑證，請遵循此處的說明。

初始組態

首先，執行 [設定 FreeRTOS 示範 \(p. 18\)](#) 中的步驟，但略過最後一個步驟（亦即不要進行格式化您的 AWS IoT 登入資料）。最終結果應是 `demos/include/aws_clientcredential.h` 檔案以您的設定更新，但 `demos/include/aws_clientcredential_keys.h` 文件則否。

示範專案組態

如 [coreMQTT 主機板特定的入門指南 \(p. 28\)](#) 中您的主機板指南所述，打開 交互身份驗證示範。在專案中，開啟檔案 `aws_dev_mode_key_provisioning.c`，並將預設為零的 `keyprovisioningFORCE_GENERATE_NEW_KEY_PAIR` 定義變更為一：

```
#define keyprovisioningFORCE_GENERATE_NEW_KEY_PAIR 1
```

然後建立並執行示範專案，再繼續下一個步驟。

擷取公有金鑰

由於裝置尚未使用私有金鑰和使用者憑證來配置，因此示範將無法驗證 AWS IoT。不過，coreMQTT 交互身份驗證示範從執行開發人員模式金鑰設定開始，導致建立私有金鑰（如果私密金鑰尚未存在）。您應該會看到類似下列的序列主控台輸出開頭。

```
7 910 [IP-task] Device public key, 91 bytes:  
3059 3013 0607 2a86 48ce 3d02 0106 082a  
8648 ce3d 0301 0703 4200 04cd 6569 ceb8  
1bb9 1e72 339f e8cf 60ef 0f9f b473 33ac  
6f19 1813 6999 3fa0 c293 5fae 08f1 1ad0  
41b7 345c e746 1046 228e 5a5f d787 d571  
dcbb 4e8d 75b3 2586 e2cc 0c
```

將六行金鑰位元組複製到名為 `DevicePublicKeyAsciiHex.txt` 的檔案。然後，使用命令行工具「`xxd`」將十六進位位元組剖析為二進位：

```
xxd -r -ps DevicePublicKeyAsciiHex.txt DevicePublicKeyDer.bin
```

使用“`openssl`”將二進位編碼的 (DER) 裝置公有金鑰格式化為 PEM：

```
openssl ec -inform der -in DevicePublicKeyDer.bin -pubin -pubout -outform pem -out  
DevicePublicKey.pem
```

不要忘記停用您在前文中啟用的暫時金鑰產生設定。否則，此裝置會建立另一個金鑰對，而您則必須重複上述步驟：

```
#define keyprovisioningFORCE_GENERATE_NEW_KEY_PAIR 0
```

公有金鑰基礎設施設定

遵循 [登記 CA 憑證](#) 中的說明，建立您裝置實驗室憑證的憑證階層。請先停止，再依「使用 CA �凭證建立裝置憑證」一節中所述順序執行作業。

在本例中，裝置不會簽署憑證請求（即憑證服務請求或 CSR），因為為減少 ROM 大小，已從 FreeRTOS 示範專案中排除建立及簽署 CSR 所需的 X.509 編碼邏輯。而會基於實驗室測試目的，改在您的工作站上建立私有金鑰，用它簽署 CSR。

```
openssl genrsa -out tempCsrSigner.key 2048  
openssl req -new -key tempCsrSigner.key -out deviceCert.csr
```

建立憑證授權單位並向 AWS IoT 登記之後，請根據上一步驟中簽署的裝置 CSR，使用下列命令核發用戶端憑證：

```
openssl x509 -req -in deviceCert.csr -CA rootCA.pem -CAkey rootCA.key -CAcreateserial -  
out deviceCert.pem -days 500 -sha256 -force_pubkey DevicePublicKey.pem
```

即使已使用暫時私有金鑰簽署 CSR，所核發的憑證也只能搭配實際的裝置私有金鑰使用。如果您將 CSR 簽署者金鑰存放在不同的硬體中，並設定憑證授權單位僅針對已由該特定金鑰簽署的請求核發憑證，則可在生產環境中使用相同的機制。該金鑰應該也由指定管理員控制。

憑證匯入

核發憑證後，下一步就是將其匯入裝置。您還需要匯入憑證授權單位 (CA) 憑證，因為必須如此，才能在使用 JITP 時成功完成第一次的 AWS IoT 驗證。在專案的 `aws_clientcredential_keys.h` 檔案中，將 `keyCLIENT_CERTIFICATE_PEM` 巨集設定為 `deviceCert.pem` 的單位，並將 `keyJITR_DEVICE_CERTIFICATE_AUTHORITY_PEM` 巨集設定為 `rootCA.pem` 的項目。

裝置授權

將 `deviceCert.pem` 汇入至 AWS IoT 登記，如[使用自有憑證](#)中所述。您必須建立新的 AWS IoT 物件，將 PENDING 憑證和政策連接到您的物件，然後將憑證標記為 ACTIVE (作用中)。所有這些步驟都可在 AWS IoT 主控台中手動執行。

一旦新的使用者端憑證為 ACTIVE 並關聯至物件和政策，請再次執行 coreMQTT 交互身份驗證示範。現在，AWS IoT IoT MQTT 中介裝置的連線成功建立。

主機板特定的入門指南

完成[首要步驟 \(p. 14\)](#)後，請參閱主機板指南中對於 FreeRTOS 入門的主機板特定指示：

- Cypress CYW943907AEVAL1F 開發套件入門 (p. 29)
- Cypress CYW954907AEVAL1F 開發套件入門 (p. 31)
- Cypress CY8CKIT-064S0S2-4343W 套件入門 (p. 34)
- Infineon XMC4800 Connectivity Kit 入門 IoT (p. 61)
- 入門套件入門 MW32xAWS IoT (p. 69)
- 開發套件入門 MediaTek MT7697Hx (p. 84)
- Microchip Curiosity PIC32MZ EF 入門 (p. 88)
- Nuvoton NuMaker-IoT-M487 入門 (p. 94)
- NXP LPC54018 IoT 模組入門 (p. 100)
- Renesas Starter Kit+ for RX65N-2MB 入門 (p. 103)
- STM32L4 探索套件 STMicroelectronics 節點入門 IoT (p. 106)
- Texas Instruments CC3220SF-LAUNCHXL 入門 (p. 108)
- Windows 裝置模擬器入門 (p. 111)
- Xilinx Avnet MicroZed Industrial IoT 套件入門 (p. 114)

Note

您不需要完成以下獨立《FreeRTOS 入門指南》的[首要步驟 \(p. 14\)](#)：

- 開始使用 Microchip ATECC608A 安全元素與 Windows 模擬器 (p. 38)
- Espressif ESP32-DevKitC 和 ESP-WROVER-KIT 入門 (p. 42)
- Espressif ESP32-WROOM-32SE 入門 (p. 56)
- 開始使用 Infineon OPTIGA Trust X 和 XMC4800 IoT Connectivity Kit (p. 65)
- Nordic nRF52840-DK 入門 (p. 91)

Cypress CYW943907AEVAL1F 開發套件入門

本教學課程提供 Cypress CYW943907AEVAL1F 開發套件入門的指示。如果您沒有 Cypress CYW943907AEVAL1F 開發套件，請造訪 AWS Partner Device Catalog，向我們的[合作夥伴](#)購買。

Note

此教學會逐步引導您設定和執行 coreMQTT 交互身份驗證示範。此主機板的 FreeRTOS 連接埠目前不支援 TCP 伺服器和用戶端示範。

開始之前，請您務必要設定 AWS IoT 和 FreeRTOS 下載目錄，才能將裝置連接至 AWS 雲端。如需說明，請參閱[首要步驟 \(p. 14\)](#)。

Important

- 在本教學課程中，FreeRTOS 下載目錄的路徑會以 *freertos* 表示。
- freertos* 路徑中的空格字元可能會導致建置失敗。當您複製或拷貝儲存庫時，請確定您建立的路徑不包含空格字元。
- Microsoft Windows 的檔案路徑長度上限為 260 個字元。FreeRTOS 下載目錄路徑過長可能會導致建置失敗。
- 如[下載 FreeRTOS \(p. 17\)](#)所述，Cypress 的 FreeRTOS 連接埠目前僅適用於 [GitHub](#)。

Overview

本教學課程包含以下入門步驟的指示：

- 在主機機器上安裝軟體以對微控制器主機板的內嵌應用程式進行開發和除錯。
- 將 FreeRTOS 示範應用程式跨編譯為二進位映像。
- 將應用程式二進位映像載入主機板，然後執行應用程式。
- 透過序列連線與在開發板上執行的應用程式互動，以便進行監控和除錯。

設定開發環境

下載並安裝 WICED Studio 軟體開發套件

在本入門指南中，您可以搭配使用 Cypress WICED Studio 軟體開發套件與 FreeRTOS 示範專案，藉此對開發板進行程式設計。請前往 [WICED Software](#) 網站，下載 Cypress 提供的 WICED Studio 軟體開發套件。您需要註冊 Cypress 免費帳戶，才能下載該軟體。WICED Studio 軟體開發套件與 Windows、macOS 和 Linux 作業系統相容。

Note

部分作業系統需進行額外的安裝步驟。請確保您已詳閱適用於作業系統和所安裝 WICED Studio 版本的所有安裝說明，並遵循指示操作。

設定環境變數

使用 WICED Studio 對開發版進行程式設計前，您必須建立 WICED Studio 軟體開發套件安裝目錄的環境變數。如果 WICED Studio 在建立變數的過程中仍持續運作，則您需要在完成變數設定後重新啟動應用程式。

Note

WICED Studio 安裝程式會在您的機器上建立兩個名為 WICED-Studio-*m.n* 上的個別資料夾，其中 *m* 和 *n* 各自為主要與次要版本編號。此文件會假定 WICED-Studio-6.2 的資料夾名稱，但務必使用您所安裝版本的正確名稱。當您定義 WICED_STUDIO_SDK_PATH 環境變數時，請務必指定 WICED Studio 軟體開發套件的完整安裝路徑，而不是 WICED Studio UI 的安裝路徑。在 Windows 和 macOS 中，依預設會在 WICED-Studio-*m.n* 資料夾中為開發套件建立 Documents 資料夾。

在 Windows 上建立環境變數

1. 開啟 Control Panel (控制台) 並選擇 System (系統)，接著選擇 Advanced System Settings (進階系統設定)。
2. 在 Advanced (進階) 索引標籤上，選擇 Environment Variables (環境變數)。
3. 在 User variables (使用者變數) 下方，選擇 New (新增)。
4. 針對 Variable name (變數名稱)，輸入 **WICED_STUDIO_SDK_PATH**。針對 Variable value (變數值)，輸入 WICED Studio 軟體開發套件安裝資料夾。

在 Linux 或 macOS 上建立環境變數

1. 在機器上開啟 `/etc/profile` 檔案，然後在檔案的最後一行新增下列內容：

```
export WICED_STUDIO_SDK_PATH=installation-path/WICED-Studio-6.2
```

2. 重新啟動機器。
3. 開啟終端機並執行下列命令：

```
cd freertos/vendors/cypress/WICED_SDK
```

```
perl platform_adjust_make.pl
```

```
chmod +x make
```

建立序列連線

在主機機器和開發板之間建立序列連線

1. 使用 USB Standard-A 對 Micro-B 纜線，將開發板連接至主機電腦。
2. 找出主機電腦上連接開發板的 USB 序列埠號。
3. 啟動序列終端機，並使用以下設定開啟連線：
 - 傳輸速率：115200
 - 資料：8 位元
 - 同位：無
 - 停止位元：1
 - 流量控制：無

如需安裝終端機與設定序列連線的詳細資訊，請參閱[安裝終端機模擬器 \(p. 20\)](#)。

建置並執行 FreeRTOS 示範專案

當您為主機板設定序列連線後，您可以建置 FreeRTOS 示範專案，將示範更新至您的主機板，然後執行示範。

在 WICED Studio 中，建置和執行 FreeRTOS 示範專案

1. 啟動 WICED Studio。
2. 從 File (檔案) 功能表中，選擇 Import (匯入)。展開 General 資料夾，選擇 Existing Projects into Workspace (現有專案到工作空間)，然後選擇 Next (下一步)。

3. 在 Select root directory (選取根目錄) 中，選取 Browse... (瀏覽...)，導覽至路徑 `freertos/projects/cypress/CYW943907AEVAL1F/wicedstudio`，然後選取 OK (確定)。
4. 在 Projects (專案) 下，僅勾選 aws_demo 專案的方塊。選擇 Finish (完成) 來匯入專案。目標專案 aws_demo 應該會出現在 Make Target (製作目標) 視窗中。
5. 展開 WICED Platform (WICED 平台) 功能表，然後選擇 WICED Filters off (WICED 篩選條件關閉)。
6. 在 Make Target (製作目標) 視窗中，展開 aws_demo，以滑鼠右鍵按一下 `demo.aws_demo` 檔案，然後選擇 Build Target (建置目標) 來建置示範並將其下載至您的主機板。示範應會在建置和下載到您的主機板後自動執行。

監控雲端的 MQTT 訊息

您可以在 AWS IoT 主控台中使用 MQTT 用戶端來監控裝置傳送至 AWS 雲端的訊息。

使用 AWS IoT MQTT 用戶端訂閱 MQTT 主題

1. 登入 [AWS IoT 主控台](#)。
2. 在導覽窗格中，選擇 Test (測試) 以開啟 MQTT 用戶端。
3. 在訂閱主題中輸入 `iotdemo/#`，然後選擇訂閱主題。

Troubleshooting

- 如果您是使用 Windows，則建置和執行示範專案時可能會收到以下錯誤：

```
: recipe for target 'download_dct' failed
make.exe[1]: *** [download_dct] Error 1
```

若要排除這個錯誤，請執行以下動作：

1. 瀏覽至 `WICED-Studio-SDK-PATH\WICED-Studio-6.2\43xxx_Wi-Fi\tools\OpenOCD\Win32`，然後按兩下 `openocd-all-brcm-libftdi.exe`。
 2. 瀏覽至 `WICED-Studio-SDK-PATH\WICED-Studio-6.2\43xxx_Wi-Fi\tools\drivers\CYW9WCD1EVAL1`，然後按兩下 `InstallDriver.exe`。
- 如果您是使用 Linux 或 macOS，則建置和執行示範專案時可能會收到以下錯誤：

```
make[1]: *** [download_dct] Error 127
```

若要排除這個錯誤，請使用下列命令來更新 libusb-dev 套件：

```
sudo apt-get install libusb-dev
```

如需 FreeRTOS 入門的一般疑難排解資訊，請參閱 [故障診斷入門 \(p. 19\)](#)。

Cypress CYW954907AEVAL1F 開發套件入門

本教學課程提供 Cypress CYW954907AEVAL1F 開發套件入門的指示。如果您沒有 Cypress CYW954907AEVAL1F 開發套件，請造訪 AWS Partner Device Catalog，向我們的[合作夥伴](#)購買。

Note

此教學會逐步引導您設定和執行 coreMQTT 交互身份驗證示範。此主機板的 FreeRTOS 連接埠目前不支援 TCP 伺服器和用戶端示範。

開始之前，請您務必要設定 AWS IoT 和 FreeRTOS 下載目錄，才能將裝置連接至 AWS 雲端。如需說明，請參閱 [首要步驟 \(p. 14\)](#)。在本教學課程中，FreeRTOS 下載目錄的路徑會以 *freertos* 表示。

Important

- 在本教學課程中，FreeRTOS 下載目錄的路徑會以 *freertos* 表示。
- *freertos* 路徑中的空格字元可能會導致建置失敗。當您複製或拷貝儲存庫時，請確定您建立的路徑不包含空格字元。
- Microsoft Windows 的檔案路徑長度上限為 260 個字元。FreeRTOS 下載目錄路徑過長可能會導致建置失敗。
- 如 [下載 FreeRTOS \(p. 17\)](#) 所述，Cypress 的 FreeRTOS 連接埠目前僅適用於 [GitHub](#)。

Overview

本教學課程包含以下入門步驟的指示：

1. 在主機機器上安裝軟體以對微控制器主機板的內嵌應用程式進行開發和除錯。
2. 將 FreeRTOS 示範應用程式跨編譯為二進位映像。
3. 將應用程式二進位映像載入主機板，然後執行應用程式。
4. 透過序列連線與在開發板上執行的應用程式互動，以便進行監控和除錯。

設定開發環境

下載並安裝 WICED Studio 軟體開發套件

在本入門指南中，您可以搭配使用 Cypress WICED Studio 軟體開發套件與 FreeRTOS 示範專案，藉此對開發板進行程式設計。請前往 [WICED Software](#) 網站，下載 Cypress 提供的 WICED Studio 軟體開發套件。您需要註冊 Cypress 免費帳戶，才能下載該軟體。WICED Studio 軟體開發套件與 Windows、macOS 和 Linux 作業系統相容。

Note

部分作業系統需進行額外的安裝步驟。請確保您已詳閱適用於作業系統和所安裝 WICED Studio 版本的所有安裝說明，並遵循指示操作。

設定環境變數

使用 WICED Studio 對開發版進行程式設計前，您必須建立 WICED Studio 軟體開發套件安裝目錄的環境變數。如果 WICED Studio 在建立變數的過程中仍持續運作，則您需要在完成變數設定後重新啟動應用程式。

Note

WICED Studio 安裝程式會在您的機器上建立兩個名為 *WICED-Studio-m.n* 上的個別資料夾，其中 m 和 n 各自為主要與次要版本編號。此文件會假定 *WICED-Studio-6.2* 的資料夾名稱，但務必使用您所安裝版本的正確名稱。當您定義 *WICED_STUDIO_SDK_PATH* 環境變數時，請務必指定 WICED Studio 軟體開發套件的完整安裝路徑，而不是 WICED Studio UI 的安裝路徑。在 Windows 和 macOS 中，預設會在 *WICED-Studio-m.n* 資料夾中為開發套件建立 *Documents* 資料夾。

在 Windows 上建立環境變數

1. 開啟 Control Panel (控制台) 並選擇 System (系統)，接著選擇 Advanced System Settings (進階系統設定)。
2. 在 Advanced (進階) 索引標籤上，選擇 Environment Variables (環境變數)。
3. 在 User variables (使用者變數) 下方，選擇 New (新增)。
4. 針對 Variable name (變數名稱)，輸入 *WICED_STUDIO_SDK_PATH*。針對 Variable value (變數值)，輸入 WICED Studio 軟體開發套件安裝資料夾。

在 Linux 或 macOS 上建立環境變數

1. 在機器上開啟 /etc/profile 檔案，然後在檔案的最後一行新增下列內容：

```
export WICED_STUDIO_SDK_PATH=installation-path/WICED-Studio-6.2
```

2. 重新啟動機器。
3. 開啟終端機並執行下列命令：

```
cd freertos/vendors/cypress/WICED_SDK
```

```
perl platform_adjust_make.pl
```

```
chmod +x make
```

建立序列連線

在主機機器和開發板之間建立序列連線

1. 使用 USB Standard-A 對 Micro-B 纜線，將開發板連接至主機電腦。
2. 找出主機電腦上連接開發板的 USB 序列埠號。
3. 啟動序列終端機，並使用以下設定開啟連線：
 - 傳輸速率：115200
 - 資料：8 位元
 - 同位：無
 - 停止位元：1
 - 流量控制：無

如需安裝終端機與設定序列連線的詳細資訊，請參閱[安裝終端機模擬器 \(p. 20\)](#)。

建置並執行 FreeRTOS 示範專案

當您為主機板設定序列連線後，您可以建置 FreeRTOS 示範專案，將示範更新至您的主機板，然後執行示範。

在 WICED Studio 中，建置和執行 FreeRTOS 示範專案

1. 啟動 WICED Studio。
2. 從 File (檔案) 功能表中，選擇 Import (匯入)。展開 General 資料夾，選擇 Existing Projects into Workspace (現有專案到工作空間)，然後選擇 Next (下一步)。
3. 在 Select root directory (選取根目錄) 中，選取 Browse... (瀏覽...)，導覽至路徑 *freertos*/projects/cypress/CYW954907AEVAL1F/wicedstudio，然後選取 OK (確定)。
4. 在 Projects (專案) 下，僅勾選 aws_demo 專案的方塊。選擇 Finish (完成) 來匯入專案。目標專案 aws_demo 應該會出現在 Make Target (製作目標) 視窗中。
5. 展開 WICED Platform (WICED 平台) 功能表，然後選擇 WICED Filters off (WICED 篩選條件關閉)。
6. 在 Make Target (製作目標) 視窗中，展開 aws_demo，以滑鼠右鍵按一下 demo.aws_demo 檔案，然後選擇 Build Target (建置目標) 來建置示範並將其下載至您的主機板。示範應會在建置和下載到您的主機板後自動執行。

監控雲端的 MQTT 訊息

您可以在 AWS IoT 主控台中使用 MQTT 用戶端來監控裝置傳送至 AWS 雲端的訊息。

使用 AWS IoT MQTT 用戶端訂閱 MQTT 主題

1. 登入 [AWS IoT 主控台](#)。
2. 在導覽窗格中，選擇 Test (測試) 以開啟 MQTT 用戶端。
3. 在訂閱主題中輸入 `iotdemo/#`，然後選擇訂閱主題。

Troubleshooting

- 如果您是使用 Windows，則建置和執行示範專案時可能會收到以下錯誤：

```
: recipe for target 'download_dct' failed
make.exe[1]: *** [download_dct] Error 1
```

若要排除這個錯誤，請執行以下動作：

1. 瀏覽至 **WICED-Studio-SDK-PATH\WICED-Studio-6.2\43xxx_Wi-Fi\tools\OpenOCD\Win32**，然後按兩下 `openocd-all-brcm-libftdi.exe`。
 2. 瀏覽至 **WICED-Studio-SDK-PATH\WICED-Studio-6.2\43xxx_Wi-Fi\tools\drivers\CYW9WCD1EVAL1**，然後按兩下 `InstallDriver.exe`。
- 如果您是使用 Linux 或 macOS，則建置和執行示範專案時可能會收到以下錯誤：

```
make[1]: *** [download_dct] Error 127
```

若要排除這個錯誤，請使用下列命令來更新 libusb-dev 套件：

```
sudo apt-get install libusb-dev
```

如需 FreeRTOS 入門的一般疑難排解資訊，請參閱 [故障診斷入門 \(p. 19\)](#)。

Cypress CY8CKIT-064S0S2-4343W 套件入門

本教學課程提供 [CY8CKIT-064S0S2-4343W](#) 套件的入門指示。如果您尚未擁有，您可以使用該連結來購買套件。您也可以使用該連結來存取套件使用者指南。

入門

開始之前，您必須設定 AWS IoT 和 FreeRTOS 以將您的裝置連接到 AWS 雲端。如需說明，請參閱 [首要步驟 \(p. 14\)](#)。完成必要條件之後，您將擁有包含 FreeRTOS 登入資料的 AWS IoT Core 套件。

Note

在本教學課程中，「第一個步驟」一節中所建立 FreeRTOS 下載資料夾的路徑稱為 `freertos`。

設定開發環境

可與 FreeRTOS 或 Make build flow (使建置流程) 搭配使用。CMake 您可以將 ModusToolbox 用於建立流程。您可以使用透過 ModusToolbox 或合作夥伴 IDE (例如 IAR EW-Arm、Arm MDK 或 Microsoft Visual Studio Code) 所交付的 Eclipse IDE。Eclipse IDE 與 Windows、macOS 和 Linux 作業系統相容。

開始之前，請下載並安裝最新的 [軟體ModusToolbox](#)。如需詳細資訊，請參閱 [安裝指南ModusToolbox](#)。

更新 ModusToolbox 2.1 或更新版本的工具

如果您使用的是 ModusToolbox 2.1 Eclipse IDE 程式設計此套件，則需要更新 OpenOCD 和韌體載入器工具。

在下列步驟中，根據預設，*ModusToolbox* 路徑為：

- Windows 是 C:\Users\user_name\ModusToolbox。
- Linux 是 user_home/ModusToolbox 或您在其中選擇擷取封存檔的位置。
- MacOS 位於您在精靈中選取之磁碟區的 Applications (應用程式) 資料夾下方。

更新OpenOCD

此套件需要 Cypress OpenOCD 4.0.0 或更新版本才能成功清除和程式設計晶片。

更新 Cypress OpenOCD

1. 前往 [Cypress OpenOCD 發行版本](#)。
2. 下載適用於您作業系統的封存檔 (Windows/Mac/Linux) 。
3. 刪除 *ModusToolbox/tools_2.x/openocd* 中現有的檔案。
4. 以您在先前步驟中下載的封存解壓縮的項目取代 *ModusToolbox/tools_2.x/openocd* 中的檔案。

更新韌體載入器

此套件需要 Cypress Firmware-loader 3.0.0 或更新版本。

更新 Cypress Firmware-loader

1. 前往 [Cypress Firmware-loader 發行版本頁面](#)。
2. 下載適用於您作業系統的封存檔 (Windows/Mac/Linux) 。
3. 刪除 *ModusToolbox/tools_2.x/fw-loader* 中現有的檔案。
4. 以您在先前步驟中下載的封存解壓縮的項目取代 *ModusToolbox/tools_2.x/fw-loader* 中的檔案。

或者，您可以使用 CMake 從 FreeRTOS 應用程式原始碼建立專案建置檔案、使用您偏好的建置工具建置專案，然後使用 OpenOCD 來進行套件的程式設計。如果您偏好使用 GUI 工具搭配 CMake 流程進行程式設計，請從 [Cypress Programming Solution](#) 網頁下載並安裝 Cypress Programramer。如需詳細資訊，請參閱 [使用 CMake 配 FreeRTOS \(p. 20\)](#)。

設定您的硬體

依照以下步驟來設定套件的硬體。

1. 發行您的套件

請遵循 [Provisioning Guide for CY8CKIT-064S0S2-4343W Kit](#) 指示，安全地為您的 AWS IoT 實作您的套件。

2. 設定序列連接

- a. 將套件連接至主機電腦。
- b. 此套件的 USB 序列連接埠會在主機電腦上自動列舉。找出連接埠號碼。在 Windows 中，您可以在 Ports (連接埠) (COM 和 LPT) 下使用 Device Manager (裝置管理員) 來辨識它。
- c. 啟動序列終端機，並使用以下設定開啟連線：

- 傳輸速率：115200

- 資料：8位元
 - 同位：無
 - 停止位元：1
 - 流量控制：無

建置並執行 FreeRTOS 示範專案

在本節中，您會建置並執行示範。

1. 請務必遵循 CY8CKIT-064S0S2-4343W 套件 的供應指南步驟。
 2. 建置 FreeRTOS 示範。

- a. 開立適用於 ModusToolbox 的 Eclipse IDE，並選擇或建立工作空間。
 - b. 從 File (檔案) 功能表中，選擇 Import (匯入)。

展開 General (一般)，選擇 Existing Project Into Workspace (現有專案至工作空間)，然後選擇 Next (下一步)。

- c. 在 Root Directory (根 Directory) 中，輸入 `freertos/projects/cypress/CY8CKIT-064S0S2-4343W/mtb/aws_demos`，然後選取專案名稱 `aws_demos`。預設應會選取它。
 - d. 選擇 Finish (完成)，以將專案匯入工作區。
 - e. 請執行下列其中一個項目以建置應用程式：
 - 從 Quick Panel (快速面板)，選取 Build aws_demos Application (建置 aws_demos 應用程式)。
 - 選擇 Project (專案)，然後選擇 Build All (全部建置)。

請確定專案編譯時未發生錯誤。

- ### 3. 執行 FreeRTOS 示範專案

- a. 在工作空間中選取專案 aws_demos。
 - b. 從 Quick Panel (快速面板)，選取 aws_demos Program (KitProg3)。此程式設計主機板和示範應用程式會在程式設計完成後開始執行。
 - c. 您可以在序列終端機中檢視執行中應用程式的狀態。下圖顯示終端機輸出的一部分。

```
COM5 - Tera Term VT
File Edit Setup Control Window Help

ULAN MAC Address : CC:C8:79:24:DB:8B
ULAN Firmware : v1.0.0-dжу 30 2019 01:54:48 version 7.45.98.89 <r718486 CY> FWID 01-81376c4b
ULAN CLM : API: 12.2 Data: 9.10.39 Compiler: 1.29.4 ClmImport: 1.36.3 Creation: 2019-07-30 01:43:02
WHD VERSION : v1.30.0-rc3-dirty : v1.30.0-rc3 : GCC 7.2 : 2019-08-27 16:29:32 +0000
3518 [Info] Wi-Fi Connected to AP Creating tasks which use network...
3518 [Info] Mac address acquired 192.168.43.207
5895 [Info] User connected
45623 [Info] Device credential provisioning succeeded.
5 5627 [iot_thread] [INFO] INITIALLu SDK successfully initialized.
6 8504 [iot_thread] [INFO] IDEMO1lllu Successfully initialized the demo. Network type for the demo: 1
7 8504 [iot_thread] [INFO] IMQTT1lllu MQTT library successfully initialized.
8 8504 [iot_thread] [INFO] IDEMO1lllu demo client identifier is cysproto-kit <length 13>.
13409 [iot_thread] [INFO] IMQTT1lllu Establishing new MQTT connection.
1904 [iot_thread] [INFO] IMQTT1lllu AWS IoT Metrics connection established.
1904 [iot_thread] [INFO] IMQTT1lllu AWS IoT Metrics (SDN language, SDN version) will be provided to AWS IoT. Recompile with AWS_IOT_MOTT_ENROLLMENT_METRICS set to 0 to disable.
11 13412 [iot_thread] [INFO] IMQTT1lllu <MQTT connection 0x800b4c8> CONNECT operation 0x800b2d0> Waiting for operation completion.
12 13753 [iot_thread] [INFO] IMQTT1lllu <MQTT connection 0x800b4c8> CONNECT operation 0x800b2d0> Wait complete with result SUCCESS.
13 13754 [iot_thread] [INFO] IMQTT1lllu New MQTT connection 0x800bc84 established.
14 13755 [iot_thread] [INFO] IMQTT1lllu <MQTT connection 0x800b4c8> SUBSCRIBE operation scheduled.
15 13756 [iot_thread] [INFO] IMQTT1lllu <MQTT connection 0x800b4c8> SUBSCRIBE operation 0x800b5e0> Waiting for operation completion.
16 14065 [iot_thread] [INFO] IMQTT1lllu <MQTT connection 0x800b4c8> SUBSCRIBE operation 0x800b5e0> Wait complete with result SUCCESS.
17 14065 [iot_thread] [INFO] IDEMO1lllu All demo topic filter subscriptions accepted.
18 14065 [iot_thread] [INFO] IDEMO1lllu Publishing messages 0 to 1.
19 14067 [iot_thread] [INFO] IMQTT1lllu <MQTT connection 0x800b4c8> MQTT PUBLISH operation queued.
20 14067 [iot_thread] [INFO] IMQTT1lllu <MQTT connection 0x800b4c8> MQTT PUBLISH operation queued.
21 14067 [iot_thread] [INFO] IDEMO1lllu Waiting for 2 publishes to be received.
22 14324 [iot_thread] [INFO] IDEMO1lllu MQTT PUBLISH 0 successfully sent.
23 14424 [iot_thread] [INFO] IDEMO1lllu Incoming PUBLISH received:
Subscription topic filter: iotdemo/topic/1
Publish topic name: iotdemo/topic/1
Publish retain flag: 0
Publish QoS: 1
Publish pay24 14424 [iot_thread] [INFO] IMQTT1lllu <MQTT connection 0x800b4c8> MQTT PUBLISH operation queued.
24 14426 [iot_thread] [INFO] IDEMO1lllu Acknowledgment message for PUBLISH 0 will be sent.
25 14428 [iot_thread] [INFO] IMQTT1lllu <MQTT connection 0x800b4c8> MQTT PUBLISH 0 successfully sent.
27 14708 [iot_thread] [INFO] IDEMO1lllu Incoming PUBLISH received:
Subscription topic filter: iotdemo/topic/2
Publish topic name: iotdemo/topic/2
Publish retain flag: 0
Publish QoS: 1
Publish pay28 14708 [iot_thread] [INFO] IMQTT1lllu <MQTT connection 0x800b4c8> MQTT PUBLISH operation queued.
28 14710 [iot_thread] [INFO] IDEMO1lllu Acknowledgment message for PUBLISH 1 will be sent.
29 14710 [iot_thread] [INFO] IDEMO1lllu 2 publishes received.
30 14710 [iot_thread] [INFO] IDEMO1lllu Publishing messages 2 to 3.
31 14710 [iot_thread] [INFO] IDEMO1lllu Publishing messages 2 to 3.
```

MQTT 示範在四個不同主題 (`iotdemo/topic/n` , 其中 $n=1$ 到 4) 上發布訊息，並訂用所有主題以接收相同訊息。收到訊息時，示範會在主題 `iotdemo/acknowledgements` 上發布確認訊息。以下清單描述顯示在終端機輸出中的偵錯訊息，並參考訊息的序號。在輸出中，會先列印 WICED Host Driver (WHD) 驅動程式詳細資訊，而不使用序號。

1. 1 到 4 個 – 裝置連接到設定的存取點 (AP)，並可透過使用設定的端點和憑證連接到 AWS 伺服器來設定。
2. 5 到 13 – coreMQTT 程式庫已初始化，而且裝置會建立 MQTT 連接。
3. 14 到 17 個 – 裝置會訂用所有主題，以接收已發布的訊息。
4. 18 到 30 個 – 裝置會發布兩則訊息並等待接收它們。收到每則訊息時，裝置會傳送確認訊息。

同樣的發布、接收和確認循環會持續到所有訊息發布為止。每個循環會發布兩則訊息，直到設定的循環次數完成。

4. 將 CMake 與 FreeRTOS 搭配使用

您也可以使用 CMake 來建置和執行示範應用程式。若要設定 CMake 和原生建置系統，請前往 [Prerequisites \(p. 21\)](#)。

- a. 使用下列命令來生成組建檔案。使用 `-DBOARD` 選項指定目標電路板。

```
cmake -DVENDOR=cypress -DBOARD=CY8CKIT_064S0S2_4343W -DCOMPILER=arm-gcc -S freertos  
-B build_dir
```

如果您使用 Windows，則必須使用 `-G` 選項來指定原生建置系統，因為 CMake 預設會使用 Visual Studio。

Example

```
cmake -DVENDOR=cypress -DBOARD=CY8CKIT_064S0S2_4343W -DCOMPILER=arm-gcc -S freertos  
-B build_dir -G Ninja
```

如果 `arm-none-eabi-gcc` 不在您的 shell 路徑中，您也需要設定 `AFR_TOOLCHAIN_PATH` CMake 變數。

Example

```
-DAFR_TOOLCHAIN_PATH=/home/user/opt/gcc-arm-none-eabi/bin
```

- b. 使用下列命令來使用 CMake 建置專案。

```
cmake --build build_dir
```

- c. 最後，使用 Cypress Programler 編寫 `cm0.hex` 和 `cm4.hex` 檔案的程式。`build_dir`

5. 監控雲端的 MQTT 訊息

您可以在 AWS IoT 主控台中使用 MQTT 用戶端來監控裝置傳送至 AWS 雲端的訊息。若要使用 AWS IoT MQTT 使用者端訂用 MQTT 主題，請依照以下步驟。

- a. 登入 [AWS IoT 主控台](#)。
- b. 在導覽窗格中，選擇 Test (測試) 以開啟 MQTT 用戶端。
- c. 針對 Subscription topic (子標主題)，輸入 `iotdemo/#`，然後選擇 Subscribe to topic (描述主題)。
- d. 重設套件以讓它發布 MQTT 訊息，然後在主控台測試使用者看到該訊息。

執行其他示範

以下示範應用程式已經過測試和驗證，可與目前版本搭配使用。您可以在 `freertos/demos` 下方找到這些示範。如需執行這些示範的資訊，請前往 [FreeRTOS 示範 \(p. 210\)](#)。

- 低功耗藍牙示範
- 即時更新示範
- Secure Sockets Echo Client 示範
- AWS IoT 裝置影子示範

Debugging

套件上的 KitProg3 支援透過 SWD 通訊協定進行偵錯。

- 若要偵錯 FreeRTOS 應用程式，請選取工作空間中的 `aws_demos` 專案，然後從 Quick Panel (快速面板) 選取 `aws_demos Debug` (KitPig3) 。

OTA 更新

PSoC 64 MCUs 已通過所有必要的 FreeRTOS 資格測試。不過，在 PSoC 64 Standard Secure AWS 韌體程式庫中實作的選用遠端 (OTA) 功能仍處於等待評估的狀態。做為實作的 OTA 功能目前會通過所有 OTA 資格測試，但 [aws_ota_test_case_rollback_if_unable_to_connect_after_update.py](#) 除外。

使用 PSoC64 Standard Secure – AWS MCU 將成功驗證 OTA 映像套用至裝置且裝置無法與 AWS IoT Core 通訊時，裝置無法自動轉返到原始已知良好的映像。這可能會導致裝置無法從 AWS IoT Core 觸達，以進行進一步的更新。Cypress 團隊仍在開發此功能。

如需詳細資訊，請查看 [OTA Updates with AWS 和 CY8CKIT-064S0S2-4343W 套件](#)。如果您有進一步的問題或需要技術支援，請聯絡 [Cypress 開發人員社群](#)。

開始使用 Microchip ATECC608A 安全元素與 Windows 模擬器

本教學課程提供 Microchip ATECC608A 安全元素與 Windows 模擬器的入門說明。

您需要下列硬體：

- Microchip ATECC608A 安全元素點擊板
- SAMD21 XPlained Pro
- mikroBUS Xplained Pro Adapter

開始之前，請務必設定 AWS IoT 和 FreeRTOS 下載將裝置連線至 AWS 雲端。在本教學課程中，FreeRTOS 下載目錄的路徑會以 `freertos` 表示。

概觀

本教學課程包含以下步驟：

1. 將主機板連線到主機機器。
2. 在主機機器上安裝軟體，以開發和偵錯微控制器主機板的內嵌應用程式。

3. 將 FreeRTOS 示範應用程式跨編譯為二進位映像。
4. 將應用程式二進位映像載入主機板，然後執行應用程式。

設定 Microchip ATECC608A 硬體

您必須先程式化 SAMD21，才能與 Microchip ATECC608A 裝置互動。

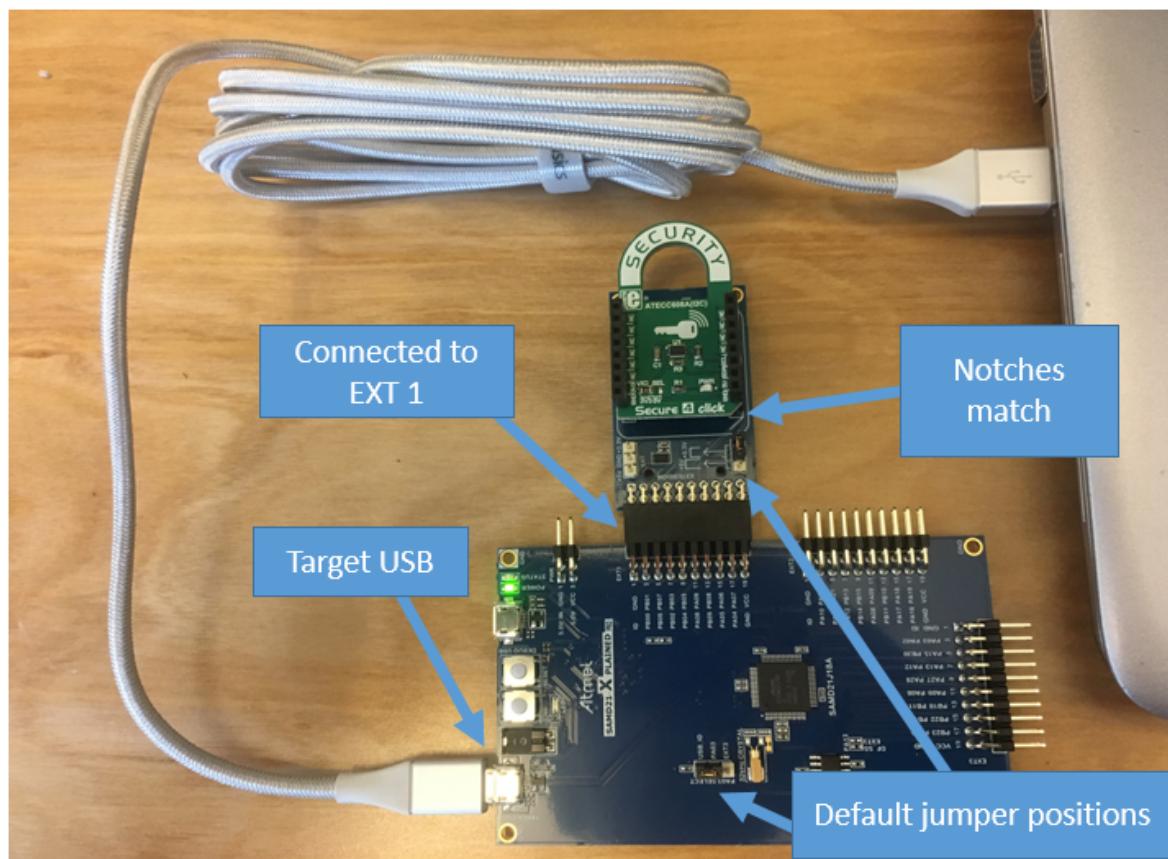
設定 SAMD21 XPlained Pro 面板

1. 遵循 [CryptoAuthSSH-XSTK \(DM320109\) - Latest Firmware](#) 連接，下載包含說明 (PDF) 的和可程式化為 D21 之二進位的 .zip 檔案。
2. 下載並安裝 [Atmel Studio 7](#) IDP。請務必在安裝期間選取 SMART ARM MCU 驅動程式架構。
3. 使用 USB 2.0 Micro B 連接線將「偵錯 USB」連接器連接到您的電腦，並遵循 PDF 中的說明進行。（「偵錯 USB」連接器是最接近電源 led 燈和接腳的 USB 連接埠。）

連接硬體

1. 從偵錯 USB 拔除微型 USB 連接線。
2. 將 mikroBUS XPlained Pro Adapter 插入 EXT1 位置的 SAMD21 主機板。
3. 將 ATECC608A Secure 4 點擊板插入 mikroBUSX XPlained Pro Adapter。確定點擊板的凹口角落與轉接器板的凹口圖示相符。
4. 將微型 USB 連接線插入目標 USB。

您的設定看起來應該如下所示。



設定開發環境

1. 若尚未如此做，請建立 AWS 帳戶。若要新增 IAM 使用者至您的 AWS 帳戶，請參閱 [IAM 使用者指南](#)。
若要將您的 IAM 使用者帳戶存取權授予 AWS IoT 和 FreeRTOS，請在這些步驟中連接下列 IAM 政策至您的 IAM 使用者帳戶：
 - AmazonFreeRTOSFullAccess
 - AWSIoTFullAccess
2. 將 AmazonFreeRTOSFullAccess 政策連接到您的 IAM 使用者。
 - a. 瀏覽至 [IAM 主控台](#)，並從導覽窗格中，選擇 Users (使用者)。
 - b. 在搜尋文字方塊中，輸入您的使用者名稱，然後從清單中選擇它。
 - c. 選擇 Add permissions (新增許可)。
 - d. 選擇 Attach existing policies directly (直接連接現有政策)。
 - e. 在搜尋方塊中，輸入 **AmazonFreeRTOSFullAccess**，從清單中選擇它，然後選擇 Next: Review (下一步：檢閱)。
 - f. 選擇 Add permissions (新增許可)。
3. 將 AWSIoTFullAccess 政策連接到您的 IAM 使用者。
 - a. 瀏覽至 [IAM 主控台](#)，並從導覽窗格中，選擇 Users (使用者)。
 - b. 在搜尋文字方塊中，輸入您的使用者名稱，然後從清單中選擇它。
 - c. 選擇 Add permissions (新增許可)。
 - d. 選擇 Attach existing policies directly (直接連接現有政策)。
 - e. 在搜尋方塊中，輸入 **AWSIoTFullAccess**，從清單中選擇它，然後選擇 Next: Review (下一步：檢閱)。
 - f. 選擇 Add permissions (新增許可)。

如需 IAM 的詳細資訊，請參閱[IAM 使用者指南](#)中的 IAM 許可和政策。

4. 從 [FreeRTOS GitHub 儲存庫](#) 下載 FreeRTOS 儲存庫。

從 GitHub 下載 FreeRTOS：

1. 瀏覽到 [FreeRTOS GitHub 儲存庫](#)。
2. 選擇 Clone or download (複製或下載)。
3. 透過您電腦上的命令列，將儲存庫複製到您主機上的目錄。

```
git clone https://github.com/aws/amazon-freertos.git --recurse-submodules
```

Important

- 在本主題中，FreeRTOS 下載目錄的路徑會以 **freertos** 表示。
- **freertos** 路徑中的空格字元可能會導致建置失敗。當您複製或拷貝儲存庫時，請確定您建立的路徑不包含空格字元。
- Microsoft Windows 的檔案路徑長度上限為 260 個字元。FreeRTOS 下載目錄路徑過長可能會導致建置失敗。

4. 從 **freertos** 目錄中，查看要使用的分支。
5. 設定開發環境。
 - a. 安裝最新版的 [WinPCap](#)。
 - b. 安裝 Microsoft Visual Studio。

Visual Studio 版本 2017 和 2019 已知可運作。支援所有這些 Visual Studio 版本 (Community、Professional 或 Enterprise)。

除 IDE 外，安裝使用 C++ 的桌面開發元件。然後，在 Optional (選用) 下，安裝最新的 Windows 10 軟體開發套件。

- c. 請確認您的有線乙太網路連線為作用中。

建置並執行 FreeRTOS 示範專案

Important

Microchip ATECC608A 裝置具有一次性的初始化功能，會在專案第一次執行時 (在對 `C_InitToken` 的呼叫期間) 鎖定到裝置上。不過，FreeRTOS 示範專案和測試專案有不同的組態。如果裝置在示範專案組態期間鎖定，測試專案中的所有測試將無法成功。

使用 Visual Studio IDE 建立及執行 FreeRTOS 示範專案

1. 將專案載入到 Visual Studio。

在 File (檔案) 功能表上，選擇 Open (開啟)。選擇 File/Solution (檔案/解決方案)，導覽至 `freertos\projects\microchip\ecc608a_plus_winsim\visual_studio\aws_demos\aws_demos.sln` 檔案，然後選擇 Open (開啟)。

2. 重新定向示範專案。

示範專案取決於 Windows 開發套件，但該專案沒有指定的 Windows 開發套件版本。在預設情況下，IDE 可能會嘗試使用您電腦中未呈現的軟體開發套件版本來建置示範。若要設定 Windows 開發套件版本，請用滑鼠右鍵按一下 `aws_demos`，然後選擇 Retarget Projects (重定向專案)。這會開啟 Review Solution Actions (檢閱解決方案動作) 視窗。選擇您電腦上呈現的 Windows 軟體開發套件版本 (使用下拉式清單中的初始值)，然後選擇 OK (確定)。

3. 建置並執行專案。

從 Build (建立) 選單中，選擇 Build Solution (建立解決方案)，並確認建立解決方案時未發生錯誤。選擇 Debug (偵錯)、Start Debugging (開始偵錯) 以執行專案。在第一次執行時，您需要設定您的裝置界面並重新編譯。如需詳細資訊，請參閱 [設定網路界面 \(p. 113\)](#)。

4. 佈建 Microchip ATECC608A。

微晶片提供了多種指令碼工具，協助您設定 ATECC608A 組件。導覽至 `freertos\vendors\microchip\secure_elements\app\example_trust_chain_tool`，然後開啟 README.md 檔案。

遵循 README.md 檔案中的指示，佈建您的裝置。這些步驟如下：

1. 建立憑證授權機構並向 AWS 註冊。
2. 在 Microchip ATECC608A 上產生您的金鑰，並匯出公有金鑰和裝置序號。
3. 產生裝置的憑證並向 AWS 註冊該憑證。
4. 將 CA 憑證和裝置憑證載入至裝置。
5. 建置並執行 FreeRTOS 範例。

再執行一次示範專案。這次應該能連線了！

故障診斷

如需一般疑難排解資訊，請參閱 [故障診斷入門 \(p. 19\)](#)。

Espressif ESP32-DevKitC 和 ESP-WROVER-KIT 入門

本教學提供配備 ESP32-WROOM-32、ESP32-SOLO-1 或 ESP-WROVER 模組及 ESP-WROVER-KIT-VB 的 Espressif ESP32-DevKitC 入門說明。要向我們的合作伙伴購買AWSPartnerDevice目錄,請使用以下鏈接:[ESP32-WROOM-32\(ESP32-WROOM-32\) DevKitC, 另迄革32-迄連在連-1,或](#) [ESP32-WROVER-KIT開關](#). 這些版本的開發面板, FreeRTOS 都支援。如需這些面板的詳細資訊, 請參閱 Espressif 網站上的 [ESP32-DevKitC](#) 或 [ESP-WROVER-KIT](#)。

Note

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

- 對稱式多工處理 (SMP)。

Overview

本教學課程包含以下入門步驟的指示：

1. 將主機板連線到主機機器。
2. 在主機機器上安裝軟體以對微控制器主機板的內嵌應用程式進行開發和除錯。
3. 將 FreeRTOS 示範應用程式跨編譯為二進位映像。
4. 將應用程式二進位映像載入主機板，然後執行應用程式。
5. 透過序列連線與在開發板上執行的應用程式互動，以便進行監控和除錯。

Prerequisites

在 Espressif 主機板上開始使用 FreeRTOS 前，您需要先設定 AWS 帳戶和許可。

若要建立 AWS 帳戶，請參閱[建立並啟用 AWS 帳戶](#)。

若要新增 IAM 使用者至您的 AWS 帳戶，請參閱[IAM 使用者指南](#)。若要授予 IAM 使用者帳戶存取 AWS IoT 和 FreeRTOS 的許可，請將以下 IAM 政策連接到您的 IAM 使用者帳戶：

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

將 AmazonFreeRTOSFullAccess 政策連接到您的 IAM 使用者

1. 瀏覽至[IAM 主控台](#)，並從導覽窗格中，選擇 Users (使用者)。
2. 在搜尋文字方塊中，輸入您的使用者名稱，然後從清單中選擇它。
3. 選擇 Add permissions (新增許可)。
4. 選擇 Attach existing policies directly (直接連接現有政策)。
5. 在搜索框中,輸入 [AmazonFreeRTOSFullAccess](#),從列表中選擇,然後選擇 下一步: 回顧.
6. 選擇 Add permissions (新增許可)。

將 AWSIoTFullAccess 政策連接到您的 IAM 使用者

1. 瀏覽至[IAM 主控台](#)，並從導覽窗格中，選擇 Users (使用者)。
2. 在搜尋文字方塊中，輸入您的使用者名稱，然後從清單中選擇它。
3. 選擇 Add permissions (新增許可)。
4. 選擇 Attach existing policies directly (直接連接現有政策)。

5. 在搜索框中，輸入 **AWSIoTFullAccess**，從列表中選擇，然後選擇 下一步：回顧。
6. 選擇 Add permissions (新增許可)。

如需 IAM 和使用者帳戶的詳細資訊，請參閱 [IAM 使用者指南](#)。

如需政策的詳細資訊，請參閱 [IAM 許可與政策](#)。

設定 Espressif 硬體

請參閱 [ESP32-DevKitC 入門指南](#)，以取得有關設定 ESP32-DevKitC 開發板硬體的資訊。

請參閱 [ESP-WROVER-KIT 入門指南](#)，以取得有關設定 ESP-WROVER-KIT 開發板硬體的資訊。

Note

請勿繼續進行 Espressif 指南的 Get Started (開始使用) 部分。反之，請依照以下步驟進行。

設定開發環境

若要與主機板通訊，您必須下載並安裝工具鏈。

設定工具鏈

若要設定工具鏈，請依照您主機作業系統的指示進行：

- [Windows 工具鏈的標準設定](#)
- [工具鏈的標準設置 macOS](#)
- [Linux 工具鏈的標準設定](#)

Important

當您到達 Next Steps (後續步驟) 下的「取得 ESP-IDF」指示，請停止和返回此頁面上的指示。

請確定 `IDF_PATH` 環境變數已從您的系統中清除，然後再繼續。如果您遵循了 Next Steps (後續步驟) 下的「取得 ESP-IDF」指示，則會自動設定此環境變數。

Note

ESP-IDF 3.3 版本 (FreeRTOS 使用的版本) 不支援 ESP32 編譯器的最新版本。您必須使用與 ESP-IDF 3.3 版本相容的編譯器。請參閱上述連結。若要檢查編譯器的版本，請執行下列命令。

```
xtensa-esp32-elf-gcc --version
```

安裝 CMake

的 CMake 構建系統是構建 FreeRTOS 演示和測試此設備的應用程序。FreeRTOS 支持 3.13 版和更高版本。

您可以下載最新版本的 CMake 從 [CMake.org 公司](#)，同時提供來源和二進位發佈。

有關使用的更多詳細信息 CMake 配 FreeRTOS，請參閱 [使用 CMake 配 FreeRTOS \(p. 20\)](#)。

建立序列連線

要在主機和 ESP32-DevKitC 之間建立串行連接，您必須安裝 CP210x USB 至 UART 橋接 VCP 驅動器。您可以從 [Silicon Labs](#) 下載這些驅動程式。

若要在您的主機與 ESP32-WROVER-KIT 之間建立序列連線，您必須安裝一些 FTDI virtual COM 連接埠驅動程式。您可以從 [FTDI](#) 下載這些驅動程式。

如需詳細資訊，請參閱[建立與 ESP32 的序列連線](#)。在您建立序連接之後，請記下開發板連接的序列連接埠。您在建立示範時會需要它。

下載和設定 FreeRTOS

設定您的環境之後，即可從 [GitHub](#) 或 [FreeRTOS 主控台](#) 下載 FreeRTOS。如需說明，請參閱 [README.md](#) 檔案。

設定 FreeRTOS 示範應用程式

1. 如果你正在運行 macOS 或 Linux, 打開終端提示符。如果您正在運行 Windows, 請打開 mingw32.exe。(人最小GW 是適用於原生 Microsoft Windows 應用程序的極簡開發環境。)
2. 要驗證您是否安裝了 Python 2.7.10 或更高版本, 請運行 python --version。已安裝的版本顯示。如果您沒有安裝 Python 2.7.10 或更新版本，您可以從 [Python 網站](#) 進行安裝。
3. 若要執行 AWS IoT 命令，您需要 AWS CLI。如果您執行的是 Windows，請使用 easy_install awscli 以在 mingw32 環境中安裝 AWS CLI。

如果你正在運行 macOS 或 Linux, 請參閱 [安裝 AWS 命令行界面](#)。

4. 執行 aws configure 並以您的 AWS 存取金鑰 ID、私密存取金鑰和預設區域名稱來設定 AWS CLI。如需詳細資訊，請參閱[設定 AWS CLI](#)。
5. 使用以下命令來安裝適用於 Python 的 AWS 開發套件 (boto3)：
 - 在 Windows 的 mingw32 環境中，執行 easy_install boto3。
 - 開啟 macOS 或 Linux, 運行 pip install tornado nose --user 然後運行 pip install boto3 --user.

FreeRTOS 包含 SetupAWS.py 指令碼，可讓您更輕鬆地設定 Espressif 板以連接至 AWS IoT。若要設定此指令碼，請開啟 [freertos/tools/aws_config_quick_start/configure.json](#) 並設定下列屬性：

afr_source_dir

通往 [freertos](#) 計算機上的目錄。請確定您使用斜線來指定此路徑。

thing_name

您想要指派至代表您開發板之 AWS IoT 物的名稱。

wifi_ssid

您的 Wi-Fi 網路 SSID。

wifi_password

您的 Wi-Fi 網路的密碼。

wifi_security

您的 Wi-Fi 網路的安全類型。

有效安全類型為：

- eWiFiSecurityOpen (開放，不具安全性)
- eWiFiSecurityWEP (WEP 安全性)
- eWiFiSecurityWPA (WPA 安全性)
- eWiFiSecurityWPA2 (WPA2 安全性)

執行組態指令碼

1. 如果你正在運行 macOS 或 Linux, 打開終端提示符。如果您執行的是 Windows, 請開啟 mingw32.exe。
2. 前往 `freertos/tools/aws_config_quick_start` 目錄並執行 `python SetupAWS.py setup`。

此指令碼會執行下列操作：

- 創建 IoT 證書和政策
- 將 IoT 證書和證書的策略 AWS IoT 事情
- 使用您的 AWS IoT 端點、Wi-Fi SSID 和登入資料來填入 `aws_clientcredential.h` 檔案
- 格式化您的憑證與私密金鑰，並將其寫入 `aws_clientcredential_keys.h` 標頭檔案

Note

憑證硬式編碼只是為了示範。生產層級應用程式必須將這些檔案存放在安全的位置。

如需 `SetupAWS.py` 的詳細資訊，請參閱 `freertos/tools/aws_config_quick_start` 目錄中的 `README.md`。

建置、刷新和執行 FreeRTOS 示範專案

您可以使用 CMake 以生成構建文件、Make 以構建應用程序二進制文件以及 Espressif 的 IDF 實用程序以刷寫您的板。

構建 FreeRTOS 在 Linux 和上 MacOS

(如果您使用的是 Windows，請參閱下一節。)

使用 CMake 以生成構建文件，然後使用 Make 構建應用程序。

要生成演示應用程序的構建文件，請使用 CMake

1. 將目錄變更到您 FreeRTOS 下載目錄的根。
2. 使用下列命令來產生建置檔案：

```
cmake -DVENDOR=espressif -DBOARD=esp32_wrover_kit -DCOMPILER=xtensa-esp32 -S . -B your-build-directory
```

Note

如果您想要建置應用程式以進行偵錯，請將 `-DCMAKE_BUILD_TYPE=Debug` 旗標新增到此命令。

如果您想要產生測試應用程式建置檔案，請新增 `-DAFR_ENABLE_TESTS=1` 旗標。

Espressif 提供的程式碼會使用輕量型 IP (LWIP) 堆疊作為預設網路堆疊。要使用 FreeRTOS + TCP 網絡堆棧，添加 `-DAFR_ESP_FREERTOS_TCP` 標誌到 CMake 命令。

要添加 lwIP 與非供應商提供的代碼的依賴關係，請將以下行添加到 CMake 依賴文件，`CMakeLists.txt`，用於您的自定義 WiFi 組件。

```
# Add a dependency on the bluetooth espressif component to the common component
set(COMPONENT_REQUIRES lwip)
```

使用 Make 建置應用程式

1. 將目錄變更為 build 目錄。

2. 使用下列命令以透過 Make 建置應用程式：

```
make all -j4
```

Note

每次在 aws_demos 專案與 aws_tests 專案之間切換時，您都必須使用 cmake 命令產生建置檔案。

在 Windows 上建置 FreeRTOS

在Windows上,您必須為 CMake,否則 CMake 默認為VisualStudio。Espressif正式推薦Ninja構建系統,因為它適用於Windows、Linux和MacOS. 您必須運行 CMake 本機Windows環境中的命令,例如cmd或 PowerShell. 正在運行 CMake 不支持虛擬Linux環境中的命令,例如MSYS2或WSL.

使用 CMake 以生成構建文件,然後使用Make構建應用程序。

要生成演示應用程序的構建文件,請使用 CMake

1. 將目錄變更您 FreeRTOS 下載目錄的根。
2. 使用下列命令來產生建置檔案：

```
cmake -DVENDOR=espressif -DBOARD=esp32_wrover_kit -DCOMPILER=xtensa-esp32 -GNinja -S .  
-B build-directory
```

Note

如果您想要建置應用程式以進行偵錯，請將 -DCMAKE_BUILD_TYPE=Debug 旗標新增到此命令。

如果您想要產生測試應用程式建置檔案，請新增 -DAFR_ENABLE_TESTS=1 旗標。

Espressif 提供的程式碼會使用輕量型 IP (LWIP) 堆疊做為預設網路堆疊。要使用 FreeRTOS +TCP 網絡堆棧,添加 -DAFR_ESP_FREERTOS_TCP 標誌到 CMake 命令。

要添加 lwIP 與非供應商提供的代碼的依賴關係,請將以下行添加到 CMake 依賴文件, CMakeLists.txt,用於您的自定義 WiFi 組件。

```
# Add a dependency on the bluetooth espressif component to the common component  
set(COMPONENT_REQUIRES lwip)
```

建置應用程式

1. 將目錄變更為 build 目錄。
2. 叫用 Ninja 以建置應用程式：

```
ninja
```

或者,使用通用 CMake 界面以構建應用程序:

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

Note

每次在 aws_demos 專案與 aws_tests 專案之間切換時，您都必須使用 cmake 命令產生建置檔案。

刷新和執行 FreeRTOS

使用 Espressif 的 IDF 公用程式 (`freertos/vendors/espressif/esp-idf/tools/idf.py`) 刷新您的電路板、執行應用程式並查看日誌。

若要清除主機板的快閃記憶體，請移至 `freertos` 目錄並使用下列命令：

```
./vendors/espressif/esp-idf/tools/idf.py erase_flash -B build-directory
```

若要將應用程式二進位刷新到您的電路板，請使用 make：

```
make flash
```

您也可以使用 IDF 指令碼來刷新您的電路板：

```
./vendors/espressif/esp-idf/tools/idf.py flash -B build-directory
```

監控：

```
./vendors/espressif/esp-idf/tools/idf.py monitor -p /dev/ttyUSB1 -B build-directory
```

Note

您可以結合這些命令。例如：

```
./vendors/espressif/esp-idf/tools/idf.py erase_flash flash monitor -p /dev/ttyUSB1  
-B build-directory
```

監控雲端的 MQTT 訊息

您可以在 AWS IoT 主控台中使用 MQTT 用戶端來監控裝置傳送至 AWS 雲端的訊息。

使用 AWS IoT MQTT 用戶端訂閱 MQTT 主題

1. 登入 [AWS IoT 主控台](#)。
2. 在導覽窗格中，選擇 Test (測試) 以開啟 MQTT 用戶端。
3. 在訂閱主題中輸入 `iotdemo/#`，然後選擇訂閱主題。

執行低功耗藍牙示範

FreeRTOS 支援[低功耗藍牙](#)連線能力。

您需要在 iOS 或 Android 行動裝置上執行 FreeRTOS 低功耗藍牙 Mobile SDK 示範應用程式，才能跨低功耗藍牙執行 FreeRTOS 示範專案。

設定 FreeRTOS 低功耗藍牙 Mobile SDK 示範應用程式

1. 按照中的說明操作 移動 SDKs 為 FreeRTOS 藍牙設備 在主機上下載和安裝移動平臺的SDK。
2. 依照 [FreeRTOS 低功耗藍牙 Mobile SDK 示範應用程式](#) 中的指示，在您的行動裝置上設定示範行動應用程式。

如需有關如何在您的主機板上執行透過低功耗藍牙的 MQTT 示範，請參閱[透過低功耗藍牙的 MQTT 示範應用程式](#)。

如需有關如何在您的主機板上執行 Wi-Fi 佈建示範的指示，請參閱 [Wi-Fi 佈建示範應用程式](#)。

使用 FreeRTOS 自己 CMake ESP32項目

如果您想 FreeRTOS 自己 CMake 項目，您可以將其設置為子目錄，並將其與您的應用程序一起構建。首先，從 GitHub 或從 FreeRTOS 主控台取得 FreeRTOS 的複本。如果您使用的是 git，也可以使用以下命令將其設定為 git 子模組，以便將來更容易更新。

```
git submodule add -b release https://github.com/aws/amazon-freertos.git freertos
```

如果發行更新的版本，您可以使用這些命令更新本機複本。

```
# Pull the latest changes from the remote tracking branch.  
git submodule update --remote -- amazon-freertos  
# Commit the submodule change because it is pointing to a different revision now.  
git add amazon-freertos  
git commit -m "Update FreeRTOS to a new release"
```

假設您的專案具有以下目錄結構：

```
- freertos (the copy that you obtained from GitHub or the AWS IoT console)  
- src  
  - main.c (your application code)  
- CMakeLists.txt
```

以下是可用來一起建置應用程式與 FreeRTOS 的頂層 CMakeLists.txt 檔案範例。

```
cmake_minimum_required(VERSION 3.13)  
project(freertos_examples)  
add_executable(my_app src/main.c)  
# Tell IDF build to link against this target.  
set(IDF_PROJECT_EXECUTABLE my_app)  
  
# Add FreeRTOS as a subdirectory. AFR_BOARD tells which board to target.  
set(AFR_BOARD espressif.esp32_devkitc CACHE INTERNAL "")  
add_subdirectory(freertos)  
  
# Link against the mqtt library so that we can use it. Dependencies are transitively  
# linked.  
target_link_libraries(my_app PRIVATE AFR::mqtt)
```

要構建項目，請運行以下操作 CMake 命令。確保 ESP32 編譯器位於 PATH 環境變數中。

```
cmake -S . -B build-directory -DCMAKE_TOOLCHAIN_FILE=freertos/tools/cmake/toolchains/  
xtensa-esp32.cmake -GNinja  
cmake --build build
```

若要將應用程式刷入至您的電路板，請執行

```
cmake --build build-directory --target flash
```

使用來自 FreeRTOS 的元件

運行後 CMake，您可以在摘要輸出中找到所有可用組件。此 URL 看起來如下：

```
=====Configuration for FreeRTOS=====
```

```

Version: 201910.00
Git version: 201910.00-388-gccb3612cb7

Target microcontroller:
  vendor: Espressif
  board: ESP32-DevKitC
  description: Development board produced by Espressif that comes in two variants either with ESP-WROOM-32 or ESP32-WROVER module
  family: ESP32
  data ram size: 520KB
  program memory size: 4MB

Host platform:
  OS: Linux-4.15.0-66-generic
  Toolchain: xtensa-esp32
  Toolchain path: /opt/xtensa-esp32-elf
  CMake generator: Ninja

FreeRTOS modules:
  Modules to build: ble, ble_hal, ble_wifi_provisioning, common, crypto, defender, dev_mode_key_provisioning, freertos_plus_tcp, greengrass, https, kernel, mqtt, ota, pkcs11, pkcs11_implementation, platform, secure_sockets, serializer, shadow, tls, wifi
  Enabled by user: ble, ble_hal, ble_wifi_provisioning, defender, greengrass, https, mqtt, ota, pkcs11, pkcs11_implementation, platform, secure_sockets, shadow, wifi
  Enabled by dependency: common, crypto, demo_base, dev_mode_key_provisioning, freertos, freertos_plus_tcp, kernel, pkcs11_mbedtls, secure_sockets_freertos_plus_tcp, serializer, tls, utils
  3rdparty dependencies: http_parser, jsmn, mbedtls, pkcs11, tinycc
  Available demos: demo_ble, demo_ble_numeric_comparison, demo_defender, demo_greengrass_connectivity, demo_https, demo_mqtt, demo_ota, demo_shadow, demo_tcp, demo_wifi_provisioning
  Available tests: =====
=====
```

您可以參考「要建置的模組」清單中的任何元件。若要將它們連結至您的應用程式，請將 AFR::: 命名空間放在名稱前面，例如，AFR:::mqtt、AFR:::ota 等等。

將自訂元件新增至 ESP-IDF

您可以將更多元件新增至 ESP-IDF 建置環境。例如，假設你想要新增一個名為 foo 的元件，並且你的專案看起來像這樣：

```

- freertos
- components
  - foo
    - include
      - foo.h
    - src
      - foo.c
    - CMakeLists.txt
- src
  - main.c
- CMakeLists.txt
```

以下是 CMakeLists 您的組件的.txt文件：

```

# include paths of this components.
set(COMPONENT_ADD_INCLUDEDIRS include)

# source files of this components.
set(COMPONENT_SRCDIRS src)
```

```
# Alternatively, use COMPONENT_SRCS to specify source files explicitly
# set(COMPONENT_SRCS src/foo.c)

# add this components, this will define a CMake library target.
register_component()
```

您還可以使用標準 CMake 功能 `target_link_libraries`。請注意，組件的目標名稱存儲在變量中 `COMPONENT_TARGET`，由 ESP-IDF 定義。

```
# add this component, this will define a CMake library target.
register_component()

# standard CMake function can be used to specify dependencies. ${COMPONENT_TARGET} is
defined
# from esp-idf when you call register_component, by default it's
idf_component_<folder_name>.
target_link_libraries(${COMPONENT_TARGET} PRIVATE AFR::mqtt)
```

對於 ESP 組件，可通過設置 2 個變量來完成 `COMPONENT_REQUIRES` 和 `COMPONENT_PRIV_REQUIRES`。參見 [構建系統\(CMake\)](#) 在 ESP-IDF 編程指南 v3.3。

```
# If the dependencies are from ESP-IDF, use these 2 variables. Note these need to be
# set before calling register_component().
set(COMPONENT_REQUIRES log)
set(COMPONENT_PRIV_REQUIRES lwip)
```

然後，在頂層 `CMakeLists.txt` 檔案中，告訴 ESP-IDF 在哪裡可以找到這些元件。在 `add_subdirectory(freertos)` 前面的任意位置插入以下幾行：

```
# Add some extra components. IDF_EXTRA_COMPONENT_DIRS is a variable used by ESP-IDF
# to collect extra components.
get_filename_component(
    EXTRA_COMPONENT_DIRS
    "components/foo" ABSOLUTE
)
list(APPEND IDF_EXTRA_COMPONENT_DIRS ${EXTRA_COMPONENT_DIRS})
```

根據預設，此元件現在會自動連結至您的應用程式碼。你應該能夠包含它的標頭檔案，並呼叫其定義的函數。

覆寫 FreeRTOS 的組態

目前沒有妥善的方法來重新定義 FreeRTOS 來源樹狀目錄之外的組態。依默認，CMake 會尋找 `freertos/vendors/espressif/boards/esp32/aws_demos/config_files/` 和 `freertos/demos/include/` 目錄。不過，您可以使用因應措施，告訴編譯器首先搜尋其他目錄。例如，您可以為 FreeRTOS 組態新增另一個資料夾：

```
- freertos
- freertos-configs
  - aws_clientcredential.h
  - aws_clientcredential_keys.h
  - iot_mqtt_agent_config.h
  - iot_config.h
- components
- src
- CMakeLists.txt
```

`freertos-configs` 下的檔案是複製自 `freertos/vendors/espressif/boards/esp32/aws_demos/config_files/` 和 `freertos/demos/include/i` 目錄。接著，在您頂層的

CMakeLists.txt 檔案中，將此行加在 add_subdirectory(freertos) 前面，以便編譯器能先搜尋此目錄：

```
include_directories(BEFORE freertos-configs)
```

為 ESP-IDF 提供您自己的 sdkconfig

如果你想提供自己的 sdkconfig.default，您可以設置 CMake 變量 IDF_SDKCONFIG_DEFAULTS，從命令行：

```
cmake -S . -B build-directory -IDF_SDKCONFIG_DEFAULTS=path_to_your_sdkconfig_defaults -DCMAKE_TOOLCHAIN_FILE=freertos/tools/cmake/toolchains/xtensa-esp32.cmake -GNinja
```

如果您沒有為自己的 sdkconfig.default 檔案指定位置，FreeRTOS 將使用位於 *freertos/vendors/espressif/boards/esp32/aws_demos/sdkconfig.defaults* 的預設檔案。

Summary

如果你有一個專案具有名為 foo 的元件，而且你想要覆寫一些組態，則以下是頂層 CMakeLists.txt 檔案的完整範例。

```
cmake_minimum_required(VERSION 3.13)

project(freertos_examples)

add_executable(my_app src/main.c)

# Tell IDF build to link against this target.
set(IDF_PROJECT_EXECUTABLE my_app)

# Add some extra components. IDF_EXTRA_COMPONENT_DIRS is a variable used by ESP-IDF
# to collect extra components.
get_filename_component(
    EXTRA_COMPONENT_DIRS
    "components/foo" ABSOLUTE
)
list(APPEND IDF_EXTRA_COMPONENT_DIRS ${EXTRA_COMPONENT_DIRS})

# Override the configurations for FreeRTOS.
include_directories(BEFORE freertos-configs)

# Add FreeRTOS as a subdirectory. AFR_BOARD tells which board to target.
set(AFR_BOARD espressif.esp32_devkitc CACHE INTERNAL "")
add_subdirectory(freertos)

# Link against the mqtt library so that we can use it. Dependencies are transitively
# linked.
target_link_libraries(my_app PRIVATE AFR::mqtt)
```

Troubleshooting

- 如果你正在運行 macOS 操作系統無法識別您的ESP-WROVER-KIT，請確保未安裝D2XX驅動程序。要卸載它們，請按照 [FTDI驅動程序安裝指南 macOS +](#)。
- ESP-IDF 提供的監控公用程式（並使用 make monitor 叫用）可協助您解碼地址。因此，它可以協助您在應用程式當機時取得一些有意義的回溯。如需詳細資訊，請參閱 Espressif 網站上的 [Automatically Decoding Addresses](#)。
- 也可以啟用 GDBstub 用於與gdb通信，無需任何特殊JTAG硬件。有關更多信息，請參閱 [啟動GDB GDBStub](#) 在Espressif網站上。

- 有關設置 OpenOCD-如果需要基於JTAG硬件的調試,請參見文檔 JTAG調試用於ESP32 可在 [Espressif網站](#).
- 如果 pyserial 無法使用進行安裝 pip 於 macOS,請從下載 [pyserial網站](#).
- 如果電路板持續重設 , 請嘗試在終端機中輸入下列命令 , 以清除快閃記憶體 :

```
make erase_flash
```

- 如果您在執行 idf_monitor.py 時看到錯誤 , 請使用 Python 2.7。
- FreeRTOS 已隨附 ESP-IDF 的必要程式庫 , 因此您不需要從外部下載。如果已設定 IDF_PATH 環境變數 , 建議您先將其清除再建置 FreeRTOS。
- 在 Windows 上 , 系統可能需要 3-4 分鐘來建置專案。您可以在 make 命令上使用 -j4 參數 , 以縮短建置時間 :

```
make flash monitor -j4
```

- 如果您的設備無法連接到 AWS IoT,打開 aws_clientcredential.h 文件,並驗證配置變量是否已在文件中正確定義。clientcredentialMQTT_BROKER_ENDPOINT[] 應該看起來像 1234567890123-ats.iot.us-east-1.amazonaws.com.
- 如果您遵循 [使用 FreeRTOS 自己 CMake ESP32項目 \(p. 48\)](#) 中的步驟 , 並且從連結器中看到未定義的參照錯誤 , 這通常是因為缺少相依程式庫或示範導致。要添加它們,請更新 CMakeLists.txt 文件(根目錄下),使用標準 CMake 功能 target_link_libraries.

如需故障診斷資訊 , 請參閱 [故障診斷入門 \(p. 19\)](#)。

在 Espressif ESP32-DevKitC 和 ESP-WROVER-KIT 上偵錯程式碼

您需要 JTAG 對 USB 纜線。我們會使用 USB 對 MPSSE 纜線 (例如 [FTDI C232HM-DDHSL-0](#))。

ESP-DevKitC JTAG 設定

對於FTDIC232HM-DDHSL-0電纜,這些是到ESP32的連接 DevkitC:

C232HM-DDHSL-0 電線顏色	ESP32 GPIO Pin	JTAG 訊號名稱
棕色 (pin 5)	IO14	TMS
黃色 (pin 3)	IO12	TDI
黑色 (pin 10)	GND	GND
橘色 (pin 2)	IO13	TCK
綠色 (pin 4)	IO15	TDO

ESP-WROVER-KIT JTAG 設定

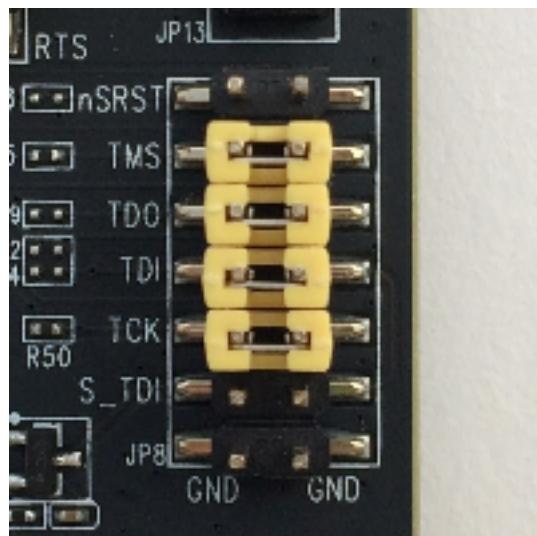
針對 FTDI C232HM-DDHSL-0 纜線 , 下列為 ESP32-WROVER-KIT 的連接方式 :

C232HM-DDHSL-0 電線顏色	ESP32 GPIO Pin	JTAG 訊號名稱
棕色 (pin 5)	IO14	TMS
黃色 (pin 3)	IO12	TDI
橘色 (pin 2)	IO13	TCK

C232HM-DDHSL-0 電線顏色	ESP32 GPIO Pin	JTAG 訊號名稱
綠色 (pin 4)	IO15	TDO

這些資料表是從 [FTDI C232HM-DDHSL-0 datasheet](#) 開發而來。如需詳細資訊，請參閱該資料表中的 C232HM MPSSE Cable Connection and Mechanical Details。

若要在 ESP-WROVER-KIT 上啟用 JTAG，請將跳接器調到 TMS、TDO、TDI、TCK 和 S_TDI 的 pin，如下所示：



在 Windows 中進行除錯

在 Windows 中進行除錯設定

1. 將 FTDI C232HM-DDHSL-0 的 USB 一端接到您的電腦，而另一端則按在 [在 Espressif ESP32-DevKitC 和 ESP-WROVER-KIT 上偵錯程式碼 \(p. 52\)](#) 中所述進行。FTDI C232HM-DDHSL-0 裝置應該會出現在 Universal Serial Bus Controllers (通用序列匯流排控制器) 下方的 Device Manager (裝置管理員) 中。
2. 在通用序列匯流排裝置清單下，請以滑鼠右鍵按一下 C232HM-DDHSL-0 裝置，然後選擇 Properties (屬性)。

Note

裝置可能會列為 USB Serial Port (USB 序列連接埠)。

在屬性視窗中，選擇 Details (詳細資訊) 標籤，以查看裝置的屬性。如果未列出裝置，請安裝適用於 [FTDI C232HM-DDHSL-0 的 Windows 驅動程式](#)。

3. 在 詳情 選項卡，選擇 財產，然後選擇 硬件 IDs。您應會在 Value (值) 欄位中看到類似以下的內容：

```
FTDIBUS\COMPORT&VID_0403&PID_6014
```

在此範例中，廠商 ID 為 0403，產品 ID 為 6014。

驗證這些 IDs 匹配 IDs 英寸 `projects/espressif/esp32/make/aws_demos/esp32_devkitj_v1.cfg`。的 IDs 在以 `ftdi_vid_pid` 後跟供應商ID和產品ID:

```
ftdi_vid_pid 0x0403 0x6014
```

4. 下載 [OpenOCD for Windows](#)。

5. 將檔案解壓縮至 C:\，並新增 C:\openocd-esp32\bin 到您的系統路徑。
6. OpenOCD 需要 libusb，但預設不會在 Windows 中安裝。

安裝 libusb

- a. 請下載 [zadig.exe](#)。
 - b. 運行 zadig.exe。從選項菜單，選擇列出所有設備。
 - c. 從下拉式選單中，選擇 C232HM-DDHSL-0。
 - d. 在目標驅動程式欄位中，選擇綠色箭頭右側的 WinUSB (WinUSB)。
 - e. 從目標驅動程式欄位的下拉式方塊中，選擇箭頭，然後選擇 Install Driver (安裝驅動程式)。選擇 Replace Driver (取代驅動程式)。
7. 開啟命令提示，導覽至 projects/espressif/esp32/make/aws_demos 並執行：

對於 ESP32-WROOM-32 and ESP32-WROVER：

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

對於 ESP32-SOLO-1：

```
openocd.exe -f esp32_devkitj_v1.cfg -f esp-solo-1.cfg
```

將此命令提示保持開啟。

8. 打開新的命令提示符，導航至您的 msys32 目錄，並運行 mingw32.exe。在 mingw32 端子中，導航到 projects/espressif/esp32/make/aws_demos 並運行 make flash monitor。
9. 開啟另一個 mingw32 終端機，導覽至 projects/espressif/esp32/make/aws_demos，並等待您主機板上的示範開始執行。這樣做時，運行 xtensa-esp32-elf-gdb -x gdbinit build/aws_demos.elf。程序應停止於 main 功能。

Note

ESP32 支援最多兩個中斷點。

調試開 macOS

1. 下載 [FTDI驅動程序 macOS](#)。
2. 下載 [OpenOCD](#)。
3. 解壓縮已下載的 .tar 檔案，並將路徑設在 .bash_profile 到 *OCD_INSTALL_DIR*/openocd-esp32/bin 中。
4. 使用以下命令安裝 libusb 於 macOS：

```
brew install libusb
```

5. 請使用下列命令來卸載序列埠驅動程式：

```
sudo kextunload -b com.FTDI.driver.FTDIUSBSerialDriver
```

6. 如果您正在運行 macOS 版本低於 10.9 時，使用以下命令卸載 AppleFTDI 驅動程序：

```
sudo kextunload -b com.apple.driver.AppleUSBFTDI
```

7. 使用下列命令來取得 FTDI 纜線的產品 ID 和廠商 ID。它會列出連接的 USB 裝置：

```
system_profiler SPUSBDataType
```

system_profiler 中的輸出應類似以下內容：

```
DEVICE:  
  
Product ID: product-ID  
Vendor ID: vendor-ID (Future Technology Devices International Limited)
```

8. 打開 projects/espressif/esp32/make/aws_demos/esp32_devkitj_v1.cfg。設備的供應商 ID 和產品 ID 以 ftdi_vid_pid。更改 IDs 來匹配 IDs 從 system_profiler 輸出。
9. 打開終端窗口，導航到 projects/espressif/esp32/make/aws_demos，並使用以下命令運行 OpenOCD。

對於 ESP32-WROOM-32 and ESP32-WROVER：

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

對於 ESP32-SOLO-1：

```
openocd -f esp32_devkitj_v1.cfg -f esp-solo-1.cfg
```

10. 開啟新的終端機，並使用下列命令載入 FTDI 序列埠驅動程式：

```
sudo kextload -b com.FTDI.driver.FTDIUSBSerialDriver
```

11. 導覽至 projects/espressif/esp32/make/aws_demos，然後執行下列命令：

```
make flash monitor
```

12. 開啟另一個新的終端機，導覽至 projects/espressif/esp32/make/aws_demos，並執行下列命令：

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

程式應該會在 main() 停止。

在 Linux 中除錯

1. 下載 [OpenOCD](#)。解壓縮 tarball 並遵循讀我檔中的安裝指示。
2. 使用下列命令，在 Linux 上安裝 libusb：

```
sudo apt-get install libusb-1.0
```

3. 開啟終端機，並輸入 `ls -l /dev/ttyUSB*` 以列出所有連接到您電腦的 USB 裝置。這可協助您檢查作業系統是否能辨識電路板的 USB 連接埠。您應該會看到輸出，如下所示：

```
$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. 登出再登入，並重新啟動電路板的電源，以讓變更生效。在終端機提示中，列出 USB 裝置。確認群組擁有者已從 dialout 變更為 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
```

數字小的 /dev/ttyUSBn 界面用於 JTAG 通訊。另一個界面則會路由到 ESP32 的序列埠 (UART)，並用來上傳程式碼到 ESP32 的快閃記憶體。

- 在終端窗口中，導航到 projects/espressif/esp32/make/aws_demos，並使用以下命令運行 OpenOCD。

對於 ESP32-WROOM-32 和 ESP32-WROVER：

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

對於 ESP32-SOLO-1：

```
openocd -f esp32_devkitj_v1.cfg -f esp-solo-1.cfg
```

- 開啟另一個終端機，導覽至 projects/espressif/esp32/make/aws_demos，並執行下列命令：

```
make flash monitor
```

- 開啟另一個終端機，導覽至 projects/espressif/esp32/make/aws_demos，並執行下列命令：

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

程式應該會在 main() 中停止。

Espressif ESP32-WROOM-32SE 入門

遵循此教學課程，開始使用 Espressif ESP32-WROOM-32SE。若要在 AWS Partner Device Catalog 中向我們的合作夥伴購買，請查看 [ESP32-WROOM-32SE](#)。

Note

ESP32-WROOM-32SE 的 FreeRTOS 連接埠不支援對稱式多工處理 (SMP)。

Overview

本教學課程將指引您完成下列步驟：

- 將主機板連線到主機機器。
- 在主機機器上安裝軟體，以開發和偵錯微控制器面板的內嵌應用程式。
- 將 FreeRTOS 示範應用程式跨編譯為二進位映像。
- 將應用程式二進位映像載入主機板，然後執行應用程式。
- 使用序列連線監控執行中的應用程式並進行偵錯。

Prerequisites

在 Espressif 主機板上開始使用 FreeRTOS 前，您需要先設定 AWS 帳戶和許可。

若要建立 AWS 帳戶，請參閱 [建立並啟用 AWS 帳戶](#)。

若要新增 IAM 使用者到您的 AWS 帳號，請查看 [中的新增使用者](#)。IAM 使用者指南若要授予 IAM 使用者 AWS IoT 和 FreeRTOS 的許可，請將下列 IAM 受管政策連接至 IAM 使用者：

- **AmazonFreeRTOSFullAccess**
允許完整存取所有 IAM 使用者的 FreeRTOS 資源 (`freertos:*`)。
- **AWSIoTFullAccess**
允許完整存取所有 IAM 使用者的 IoT 資源 (`iot:*`)。

將 **AmazonFreeRTOSFullAccess** 政策連接到您的 IAM 使用者

1. 導覽至 [IAM 主控台](#)。
2. 在導覽窗格中，選擇 Users (使用者)。
3. 在搜尋文字方塊中，輸入您的使用者名稱，然後從清單中選擇它。
4. 選擇新增許可。
5. 選擇 Attach existing policies directly (直接連接現有政策)。
6. 在搜尋方塊中，輸入 **AmazonFreeRTOSFullAccess**，從清單中選擇它，然後選擇 Next : (下一步：)。審核。
7. 選擇新增許可。

將 **AWSIoTFullAccess** 政策連接到您的 IAM 使用者

1. 導覽至 [IAM 主控台](#)。
2. 在導覽窗格中，選擇 Users (使用者)。
3. 在搜尋文字方塊中，輸入您的使用者名稱，然後從清單中選擇它。
4. 選擇新增許可。
5. 選擇 Attach existing policies directly (直接連接現有政策)。
6. 在搜尋方塊中，輸入 **AWSIoTFullAccess**，從清單中選擇它，然後選擇 Next : (下一步：)。檢視。
7. 選擇新增許可。

如需 IAM 的詳細資訊，請參閱 [IAM 使用者指南](#)。

如需政策的詳細資訊，請參閱 [IAM 許可與政策](#)。

設定 Espressif 硬體

如需有關設定 ESP32-WROOM-32SE 開發板硬體的資訊，請參閱 [ESP32-DevKitC 入門指南](#)。

Note

請勿按照 Espressif 指南的 Get Started (開始使用) 章節進行。反之，請依照以下步驟進行。

設定開發環境

若要與主機板通訊，您必須下載並安裝工具鏈。

設定工具鏈

若要設定工具鏈，請依照您主機作業系統的指示進行：

- [Windows 工具鏈的標準設定](#)
- [工具鏈的標準設定 macOS](#)

- [Linux 工具鏈的標準設定](#)

Important

當您到達 Next Steps (後續步驟) 下的「取得 ESP-IDF」指示，請停止和返回此頁面上的指示。

請確定 `IDF_PATH` 環境變數已從您的系統中清除，然後再繼續。如果您遵循了 Next Steps (後續步驟) 下的「取得 ESP-IDF」指示，則會自動設定此環境變數。

Note

ESP-IDF 3.3 版本 (FreeRTOS 使用的版本) 不支援 ESP32 編譯器的最新版本。您必須使用與 ESP-IDF 3.3 版本相容的編譯器。請參閱上述連結。若要檢查編譯器的版本，請執行下列命令。

```
xtensa-esp32-elf-gcc --version
```

安裝 CMake

需要 CMake 建置系統，才能為此裝置建置 FreeRTOS 示範和測試應用程式。FreeRTOS 支援 3.13 版和更新版本。

您可以從 CMake [CMake.org](#)下載最新版本的。來源和二進位分發可供使用。

如需將 CMake 與 FreeRTOS 搭配使用的詳細資訊，請前往 [使用 CMake 配 FreeRTOS \(p. 20\)](#)。

建立序列連線

1. 若要在您的主機與 ESP32-WROOM-32SE 之間建立序列式連接，請安裝 CP210x USB to UART Bridge VCP 驅動程式。您可以從 [Silicon Labs](#) 下載這些驅動程式。
2. 請依照步驟來[建立與 ESP32 的序列連線](#)。
3. 在您建立序連接之後，請記下開發板連接的序列連接埠。您在建立示範時會需要它。

下載和設定 FreeRTOS

設定您的環境之後，即可從 FreeRTOS [GitHub](#) 或 主控台[下載 FreeRTOS](#)。如需指示，請查看 [README.md](#) 檔案。

Important

ATECC608A 裝置具有一次性的初始化，會在專案第一次執行時（在對 `C_InitToken` 的呼叫期間）鎖定到裝置上。不過，FreeRTOS 示範專案和測試專案有不同的組態。如果裝置在示範專案組態期間鎖定，並非測試專案中的所有測試都會成功。

1. 遵循[設定 FreeRTOS 示範 \(p. 18\)](#)中的步驟來設定 FreeRTOS 示範專案。略過最後一個步驟 To format your AWS IoT credentials（格式化您的登入資料），然後改為遵循下列步驟。
2. 微晶片提供了多種指令碼工具，協助您設定 ATECC608A 組件。導覽至 `freertos/vendors/microchip/secure_elements/app/example_trust_chain_tool` 目錄並開啟 `README.md` 檔案。

遵循 `README.md` 檔案中的指示，佈建您的裝置。步驟包括：

1. 建立憑證授權機構並向 AWS 註冊。
2. 在 ATECC608A 上產生您的金鑰，並匯出公有金鑰和裝置序號。

3. 產生裝置的憑證並向 AWS 註冊該憑證。
3. 按照[開發人員模式金鑰佈建 \(p. 26\)](#)的指示，將憑證授權機構憑證和裝置憑證載入至裝置。

建置、刷新和執行 FreeRTOS 示範專案

您可以使用 CMake 來製作建置檔案、使用 Make 來建置應用程式二進位檔，以及使用 Espressif IDF 公用程式來刷新您的電路板。

在 Linux 或 FreeRTOS 上建置MacOS

如果您使用的是 Windows，您可以跳到[在 Windows 上建置 FreeRTOS \(p. 59\)](#)。

使用 CMake 來製作建置檔案，然後使用 Make 來建置應用程式。

若要使用 CMake 建立示範應用程式的建置檔案

1. 導覽至 FreeRTOS 下載目錄的根。
2. 在命令列視窗中，輸入下列命令以產生建置檔案。

```
cmake -DVENDOR=espressif -DBOARD=esp32_plus_ecc608a_devkitc -DCOMPILER=xtensa-esp32 -S . -B your-build-directory
```

Note

若要建置應用程式以進行偵錯，請新增 `-DCMAKE_BUILD_TYPE=Debug` 旗標。

若要產生測試應用程式建置檔案，請新增 `-DAFR_ENABLE_TESTS=1` 旗標。

Espressif 提供的程式碼會使用輕量型 IP (LWIP) 堆疊做為預設網路堆疊。若要改用 FreeRTOS +TCP 聯網堆疊，請將 `-DAFR_ESP_FREERTOS_TCP` 旗標新增至 CMake 命令。

若要為非供應商提供的程式碼新增 lwIP 相依性，請將下列幾行新增至您自訂 CMake 元件的 `CMakeLists.txt` 相依性檔案 WiFi。

```
# Add a dependency on the bluetooth espressif component to the common component
set(COMPONENT_REQUIRES lwip)
```

使用 Make 建置應用程式

1. 請前往 build 目錄。
2. 在命令列視窗中，輸入下列命令以使用 Make 建置應用程式。

```
make all -j4
```

Note

每次在 `aws_demos` 專案與 `aws_tests` 專案之間切換時，您都必須使用 `cmake` 命令產生建置檔案。

在 Windows 上建置 FreeRTOS

在 Windows 上，您必須指定 CMake 的建置器。否則，CMake 會預設為 Visual Studio。Espressif 正式建議 Ninja 建置系統，因為它適用於 Windows、Linux 和 MacOS。您必須在原生 Windows 環境（例如 cmd 或 CMake）中執行 PowerShell 命令。不支援在 virtual Linux 環境中執行 CMake 命令，例如 MSYS2 或 WSL。

使用 CMake 來製作建置檔案，然後使用 Make 來建置應用程式。

若要使用 CMake 建立示範應用程式的建置檔案

1. 導覽至 FreeRTOS 下載目錄的根。
2. 在命令列視窗中，輸入下列命令以產生建置檔案。

```
cmake -DVENDOR=espressif -DBOARD=esp32_plus_ecc608a_devkitc -DCOMPILER=xtensa-esp32 -  
GNinja -S . -B your-build-directory
```

Note

若要建置應用程式以進行偵錯，請新增 `-DCMAKE_BUILD_TYPE=Debug` 旗標。

若要產生測試應用程式建置檔案，請新增 `-DAFR_ENABLE_TESTS=1` 旗標。

Espressif 提供的程式碼會使用輕量型 IP (LWIP) 堆疊做為預設網路堆疊。若要改用 FreeRTOS +TCP 聯網堆疊，請將 `-DAFR_ESP_FREERTOS_TCP` 旗標新增至 CMake 命令。

若要為非供應商提供的程式碼新增 lwIP 相依性，請將下列幾行新增至您自訂 CMake 元件的 `CMakeLists.txt` 相依性檔案 WiFi。

```
# Add a dependency on the bluetooth espressif component to the common component  
set(COMPONENT_REQUIRES lwip)
```

建置應用程式

1. 請前往 build 目錄。
2. 在命令列視窗中，輸入下列命令以叫用 Ninja 來建置應用程式。

```
ninja
```

或者，使用一般的 CMake 界面來建置應用程式。

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

Note

每次在 `aws_demos` 專案與 `aws_tests` 專案之間切換時，您都必須使用 `cmake` 命令產生建置檔案。

刷新和執行 FreeRTOS

使用 Espressif 的 IDF 公用程式 (`freertos/vendors/espressif/esp-idf/tools/idf.py`) 刷新您的電路板、執行應用程式並查看日誌。

若要清除電路板的快閃記憶體，請導覽至 `freertos` 目錄並輸入下列命令。

```
./vendors/espressif/esp-idf/tools/idf.py erase_flash -B build-directory
```

若要將應用程式二進位刷新到您的電路板，請使用 `make`。

```
make flash
```

您也可以使用 IDF 指令碼來刷新您的電路板。

```
./vendors/espressif/esp-idf/tools/idf.py flash -B build-directory
```

監控：

```
./vendors/espressif/esp-idf/tools/idf.py monitor -p /dev/ttyUSB1 -B build-directory
```

Tip

您也可以結合這些命令。

```
./vendors/espressif/esp-idf/tools/idf.py erase_flash flash monitor -p /dev/ttyUSB1  
-B build-directory
```

監控 AWS 雲端的 MQTT 訊息

您可以在 AWS IoT 主控台中使用 MQTT 用戶端來監控裝置傳送至 AWS 雲端的訊息。

使用 AWS IoT MQTT 用戶端訂閱 MQTT 主題

1. 登入 [AWS IoT 主控台](#)。
2. 在導覽窗格中，選擇 Test (測試) 以開啟 MQTT 用戶端。
3. 在訂閱主題中輸入 `iotdemo/#`，然後選擇訂閱主題。

Infineon XMC4800 Connectivity Kit 入門 IoT

本教學課程提供 Infineon XMC4800 IoT Connectivity Kit 入門指示。如果您未擁有 Infineon XMC4800 IoT Connectivity Kit，請前往 AWS Partner Device Catalog，向我們的[合作夥伴](#)購買。

如果您想打開與電路板的序列連接以檢視登入和偵錯資訊，除了 XMC4800 IoT Connectivity Kit 之外，您還需要 3.3V 的 USB/序列轉換器。CP2104 是一種常見的 USB/Serial 轉換器，各種電路板均有提供，例如 Adafruit 的 [CP2104 Friend](#)。

開始之前，請您務必要設定 AWS IoT 和 FreeRTOS 下載目錄，才能將裝置連接至 AWS 雲端。如需說明，請參閱[首要步驟 \(p. 14\)](#)。在本教學課程中，FreeRTOS 下載目錄的路徑會以 `freertos` 表示。

Overview

本教學課程包含以下入門步驟的指示：

1. 在主機機器上安裝軟體以對微控制器主機板的內嵌應用程式進行開發和除錯。
2. 將 FreeRTOS 示範應用程式跨編譯為二進位映像。
3. 將應用程式二進位映像載入主機板，然後執行應用程式。
4. 透過序列連線與在開發板上執行的應用程式互動，以便進行監控和除錯。

設定開發環境

FreeRTOS 使用 Infineon 的 DAVE 開發環境來進行 XMC4800 的程式設計。開始之前，您需要下載並安裝 DAVE 和一些 J-Link 驅動程式，以與內建除錯器通訊。

安裝 DAVE

1. 前往 Infineon 的 [DAVE software download](#) 頁面。

2. 選擇適用於您作業系統的 DAVE 套件，並提交您的註冊資訊。註冊 Infineon 之後，您應該會收到一封確認電子郵件，其中包含 .zip 檔案的下載連結。
3. 下載 DAVE 套件 .zip 檔案 (DAVE_**version**_os_**date**.zip)，並解壓縮至您要安裝 DAVE 的位置 (例如 C:\DAVE4)。

Note

部分 Windows 使用者曾回報使用 Windows 檔案總管解壓縮檔案時會發生問題。我們建議您使用第三方程式，例如 7-Zip。

4. 若要啟動 DAVE，請執行解壓縮 DAVE_**version**_os_**date**.zip 資料夾中的可執行檔。

如需詳細資訊，請參閱 [DAVE Quick Start Guide](#)。

安裝 Segger J-Link 驅動程式

若要與 XMC4800 Relax EtherCAT 主機板的主機板除錯探查通訊，您需要 J-Link 軟體和文件套件中包含的驅動程式。您可以從 Segger 的 [J-Link software download](#) 頁面下載 J-Link 軟體和文件套件。

建立序列連線

設定序列連接是選用的步驟，但仍建議您執行。序列連接可讓電路板使用可供您在開發機器上檢視的格式，傳送記錄和除錯資訊。

XMC4800 示範應用程式在 P0.0 和 P0.1 pin 上使用 UART 序列連接，這兩個 pin 在 XMC4800 Relax EtherCAT 電路板的絲印層上有標記。設定序列連接：

1. 將標示 "RX<P0.0" 的 pin 接到您 USB/Serial 轉換器的 "TX" pin。
2. 將標示 "TX>P0.1" 的 pin 接到您 USB/Serial 轉換器的 "RX" pin。
3. 將序列轉換器的接地 pin 接到電路板上其中一個標示 "GND" 的 pin。裝置必須共用共同的接地線。

電源是由 USB 除錯連接埠提供，因此請勿將序列界面卡的正電壓 pin 接到電路板。

Note

有些序列纜線使用 5V 訊號層級。XMC4800 電路板和 Wi-Fi Click 模組需要 3.3V。請不要使用電路板的 IOREF 跳接器，來將電路板的訊號變更為 5V。

連接纜線時，您可以在終端機模擬器 (例如 [GNU Screen](#)) 上開啟序列連接。傳輸速率預設為 115200，含 8 個資料位元、無同位與 1 個停止位元。

建置並執行 FreeRTOS 示範專案

將 FreeRTOS 示範匯入至 DAVE

1. 啟動 DAVE。
2. 在 DAVE 中，選擇 File (檔案)、Import (匯入)。在 Import (匯入) 視窗中，展開 Infineon 資料夾，選擇 DAVE Project (DAVE 專案)，然後選擇 Next (下一步)。
3. 在 Import DAVE Projects (匯入 DAVE 專案) 視窗中，選擇 Select Root Directory (選取根目錄) 和 Browse (瀏覽)，然後選擇 XMC4800 示範專案。

在您解壓縮 FreeRTOS 下載的目錄中，示範專案位於 projects/infineon/xmc4800_iotkit/dave4/aws_demos。

確定取消核取 Copy Projects Into Workspace (複製專案至工作區)。

4. 選擇 Finish (完成)。

aws_demos 專案應會匯入您的工作空間並啟用。

- 在 Project (專案) 功能表中，選擇 Build Active Project (建置作用中的專案)。

確認專案建置時未發生錯誤。

執行 FreeRTOS 示範專案

- 使用 USB 電波將 XMC4800 IoT Connectivity Kit 連接到您的電腦。電路板有兩個 microUSB 連接器。請使用標示 "X101" 的連接器，該連接器旁的電路板絲印層上顯示 Debug。
- 從 Project (專案) 功能表中，選擇 Rebuild Active Project (重建作用中的專案) 以重建 aws_demos，並確認您的組態變更生效。
- 從 Project Explorer (專案瀏覽器) 中，以滑鼠右鍵按一下 aws_demos，選擇 Debug As (除錯工具)，然後選擇 DAVE C/C++ Application (DAVE C/C++ 應用程式)。
- 按兩下 GDB SEGGER J-Link Debugging (GDB SEGGER J-Link 除錯) 以建立除錯確認。選擇 Debug (除錯)。
- 當除錯器停在 main() 中斷點時，請從 Run (執行) 功能表，選擇 Resume (繼續)。

在步驟 4-5 之後，AWS IoT 主控台的 MQTT 用戶端中，應該會顯示您裝置傳送的 MQTT 訊息。如果您使用序列連接，您會看到類似如下的 UART 輸出：

```
0 0 [Tmr Svc] Starting key provisioning...
1 1 [Tmr Svc] Write root certificate...
2 4 [Tmr Svc] Write device private key...
3 82 [Tmr Svc] Write device certificate...
4 86 [Tmr Svc] Key provisioning done...
5 291 [Tmr Svc] Wi-Fi module initialized. Connecting to AP...
6 8046 [Tmr Svc] Wi-Fi Connected to AP. Creating tasks which use network...
7 8058 [Tmr Svc] IP Address acquired [IP Address]
8 8058 [Tmr Svc] Creating MQTT Echo Task...
9 8059 [MQTTEcho] MQTT echo attempting to connect to [MQTT Broker].
...10 23010 [MQTTEcho] MQTT echo connected.
11 23010 [MQTTEcho] MQTT echo test echoing task created.
12 26011 [MQTTEcho] MQTT Echo demo subscribed to iotdemo/# 
13 29012 [MQTTEcho] Echo successfully published 'Hello World 0'
14 32096 [Echoing] Message returned with ACK: 'Hello World 0 ACK'
15 37013 [MQTTEcho] Echo successfully published 'Hello World 1'
16 40080 [Echoing] Message returned with ACK: 'Hello World 1 ACK'
17 45014 [MQTTEcho] Echo successfully published 'Hello World 2'
18 48091 [Echoing] Message returned with ACK: 'Hello World 2 ACK'
19 53015 [MQTTEcho] Echo successfully published 'Hello World 3'
20 56087 [Echoing] Message returned with ACK: 'Hello World 3 ACK'
21 61016 [MQTTEcho] Echo successfully published 'Hello World 4'
22 64083 [Echoing] Message returned with ACK: 'Hello World 4 ACK'
23 69017 [MQTTEcho] Echo successfully published 'Hello World 5'
24 72091 [Echoing] Message returned with ACK: 'Hello World 5 ACK'
25 77018 [MQTTEcho] Echo successfully published 'Hello World 6'
26 80085 [Echoing] Message returned with ACK: 'Hello World 6 ACK'
27 85019 [MQTTEcho] Echo successfully published 'Hello World 7'
28 88086 [Echoing] Message returned with ACK: 'Hello World 7 ACK'
29 93020 [MQTTEcho] Echo successfully published 'Hello World 8'
30 96088 [Echoing] Message returned with ACK: 'Hello World 8 ACK'
31 101021 [MQTTEcho] Echo successfully published 'Hello World 9'
32 104102 [Echoing] Message returned with ACK: 'Hello World 9 ACK'
33 109022 [MQTTEcho] Echo successfully published 'Hello World 10'
34 112047 [Echoing] Message returned with ACK: 'Hello World 10 ACK'
35 117023 [MQTTEcho] Echo successfully published 'Hello World 11'
36 120089 [Echoing] Message returned with ACK: 'Hello World 11 ACK'
37 122068 [MQTTEcho] MQTT echo demo finished.
```

```
38 122068 [MQTTEcho] ----Demo finished----
```

使用 FreeRTOS 建置 CMake 示範

如果您不想使用 IDE 進行 FreeRTOS 開發，您也可以改為使用 CMake 來建置和執行示範應用程式，或使用第三方程式碼編輯器和除錯工具所開發的應用程式。

Note

本節涵蓋在 Windows 上使用 CMake 做為原生建置系統。MingW 如需使用 CMake 與其他作業系統和選項的詳細資訊，請前往 [使用 CMake 配 FreeRTOS \(p. 20\)](#)。（MinGW 是本機 Microsoft Windows 應用程式的簡約開發環境。）

使用 FreeRTOS 建置 CMake 示範

1. 設定 GNU Arm 內嵌式工具鏈。

- a. 從 [Arm 內嵌式工具鏈下載頁面](#)下載 Windows 版本的工具鏈。

Note

我們建議您下載「8-2018-q4-major」以外的版本，因為該版本發生「objcopy」公用程式的錯誤回報。

- b. 開啟下載的工具鏈安裝程式，並依照安裝精靈的指示安裝工具鏈。

Important

在安裝精靈的最終頁面中，選擇 Add path to environment variable (新增路徑至環境變數) 以新增工具鏈路徑到系統路徑環境變數。

2. 安裝 CMake 和 MingW。

如需指示，請查看 [CMake事前準備 \(p. 21\)](#)。

3. 建立資料夾以包含生成的建置檔案 (*build-folder*)。
4. 將目錄變更到您的 FreeRTOS 下載目錄 (*freertos*)，然後使用下列命令來產生建立檔案：

```
cmake -DVENDOR=infineon -DBOARD=xmc4800_iotkit -DCOMPILER=arm-gcc -S . -B build-folder  
-G "MinGW Makefiles" -DAFR_ENABLE_TESTS=0
```

5. 將 建置資料夾變更至 (*build-folder*)，並使用下列命令來建置二進位檔案：

```
cmake --build . --parallel 8
```

此命令會建置輸出二進位 *aws_demos.hex* 到建置目錄。

6. 使用 [JLINK \(p. 62\)](#) 刷新並執行映像。

- a. 從組建資料夾中 (*build-folder*)，使用下列命令來建立刷新指令碼：

```
echo loadfile aws_demos.hex > flash.jlink
```

```
echo r >> flash.jlink
```

```
echo g >> flash.jlink
```

```
echo q >> flash.jlink
```

- b. 使用 JLINK 可執行檔來刷新映像。

```
JLINK_PATH\JLink.exe -device XMC4800-2048 -if SWD -speed auto -CommanderScript flash.jlink
```

您應該可透過使用面板建立的 [序列連線 \(p. 62\)](#) 看到應用程式日誌。

監控雲端的 MQTT 訊息

您可以在 AWS IoT 主控台中使用 MQTT 用戶端來監控裝置傳送至 AWS 雲端的訊息。

使用 AWS IoT MQTT 用戶端訂閱 MQTT 主題

1. 登入 [AWS IoT 主控台](#)。
2. 在導覽窗格中，選擇 Test (測試) 以開啟 MQTT 用戶端。
3. 在訂閱主題中輸入 `iotdemo/#`，然後選擇訂閱主題。

Troubleshooting

若尚未如此做，請務必設定 AWS IoT 和您的 FreeRTOS 下載，將裝置連線到 AWS 雲端。如需說明，請參閱 [首要步驟 \(p. 14\)](#)。

如需 FreeRTOS 入門的一般疑難排解資訊，請參閱 [故障診斷入門 \(p. 19\)](#)。

開始使用 Infineon OPTIGA Trust X 和 XMC4800 IoT Connectivity Kit

本教學課程提供 Infineon OPTIGA Trust X 安全元素和 XMC4800 IoT Connectivity Kit 入門指示。與 [Infineon XMC4800 Connectivity Kit 入門 IoT \(p. 61\)](#) 教學相比，本指南會示範如何使用 Infineon OPTIGA Trust X 安全元素提供安全登入資料。

您需要下列硬體：

1. 主機 MCU - Infineon XMC4800 IoT Connectivity Kit，請前往 AWS Partner Device Catalog，向我們的 [合作夥伴](#) 購買。
2. 安全性延伸套件：
 - 安全元素 - Infineon OPTIGA Trust X。

請前往 AWS Partner Device Catalog，向我們的 [合作夥伴](#) 購買。

- 個人化面板 - Infineon OPTIGA 個人化面板。
- 轉接器板 - Infineon MyIoT 轉接器。

若要遵循此處的步驟，您必須使用面板開啟序列連線，才能檢視記錄和偵錯資訊。（其中一個步驟需要您從開發板複製序列偵錯輸出的公有金鑰，並將其貼入檔案。）若要這樣做，您需要一個 3.3V 的 USB/序列轉換器以及 XMC4800 IoT Connectivity Kit。已知此示範使用 [JBtek EL-PN-47310126](#) USB/序列轉換器。您也需要三條公對公的電波（適用於接收 (RX)、傳輸 (TX) 和地線 (GND)），將序列表接至 Infineon MyIoT 轉接器板。

開始之前，請您務必要設定 AWS IoT 和 FreeRTOS 下載目錄，才能將裝置連接至 AWS 雲端。如需說明，請參閱 [選項 #2：產生內建私有金鑰 \(p. 26\)](#)。在本教學課程中，FreeRTOS 下載目錄的路徑會以 `freertos` 表示。

Overview

本教學課程包含以下步驟：

1. 在主機機器上安裝軟體，以開發和偵錯微控制器面板的內嵌應用程式。
2. 將 FreeRTOS 示範應用程式跨編譯為二進位映像。
3. 將應用程式二進位映像載入主機板，然後執行應用程式。
4. 如需監控與偵錯，請透過序列連線與在面板上執行的應用程式互動。

設定開發環境

FreeRTOS 使用 Infineon 的 DAVE 開發環境來進行 XMC4800 的程式設計。開始之前，請下載並安裝 DAVE 和一些 J-Link 驅動程式，以與內建除錯器通訊。

安裝 DAVE

1. 前往 Infineon 的 [DAVE software download](#) 頁面。
2. 選擇適用於您作業系統的 DAVE 套件，並提交您的註冊資訊。註冊之後，您應該會收到一封確認電子郵件，其中包含 .zip 檔案的下載連結。
3. 下載 DAVE 套件 .zip 檔案 (`DAVE_version_os_date.zip`)，並解壓縮至您要安裝 DAVE 的位置 (例如 `C:\DAVE4`)。

Note

部分 Windows 使用者曾回報使用 Windows 檔案總管解壓縮檔案時會發生問題。我們建議您使用第三方程式，例如 7-Zip。

4. 若要啟動 DAVE，請執行解壓縮 `DAVE_version_os_date.zip` 資料夾中的可執行檔。

如需詳細資訊，請參閱 [DAVE Quick Start Guide](#)。

安裝 Segger J-Link 驅動程式

若要與 XMC4800 IoT Connectivity Kit 的機上除錯探查通訊，您需要 J-Link 軟體和文件套件中包含的驅動程式。您可以從 Segger 的 [J-Link software download](#) 頁面下載 J-Link 軟體和文件套件。

建立序列連線

將 USB/Serial 轉換器接到 Infineon Shield2Go 轉接器。這可讓您的主機板以您可以在開發機器上檢視的形式傳送登入和偵錯資訊。設定序列連接：

1. 將 RX pin 接到您 USB/序列轉換器的 TX pin。
2. 將 TX pin 接到您 USB/序列轉換器的 RX pin。
3. 將序列轉換器的接地 pin 接到面板其中一個 GND pin。裝置必須共用共同的接地線。

電源是由 USB 除錯連接埠提供，因此請勿將序列界面卡的正電壓 pin 接到電路板。

Note

有些序列纜線使用 5V 訊號層級。XMC4800 電路板和 Wi-Fi Click 模組需要 3.3V。請不要使用電路板的 IOREF 跳接器，來將電路板的訊號變更為 5V。

連接纜線時，您可以在終端機模擬器 (例如 [GNU Screen](#)) 上開啟序列連接。傳輸速率預設為 115200，含 8 個資料位元、無同位與 1 個停止位元。

監控雲端的 MQTT 訊息

您可以在 AWS IoT 主控台中使用 MQTT 用戶端來監控裝置傳送至 AWS 雲端的訊息。您可能希望在裝置執行示範專案之前，先設定此項目。

使用 AWS IoT MQTT 用戶端訂閱 MQTT 主題

1. 登入 [AWS IoT 主控台](#)。
2. 在導覽窗格中，選擇 Test (測試) 以開啟 MQTT 用戶端。
3. 在訂閱主題中輸入 `iotdemo/#`，然後選擇訂閱主題。

建置並執行 FreeRTOS 示範專案

將 FreeRTOS 示範匯入至 DAVE

1. 啟動 DAVE。
2. 在 DAVE 中，選擇 File (檔案)，然後選擇 Import (匯入)。展開 Infineon 資料夾、選擇 DAVE Project (DAVE 專案)，然後選擇 Next (下一步)。
3. 在 Import DAVE Projects (匯入 DAVE 專案) 視窗中，選擇 Select Root Directory (選取根目錄) 和 Browse (瀏覽)，然後選擇 XMC4800 示範專案。

在您解壓縮 FreeRTOS 下載的目錄中，示範專案位於 `projects/infineon/xmc4800_plus_optiga_trust_x/dave4/aws_demos/dave4`。

確定清除 Copy Projects Into Workspace (複製專案至工作區)。

4. 選擇 Finish (完成)。

`aws_demos` 專案應會匯入您的工作空間並啟用。

5. 在 Project (專案) 功能表中，選擇 Build Active Project (建置作用中的專案)。

確認專案建置時未發生錯誤。

執行 FreeRTOS 示範專案

1. 從 Project (專案) 功能表中，選擇 Rebuild Active Project (重建作用中的專案) 以重建 `aws_demos`，並確認您的組態變更生效。
2. 從 Project Explorer (專案瀏覽器) 中，以滑鼠右鍵按一下 `aws_demos`，選擇 Debug As (除錯工具)，然後選擇 DAVE C/C++ Application (DAVE C/C++ 應用程式)。
3. 按兩下 GDB SEGGER J-Link Debugging (GDB SEGGER J-Link 除錯) 以建立除錯確認。選擇 Debug (除錯)。
4. 當除錯器停在 `main()` 中斷點時，請從 Run (執行) 功能表，選擇 Resume (繼續)。

此時，請繼續執行選項 #2：產生內建私有金鑰 (p. 26) 中的擷取公用金鑰步驟。完成所有步驟後，請移至 AWS IoT 主控台。您之前設定的 MQTT 用戶端應該會顯示您裝置傳送的 MQTT 訊息。透過裝置的序列連線，您應該會在 UART 輸出中看到與下雷同的內容：

```
0 0 [Tmr Svc] Starting key provisioning...
1 1 [Tmr Svc] Write root certificate...
2 4 [Tmr Svc] Write device private key...
3 82 [Tmr Svc] Write device certificate...
4 86 [Tmr Svc] Key provisioning done...
```

```
5 291 [Tmr Svc] Wi-Fi module initialized. Connecting to AP...
.6 8046 [Tmr Svc] Wi-Fi Connected to AP. Creating tasks which use network...
7 8058 [Tmr Svc] IP Address acquired [IP Address]
8 8058 [Tmr Svc] Creating MQTT Echo Task...
9 8059 [MQTTEcho] MQTT echo attempting to connect to [MQTT Broker].
...10 23010 [MQTTEcho] MQTT echo connected.
11 23010 [MQTTEcho] MQTT echo test echoing task created.
.12 26011 [MQTTEcho] MQTT Echo demo subscribed to iotdemo/# 
13 29012 [MQTTEcho] Echo successfully published 'Hello World 0'
.14 32096 [Echoing] Message returned with ACK: 'Hello World 0 ACK'
.15 37013 [MQTTEcho] Echo successfully published 'Hello World 1'
16 40080 [Echoing] Message returned with ACK: 'Hello World 1 ACK'
.17 45014 [MQTTEcho] Echo successfully published 'Hello World 2'
.18 48091 [Echoing] Message returned with ACK: 'Hello World 2 ACK'
.19 53015 [MQTTEcho] Echo successfully published 'Hello World 3'
.20 56087 [Echoing] Message returned with ACK: 'Hello World 3 ACK'
.21 61016 [MQTTEcho] Echo successfully published 'Hello World 4'
22 64083 [Echoing] Message returned with ACK: 'Hello World 4 ACK'
.23 69017 [MQTTEcho] Echo successfully published 'Hello World 5'
.24 72091 [Echoing] Message returned with ACK: 'Hello World 5 ACK'
.25 77018 [MQTTEcho] Echo successfully published 'Hello World 6'
26 80085 [Echoing] Message returned with ACK: 'Hello World 6 ACK'
.27 85019 [MQTTEcho] Echo successfully published 'Hello World 7'
.28 88086 [Echoing] Message returned with ACK: 'Hello World 7 ACK'
.29 93020 [MQTTEcho] Echo successfully published 'Hello World 8'
.30 96088 [Echoing] Message returned with ACK: 'Hello World 8 ACK'
.31 101021 [MQTTEcho] Echo successfully published 'Hello World 9'
32 104102 [Echoing] Message returned with ACK: 'Hello World 9 ACK'
.33 109022 [MQTTEcho] Echo successfully published 'Hello World 10'
.34 112047 [Echoing] Message returned with ACK: 'Hello World 10 ACK'
.35 117023 [MQTTEcho] Echo successfully published 'Hello World 11'
36 120089 [Echoing] Message returned with ACK: 'Hello World 11 ACK'
.37 122068 [MQTTEcho] MQTT echo demo finished.
38 122068 [MQTTEcho] ----Demo finished----
```

使用 FreeRTOS 建置 CMake 示範

本節涵蓋在 Windows 上使用 CMake 做為原生建置系統。MingW 如需使用 CMake 與其他作業系統和選項的詳細資訊，請前往 [使用 CMake 配 FreeRTOS \(p. 20\)](#)。（MinGW 是本機 Microsoft Windows 應用程式的簡約開發環境。）

如果您不想使用 IDE 來開發 FreeRTOS，可以使用 CMake 來建置和執行示範應用程式，或使用第三方程式碼編輯器和除錯工具開發的應用程式。

使用 FreeRTOS 建置 CMake 示範

1. 設定 GNU Arm 內嵌式工具鏈。

- 從 [Arm 內嵌式工具鏈下載頁面](#)下載 Windows 版本的工具鏈。

Note

因為 objcopy 公用程式有[錯誤回報](#)，因此建議您下載 "8-2018-q4-major" 以外的版本。

- 開啟下載的工具鏈安裝程式，然後依照精靈中的指示進行。
- 在安裝精靈的最終頁面中，選擇 Add path to environment variable (新增路徑至環境變數) 以新增工具鏈路徑到系統路徑環境變數。

2. 安裝 CMake 和 MingW。

如需指示，請查看[CMake事前準備 \(p. 21\)](#)。

- 建立資料夾以包含生成的建置檔案 (*build-folder*)。
- 將目錄變更到您的 FreeRTOS 下載目錄 (*freertos*)，然後使用下列命令來產生建立檔案：

```
cmake -DVENDOR=infineon -DBOARD=xmc4800_plus_optiga_trust_x -DCOMPILER=arm-gcc -S . -B build-folder -G "MinGW Makefiles" -DAFR_ENABLE_TESTS=0
```

- 將建置資料夾變更至建置資料夾 (*build-folder*)，並使用下列命令來建置二進位檔案：

```
cmake --build . --parallel 8
```

此命令會建置輸出二進位 `aws_demos.hex` 到建置目錄。

- 使用 [JLINK \(p. 62\)](#) 刷新並執行映像。

- 從組建資料夾中 (*build-folder*)，使用下列命令來建立刷新指令碼：

```
echo loadfile aws_demos.hex > flash.jlink
echo r >> flash.jlink
echo g >> flash.jlink
echo q >> flash.jlink
```

- 使用 `JLINK` 可執行檔來刷新映像。

```
JLINK_PATH\JLink.exe -device XMC4800-2048 -if SWD -speed auto -CommanderScript flash.jlink
```

您應該可透過使用面板建立的 [序列連線 \(p. 62\)](#) 看到應用程式日誌。繼續執行 [選項 #2：產生內建私有金鑰 \(p. 26\)](#) 中的擷取公用金鑰步驟。完成所有步驟之後，請移至 AWS IoT 主控台。您之前設定的 MQTT 用戶端應該會顯示您裝置傳送的 MQTT 訊息。

Troubleshooting

如需一般疑難排解資訊，請參閱 [故障診斷入門 \(p. 19\)](#)。

入門套件入門MW32xAWS IoT

入門套件是一種開發套件，其以 88MW320/88MW322 為基礎，是 NXP 中最新整合的 Cortex M4 微控制器，已將 802.11b/g/n Wi-Fi 整合在單一微控制器晶片中。AWS IoT 開發套件通過 FCC 認證。如需詳細資訊，請參考 [Partner Device CatalogAWS](#)，向我們的合作夥伴購買。88MW320/88MW322 模組也通過 FCC 認證，可供自訂和批發銷售。

這個入門指南會示範如何在主機電腦上將應用程式與軟體開發套件交互編譯在一起，然後使用軟體開發套件隨附的工具將生成的二進位檔案載入電路板。當應用程式開始在電路板上執行時，您可以從主機電腦上的 Serial 主控台偵錯或與其互動。

Ubuntu 16.04 是支援開發和偵錯的主機平台。您可能可以使用其他平台，但並未正式支援。您必須具有在主機平台上安裝軟體的許可。建置軟體開發套件所需的下列外部工具：

- Ubuntu 16.04 主機平台
- ARM 工具鏈版本 4_9_2015q3
- Eclipse 4.9.0 IDE

需要 ARM 工具鏈才能跨編譯您的應用程式和軟體開發套件。開發套件利用工具鏈的最新版本來最佳化影像使用量，並將更多功能納入較少的空間中。本指南假設您使用工具鏈的 4_9_2015q3 版。不建議使用較舊版本的工具鏈。開發套件已預先使用 Wireless Microcontroller Demo 專案韌體進行快閃處理。

主題

- 設定您的硬體 (p. 70)
- 設定開發環境 (p. 70)
- 建置並執行 FreeRTOS 示範專案 (p. 73)
- Debugging (p. 83)
- Troubleshooting (p. 84)

設定您的硬體

使用 Mini-USB 到 USB 傳輸串,將 MW32x 電路板連接到筆記型電腦。將迷你 USB 連接器連接至電路板上的唯一迷你 USB 連接器。您不需要變更跳接器。

如果電路板連接到筆記型電腦或桌上型電腦,則不需要外部電源供應器。

此 USB 連接提供下列項目:

- 主控台存取電路板。virtty/com 連接埠是在可用來存取 主控台的開發主機登記。
- JTAG 存取電路板。這可以用來將韌體映像載入或卸載到開發板的 RAM 或刷新,或用於偵錯。

設定開發環境

基於開發目的,最低要求是 ARM 工具鏈和隨附的工具。下列各節提供 ARM 工具鏈設定的詳細資訊。

GNU 工具鏈

開發套件正式支援 GCC 編譯器工具鏈。GNU ARM 的跨編譯器工具鏈可在 [GNU Arm Embedded Toolchain 4.9-2015-q3-update](#) 取得。

建置系統預設為使用 GNU 工具鏈。Makefiles 假設 GNU 編譯器工具鏈二進位檔可在使用者的 PATH 上取得,並從 Makefiles 叫用。Makefile 也會假設 GNU 工具鏈二進位檔的檔案名稱字首為 arm-none-eabi-。

GCC 工具鏈可搭配 GDB 使用 OpenOCD (與軟體開發套件綁定) 進行偵錯。這可提供與 JTAG 連接的軟體。

我們建議使用 gcc-arm-embedded 工具鏈的 4_9_2015q3 版。

Linux 工具鏈設定程序

請依照下列步驟,在 Linux 中設定 GCC 工具鏈。

1. 下載 [GNU Arm Embedded Toolchain 4.9-2015-q3-update](#) 提供的工具鏈 tarball。此檔案為 gcc-arm-none-eabi-4_9-2015q3-20150921-linux.tar.bz2。
2. 將檔案複製到您選擇的目錄。請確定此資料夾名稱中無空格。
3. 使用以下命令將檔案解壓縮。

```
tar -vxf filename
```

4. 將已安裝工具鏈的路徑新增至系統 PATH。例如,在位於 .profile 的 /home/*user-name* 檔案結尾附加下列行。

```
PATH=$PATH:path to gcc-arm-none-eabit-4_9_2015_q3/bin
```

Note

Ubuntu 的新發行版本可能隨附 Debian 版本的 GCC Cross Compiler。若是如此,您必須移除原生 Cross Compiler,並遵循上述設定程序。

使用 Linux 開發主機

可使用任何現代 Linux 桌面分發,例如 Ubuntu 或 Fedora。不過,我們建議您升級至最新的版本。下列步驟已驗證為可在 Ubuntu 16.04 上運作,並假設您使用該版本。

安裝套件

開發套件包含指令碼,可讓您在新設定 Linux 機器上快速設定您的開發環境。指令碼會嘗試自動偵測機器類型,並安裝適當的軟體,包括 C 程式庫、USB 程式庫、FTDI 程式庫、遞迴、Python 和 Lollate。在本節中,一般資料夾名稱 **amzsdk_bundle-x.y.z** 表示 AWS 開發套件根資料夾。實際的目標名稱可能不同。您必須具有根權限。

- 導覽至 **amzsdk_bundle-x.y.z/** 並執行此命令。

```
./lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/bin/installpkgs.sh
```

避免 sudo

在本指南中,flashprog 操作使用 flashprog.py 指令碼來刷新電路板的 NVMe,如下所述。同樣地,ramload 操作使用 ramload.py 指令碼將韌體映像從主機直接複製到微控制器的 RAM,而無須刷入 NAND。

您可以設定您的 Linux 開發主機執行 flashprog 和 ramload 操作,而不需要每次都使用 sudo 命令。若要進行這項動作,請執行以下命令。

```
./lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/bin/perm_fix.sh
```

Note

您必須以此方式設定您的 Linux 開發主機許可,以確保順暢的 Eclipse IDE 體驗。

設定序列主控台

將 USB 電波插入 Linux 主機 USB 插槽。這會觸發裝置的偵測。您應該會在 /var/log/messages 檔案中或在執行 dmesg 命令之後看到類似以下的訊息。

```
Jan 6 20:00:51 localhost kernel: usb 4-2: new full speed USB device using uhci_hcd and
address 127
Jan 6 20:00:51 localhost kernel: usb 4-2: configuration #1 chosen from 1 choice
Jan 6 20:00:51 localhost kernel: ftdi_sio 4-2:1.0: FTDI USB Serial Device converter
detected
Jan 6 20:00:51 localhost kernel: ftdi_sio: Detected FT2232C
Jan 6 20:00:51 localhost kernel: usb 4-2: FTDI USB Serial Device converter now attached to
ttyUSB0
Jan 6 20:00:51 localhost kernel: ftdi_sio 4-2:1.1: FTDI USB Serial Device converter
detected
Jan 6 20:00:51 localhost kernel: ftdi_sio: Detected FT2232C
Jan 6 20:00:51 localhost kernel: usb 4-2: FTDI USB Serial Device converter now attached to
ttyUSB1
```

確認已建立兩個 ttyUSB 裝置。第二個 ttyUSB 是序列主控台。在上述範例中,名為「ttyUSB1」。

在本指南中,我們使用 Minicom 來查看序列主控台輸出。您也可以使用其他序列程式,例如 putty。執行下列命令,在設定模式中執行 Minicom。

```
minicom -s
```

在 Minicom 中,導覽至 Serial Port Setup (序列埠設定) 並擷取以下設定。

```
| A - Serial Device : /dev/ttyUSB1
| B - Lockfile Location : /var/lock
| C - Callin Program :
| D - Callout Program :
| E - Bps/Par/Bits : 115200 8N1
| F - Hardware Flow Control : No
| G - Software Flow Control : No
```

您可以在 Minicom 中儲存這些設定以供日後使用。Minicom 視窗現在會顯示來自序列主控台的訊息。

選擇序列主控台視窗,然後按下 Enter 鍵。這會在螢幕上顯示井字號 (#)。

Note

開發板包含 FTDI 晶片裝置。FTDI 裝置會公開主機的兩個 USB 界面。第一個界面與 MCU 的 JTAG 功能相關聯,第二個界面與 MCU 的實體 UARTx 連接埠相關聯。

安裝 OpenOCD

OpenOCD 是提供嵌入式目標裝置除錯、系統程式設計和邊界掃描測試的軟體。

需要 OpenOCD 0.9 版。Eclipse 功能也需要此元件。如果舊版 (如 0.7 版) 已安裝在 Linux 主機上,請使用您目前使用之 Linux 發行版本的適當命令移除該儲存庫。

執行標準 Linux 命令來安裝 OpenOCD,

```
apt-get install openocd
```

如果上述命令未安裝 0.9 版或更新版本,請使用下列程序來下載和編譯 openocd 來源碼。

安裝 OpenOCD

1. 執行以下命令來安裝 libusb-1.0。

```
sudo apt-get install libusb-1.0
```

2. 從 <http://openocd.org/> 下載 openocd 0.9.0 原始程式碼。

3. 解壓縮 openocd,並導覽至您解壓縮該的 directory。

4. 使用下列命令設定 openocd。

```
./configure --enable-ftdi --enable-jlink
```

5. 執行 make 公用程式以編譯 openocd。

```
make install
```

設定 Eclipse

Note

本節假設您已完成避免 sudo (p. 71) 中的步驟。

Eclipse 是應用程式開發和除錯的首選 IDE。它提供了豐富的使用者易用 IDE 與整合的除錯支援,包括執行緒感知除錯。本節描述所有受支援之開發主機的常見 Eclipse 設定。

設定 Eclipse

1. 下載並安裝 Java Run Time Environment (JRE)。

Eclipse 需要您安裝 JRE。我們建議您先安裝此項目，不過它在您安裝 Eclipse 後即可安裝。JRE 版本 (32/64 位元) 必須符合 Eclipse (32/64 位元) 版本。您可以從 Oracle 網站的 [Java SE Runtime Environment 8 下載](#) JRE。

2. 從 <http://www.eclipse.org> 下載並安裝「Eclipse IDE for C/C++ Developers」。支援 Eclipse 4.9.0 版或更新版本。安裝只需要您解壓縮下載的封存。您執行平台特定的 Eclipse 可執行檔來開始應用程式。

建置並執行 FreeRTOS 示範專案

執行 FreeRTOS 示範專案的方式有兩種：

- 使用命令列。
- 使用 Eclipse IDE。

本主題涵蓋這兩種選項。

Provisioning

- 視您是否使用測試或示範應用程式而定，在下列其中一個檔案設定配置資料：

- ./tests/common/include/aws_clientcredential.h
- ./demos/common/include/aws_clientcredential.h

例如：

```
#define clientcredentialWIFI_SSID "Wi-Fi SSID"
#define clientcredentialWIFI_PASSWORD "Wi-Fi password"
#define clientcredentialWIFI_SECURITY "Wi-Fi security"
```

Note

您可以輸入下列 Wi-Fi 安全值：

- eWiFiSecurityOpen
- eWiFiSecurityWEP
- eWiFiSecurityWPA
- eWiFiSecurityWPA2

SSID 和密碼應用雙引號括住。

使用命令列建置並執行 FreeRTOS 示範

1. 使用下列命令來開始建置示範應用程式。

```
cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S . -Bbuild -
DAFR_ENABLE_TESTS=0
```

請確定您取得與下列範例相同的輸出。

```
marvell@pe-lt586:amzsdk$ cmake -DVENDOR=marvell -DBOARD=mw300_rd -DCOMPILER=arm-gcc -S . -Bbuild -DAFR_ENABLE_TESTS=0
=====
Configuration for Amazon FreeRTOS=====
Version:          v1.2.4
Git version:      AMZSDK_V1.2.r6.p1-12-gdd17d10

Target microcontroller:
  vendor:          Marvell
  board:           mw300_rd
  description:    Marvell Board for AmazonFreeRTOS
  family:          Wireless Microcontroller
  data ram size:  512KB
  program memory size: 2MB

Host platform:
  OS:              Linux-4.15.0-47-generic
  Toolchain:       arm-gcc
  Toolchain path: /home/marvell/Software/gcc-arm-none-eabi-4_9-2015q3
  CMake generator: Unix Makefiles

Amazon FreeRTOS modules:
  Modules to build: kernel, freertos_plus_tcp, bufferpool, crypto, greengrass, mqtt,
  ota, pkcs11, secure_sockets, shadow, tls, wifi
  Disabled by user:
  Disabled by dependency: posix

  Available demos:   demo_key_provisioning, demo_logging, demo_mqtt_hello_world, demo_mqtt_pubsub, demo_tcp, demo_shadow, demo_greengrass, demo_ota
  Available tests:   test_crypto, test_greengrass, test_mqtt, test_ota, test_pkcs11,
  test_secure_sockets, test_tls, test_shadow, test_wifi, test_memory
=====

-- Configuring done
-- Generating done
-- Build files have been written to: /home/marvell/gerrit/amzsdk/build
marvell@pe-lt586:amzsdk$
```

2. 導覽至建置資料夾。

```
cd build
```

3. 執行 make 公用程式以建置應用程式。

```
make all -j4
```

請確定您取得與下圖相同的輸出:

```
[ 92%] Building C object CMakeFiles/afr_ota.dir/lib/third_party/tinycbor/cborencoder.c.obj
[ 93%] Building C object CMakeFiles/afr_ota.dir/lib/third_party/tinycbor/cborencoder_close_container_checked.c.obj
[ 93%] Building C object CMakeFiles/afr_ota.dir/lib/third_party/tinycbor/cborerrorstrings.c.obj
[ 93%] Building C object CMakeFiles/afr_ota.dir/lib/third_party/tinycbor/cborparser.c.obj
[ 94%] Building C object CMakeFiles/afr_ota.dir/lib/third_party/tinycbor/cborparser_dup_string.c.obj
[ 94%] Building C object CMakeFiles/afr_ota.dir/lib/third_party/tinycbor/cborpretty.c.obj
[ 94%] Building C object CMakeFiles/afr_ota.dir/lib/third_party/jsmn/jsmn.c.obj
[ 95%] Building C object CMakeFiles/afr_ota.dir/demos/common/logging/aws_logging_task_dynamic_buffers.c.obj
[ 95%] Building C object CMakeFiles/afr_ota.dir/demos/common/demo_runner/aws_demo_runner.c.obj
[ 95%] Linking C static library afr_ota.a
[ 95%] Built target afr_ota
[ 96%] Linking C executable aws_demos.axf
[100%] Built target aws_demos
marvell@pe-lt586:build$
```

4. 使用下列命令來建置測試應用程式。

```
cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S . -Bbuild -  
DAFR_ENABLE_TESTS=1  
cd build  
make all -j4
```

每次在 cmake 和 aws_demos project 之間切換時,請執行 aws_tests project 命令。

5. 將韌體映像寫入開發電路板的快閃記憶體。韌體將在重設開發電路板後執行。您必須先建置軟體開發套件,才能將映像存至微控制器。

- a. 在您刷新韌體映像前,請使用常見的元件 Layout 和 Boot2 準備開發板的快閃。請使用以下命令。

```
cd amzsdk_bundle-x.y.z  
.vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py -l ./vendors/marvell/  
WMSDK/mw320/sdk/tools/OpenOCD/mw300/layout.txt --boot2 ./vendors/marvell/WMSDK/  
mw320/boot2/bin/boot2.bin
```

命令會起始以下項目:flashprog

- Layout (配置)– 第一次指示 Flashprog 公用程式將配置寫入至快閃記憶體。配置類似於快閃記憶體的分割區資訊。預設配置位於 /lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/OpenOCD/mw300/layout.txt。
- Boot2 – 這是 WMSDK 所使用的開機載入器。命令也會將開機載入器寫入至快閃記憶體。flashprog開機載入器的任務是在微型控制器的韌體映像刷入後將其載入。請確定您取得與下圖相同的輸出。

```
target state: halted  
target halted due to debug-request, current mode: Thread  
xPSR: 0x01000000 pc: 0x00007f14 msp: 0x20001000  
29088 bytes written at address 0x00100000  
downloaded 29088 bytes in 0.245458s (115.728 KiB/s)  
verified 29088 bytes in 0.350004s (81.160 KiB/s)  
semihosting is enabled  
  
Flashprog version: 2.1.0  
Erasing primary flash...done  
Writing new flash layout...done  
Writing "boot2" @0x0 (primary)...done  
semihosting: *** application exited ***  
Flashprog Complete  
shutdown command invoked  
  
target state: halted  
target halted due to breakpoint, current mode: Thread  
xPSR: 0x21000000 pc: 0x00100658 msp: 0x0015ffe4, semihosting
```

- b. 韌體會針對其功能使用 Wi-Fi 晶片組,而且 Wi-Fi 晶片組有自己的韌體,同時也必須存在於快閃記憶體中。使用 flashprog.py 公用程式刷新 Wi-Fi 韌體的方式與刷新 Boot2 開機載入器和 MCU 韌體相同。請使用以下命令來刷新 Wi-Fi 韌體。

```
cd amzsdk_bundle-x.y.z  
.vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py --wififw ./vendors/  
marvell/WMSDK/mw320/wifi-firmware/mw30x/mw30x_uapsta_W14.88.36.p135.bin
```

請確定命令的輸出與下圖類似。

```
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00007f14 msp: 0x20001000
29088 bytes written at address 0x00100000
downloaded 29088 bytes in 0.245498s (115.709 KiB/s)
verified 29088 bytes in 0.350229s (81.108 KiB/s)
semihosting is enabled

Flashprog version: 2.1.0
Writing "wififw" @0x12a000 (primary).....done
semihosting: *** application exited ***
Flashprog Complete
shutdown command invoked

target state: halted
target halted due to breakpoint, current mode: Thread
xPSR: 0x21000000 pc: 0x00100658 msp: 0x0015ffe4, semihosting
```

- c. 使用下列命令來刷新 MCU 韌體。

```
cd amzsdk_bundle-x.y.z
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py --mcufw build/cmake/
vendors/marvell/mw300_rd/aws_demos.bin -r
```

- d. 重設開發板。您應該會看到示範應用程式的 logs。
e. 若要執行測試應用程式,請刷新位於相同資料夾的 aws_tests.bin 二進位檔。

```
cd amzsdk_bundle-x.y.z
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py --mcufw build/cmake/
vendors/marvell/mw300_rd/aws_tests.bin -r
```

命令輸出應該會類似下圖所示的輸出。

```
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00007f14 msp: 0x20001000
29088 bytes written at address 0x00100000
downloaded 29088 bytes in 0.245499s (115.708 KiB/s)
verified 29088 bytes in 0.350231s (81.107 KiB/s)
semihosting is enabled

Flashprog version: 2.1.0
Writing "mcufw" @0x6a000 (primary)...done
semihosting: *** application exited ***
Flashprog Complete
shutdown command invoked

target state: halted
target halted due to breakpoint, current mode: Thread
xPSR: 0x21000000 pc: 0x00100658 msp: 0x0015ffe4, semihosting
Resetting board...
Using OpenOCD interface file ftdi.cfg
Open On-Chip Debugger 0.9.0 (2015-07-15-15:28)
Licensed under GNU GPL v2
For bug reports, read
    http://openocd.org/doc/doxygen/bugs.html
adapter speed: 3000 kHz
adapter_nsrst_delay: 100
Info : auto-selecting first available session transport "jtag". To override use 'transport
select <transport>'.
jtag_ntrst_delay: 100
cortex_m reset_config sysresetreq
sh_load
Info : clock speed 3000 kHz
Info : JTAG tap: wmcore.cpu tap/device found: 0x4ba00477 (mfg: 0x23b, part: 0xba00, ver: 0
x4)
Info : wmcore.cpu: hardware has 6 breakpoints, 4 watchpoints
Info : JTAG tap: wmcore.cpu tap/device found: 0x4ba00477 (mfg: 0x23b, part: 0xba00, ver: 0
x4)
shutdown command invoked
Resetting board done...
```

6. 重新整理韌體並重設電路板後,示範應用程式應該會開始,如下圖所示。

```
Network connection successful.  
Wi-Fi Connected to AP. Creating tasks which use network...  
2 6293 [Startup Hook] Write certificate...  
3 6296 [Startup Hook] Write device private key...  
4 6362 [Startup Hook] Creating MQTT Echo Task...  
6 11668 [MQTTEcho] MQTT echo connected.to connect to a2wtm15blvjj8-ats.iot.us-east-2.amazonaws.com  
7 11668 [MQTTEcho] MQTT echo test echoing task created.  
8 11961 [MQTTEcho] MQTT Echo demo subscribed to freertos/demos/echo  
9 12248 [MQTTEcho] Echo successfully published 'Hello World 0'  
10 12591 [Echoing] Message returned with ACK: 'Hello World 0 ACK'  
11 17633 [MQTTEcho] Echo successfully published 'Hello World 1'  
12 17927 [Echoing] Message returned with ACK: 'Hello World 1 ACK'  
13 22953 [MQTTEcho] Echo successfully published 'Hello World 2'  
14 23276 [Echoing] Message returned with ACK: 'Hello World 2 ACK'  
15 28245 [MQTTEcho] Echo successfully published 'Hello World 3'  
16 28575 [Echoing] Message returned with ACK: 'Hello World 3 ACK'  
17 33542 [MQTTEcho] Echo successfully published 'Hello World 4'  
18 33980 [Echoing] Message returned with ACK: 'Hello World 4 ACK'  
19 38823 [MQTTEcho] Echo successfully published 'Hello World 5'  
20 39279 [Echoing] Message returned with ACK: 'Hello World 5 ACK'  
21 44139 [MQTTEcho] Echo successfully published 'Hello World 6'  
22 44501 [Echoing] Message returned with ACK: 'Hello World 6 ACK'  
23 49516 [MQTTEcho] Echo successfully published 'Hello World 7'  
24 50270 [Echoing] Message returned with ACK: 'Hello World 7 ACK'  
25 54796 [MQTTEcho] Echo successfully published 'Hello World 8'  
26 55129 [Echoing] Message returned with ACK: 'Hello World 8 ACK'  
27 60080 [MQTTEcho] Echo successfully published 'Hello World 9'  
28 60389 [Echoing] Message returned with ACK: 'Hello World 9 ACK'  
29 65378 [MQTTEcho] Echo successfully published 'Hello World 10'  
30 65998 [Echoing] Message returned with ACK: 'Hello World 10 ACK'  
31 70658 [MQTTEcho] Echo successfully published 'Hello World 11'  
32 70964 [Echoing] Message returned with ACK: 'Hello World 11 ACK'  
33 75958 [MQTTEcho] MQTT echo demo finished.  
34 75958 [MQTTEcho] ----Demo finished----
```

7. (選用) 做為測試映像的替代方法,請使用 flashprog 公用程式,將主機中的微控制器映像直接複製到微型控制器 RAM。映像不會在快閃記憶體中複製,因此在您重新開機微型控制器之後,它會遺失。

將韌體映像載入至 SRAM 是更快速的操作,因為它會立即推出執行檔案。此方法主要用於反覆開發。

使用下列命令將韌體載入至 SRAM。

```
cd amzsdk_bundle-x.y.z  
.lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/OpenOCD/ramload.py build/  
cmake/vendors/marvell/mw300_rd/aws_demos.axf
```

命令輸出顯示於下方圖表中。

```
Using OpenOCD interface file ftdi.cfg
Open On-Chip Debugger 0.9.0 (2015-07-15-15:28)
Licensed under GNU GPL v2
For bug reports, read
    http://openocd.org/doc/doxygen/bugs.html
adapter speed: 3000 kHz
adapter_nsrst_delay: 100
Info : auto-selecting first available session transport "jtag". To override use 'transport
select <transport>'.
jtag_ntrst_delay: 100
cortex_m_reset_config sysresetreq
sh_load
Info : clock speed 3000 kHz
Info : JTAG tap: wmcrc.cpu tap/device found: 0x4ba00477 (mfg: 0x23b, part: 0xba00, ver: 0
x4)
Info : wmcrc.cpu: hardware has 6 breakpoints, 4 watchpoints
Info : JTAG tap: wmcrc.cpu tap/device found: 0x4ba00477 (mfg: 0x23b, part: 0xba00, ver: 0
x4)
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00007f14 msp: 0x20001000
75072 bytes written at address 0x00100000
8 bytes written at address 0x00112540
468 bytes written at address 0x20000040
downloaded 75548 bytes in 0.636127s (115.979 KiB/s)
verified 75548 bytes in 0.959023s (76.930 KiB/s)
shutdown command invoked
```

當命令執行完成時,您應該會看到示範應用程式的 logs。

使用 Eclipse IDE 建置並執行 FreeRTOS 示範

1. 設定 Eclipse 工作空間之前,您必須執行 cmake 命令。

執行以下命令來使用 aws_demos Eclipse 專案。

```
cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S . -Bbuild -
DAFR_ENABLE_TESTS=0
```

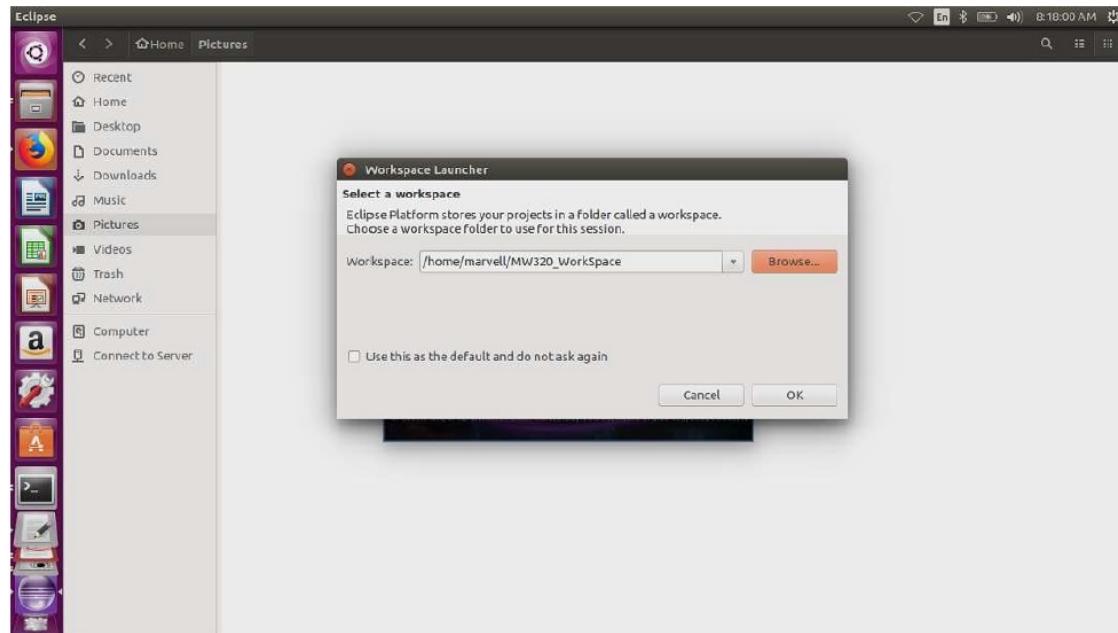
執行以下命令來使用 aws_tests Eclipse 專案。

```
cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S . -Bbuild -
DAFR_ENABLE_TESTS=1
```

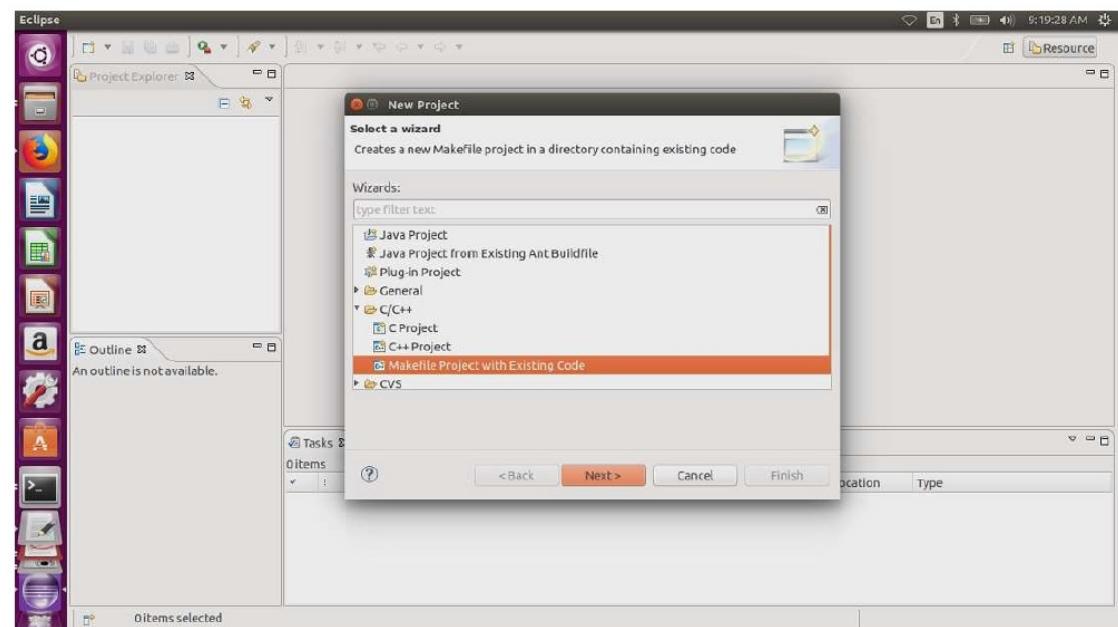
Tip

每次在 cmake 專案和 aws_demos 專案之間切換時,請執行 aws_tests 命令。

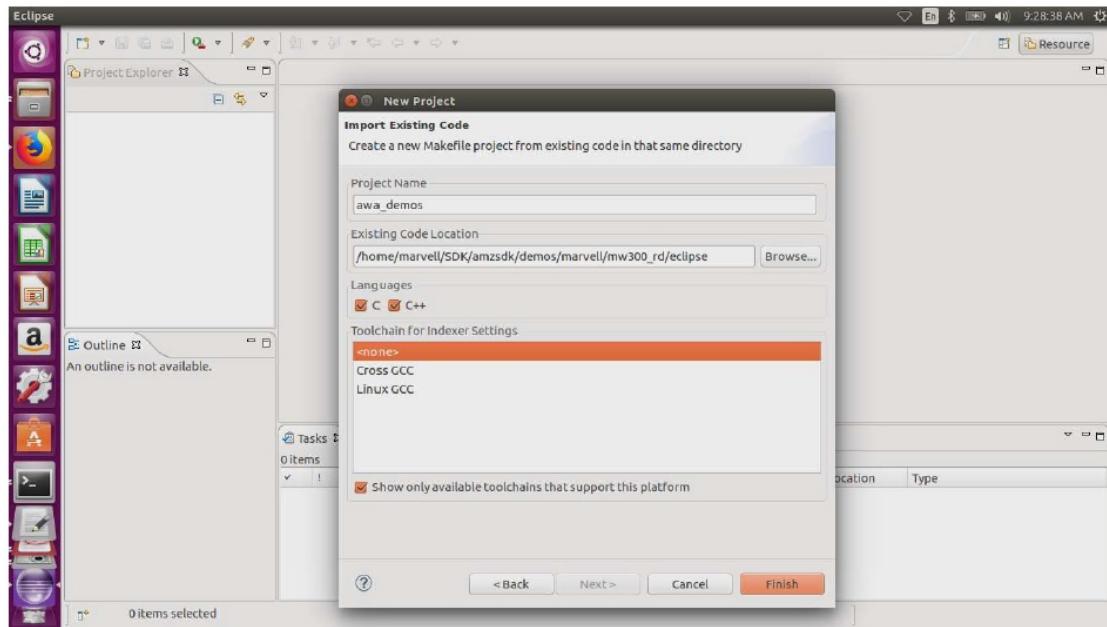
2. Open Eclipse,並在系統提示時,選擇 Eclipse 工作空間,如下圖所示。



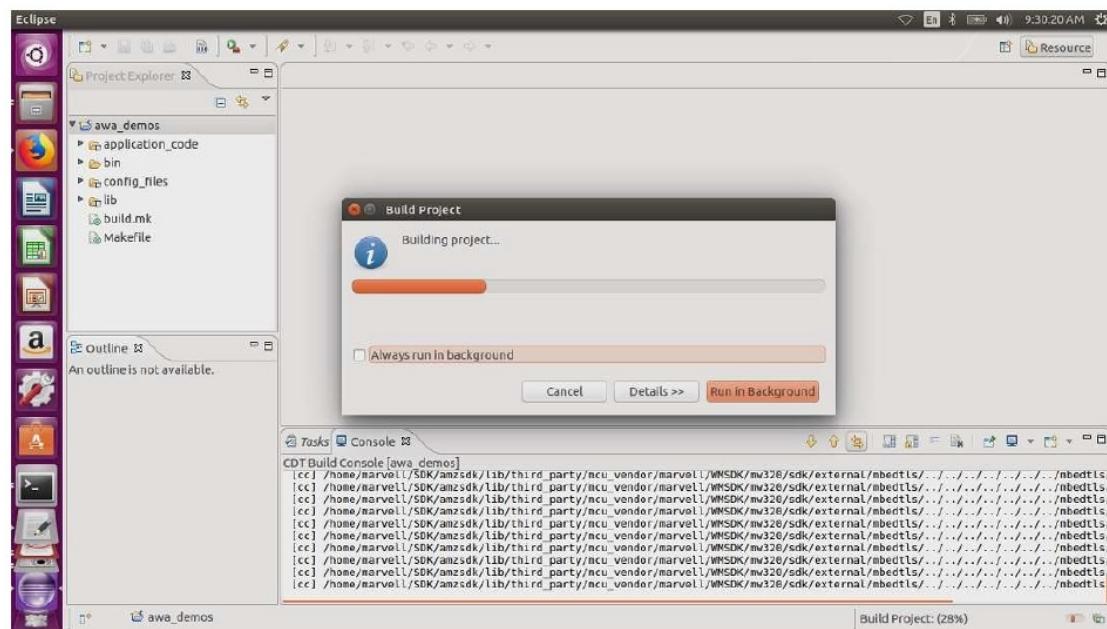
3. 選擇選項以建立 Makefile Project: with Existing Code (Makefile 專案:含現有程式碼),如下圖所示。



4. 選擇 Browse (瀏覽),指定現有程式碼的目標,然後選擇 Finish (完成)。



- 在導覽窗格中,選擇專案瀏覽器中的 `aws_demos`。在 `aws_demos` 上按一下滑鼠右鍵以打開功能表,然後選擇 Build (建置)。



如果建置成功,就會得到 `build/cmake/vendors/marvell/mw300_rd/aws_demos.bin` 檔案。

- 使用命令列工具刷新配置檔案 (`layout.txt`)、Boot2 二進位檔 (`boot2.bin`)、MCU 韌體二進位檔 (`aws_demos.bin`) 和 Wi-Fi 韌體。
 - 在您刷新韌體映像前,請使用常見的元件 Layout 和 Boot2 準備開發電路板的快閃記憶體。請使用以下命令。

```
cd amzsdk_bundle-x.y.z
```

```
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py -l ./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/mw300/layout.txt --boot2 ./vendors/marvell/WMSDK/mw320/boot2/bin/boot2.bin
```

命令會起始以下項目:flashprog

- Layout (配置)- 第一次指示 Flashprog 公用程式將配置寫入至快閃記憶體。配置類似於快閃記憶體的分割區資訊。預設配置位於 /lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/OpenOCD/mw300/layout.txt。
- Boot2 – 這是 WMSDK 所使用的開機載入器。flashprog 命令也會將開機載入器寫入至快閃記憶體。它是開機載入器的任務,在閃存之後載入微控制器的韌體映像。請確定您取得與下圖相同的輸出。

```
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x000007f14 msp: 0x20001000
29088 bytes written at address 0x00100000
downloaded 29088 bytes in 0.245458s (115.728 KiB/s)
verified 29088 bytes in 0.350004s (81.160 KiB/s)
semihosting is enabled

Flashprog version: 2.1.0
Erasing primary flash...done
Writing new flash layout...done
Writing "boot2" @0x0 (primary)...done
semihosting: *** application exited ***
Flashprog Complete
shutdown command invoked

target state: halted
target halted due to breakpoint, current mode: Thread
xPSR: 0x21000000 pc: 0x00100658 msp: 0x0015ffe4, semihosting
```

- b. 韌體會針對其功能使用 Wi-Fi 晶片組,而且 Wi-Fi 晶片組有自己的韌體,同時也必須存在於快閃記憶體中。您使用 flashprog.py 公用程式刷新 Wi-Fi 韌體,方法與刷新 boot2 開機載入器和 MCU 韌體相同。請使用以下命令來刷新 Wi-Fi 韌體。

```
cd amzsdk_bundle-x.y.z
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py --wififw ./vendors/marvell/WMSDK/mw320/wifi-firmware/mw30x/mw30x_uapsta_W14.88.36.p135.bin
```

請確定命令的輸出與下圖類似。

```
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x000007f14 msp: 0x20001000
29088 bytes written at address 0x00100000
downloaded 29088 bytes in 0.245498s (115.709 KiB/s)
verified 29088 bytes in 0.350229s (81.108 KiB/s)
semihosting is enabled

Flashprog version: 2.1.0
Writing "wififw" @0x12a000 (primary)....done
semihosting: *** application exited ***
Flashprog Complete
shutdown command invoked

target state: halted
target halted due to breakpoint, current mode: Thread
xPSR: 0x21000000 pc: 0x00100658 msp: 0x0015ffe4, semihosting
```

- c. 使用下列命令來刷新 MCU 韌體。

```
cd amzsdk_bundle-x.y.z
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py --mcufw build/cmake/
vendors/marvell/mw300_rd/aws_demos.bin -r
```

- d. 重設開發板。您應該會看到示範應用程式的 logs。
- e. 若要執行測試應用程式,請刷新位於相同資料夾的 aws_tests.bin 二進位檔。

```
cd amzsdk_bundle-x.y.z
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py --mcufw build/cmake/
vendors/marvell/mw300_rd/aws_tests.bin -r
```

命令輸出應該會類似下圖所示的輸出。

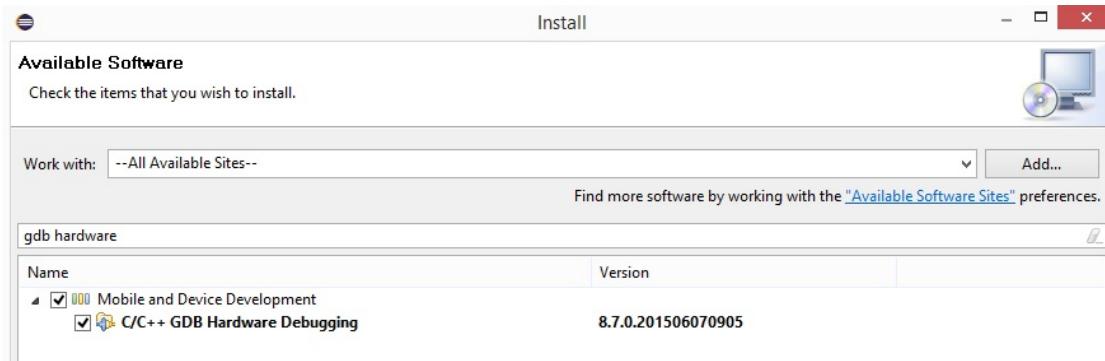
```
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00007f14 msp: 0x20001000
29088 bytes written at address 0x00100000
downloaded 29088 bytes in 0.245499s (115.708 KiB/s)
verified 29088 bytes in 0.350231s (81.107 KiB/s)
semihosting is enabled

Flashprog version: 2.1.0
Writing "mcufw" @0x6a000 (primary)...done
semihosting: *** application exited ***
Flashprog Complete
shutdown command invoked

target state: halted
target halted due to breakpoint, current mode: Thread
xPSR: 0x21000000 pc: 0x00100658 msp: 0x0015ffe4, semihosting
Resetting board...
Using OpenOCD interface file ftdi.cfg
Open On-Chip Debugger 0.9.0 (2015-07-15-15:28)
Licensed under GNU GPL v2
For bug reports, read
    http://openocd.org/doc/doxygen/bugs.html
adapter speed: 3000 kHz
adapter_nrst_delay: 100
Info : auto-selecting first available session transport "jtag". To override use 'transport
      select <transport>'.
jtag_nrst_delay: 100
cortex_m_reset_config sysresetreq
sh_load
Info : clock speed 3000 kHz
Info : JTAG tap: wmc0re.cpu tap/device found: 0x4ba00477 (mfg: 0x23b, part: 0xba00, ver: 0
x4)
Info : wmc0re.cpu: hardware has 6 breakpoints, 4 watchpoints
Info : JTAG tap: wmc0re.cpu tap/device found: 0x4ba00477 (mfg: 0x23b, part: 0xba00, ver: 0
x4)
shutdown command invoked
Resetting board done...
```

Debugging

- 開始 Eclipse,然後選擇 Help (協助),然後選擇 Install new software (安裝新軟體)。在 Work with (使用) 功能表中,選擇 All Available Sites (所有可用網站)。輸入篩選條件文字 GDB Hardware。選取 C/C++ GDB Hardware Debugging (C/C++ GDB 硬體偵錯) 選項並安裝 plugin。



Troubleshooting

網路問題

檢查您的網路登入資料。請前往 [建置並執行 FreeRTOS 示範專案 \(p. 73\)](#) 中的「Provisioning」。
能執行額外的指令

- 允許主機板特定機碼。

允許在 `wmstdio_init(UART0_ID, 0)` 檔案的函數 `prvMiscInitialization` 中呼叫 `main.c` 以供測試或示範。

- 提供 Wi-Fi 登入

在 `CONFIG_WLCMGR_DEBUG` 檔案中使用巨集 `freertos/vendors/marvell/WMSDK/mw320/sdk/src/incl/autoconf.h`。

使用 GDB

我們建議您使用隨開發套件一起封裝的 `arm-none-eabi-gdb` 和 `gdb` 命令檔案。請導覽至 目錄。

```
cd freertos/lib/third_party/mcu_vendor/marvell/WMSDK/mw320
```

執行下列命令 (在單一行中) 以連接到 GDB。

```
arm-none-eabi-gdb -x ./sdk/tools/OpenOCD/gdbinit ../../../../build/cmake/vendors/marvell/mw300 _rd/aws_demos.axf
```

開發套件入門 MediaTek MT7697Hx

本教學課程提供 MediaTek MT7697Hx 開發套件入門指示。若您尚未擁有 MediaTek MT7697Hx 開發套件，
請前往 AWS Partner Device Catalog，向我們的[合作夥伴](#)購買。

開始之前，請您務必要設定 AWS IoT 和 FreeRTOS 下載目錄，才能將裝置連接至 AWS 雲端。如需說明，
請參閱[首要步驟 \(p. 14\)](#)。在本教學課程中，FreeRTOS 下載目錄的路徑會以 `freertos` 表示。

Overview

本教學課程包含以下入門步驟的指示：

- 在主機機器上安裝軟體以對微控制器主機板的內嵌應用程式進行開發和除錯。
- 將 FreeRTOS 示範應用程式跨編譯為二進位映像。

3. 將應用程式二進位映像載入主機板，然後執行應用程式。
4. 透過序列連線與在開發板上執行的應用程式互動，以便進行監控和除錯。

設定開發環境

在您設定環境前，請將電腦連接到 MediaTek MT7697Hx 開發套件上的 USB 連接埠。

下載並安裝 Keil MDK

您可以使用 GUI 型 Keil 微型控制器開發套件 (MDK)，在電路板上設定、建置和執行 FreeRTOS 專案。Keil MDK 包含 μVision IDE 和 μVision Debugger。

Note

僅 Windows 7、Windows 8 和 Windows 10 64 位元機器支援 Keil MDK。

下載並安裝 Keil MDK

1. 移至 [Keil MDK 入門](#) 頁面，然後選擇 Download MDK-Core (下載 MDK 核心)。
2. 輸入並提交您的資訊，以向 Keil 註冊。
3. 用滑鼠右鍵按一下 MDK 可執行檔，並將 Keil MDK 安裝程式儲存到您的電腦。
4. 開啟 Keil MDK 安裝程式並遵循步驟來完成。請務必安裝 MediaTek 裝置套件 (MT76x7 系列)。

建立序列連線

若要使用 MediaTek MT7697Hx 開發套件建立序列連接，您必須安裝 Arm Mbed Windows 序列埠驅動程式。您可以從 [Mbed](#) 下載驅動程式。依照 Windows serial driver (Windows 序列驅動程式) 頁面上的步驟，下載並安裝 MediaTek MT7697Hx 開發套件的驅動程式。

在您安裝驅動程式之後，COM 連接埠會出現在 Windows 裝置管理員中。若要進行偵錯，您可以使用終端機公用程式工具（例如 HyperTerminal 或 TeraTerm）將工作階段開放給連接埠。

Note

如果您安裝驅動程式後無法連接到主機板，您可能必須重新啟動機器。

使用 Keil MDK 來建置並執行 FreeRTOS 示範專案

在 Keil 中建置 FreeRTOS 示範專案 μVision

1. 從 Start (開始) 功能表中，開啟 Keil μVision 5。
2. 開啟 projects/mediatek/mt7697hx-dev-kit/uvision/aws_demos/aws_demos.uvprojx 專案檔案。
3. 從功能表中，選擇 Project (專案)，然後選擇 Build target (建置目標)。

程式碼建置完畢後，您即可在 projects/mediatek/mt7697hx-dev-kit/uvision/aws_demos/out/Objects/aws_demo.axf 中查看示範可執行檔。

執行 FreeRTOS 示範專案

1. 將 MediaTek MT7697Hx 開發套件設定為 PROGRAM 模式。

若要將套件設定為 PROGRAM 模式，請按住 PROG 按鈕。在 PROG 按鈕仍然按住的情況下，按下並放開 RESET 按鈕，然後放開 PROG 按鈕。

2. 從功能表中，選擇 Flash，然後選擇 Configure Flash Tools (設定 Flash 工具)。

3. 在 Options for Target 'aws_demo' (目標"的選項) 中，選擇 Debug (除錯) 標籤。選取 Use (使用)、將偵錯工具設定為 CMSIS-DAP Debugger (CMSIS-DAP 偵錯器)，然後選擇 OK (確定)。
 4. 從功能表中，選擇 Flash，然後選擇 Download (下載)。
- 下載完成時 μVision 會通知您。
5. 使用終端機公用程式來開啟序列主控台視窗。將序列連接埠設定為 115200 bps、非同位、8 位元和 1 個停止位元。
 6. 選擇 開發套件上的 MediaTekRESET (重設) MT7697Hx 按鈕。

監控雲端的 MQTT 訊息

您可以在 AWS IoT 主控台中使用 MQTT 用戶端來監控裝置傳送至 AWS 雲端的訊息。

使用 AWS IoT MQTT 用戶端訂閱 MQTT 主題

1. 登入 [AWS IoT 主控台](#)。
2. 在導覽窗格中，選擇 Test (測試) 以開啟 MQTT 用戶端。
3. 在訂閱主題中輸入 `iotdemo/#`，然後選擇訂閱主題。

Troubleshooting

在 Keil FreeRTOS 中偵錯 μ 專案 Vision

目前，您必須編輯 Keil MediaTekVision 隨附的 μ 套件，然後才能使用 Keil FreeRTOSVision 偵錯 MediaTek 的 μ 示範專案。

編輯 MediaTek 套件以偵錯 FreeRTOS 專案

1. 在 Keil MDK 安裝資料夾中尋找並開啟 Keil_v5\ARM\PACK\Web\MediaTek.MTx.pdsc 檔案。
2. 將 `flag = Read32(0x20000000);` 的所有執行個體取代為 `flag = Read32(0x0010FBFC);`。
3. 將 `Write32(0x20000000, 0x76877697);` 的所有執行個體取代為 `Write32(0x0010FBFC, 0x76877697);`。

開始偵錯專案

1. 從功能表中，選擇 Flash，然後選擇 Configure Flash Tools (設定 Flash 工具)。
2. 選擇 Target (目標) 標籤，然後選擇 Read/Write Memory Areas (讀取/寫入記憶體區域)。確認已選取 IRAM1 和 IRAM2 兩者。
3. 選擇 Debug (偵錯) 標籤，然後選擇 CMSIS-DAP Debugger (CMSIS-DAP 偵錯器)。
4. 開啟 vendors MEDIATEK/boards/mt7697hx-dev-kit/aws_demos/application_code/main.c，並將 MTK_DEBUGGER 巨集設定為 1。
5. 在 μVision 中重新建置示範專案。
6. 將 MediaTek MT7697Hx 開發套件設定為 PROGRAM 模式。

若要將套件設定為 PROGRAM 模式，請按住 PROG 按鈕。在 PROG 按鈕仍然按住的情況下，按下並放開 RESET 按鈕，然後放開 PROG 按鈕。

7. 從功能表中，選擇 Flash，然後選擇 Download (下載)。

下載完成時 μVision 會通知您。

8. 按下 開發套件上的 MediaTekRESET (重設) MT7697Hx 按鈕。
9. 從 μVision 功能表中，選擇 Debug (偵錯)，然後選擇 Start/Stop Debug Session (啟動/停止偵錯工作階段)。當您啟動偵錯工作階段時，Call Stack + Locals (呼叫堆疊 + 本機) 視窗即會開啟。

10. 從選單中選擇 Debug (偵錯) , 然後選擇 Stop (停止) 暫停執行程式碼。程式計數器會停在以下一行 :

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

11. 在 Call Stack + Locals (呼叫堆疊 + 本機) 視窗中 , 將 wait_ice 的值變更為 0。

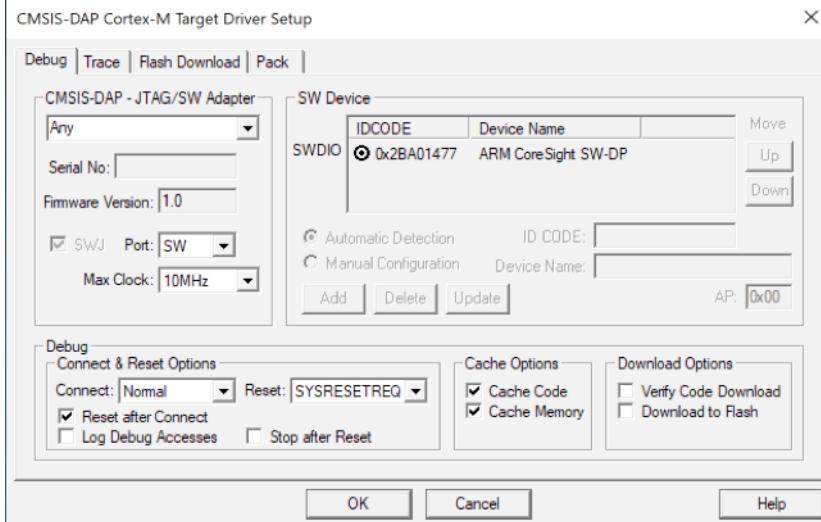
12. 在專案的原始程式碼中設定中斷點 , 然後執行程式碼。

故障診斷 IDE 除錯器設定

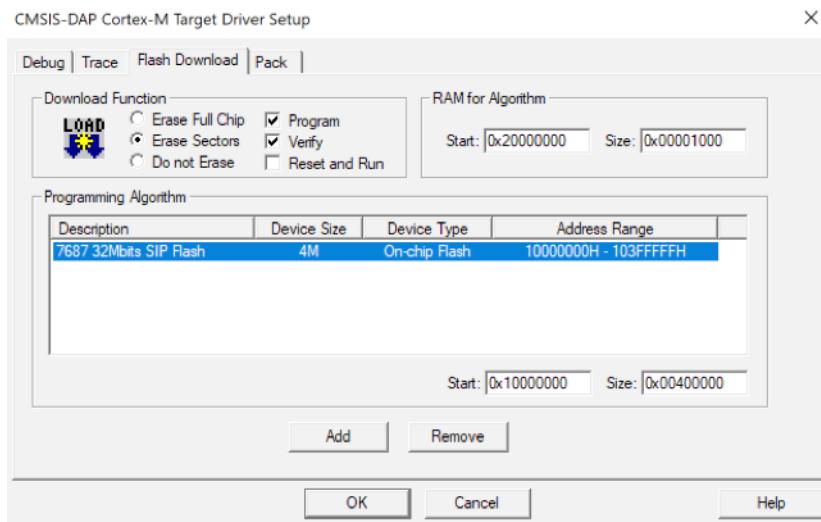
如果您無法除錯應用程式 , 您的除錯器設定可能不正確。

驗證您的除錯器設定是否正確

1. 開啟 Keil μVision。
2. 在 aws_demos 專案上按一下滑鼠右鍵 , 選擇 Options (選項) , 然後在 Utilities (公用程式) 索引標籤下 , 選擇 -- Use Debug Driver -- (-- 使用偵錯驅動程式 --) 旁的 Settings (設定) 。
3. 驗證 Debug (除錯) 索引標籤中的設定如下所示 :



4. 驗證 Flash Download (Flash 下載) 索引標籤中的設定如下所示 :



如需 FreeRTOS 入門的一般疑難排解資訊，請參閱 [故障診斷入門 \(p. 19\)](#)。

Microchip Curiosity PIC32MZ EF 入門

本教學課程提供 Microchip Curiosity PIC32MZ EF 入門指示。如果您沒有 Microchip Curiosity PIC32MZ EF 套件，請造訪 AWS Partner Device Catalog，向我們的[合作夥伴購買](#)。

套件包含下列項目：

- Curiosity PIC32MZ EF 開發板
- MikroElectronika USB UART click 電路板
- MikroElectronika WiFi 7 click 電路板
- PIC32 LAN8720 PHY 子板

您也需要使用以下項目進行偵錯：

- MPLAB Snap 電路內除錯器
- (選用) PICkit 3 Programming Cable Kit

開始之前，請您務必要設定 AWS IoT 和 FreeRTOS 下載目錄，才能將裝置連接至 AWS 雲端。請參閱 [首要步驟 \(p. 14\)](#) 以取得說明。

Important

- 在本教學課程中，FreeRTOS 下載目錄的路徑會以 *freertos* 表示。
- *freertos* 路徑中的空格字元可能會導致建置失敗。當您複製或拷貝儲存庫時，請確定您建立的路徑不包含空格字元。
- Microsoft Windows 的檔案路徑長度上限為 260 個字元。FreeRTOS 下載目錄路徑過長可能會導致建置失敗。

Overview

本教學課程包含以下入門步驟的指示：

1. 將主機板連線到主機機器。
2. 在主機機器上安裝軟體以對微控制器主機板的內嵌應用程式進行開發和除錯。
3. 將 FreeRTOS 示範應用程式跨編譯為二進位映像。
4. 將應用程式二進位映像載入主機板，然後執行應用程式。
5. 透過序列連線與在開發板上執行的應用程式互動，以便進行監控和除錯。

設定 Microchip Curiosity PIC32MZ EF 硬體

1. 將 MikroElectronika USB UART Click 電路板接到 Microchip Curiosity PIC32MZ EF 的 microBUS 1 連接器。
2. 將 PIC32 LAN8720 PHY 子板接到 Microchip Curiosity PIC32MZ EF 的 J18 接頭。
3. 使用 USB A 對 USB mini-B 纜線，將 MikroElectronika USB UART click 電路板接到您的電腦。
4. 若要將電路板連接至網際網路，請使用下列其中一個選項：
 - 若要使用 Wi-Fi，請將 MikroElectronika WiFi 7 Click 電路板接到 Microchip Curiosity PIC32MZ EF 的 microBUS 2 連接器 請參閱[設定 FreeRTOS 示範 \(p. 18\)](#)。

- 若要使用乙太網路來將 Microchip Curiosity PIC32MZ EF 電路板連接至網際網路，請將 PIC32 LAN8720 PHY 子板連接至 Microchip Curiosity PIC32MZ EF 上的 J18 標頭。將乙太網路纜線一端接到 LAN8720 PHY 子板上。將另一端接到您的路由器或其他網際網路連接埠。
5. 請將轉角聯接器 (angle connector) 焊接至 Microchip Curiosity PIC32MZ EF 的 ICSP 頂蓋 (若發現未焊接)。
 6. 將 PICkit 3 Programming Cable Kit 的 ICSP 纜線一端連接至 Microchip Curiosity PIC32MZ EF。

若您沒有 PICkit 3 Programming Cable Kit，可改為使用 M-F Dupont 跳線做連接。請注意，白色圓形表示接腳 1 (Pin 1) 的位置。

7. 將 ICSP 纜線 (或跳線) 的另一端連接到 MPLAB Snap Debugger。電路板右下方黑色三角形的標示為 8 針腳 SIL Programming Connector 的接腳 1。

請確認連接至 Microchip Curiosity PIC32MZ EF 接腳 1 (標示為白色) 的線路與 MPLAB Snap Debugger 的接腳 1 對齊。

如需 MPLAB Snap Debugger 的詳細資訊，請參閱 [MPLAB Snap In-Circuit Debugger 說明表](#)。

使用 PICkit On Board (PKOB) 設定 Microchip Curiosity PIC32MZ EF 硬體

建議您遵循上一節的設定程序。不過，您可以遵循下列步驟，使用整合式 PICkit On Board (PKOB) 程式設計工具/偵錯工具，透過基本偵錯評估並執行 FreeRTOS 示範。

1. 將 MikroElectronika USB UART Click 電路板接到 Microchip Curiosity PIC32MZ EF 的 microBUS 1 連接器。
2. 若要將電路板連接至網際網路，請執行下列其中一項操作：
 - 若要使用 Wi-Fi，請將 MikroElectronika Wi-Fi 7 Click 電路板接到 Microchip Curiosity PIC32MZ EF 的 microBUS 2 連接器 (按照 [設定 FreeRTOS 示範 \(p. 18\)](#) 中的「設定 Wi-Fi」步驟進行)。
 - 若要使用乙太網路來將 Microchip Curiosity PIC32MZ EF 電路板連接至網際網路，請將 PIC32 LAN8720 PHY 子板連接至 Microchip Curiosity PIC32MZ EF 上的 J18 標頭。將乙太網路纜線一端接到 LAN8720 PHY 子板上。將另一端接到您的路由器或其他網際網路連接埠。
3. 使用 USB type A 轉 USB micro-B 纜線，將 Microchip Curiosity PIC32MZ EF 電路板上名為「USB DEBUG」(USB 偵錯) 的 USB micro-B 連接埠連接至電腦。
4. 使用 USB A 對 USB mini-B 纜線，將 MikroElectronika USB UART click 電路板接到您的電腦。

設定開發環境。

Note

此裝置的 FreeRTOS 專案是根據 MPLAB Harmony v2。若要建置專案，您需要使用與 Harmony v2 相容的 MPLAB 工具，例如 MPLAB XC32 的 v2.10 版和 MPLAB Harmony Configurator (MHC) 的 2.X.X 版。

1. 安裝 [Python 3.x 版](#)或更新版本。
2. 安裝 MPLAB X IDE：

Note

FreeRTOS 目前僅在 MPLabv5.35 上支持 SIP 參考集成 v202007.00。MPLabv5.40 支持以前版本的 FreeRTOS 的 ASM 參考集成。

MPLabv5.35下載

- 適用於 Windows 的 MPLAB X 整合開發環境
- 適用於 macOS 的 MPLAB X 整合開發環境
- 適用於 Linux 的 MPLAB X 整合開發環境

最新MPLab下載(MPLabv5.40)

- 適用於 Windows 的 MPLAB X 整合開發環境
 - 適用於 macOS 的 MPLAB X 整合開發環境
 - 適用於 Linux 的 MPLAB X 整合開發環境
3. 安裝 MPLAB XC32 編譯器：
 - 適用於 Windows 的 MPLAB XC32/32++ 編譯器
 - 適用於 macOS 的 MPLAB XC32/32++ 編譯器
 - 適用於 Linux 的 MPLAB XC32/32++ 編譯器
 4. 啟動 UART 終端機模擬器，並使用以下設定開啟連線：
 - 傳輸速率 115200
 - 資料：8 位元
 - 同位 無.
 - 停止位: 1.
 - 流量控制 無.

建置並執行 FreeRTOS 示範專案

在 MPLAB IDE 中開啟 FreeRTOS 示範

1. 開啟 MPLAB IDE。如果您已安裝多個版本的編譯器，您需要選擇想在 IDE 內使用的編譯器。
2. 從 File (檔案) 選單中，選擇 Open Project (開啟專案)。
3. 瀏覽至 projects/microchip/curiosity_pic32mzef/mplab/aws_demos 並將它開啟。
4. 選擇 Open project (開啟專案)。

Note

在您首次開啟專案時，您可能會收到有關編譯器的錯誤訊息。在 IDE 中，導覽至 Tools (工具)、Options (選項)、Embedded (內嵌)，然後選擇您要用於專案的編譯器。

執行 FreeRTOS 示範專案

1. 重新組建專案。
2. 在 Projects (專案) 標籤中，以滑鼠右鍵按一下 aws_demos 最上層資料夾，然後選擇 Debug (除錯)。
3. 當除錯器停在 main() 中斷點時，請從 Run (執行) 功能表，選擇 Resume (繼續)。

使用 CMake 建置 FreeRTOS 示範

如果您不想使用 IDE 進行 FreeRTOS 開發，您也可以改為使用 CMake 來建置並執行示範應用程式，或使用第三方程式碼編輯器和偵錯工具所開發的應用程式。

使用 CMake 建置 FreeRTOS 示範

1. 創建文件夾以包含生成的構建文件(*build-folder*).
2. 使用下列命令以從來源程式碼產生建置檔案：

```
cmake -DVENDOR=microchip -DBOARD=curiosity_pic32mzef -DCOMPILER=xc32 -  
DMCHP_HEXMATE_PATH=path/microchip/mplabx/version/mplab_platform/bin -  
DAFR_TOOLCHAIN_PATH=path/microchip/xc32/version/bin -S freertos -B build-folder
```

Note

您必須指定正確的路徑到 Hexmate 和工具鏈二進位檔。(例如：分別為 C:\Program Files (x86)\Microchip\MPLABX\v5.35\mplab_platform\bin 和 C:\Program Files \Microchip\xc32\v2.40\bin。)

3. 將目錄更改為內部目錄(*build-folder*),並運行 make 從目錄。

如需更多詳細資訊，請參閱「[使用 CMake 配 FreeRTOS \(p. 20\)](#)」。

監控雲端的 MQTT 訊息

您可以在 AWS IoT 主控台中使用 MQTT 用戶端來監控裝置傳送至 AWS 雲端的訊息。

使用 AWS IoT MQTT 用戶端訂閱 MQTT 主題

1. 登入 [AWS IoT 主控台](#)。
2. 在導覽窗格中，選擇 Test (測試) 以開啟 MQTT 用戶端。
3. 在訂閱主題中輸入 `iotdemo/#`，然後選擇訂閱主題。

Troubleshooting

如需故障診斷資訊，請參閱 [故障診斷入門 \(p. 19\)](#)。

Nordic nRF52840-DK 入門

本教學課程提供 Nordic nRF52840-DK 的入門說明。如果還沒有 Nordic nRF52840-DK，請造訪 AWS Partner Device Catalog，向我們的[合作夥伴購買](#)。

開始之前，您需要為 FreeRTOS 低功耗藍牙設定 AWS IoT 和 Amazon Cognito (p. 211)。

若要執行 FreeRTOS 低功耗藍牙示範，則您的 iOS 或 Android 行動裝置也需具備藍牙和 Wi-Fi 功能。

Note

如果您使用 iOS 裝置，您需要 Xcode 來建置示範行動應用程式。如果您使用 Android 裝置，您可以使用 Android Studio 來建置示範行動應用程式。

Overview

本教學課程包含以下入門步驟的指示：

1. 將主機板連線到主機機器。
2. 在主機機器上安裝軟體以對微控制器主機板的內嵌應用程式進行開發和除錯。
3. 將 FreeRTOS 示範應用程式跨編譯為二進位映像。
4. 將應用程式二進位映像載入主機板，然後執行應用程式。
5. 透過序列連線與在開發板上執行的應用程式互動，以便進行監控和除錯。

設定 Nordic 硬體

請將主機電腦連接至標籤為 J2 的 USB 連接埠，其位於 Nordic nRF52840 主機板鈕釦型電池座的正上方。

如需設定 Nordic nRF52840-DK 的詳細資訊，請參閱 [nRF52840 開發套件使用者指南](#)。

設定開發環境。

下載並安裝 Segger Embedded Studio

FreeRTOS 支援將 Segger Embedded Studio 作為 Nordic nRF52840-DK 的開發環境。

若要設定您的環境，您需要在主機電腦上下載並安裝 Segger Embedded Studio。

下載並安裝 Segger Embedded Studio

1. 移至 [Segger Embedded Studio 下載](#)頁面，然後為您的作業系統選擇 Embedded Studio for ARM 選項。
2. 執行安裝程式並遵循提示來完成操作。

設定 FreeRTOS 低功耗藍牙 Mobile SDK 示範應用程式

您需要在行動裝置上執行 FreeRTOS 低功耗藍牙 Mobile SDK 示範應用程式，才能跨低功耗藍牙執行 FreeRTOS 示範專案。

設置 FreeRTOS 藍牙低功耗移動SDK演示應用

1. 按照 [藍牙裝置適用的 SDKs 行動FreeRTOS \(p. 191\)](#)中的指示，在您的主機電腦上下載並安裝適用於行動平台的軟體開發套件。
2. 按照 [FreeRTOS 低功耗藍牙 Mobile SDK 示範應用程式 \(p. 213\)](#) 中的指示，來在您的行動裝置上設定示範行動應用程式。

建立序列連線

Segger Embedded Studio 包含終端機模擬器，您可以用來透過電路板的序列連線接收日誌訊息。

建立與 Segger Embedded Studio 的序列連線

1. 開啟 Segger Embedded Studio。
2. 從上方功能表中，選擇 Target (目標)、Connect J-Link (連接 J-Link)。
3. 從上方功能表中，選擇 Tools (工具)、Terminal Emulator (終端機模擬器)、Properties (屬性)，然後依照 [安裝終端機模擬器 \(p. 20\)](#) 中的指示來設定屬性。
4. 從頂部菜單中選擇 工具， 端子仿真器， 連接 **port** (115200、N、8、1) .

Note

Segger 內嵌 Studio 終端模擬器不支持輸入功能。因此，請使用 PuTTY、Tera Term 或 GNU Screen 這類的終端模擬器。依照[安裝終端機模擬器 \(p. 20\)](#)中的指示，將終端機設定為透過序列連線連接到您的電路板。

下載和設定 FreeRTOS

在您設定好硬體和環境之後，即可下載 FreeRTOS。

下載 FreeRTOS

若要為 Nordic nRF52840-DK 下載 FreeRTOS，請移至 [FreeRTOS GitHub 頁面](#)並複製儲存庫。如需說明，請參閱 [README.md](#) 檔案。

Important

- 在本教學課程中，FreeRTOS 下載目錄的路徑會以 `freertos` 表示。
- `freertos` 路徑中的空格字元可能會導致建置失敗。複製或拷貝儲存庫時，請確定您建立的路徑不包含空格字元。
- Microsoft Windows 的檔案路徑長度上限為 260 個字元。FreeRTOS 下載目錄路徑過長可能會導致建置失敗。

設定專案

若要執行示範，您必須將專案設定為使用 AWS IoT。若要將專案設定為與 AWS IoT 搭配使用，必須將您的裝置註冊為 AWS IoT 實物。您應該會在 [為 FreeRTOS 低功耗藍牙設定 AWS IoT 和 Amazon Cognito \(p. 211\)](#) 時註冊您的裝置。

設定 AWS IoT 端點

- 登入 [AWS IoT 主控台](#)。
- 在導覽窗格中選擇 Settings (設定)。

您的 AWS IoT 端點會出現在 Endpoint (端點) 文字方塊中。應該看起來像是 `1234567890123-ats.iot.us-east-1.amazonaws.com`。記下此端點。

- 在導覽窗格中，選擇 Manage (管理)，然後選擇 Things (實物)。記下您的裝置的 AWS IoT 實物名稱。
- 備妥您的 AWS IoT 端點和您的 AWS IoT 物件名稱，在您的 IDE 中開啟 `freertos/demos/include/aws_clientcredential.h`，並指定下列 `#define` 常數的值：
 - `clientcredentialMQTT_BROKER_ENDPOINT Your AWS IoT endpoint`
 - `clientcredentialIOT_THING_NAME Your board's AWS IoT thing name`

啟用示範

- 檢查低功耗藍牙 GATT 示範是否已啟用。移至 `vendors/nordic/boards/nrf52840-dk/aws_demos/config_files/iot_ble_config.h`，並將 `#define IOT_BLE_ADD_CUSTOM_SERVICES (1)` 新增至 define 陳述式的清單。
- 開啟 `vendors/nordic/boards/nrf52840-dk/aws_demos/config_files/aws_demos_config.h`，然後定義 `CONFIG_MQTT_DEMO_ENABLED`。
- 由於 Nordic 晶片隨附的 RAM 很小 (250 KB)，可能需要變更 BLE 組態，以允許將較大的 GATT 表格項目與每個屬性大小進行比較。如此一來，您就可以調整應用程式取得的記憶體量。若要這麼做，請覆寫 `freertos/vendors/nordic/boards/nrf52840-dk/aws_demos/config_files/sdk_config.h` 檔案中下列屬性的定義：
 - `NRF_SDH_BLE_VS_UUID_COUNT`
廠商專屬 UUID 的數目。
 - `NRF_SDH_BLE_GATTS_ATTR_TAB_SIZE`
屬性表格大小 (以位元組為單位)。大小必須是 4 的倍數

(若為測試，檔案位置為 `freertos/vendors/nordic/boards/nrf52840-dk/aws_tests/config_files/sdk_config.h`)。

建置並執行 FreeRTOS 示範專案

下載 FreeRTOS 並設定示範專案後，您就可以在主機板上建置和執行示範專案。

Important

如果這是您首次在這個主機板上執行示範，請務必先刷新主機板的開機載入器，再開始執行該示範。

如果要建置和刷新開機載入器，請遵循下方步驟操作，但請使用 `projects/nordic/nrf52840-dk/ses/aws_demos/bootloader/bootloader.emProject`，而不是 `projects/nordic/nrf52840-dk/ses/aws_demos/aws_demos.emProject` 專案檔案。

從 Segger Embedded Studio 建置和執行 FreeRTOS 低功耗藍牙示範

1. 開啟 Segger Embedded Studio。在上方功能表中，選擇 File (檔案)，並選擇 Open Solution (開啟解決方案)，然後導覽至專案檔案 `projects/nordic/nrf52840-dk/ses/aws_demos/aws_demos.emProject`。
2. 如果您是使用 Segger Embedded Studio 終端機模擬器，請從上方功能表中選擇 Tools (工具)，然後選擇 Terminal Emulator (終端機模擬器)。Terminal Emulator (終端機模擬器) 即會顯示來自序列連線的資訊。

如果您是使用另一個終端工具，則可以監控該工具，以取得適用於序列連線的輸出。

3. 在 Project Explorer (專案總管) 中的 `aws_demos` 示範專案上按一下滑鼠右鍵，然後選擇 Build (建置)。

Note

如果這是您第一次使用 Segger Embedded Studio，您可能會看到警告「無商業使用授權」。

您可以將 Segger Embedded Studio 免費用於 Nordic Semiconductor 裝置。選擇 Activate Your Free License (啟用您的免費授權)，並遵循指示。

4. 選擇 Debug (除錯)，然後選擇 Go (前往)。

該示範開始執行後，即會等待以低功耗藍牙與行動裝置配對。

5. 按照 [透過低功耗藍牙示範應用程式的 MQTT](#) 的指示來完成示範，並將 FreeRTOS 低功耗藍牙 Mobile SDK 示範應用程式做為行動 MQTT Proxy。

Troubleshooting

如需 FreeRTOS 入門的一般疑難排解資訊，請參閱 [故障診斷入門 \(p. 19\)](#)。

Nuvoton NuMaker-IoT-M487 入門

本教學課程提供 Nuvoton NuMaker-IoT-M487 開發面板入門的指示。系列微型控制器，且包含嵌入式 RJ45 乙太網路和 Wi-Fi 模組。若您尚未擁有 Nuvoton NuMaker-IoT-M487，請前往 [AWS Partner Device Catalog](#)，向我們的合作夥伴購買。

開始之前，請您務必要設定 AWS IoT 和 FreeRTOS 軟體，以將您的開發面板連接到 AWS 雲端。如需說明，請參閱 [首要步驟 \(p. 14\)](#)。在本教學課程中，FreeRTOS 下載目錄的路徑會以 `freertos` 表示。

Overview

本教學課程將指引您完成下列步驟：

1. 在您的主機機器上安裝軟體以對微控制器主機板的內嵌應用程式進行開發和除錯。
2. 將 FreeRTOS 示範應用程式跨編譯為二進位映像。
3. 將應用程式二進位映像載入主機板，然後執行應用程式。

設定開發環境

Keil MDK Nuvoton 版本是適用於 Nuvoton M487 專為開發和偵錯的應用程式。Keil MDK v5 Essential、Plus 或 Pro 版本也應該適用於 Nuvoton M487 (Cortex-M4 核心) MCU。您可以下載 Keil MDK Nuvoton 版本，並獲得 Nuvoton Cortex-M4 系列 MCUs 的價格折扣。Keil MDK 僅支援 Windows。

安裝 NuMaker-IoT-M487 的開發工具

1. 從 Keil MDK 網站下載 [Keil MDK Nuvoton 版本](#)。
2. 在您的主機上使用授權安裝 Keil MDK。Keil MDK 包含 Keil μVision IDE、C/C++ 編譯工具鏈以及 μVision 除錯器。
如果您在安裝期間遇到問題，請聯絡 [Nuvoton](#) 尋求協助。
3. 安裝 Nu-Link_Keil_Driver_V3.00.6951 (或最新版本)，其在 [Nuvoton Development Tool \(Nuvoton 開發工具\)](#) 頁面上。

建置並執行 FreeRTOS 示範專案

建置 FreeRTOS 示範專案

1. 開啟 Keil μVision IDE。
2. 在 File (檔案) 功能表上，選擇 Open (開啟)。在 Open file (開啟檔案) 對話方塊中，確定檔案類型選擇器設定為 Project Files (專案檔案)。
3. 選擇要建置的 Wi-Fi 或乙太網路示範專案。
 - 若要開啟 Wi-Fi 示範專案，請在 `freertos\projects\nuvoton\numaker_iot_m487_wifi\uvision\aws_demos` 目錄中選擇目標專案 `aws_demos.uvproj`。
 - 若要開啟乙太網路示範專案，請在 `freertos\projects\nuvoton\numaker_iot_m487_wifi\uvision\aws_demos_eth` 目錄中選擇目標專案 `aws_demos_eth.uvproj`。
4. 為確保您的設定正確以刷入主機板，請在 IDE 中以滑鼠右鍵按一下 `aws_demo` 專案，然後選擇 Options (選項)。(請參閱 [Troubleshooting \(p. 97\)](#) 獲得詳細資訊。)
5. 在 Utilities (公用程式) 標籤上，確認 Use Target Driver for Flash Programming (使用目標驅動程式進行 Flash 程式設計) 已選取，且 Nuvoton Nu-Link Debugger 已設為目標驅動程式。
6. 在 Debug (除錯) 索引標籤的 Nuvoton Nu-Link Debugger (Nuvoton Nu-Link 除錯器) 旁，選擇 Settings (設定)。
7. 確認 Chip Type (晶片類型) 設定為 M480。
8. 在 Keil μVision IDE Project (Keil μVision IDE 專案) 導覽窗格中，選擇 `aws_demos` 專案。在 Project (專案) 功能表中，選擇 Build Target (建置目標)。

您可以在 AWS IoT 主控台中使用 MQTT 用戶端來監控裝置傳送至 AWS 雲端的訊息。

使用 AWS IoT MQTT 用戶端訂閱 MQTT 主題

1. 登入 [AWS IoT 主控台](#)。
2. 在導覽窗格中，選擇 Test (測試) 以開啟 MQTT 用戶端。
3. 在訂閱主題中輸入 `iotdemo/#`，然後選擇訂閱主題。

執行 FreeRTOS 示範專案

1. 將 Numaker-IoT-M487 主機板連接到主機 (電腦)。
2. 重新建置專案。
3. 在 Keil μVision IDE 的 Flash 功能表上，選擇 Download (下載)。

4. 在 Debug (偵錯) 選單上，選擇 Start/Stop Debug Session (啟動/停止偵錯工作階段)。
5. 當除錯器停在中斷點 main() 時，開啟 Run (執行) 功能表，然後選擇 Run (執行) (F5))。

在 AWS IoT 主控台的 MQTT 用戶端中，應該會顯示您裝置傳送的 MQTT 訊息。

將 CMake 與 FreeRTOS 搭配使用

您也可以使用 CMake 來建置並執行 FreeRTOS 示範應用程式，或使用第三方程式碼編輯器和除錯工具所開發的應用程式。

請確定您已安裝 CMake 建置系統。請按照 [使用 CMake 配 FreeRTOS \(p. 20\)](#) 中的指示，再遵循本主題中的步驟。

Note

請確定編譯器 (Keil) 位置的路徑位於您的 Path (路徑) 系統變數中，例如 C:\Keil_v5\ARM\ARMCC\bin。

您也可以在 AWS IoT 主控台中使用 MQTT 用戶端來監控裝置傳送至 AWS 雲端的訊息。

使用 AWS IoT MQTT 用戶端訂閱 MQTT 主題

1. 登入 [AWS IoT 主控台](#)。
2. 在導覽窗格中，選擇 Test (測試) 以開啟 MQTT 用戶端。
3. 在訂閱主題中輸入 iotdemo/#，然後選擇訂閱主題。

從來源檔案產生建置檔案並執行示範專案

1. 在您的主機機器上，打開命令提示字元並導覽至 *freertos* folder。
2. 建立一個資料夾，其中應包含產生的建置檔案 我們將此資料夾稱為 *BUILD_FOLDER*。
3. 產生 Wi-Fi 或乙太網路示範的建置檔案。

- 若為 Wi-Fi：

導覽到包含 FreeRTOS 示範專案之來源檔的目錄。然後，執行下列命令來產生建置檔案。

```
cmake -DVENDOR=nuvoton -DBOARD=numaker_iot_m487_wifi -DCOMPILER=arm-keil -S . -B BUILD_FOLDER -G Ninja
```

- 若為乙太網路：

導覽到包含 FreeRTOS 示範專案之來源檔的目錄。然後，執行下列命令來產生建置檔案。

```
cmake -DVENDOR=nuvoton -DBOARD=numaker_iot_m487_wifi -DCOMPILER=arm-keil -DAFR_ENABLE_ETH=1 -S . -B BUILD_FOLDER -G Ninja
```

4. 執行以下命令，產生二進位以刷入至 M487。

```
cmake --build BUILD_FOLDER
```

此時，二進位檔案 aws_demos.bin 應該位於 *BUILD_FOLDER*/vendors/Nuvoton/boards/numaker_iot_m487_wifi 資料夾。

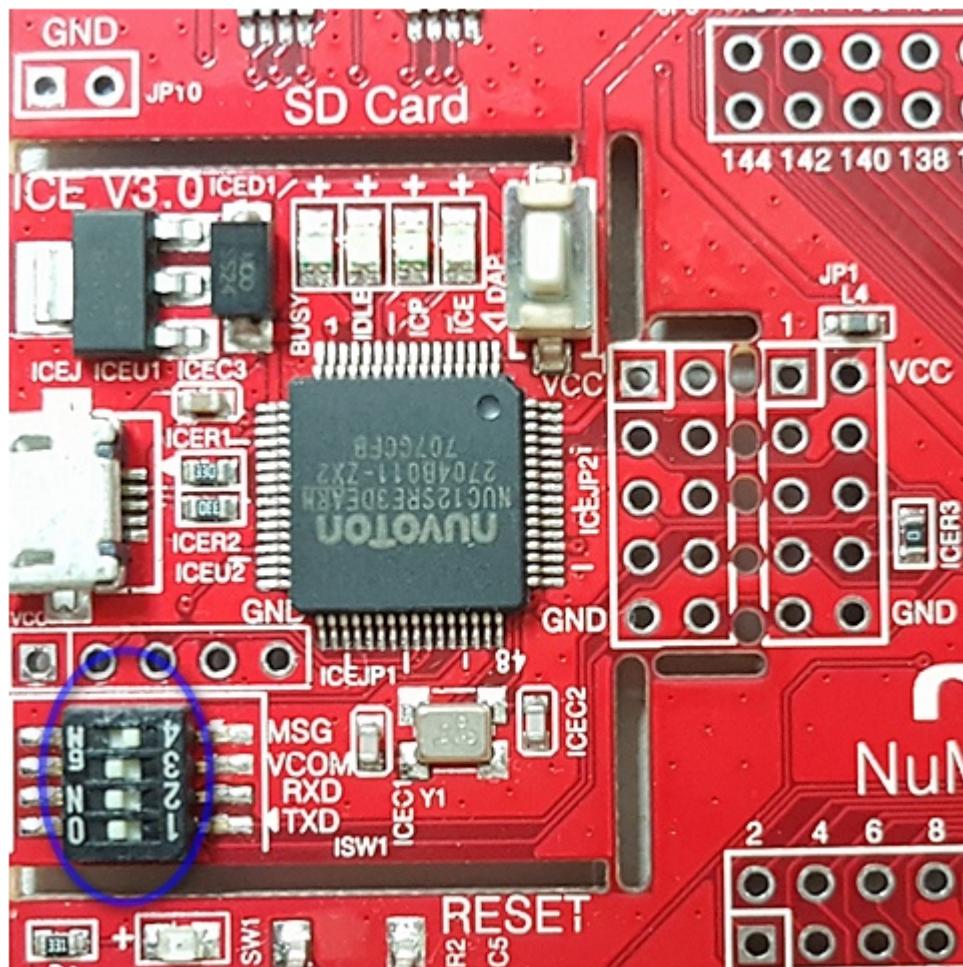
5. 若要將主機板設定為刷入模式，請確保 MSG 切換 (ICE 上 ISW1 的 4 號) 已切換為 ON (開啟)。當您插入主機板時，將會指派一個視窗 (和磁碟機)。(請參閱 [Troubleshooting \(p. 97\)](#).)
6. 開啟終端機模擬器，透過 UART 檢視訊息。按照 [安裝終端機模擬器 \(p. 20\)](#) 中的指示進行。

7. 將產生的二進位檔複製到裝置來執行示範專案。

如果您使用 AWS IoT MQTT 用戶端訂閱 MQTT 主題，您應該會在 AWS IoT 主控台中看到裝置傳送的 MQTT 訊息

Troubleshooting

- 如果您的 Windows 無法辨識裝置 VCOM，請從連結 NuMakerNu-Link USB Driver v1.6 [安裝](#) 視窗序列埠驅動程式。
- 如果您透過 Nu-Link 將裝置連接至 Keil MDK (IDE)，請確保 MSG 開關 (ICE 上 ISW1 的 4 號) 為關閉，如下所示。



如果您在設定開發環境或連接到主機板時遇到問題，請聯絡 [Nuvoton](#)。

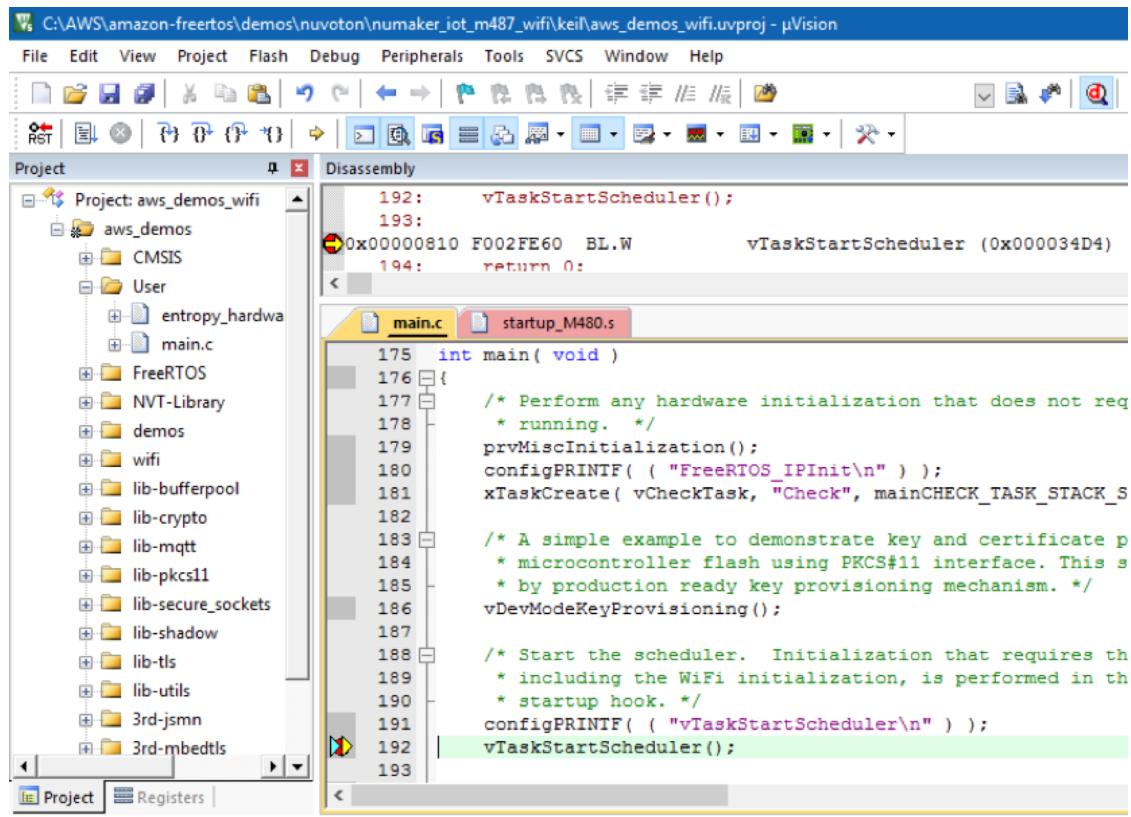
在 Keil µVision 中偵錯 FreeRTOS 專案

在 Keil µVision 中啟動偵錯工作階段

1. 開啟 Keil µVision。
2. 依照步驟在 [建置並執行 FreeRTOS 示範專案 \(p. 95\)](#) 中建置 FreeRTOS 示範專案。
3. 在 Debug (偵錯) 選單上，選擇 Start/Stop Debug Session (啟動/停止偵錯工作階段)。

當您啟動偵錯工作階段時，會顯示 Call Stack+Locals 視窗。 μ Vision 會刷入示範至主機板，執行示範，並在 main() 功能開始時停止。

4. 在專案的原始程式碼中設定中斷點，然後執行程式碼。專案應類似以下所示：

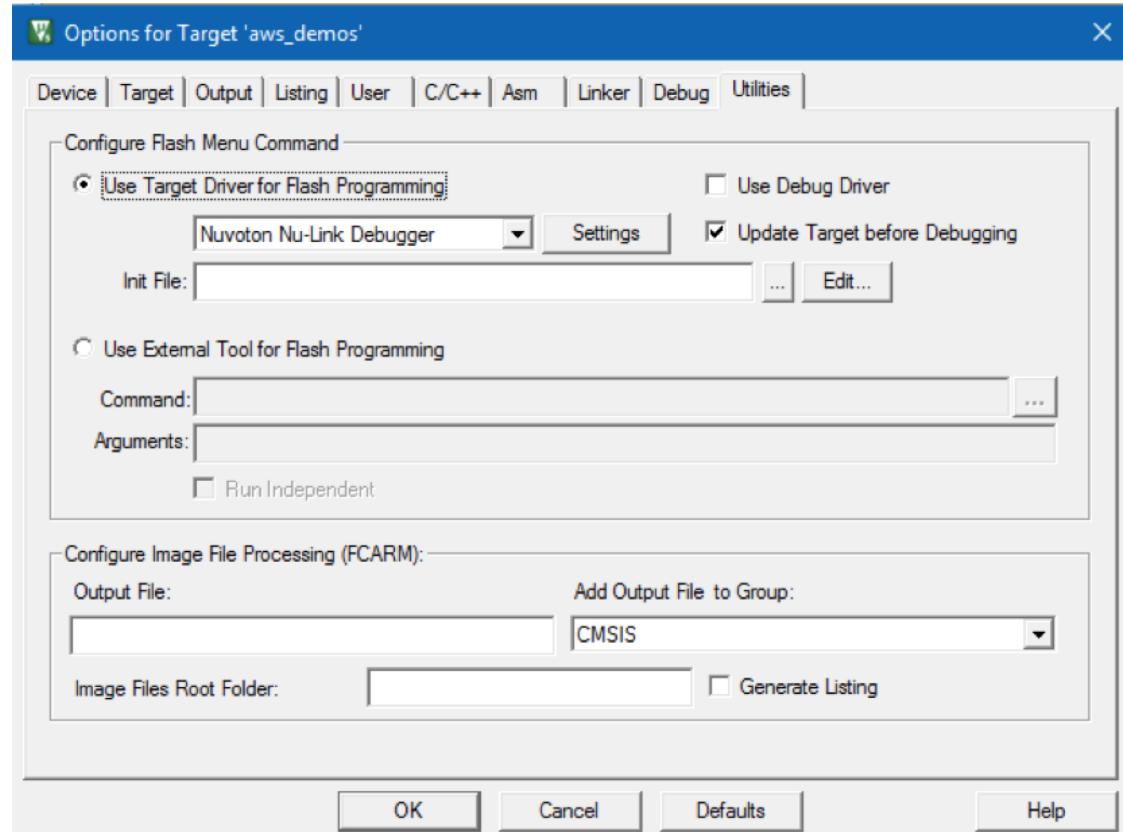


故障診斷 μ Vision 偵錯設定

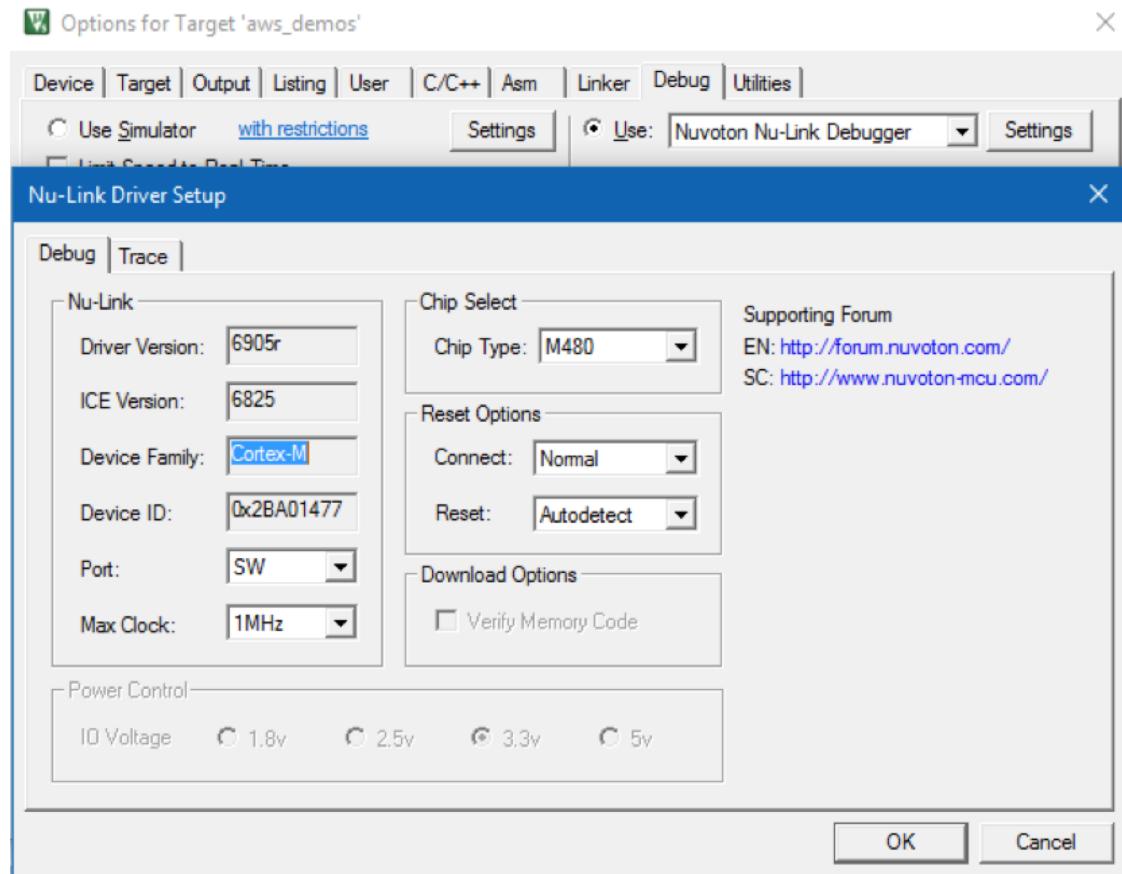
如果您在偵錯應用程式時遇到問題，請檢查您的偵錯設定是否在 Keil μ Vision 中正確設定。

若要驗證「 μ Vision」偵錯設定是否正確

1. 開啟 Keil μ Vision。
2. 在 IDE 中以滑鼠右鍵按一下 aws_demo 專案，然後選擇 Options (選項)。
3. 在 Utilities (公用程式) 標籤上，確認 Use Target Driver for Flash Programming (使用目標驅動程式進行 Flash 程式設計) 已選取，且 Nuvoton Nu-Link Debugger 已設為目標驅動程式。



4. 在 Debug (除錯) 索引標籤的 Nuvoton Nu-Link Debugger (Nuvoton Nu-Link 除錯器) 旁，選擇 Settings (設定)。



5. 確認 Chip Type (晶片類型) 設定為 M480。

NXP LPC54018 IoT 模組入門

本教學課程提供 NXP LPC54018 IoT 模組的入門指示。如果您未擁有 NXP LPC54018 IoT 模組，請前往 AWS Partner Device Catalog，向我們的[合作夥伴](#)購買。使用 USB 電波來將 NXP LPC54018 IoT 模組連接到您的電腦。

開始之前，請您務必要設定 AWS IoT 和 FreeRTOS 下載目錄，才能將裝置連接至 AWS 雲端。如需說明，請參閱[首要步驟 \(p. 14\)](#)。在本教學課程中，FreeRTOS 下載目錄的路徑會以 [freertos](#) 表示。

Overview

本教學課程包含以下入門步驟的指示：

1. 將主機板連線到主機機器。
2. 在主機機器上安裝軟體以對微控制器主機板的內嵌應用程式進行開發和除錯。
3. 將 FreeRTOS 示範應用程式跨編譯為二進位映像。
4. 將應用程式二進位映像載入主機板，然後執行應用程式。

設定 NXP 硬體

設定 NXP LPC54018

- 將您的電腦連接到 LPC54018 NXP 上的 USB 連接埠。

設定 JTAG 偵錯工具

您需要 JTAG 偵錯工具，在 NXP LPC54018 主機板上執行並偵錯您的程式碼。FreeRTOS 已使用 OM40006 IoT 模組進行測試。如需受支援除錯器的詳細資訊，請前往 OM40007 LPC54018 IoT 模組的 User Manual (使用者手動) [模組 IoT](#)，該模組可從 OM40007 LPC54018 模組頁面取得。

- 如果您使用的是 OM40006 IoT 模組偵錯工具，請使用轉換器表將 20 針連接器從偵錯工具連接到 NXP IoT 模組上的 10 針連接器。
- 使用迷你 USB 到 USB 連接的 NXP LPC54018 和 OM40006 IoT 模組除錯器，連接到電腦上的 USB 連接埠。

設定開發環境

FreeRTOS 支援兩種適用於 NXP LPC54018 IDEs 模組的 IoT：IAR Embedded Workbench 和 MCUXpresso。

開始之前，請安裝其中一個 IDEs。

安裝 IAR Embedded Workbench for ARM

- 瀏覽至 [IAR Embedded Workbench for ARM](#) 並下載軟體。

Note

IAR Embedded Workbench for ARM 必須使用 Microsoft Windows。

- 執行安裝程式並依照提示操作。
- 在 License Wizard (授權精靈) 中，選擇 Register with IAR Systems to get an evaluation license (註冊 IAR 系統以取得評估授權)。
- 在嘗試執行任何示範之前，請先將開機載入器放在裝置上。

從 NXP 安裝MCUXpresso

- 從 MCUXpressoNXP [下載並執行](#) 安裝程式。

Note

支援 10.3.x 版和更新版本。

- 瀏覽至 [MCUXpresso SDK](#)，然後選擇 Build your SDK (建置您的 SDK)。

Note

支援 2.5 版和更新版本。

- 選擇 Select Development Board (選取電路板)。
- 在 Select Development Board (選取電路板) 下方的 Search by Name (依名稱搜尋) 中，輸入 **LPC54018-IoT-Module**。
- 在 Boards (電路板) 中，選擇 LPC54018-IoT-Module (LPC54018 IoT 模組)。
- 驗證硬體詳細資訊，然後選擇 Build MCUXpresso SDK (建置 開發套件)。

7. 使用 MCUXpresso IDE 的 Windows 軟體開發套件已建置。選擇 Download SDK (下載軟體開發套件)。如果您使用其他作業系統，請在 Host OS (主機作業系統) 下方，選擇您的作業系統，然後選擇 Download SDK (下載軟體開發套件)。
8. 開始 MCUXpresso IDE，然後選擇 Installed (已安裝) SDKs 索引標籤。
9. 將下載的軟體開發套件封存檔案拖放到 Installed (已安裝) SDKs 視窗。

如果您在安裝時遇到問題，請參閱 [NXP Support](#) 或 [NXP Developer Resources](#)。

建置並執行 FreeRTOS 示範專案

將 FreeRTOS 示範匯入至您的 IDE

將 FreeRTOS 範本程式碼匯入 IAR Embedded Workbench IDE

1. 開啟 IAR Embedded Workbench，並從 File (檔案) 功能表中，選擇 Open Workspace (開啟工作空間)。
2. 在 search-directory (搜尋目錄) 文字方塊中，輸入 projects/nxp/lpc54018iotmodule/iar/aws_demos，然後選擇 aws_demos.evw。
3. 在 Project (專案) 功能表中，選擇 Rebuild All (全部重建)。

將 FreeRTOS 範本程式碼匯入 MCUXpresso IDE

1. 開放 MCUXpresso，然後從 File (檔案) 功能表中選擇 Open Projects From File System (從檔案系統開放專案)。
2. 在 Directory (目錄) 文字方塊中，輸入 projects/nxp/lpc54018iotmodule/mcuxpresso/aws_demos，然後選擇 Finish (完成)
3. 在 Project (專案) 功能表中，選擇 Build All (全部建置)。

執行 FreeRTOS 示範專案

透過 IAR 內嵌式 Workbench IDE 執行 FreeRTOS 示範專案

1. 在您 IDE 的 Project (專案) 功能表中，選擇 Make (Make)。
2. 從 Project (專案) 功能表中，選擇 Download and Debug (下載並除錯)。
3. 在 Debug (除錯) 功能表中，選擇 Start Debugging (開始除錯)。
4. 當除錯器停在 main 中斷點時，請從 Debug (除錯) 功能表，選擇 Go (開始)。

Note

如果 J-Link Device Selection (J-Link 選取裝置) 對話方塊開啟，請選擇 OK (確認) 以繼續。
在 Target Device Settings (目標裝置設定) 對話方塊中，選擇 Unspecified (未指定) 和 Cortex-M4，然後選擇 OK (確認)。您只需要執行此步驟一次。

透過 FreeRTOS IDE 執行 MCUXpresso 示範專案

1. 在您 IDE 的 Project (專案) 功能表中，選擇 Build (建置)。
2. 如果這是您第一次除錯，請選擇 aws_demos 專案，並從 Debug (除錯) 工具列，選擇藍色除錯按鈕。
3. 隨即顯示任何偵測到的除錯探查。選擇您想要使用的探查，然後選擇 OK (確認) 開始除錯。

Note



當除錯器停在 main() 中斷點時，請按下除錯重新開機按鈕一次來重設偵錯工作階段。
(這是必要的，因為 NXP54018-IoT-Module 的 MCUXpresso 除錯器出現錯誤)。

- 當除錯器停在 main() 中斷點時，請從 Debug (除錯) 功能表，選擇 Go (開始)。

監控雲端的 MQTT 訊息

您可以在 AWS IoT 主控台中使用 MQTT 用戶端來監控裝置傳送至 AWS 雲端的訊息。

使用 AWS IoT MQTT 用戶端訂閱 MQTT 主題

- 登入 [AWS IoT 主控台](#)。
- 在導覽窗格中，選擇 Test (測試) 以開啟 MQTT 用戶端。
- 在訂閱主題中輸入 `iotdemo/#`，然後選擇訂閱主題。

Troubleshooting

如需 FreeRTOS 入門的一般疑難排解資訊，請參閱 [故障診斷入門 \(p. 19\)](#)。

Renesas Starter Kit+ for RX65N-2MB 入門

本教學課程提供 RX65N-2MB 的 Renesas Starter Kit+ 入門指示。如果還沒有 Renesas RSK+ for RX65N-2MB，請造訪 AWS Partner Device Catalog，向我們的[合作夥伴](#)購買。

開始之前，請您務必要設定 AWS IoT 和 FreeRTOS 下載目錄，才能將裝置連接至 AWS 雲端。如需說明，請參閱 [首要步驟 \(p. 14\)](#)。在本教學課程中，FreeRTOS 下載目錄的路徑會以 `freertos` 表示。

Overview

本教學課程包含以下入門步驟的指示：

- 將主機板連線到主機機器。
- 在主機機器上安裝軟體以對微控制器主機板的內嵌應用程式進行開發和除錯。
- 將 FreeRTOS 示範應用程式跨編譯為二進位映像。
- 將應用程式二進位映像載入主機板，然後執行應用程式。

設定 Renesas 硬體

設定 RSK+ for RX65N-2MB

- 將正極 +5V 電源整流器連接到 RSK+RX65N-2MB 的 PWR 連接器。
- 將您的電腦連線至 RSK+ for RX65N-2MB 上的 USB 2.0 FS 連接埠。
- 將您的電腦連線至 RSK+ for RX65N-2MB 上的 USB 序列連接埠。
- 將路由器或連接網際網路的乙太網路連接埠連接至 RSK+ for RX65N-2MB 上的乙太網路連接埠。

設定 E2 Lite Debugger 模組

- 使用 14 針排線將 E2 Lite Debugger 模組連接至 RSK+ for RX65N-2MB 的「E1/E2 Lite」連接埠。

2. 使用 USB 纜線將 E2 Lite 偵錯工具模組連接到您的主機。當 E2 Lite 偵錯工具連接至開發板與您的電腦，偵錯工具上的綠色「ACT」LED 將會閃爍。
3. 將偵錯工具連接到您的主機和 RSK+ for RX65N-2MB 之後，E2 Lite 偵錯工具驅動程式將會開始安裝。

請注意，安裝驅動程式需要管理員的權限。



設定開發環境

若要設定 RSK+ for RX65N-2MB 的 FreeRTOS 組態，請使用 Renesas e²studio IDE 和 CC-RX 編譯器。

Note

支援 Renesas e²studio IDE 和 CC-RX 編譯器的只有 Windows 7、8 及 10 作業系統。

下載並安裝 e²studio。

1. 請至 [Renesas e²studio 安裝程式](#) 下載頁面，然後下載離線安裝程式。
2. 您將會導向至 Renesas 登入頁面。

如果您有 Renesas 的帳戶，請輸入您的使用者名稱和密碼，然後選擇 Login (登入)。

如果您沒有帳戶，請選擇立即 Register now (立即註冊)，然後依照第一個註冊步驟進行。您應該會收到一封電子郵件，其中包含可啟用您 Renesas 帳戶的連結。請使用此連結完成您的 Renesas 註冊，然後登入 Renesas。

3. 在您登入之後，請將 e²studio 安裝程式下載到您的電腦。
4. 開啟安裝程式並依照步驟完成安裝。

如需詳細資訊，請查看 Renesas 網站上的 [e²studio](#)。

下載並安裝 RX Family C/C++ 編譯器套件

1. 請至 [RX Family C/C++ 編譯器套件](#) 下載頁面，然後下載 V3.00.00 套件。
2. 開啟可執行檔並安裝編譯器。

如需詳細資訊，請參閱 Renesas 網站上的 [適用於 RX Family 的 C/C++ 編譯器套件](#)。

Note

此編譯器提供免費試用版，有效期為 60 天。到第 61 天，您就必須取得授權金鑰。如需詳細資訊，請參閱 [評估軟體工具](#)。

建置並執行 FreeRTOS 範例

現在您已設定示範專案，已可開始在開發板上建置並執行專案。

在 eFreeRTOS² 中建置 示範 studio

匯入並在 e² 中建置示範 studio

1. 從開始功能表啟動 e² studio。
2. 在 Select a directory as a workspace (選擇目錄做為工作空間) 視窗中，瀏覽到您希望使用的資料夾，然後選擇 Launch (啟動)。
3. 當您第一次開啟 e² studio 時，Toolchain Registry (工具鏈註冊) 視窗將會開啟。選擇 Renesas Toolchains (Renesas 工具鏈)，然後確認已選取 CC-RX v3.00.00。選擇 Register (註冊)，然後選擇 OK (確定)。
4. 如果您是第一次開啟 e² studio，將會顯示 Code Generator Registration (程式碼產生器註冊) 視窗。選擇 OK (確定)。
5. 此時將會顯示 Code Generator COM component register (程式碼產生器 COM 元件登錄) 視窗。在 Please restart e² studio to use Code Generator (請重新開機 e² 與 studio 以使用 Code Generator) 下，選擇 OK (確定)。
6. Restart e² studio (Restart e²) 視窗隨即出現。選擇 OK (確定)。
7. e² studio 重新啟動。在 Select a directory as a workspace (選擇目錄做為工作空間) 視窗中，選擇 Launch (啟動)。
8. 在 e² studio 歡迎畫面中，選擇 Go to the e² studio workbench (前往 e² 桌面工作台) 箭頭圖示。
9. 在 Project Explorer (專案瀏覽器) 視窗上按一下滑鼠右鍵，然後選擇 Import (匯入)。
10. 在匯入精靈中，選擇 General (一般)、Existing Projects into Workspace (現有專案到工作空間)，然後選擇 Next (下一步)。
11. 選擇 Browse (瀏覽)，找到目錄 projects/renesas/rx65n-rsk/e2studio/aws_demos，然後選擇 Finish (完成)。
12. 在 Project (專案) 功能表中，選擇 Project (專案)、Build All (全部建置)。

建置主控台會發出警告訊息，指出尚未安裝 License Manager。您可以忽略此訊息，除非您有 CC-RX 編譯器的授權金鑰。若要安裝 License Manager，請參閱 [License Manager](#) 下載頁面。

執行 FreeRTOS 專案

在 e² studio 中執行專案

1. 確認您已將 E2 Lite Debugger 模組連接到 RSK+ for RX65N-2MB
2. 在最上層功能表中，選擇 Run (執行)、Debug Configuration (偵錯設定)。
3. 展開 Renesas GDB Hardware Debugging (Renesas GDB 硬體偵錯)，然後選擇 aws_demos HardwareDebug。
4. 選擇 Debugger (偵錯工具) 索引標籤，然後選擇 Connection Settings (連線設定) 標籤。確認您的連線設定正確。
5. 選擇 Debug (偵錯) 將程式碼下載到您的開發板並開始偵錯。

您可能會收到 e2-server-gdb.exe 的 firewall 警告提示。檢查 Private networks, as my home or work network (私有網路，例如家用或工作網路)，然後選擇 Allow access (允許存取)。

6. e² studio 可能會要求變更為 Renesas Debug Perspective (Renesas 偵錯觀點)。選擇 Yes (是)。

E2 Lite Debugger 上的「ACT」LED 燈號將會亮起。

7. 將程式碼下載到開發板之後，請選擇 Resume (恢復) 將程式碼執行到主要函數的第一行。再次選擇 Resume (恢復) 以執行其餘程式碼。

監控雲端的 MQTT 訊息

您可以在 AWS IoT 主控台中使用 MQTT 用戶端來監控裝置傳送至 AWS 雲端的訊息。

使用 AWS IoT MQTT 用戶端訂閱 MQTT 主題

1. 登入 [AWS IoT 主控台](#)。
2. 在導覽窗格中，選擇 Test (測試) 以開啟 MQTT 用戶端。
3. 在訂閱主題中輸入 `iotdemo/#`，然後選擇訂閱主題。

如需 Renesas 發行的最新專案，請查看 [renesas-rx](#) 上 [amazon-freertos](#) 儲存庫的 [分支GitHub](#)。

Troubleshooting

如需 FreeRTOS 入門的一般疑難排解資訊，請參閱 [故障診斷入門 \(p. 19\)](#)。

STM32L4 探索套件STMicroelectronics節點入門IoT

本教學課程提供 STMicroelectronics STM32L4 Discovery Kit IoT 節點入門的指示。如果您尚未擁有 STMicroelectronics STM32L4 Discovery Kit IoT 節點，請前往 AWS Partner Device Catalog，向我們的[合作夥伴](#)購買。

確定已安裝最新的 Wi-Fi 韌體。若要下載最新的 Wi-Fi 韌體，請查看 [STM32L4 Discovery 套件IoT節點、低功耗無網路、低功耗藍牙、NFC、SubGHz、Wi-Fi](#)。在 Binary Resources (二進位資源) 中，選擇 Inventek ISM 43362 Wi-Fi module firmware update (read the readme file for instructions) (Inventek ISM 43362 Wi-Fi 模組韌體更新 (讀取讀我檔中的說明))。

開始之前，請您務必要設定 AWS IoT 和 FreeRTOS 下載目錄，才能將裝置連接至 AWS 雲端。如需說明，請參閱 [首要步驟 \(p. 14\)](#)。在本教學課程中，FreeRTOS 下載目錄的路徑會以 `freertos` 表示。

Overview

本教學課程包含以下入門步驟的指示：

1. 在主機機器上安裝軟體以對微控制器主機板的內嵌應用程式進行開發和除錯。
2. 將 FreeRTOS 示範應用程式跨編譯為二進位映像。
3. 將應用程式二進位映像載入主機板，然後執行應用程式。

設定開發環境

安裝 System Workbench for STM32

1. 瀏覽 [OpenSTM32.org](#)。
2. 在 OpenSTM32 網頁上登記。您需要登入以下載 System Workbench。
3. 瀏覽至 [System Workbench for STM32 安裝程式](#)，以下載並安裝 System Workbench。

如果您在安裝期間遇到問題，請查看 [FAQs](#) [System Workbench](#) 網站上的 [。](#)

建置並執行 FreeRTOS 示範專案

將 FreeRTOS 示範匯入至 STM32 System Workbench

1. 開啟 STM32 System Workbench，然後輸入新工作空間的名稱。

2. 從 File (檔案) 功能表中，選擇 Import (匯入)。展開 General (一般)，選擇 Existing Projects into Workspace (現有專案到工作空間)，然後選擇 Next (下一步)。
3. 在 Select Root Directory (選取根目錄) 中，輸入 projects/st/stm32l475_discovery/ac6/aws_demos。
4. 預設應選取 aws_demos 專案。
5. 選擇 Finish (完成) 以將專案匯入 STM32 System Workbench。
6. 在 Project (專案) 功能表中，選擇 Build All (全部建置)。確認專案編譯時未發生錯誤。

執行 FreeRTOS 示範專案

1. 使用 USB 連接到 STMicroelectronics STM32L4 探索套件 IoT 節點到您的電腦。
2. 從 Project Explorer (專案瀏覽器) 中，以滑鼠右鍵按一下 aws_demos，選擇 Debug As (除錯工具)，然後選擇 Ac6 STM32 C/C++ Application (Ac6 STM32 C/C++ 應用程式)。

如果第一次除錯工作階段啟動時發生除錯錯誤，請依循下列步驟進行操作：

1. 在 STM32 System Workbench 的 Run (執行) 功能表中，選擇 Debug Configurations (組態除錯)。
2. 選擇 aws_demos Debug (aws_demos 除錯)。(您可能需要展開 Ac6 STM32 Debugging (Ac6 STM32 除錯)。)
3. 選擇 Debugger (除錯器) 標籤。
4. 在 Configuration Script (組態指令碼) 中，選擇 Show Generator Options (顯示產生器選項)。
5. 在 Mode Setup (模式設定) 中，將 Reset Mode (重設模式) 設為 Software System Reset (軟體系統重設)。選擇 Apply (套用)，然後選擇 Debug (偵錯)。
3. 當除錯器停在 main() 中斷點時，請從 Run (執行) 功能表，選擇 Resume (繼續)。

將 CMake 與 FreeRTOS 搭配使用

如果您不想使用 IDE 進行 FreeRTOS 開發，您也可以改為使用 CMake 來建置和執行示範應用程式，或使用第三方程式碼編輯器和除錯工具所開發的應用程式。

首先建立一個資料夾以包含生成的建置檔案 (*build-folder*)。

使用下列命令來產生建置檔案：

```
cmake -DVENDOR=st -DBOARD=stm32l475_discovery -DCOMPILER=arm-gcc -S freertos -B build-folder
```

如果 arm-none-eabi-gcc 不在您的 shell 路徑中，您也需要設定 AFR_TOOLCHAIN_PATH CMake 變數。例如：

```
-D AFR_TOOLCHAIN_PATH=/home/user/opt/gcc-arm-none-eabi/bin
```

如需使用 CMake 搭配 FreeRTOS 的詳細資訊，請參閱 [使用 CMake 配 FreeRTOS \(p. 20\)](#)。

監控雲端的 MQTT 訊息

您可以在 AWS IoT 主控台中使用 MQTT 用戶端來監控裝置傳送至 AWS 雲端的訊息。

使用 AWS IoT MQTT 用戶端訂閱 MQTT 主題

1. 登入 [AWS IoT 主控台](#)。
2. 在導覽窗格中，選擇 Test (測試) 以開啟 MQTT 用戶端。

- 在訂閱主題中輸入 `iotdemo/#`，然後選擇訂閱主題。

Troubleshooting

如果示範應用程式的 UART 輸出中顯示以下項目，則需要更新 Wi-Fi 模組的韌體：

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

若要下載最新的 Wi-Fi 韌體，請查看 [STM32L4 Discovery 套件](#) IoT 節點、低功耗無線路、低功耗藍牙、NFC、SubGHz、Wi-Fi。在 Binary Resources (二進位資源) 中，選擇 Inventek ISM 43362 Wi-Fi module firmware update (Inventek ISM 43362 Wi-Fi 模組韌體更新) 的下載連結。

如需 FreeRTOS 入門的一般疑難排解資訊，請參閱 [故障診斷入門 \(p. 19\)](#)。

Texas Instruments CC3220SF-LAUNCHXL 入門

本教學課程提供 Texas Instruments CC3220SF-LAUNCHXL 入門指示。如果您還沒有 Texas Instruments (TI) CC3220SF-LAUNCHXL 開發套件，請瀏覽 AWS 合作夥伴裝置目錄，向我們的[合作夥伴](#)購買此裝置。

開始之前，請您務必要設定 AWS IoT 和 FreeRTOS 下載目錄，才能將裝置連接至 AWS 雲端。如需說明，請參閱 [首要步驟 \(p. 14\)](#)。在本教學課程中，FreeRTOS 下載目錄的路徑會以 `freertos` 表示。

Overview

本教學課程包含以下入門步驟的指示：

- 在主機機器上安裝軟體以對微控制器主機板的內嵌應用程式進行開發和除錯。
- 將 FreeRTOS 示範應用程式跨編譯為二進位映像。
- 將應用程式二進位映像載入主機板，然後執行應用程式。

設定開發環境

請依照下面的步驟，以設定 FreeRTOS 入門的開發環境。

請注意，FreeRTOS 支援兩種 IDEs 用於 TI CC3220SF-LAUNCHXL 開發套件：Code Composer Studio 和 IAR Embedded Workbench 8.32 版。您可以使用這兩種 IDE 來開始使用。

安裝 Code Composer Studio

- 瀏覽至 [TI Code Composer Studio](#)。
- 下載主機平台的 offline 安裝程式 (Windows、macOS 或 Linux 64 位元)。
- 解壓縮並執行離線安裝程式。依照提示進行。
- 針對 Product Famis to Install (要安裝的系列)，選擇 Wi-Fi SimpleLink Wireless CC32xx。MCUs
- 在下一個頁面中，接受預設的除錯探查設定，然後選擇 Finish (完成)。

如果您在安裝 Code Composer Studio 時遇到問題，請查看 [TI 開發工具支援](#)、[Code Composer StudioFAQs](#) 和 [故障診斷 CCS](#)。

安裝 IAR Embedded Workbench

- 下載並執行 IAR Embedded Workbench for ARM 的 [8.32 版 Windows 安裝程式](#)。在 Debug probe drivers (除錯探查驅動程式) 中，確認已選取 TI XDS。

- 完成安裝程序，然後啟動程式。在 License Wizard (授權精靈) 頁面中，選擇 Register with IAR Systems to get an evaluation license (註冊 IAR 系統以取得評估授權) 或使用自己的 IAR 授權。

安裝 SimpleLink CC3220 開發套件

安裝 [CC3220 軟體開發套件SimpleLink](#)。Wi-Fi CC3220 軟體開發套件包含適用於 CC3220SF 可程式化 MCU 的驅動程式、40 多種範例應用程式，以及使用範例所需的文件。SimpleLink

安裝 Uniflash

安裝 [Uniflash](#)。CCS Uniflash 是一種獨立工具，用於程式設計 TI MCUs 上的晶片快閃記憶體。Uniflash 具有 GUI、命令列和指令碼界面。

安裝最新的 Service Pack

- 在您的 TI CC3220SF-LAUNCHXL 中，將 SOP 跳接器調到中間的 pin (位置 = 1) 上並重設電路板。
- 啟動 Uniflash。如果 CC3220SF LaunchPad 主機板出現在 Detected Devices (已偵測的裝置) 下，請選擇 Start (開始)。如果未偵測到您的電路板，請從 New Configuration (新組態) 下的電路板清單中選擇 CC3220SF-LAUNCHXL，然後選擇 Start Image Creator (開始映像建立器)。
- 選擇 New Project (新專案)。
- 在 Start new project (開始新專案) 頁面上，輸入您專案的名稱。針對 Device Type (裝置類型)，選擇 CC3220SF。針對 Device Mode (裝置模式)，選擇 Develop (開發)，然後選擇 Create Project (建立專案)。
- 在 Uniflash 應用程式視窗的右側，選擇 Connect (連線)。
- 選擇左側欄中的 Advanced (進階)、Files (檔案)，然後選擇 Service Pack (Service Pack)。
- 選擇 Browse (瀏覽)，然後導覽至您安裝 CC3220SF SimpleLink 開發套件的位置。此服務套件位於 `ti/simplelink_cc32xx_sdk_<VERSION>/tools/cc32xx_tools/servicepack-cc3x20/sp_<VERSION>.bin`。

8.



選擇 Burn (大量) () 按鈕，然後選擇 Program Image (Create & Program) (程式映像 (建立與程式設計)) 以安裝服務套件。請記得將 SOP 跳接器切換回位置 0，並重設電路板。

設定 Wi-Fi 佈建

若要為您的電路板進行 Wi-Fi 設定，請執行以下其中一項：

- 設定 FreeRTOS 示範應用程式，如[設定 FreeRTOS 示範 \(p. 18\)](#)所述。
- 使用來自 Texas Instruments 的 [SmartConfig](#)。

建置並執行 FreeRTOS 示範專案

在 TI Code Composer 中，建置和執行 FreeRTOS 示範專案

將 FreeRTOS 示範匯入至 TI Code Composer

- 開啟 TI Code Composer，然後選擇 OK (確定) 以接受預設的工作空間名稱。
- 在 Getting Started (入門) 頁面上，選擇 Import Project (匯入專案)。
- 在 Select search-directory (選取搜尋方向) 中，輸入 `projects/ti/cc3220_launchpad/ccs/aws_demos`。預設應選取 `aws_demos` 專案。若要將專案匯入 TI Code Composer，請選擇 Finish (完成)。
- 在 Project Explorer (專案瀏覽器) 中，按兩下 `aws_demos` 將專案設為作用中。

- 從 Project (專案) 中，選擇 Build Project (建置專案)，以確保專案編譯時不會發生錯誤或警告。

在 TI Code Composer 中執行 FreeRTOS 示範

- 確認 Texas Instruments CC3220SF-LAUNCHXL 的 Sense On Power (SOP) 跳接器位於位置 0。如需詳細資訊，請前往 [Wi-Fi SimpleLink、CC3x20 網路處理器使用者指南CC3x3x](#)。
- 使用 USB 纜線將 Texas Instruments CC3220SF-LAUNCHXL 接到您的電腦。
- 在專案資源管理員中，請確定將 CC3220SF.ccxml 選取為作用中的目標組態。為了使其為作用中的狀態，請對該檔案按一下滑鼠右鍵，然後選擇 Set as active target configuration (設為作用中的目標組態)。
- 在 TI Code Composer 的 Run (執行) 中，選擇 Debug (除錯)。
- 當除錯工具停在 main() 中斷點時，請移至 Run (執行) 功能表，然後選擇 Resume (繼續)。

在 IAR Embedded Workbench 中建置和執行 FreeRTOS 示範專案

將 FreeRTOS 示範匯入至 IAR Embedded Workbench

- 開啟 IAR Embedded Workbench，選擇 File (檔案)，然後選擇 Open Workspace (開啟工作空間)。
- 導覽至 projects/ti/cc3220_launchpad/iar/aws_demos，並選擇 aws_demos.eww，然後選擇 OK (確定)。
- 用滑鼠右鍵按一下專案名稱 (aws_demos)，然後選擇 Make (設為)。

在 IAR Embedded Workbench 中，執行 FreeRTOS 示範

- 確認 Texas Instruments CC3220SF-LAUNCHXL 的 Sense On Power (SOP) 跳接器位於位置 0。如需詳細資訊，請參見 [Wi-Fi SimpleLink、CC3x20 網路處理器使用者指南CC3x3x](#)。
- 使用 USB 纜線將 Texas Instruments CC3220SF-LAUNCHXL 接到您的電腦。
- 重新組建專案。
若要重建專案，請從 Project (專案) 功能表中，選擇 Make (設為)。
- 從 Project (專案) 功能表中，選擇 Download and Debug (下載並除錯)。您可以忽略「警告：如果顯示，表示無法初始化 EnergyTrace。」如需 EnergyTrace 的詳細資訊，請前往 [MSP EnergyTrace 技術](#)。
- 當除錯工具停在 main() 中斷點時，請移至 Debug (除錯) 功能表，然後選擇 Go (開始)。

將 CMake 與 FreeRTOS 搭配使用

如果您不想使用 IDE 進行 FreeRTOS 開發，您也可以改為使用 CMake 來建置和執行示範應用程式，或使用第三方程式碼編輯器和除錯工具所開發的應用程式。

使用 FreeRTOS 建置 CMake 示範

- 建立資料夾以包含生成的建置檔案 (*build-folder*)。
- 請確定您的搜尋路徑 (\$PATH 環境變數) 包含 TI CGT 編譯器二進位檔所在位置的資料夾 (例如 c:\ti\ccs910\ccs\tools\compiler\ti-cgt-arm_18.12.2.LTS\bin)。

如果您是搭配 TI ARM 編譯器使用 TI 主機板，請透過下列命令從原始程式碼產生建置檔案：

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

如需詳細資訊，請參閱 [使用 CMake 配 FreeRTOS \(p. 20\)](#)。

監控雲端的 MQTT 訊息

您可以在 AWS IoT 主控台中使用 MQTT 用戶端來監控裝置傳送至 AWS 雲端的訊息。

使用 AWS IoT MQTT 用戶端訂閱 MQTT 主題

1. 登入 [AWS IoT 主控台](#)。
2. 在導覽窗格中，選擇 Test (測試) 以開啟 MQTT 用戶端。
3. 在訂閱主題中輸入 `iotdemo/#`，然後選擇訂閱主題。

Troubleshooting

如果您在 AWS IoT 主控台的 MQTT 用戶端中沒有看到訊息，則可能需要為電路板進行除錯設定。

若要為 TI 主機板進行偵錯設定

1. 在 Code Composer 的 Project Explorer (專案瀏覽器) 中，選擇 `aws_demos`。
2. 在 Run (執行) 選單中，選擇 Debug Configurations (組態除錯)。
3. 在導覽窗格中，選擇 `aws_demos`。
4. 在 Target (目標) 標籤中，選擇 Connection Options (連線選項) 下方的 Reset the target on a connect (重設連線目標)。
5. 選擇 Apply (套用)，然後選擇 Close (關閉)。

如果無法使用這些步驟，請查看序列終端機中的程式輸出。您應該會看到一些文字，指出問題來源。

如需 FreeRTOS 入門的一般疑難排解資訊，請參閱 [故障診斷入門 \(p. 19\)](#)。

Windows 裝置模擬器入門

本教學課程提供 FreeRTOS Windows 裝置模擬器入門指示。

開始之前，請您務必要設定 AWS IoT 和 FreeRTOS 下載目錄，才能將裝置連接至 AWS 雲端。如需說明，請參閱 [首要步驟 \(p. 14\)](#)。在本教學課程中，FreeRTOS 下載目錄的路徑會以 `freertos` 表示。

FreeRTOS 是以 zip 檔的形式發佈，其中包含您指定平台的 FreeRTOS 程式庫和範例應用程式。若要在 Windows 電腦上執行範例，請下載程式庫和隨附的範例以在 Windows 中執行。這一組檔案就是適用於 Windows 的 FreeRTOS 模擬器。

Note

本教學課程無法在 Amazon EC2 Windows 執行個體上成功執行。

設定開發環境

1. 安裝最新版本的 [WinPCap](#)。
2. 安裝 [Microsoft Visual Studio](#)。

Visual Studio 版本 2017 和 2019 已知可運作。支援所有這些 Visual Studio 版本 (Community、Professional 或 Enterprise)。

除 IDE 外，安裝使用 C++ 的桌面開發元件。

安裝最新的 Windows 10 軟體開發套件。您可以在使用 C++ 的桌面開發元件的 Optional (選用) 部分選擇此項目。

3. 請確認您的有線乙太網路連線為作用中。

4. (選用) 如果您想要使用以 CMake 為基礎的建置系統來建置您的 FreeRTOS 專案，請安裝最新版本的 CMake。FreeRTOS 需要 CMake 版本 3.13 或更新版本。

建置並執行 FreeRTOS 示範專案

您可以使用 Visual Studio 或 CMake 以建置 FreeRTOS 專案。

透過 Visual Studio IDE 建置和執行 FreeRTOS 示範專案

1. 將專案載入到 Visual Studio。

在 Visual Studio 的 File (檔案) 功能表中，選擇 Open (開啟)。選擇 File/Solution (檔案/解決方案)，導覽至 `projects/pc/windows/visual_studio/aws_demos/aws_demos.sln` 檔案，然後選擇 Open (開啟)。

2. 重新定向示範專案。

提供的示範專案取決於 Windows 開發套件，但該專案沒有指定的 Windows 開發套件版本。在預設情況下，IDE 可能會嘗試使用您電腦中未呈現的軟體開發套件版本來建置示範。若要設定 Windows 軟體開發套件版本，請在 `aws_demos` 按一下滑鼠右鍵，然後選擇 Retarget Projects (重新定向專案)。這會開啟 Review Solution Actions (檢閱解決方案動作) 視窗。選擇您電腦上呈現的 Windows 軟體開發套件版本 (使用下拉式清單中的初始值即可)，然後選擇 OK (確定)。

3. 建置並執行專案。

從 Build (建置) 功能表中，選擇 Build Solution (建置解決方案)，並確認解決方案建置時未發生錯誤或警告。選擇 Debug (偵錯)、Start Debugging (開始偵錯) 以執行專案。在初次執行時，您將需要選擇一個網路界面 (p. 113)。

透過 CMake 建置並執行 FreeRTOS 示範專案。

我們建議您使用 CMake GUI (不要使用 CMake 命令列工具) 以建置適用於 Windows 模擬器的示範專案。

在您安裝 CMake 後，開啟 CMake GUI。在 Windows 上，您可以從開始功能表下的 CMake、CMake (cmake-gui) 下找到此項目。

1. 設定 FreeRTOS 來源程式碼目錄。

在 GUI 中，將來源碼位置設定為 FreeRTOS 來源程式碼目錄 (`freertos`)。

將建置二進位程式碼的位置設定為 `freertos/build`。

2. 設定 CMake 專案。

在 CMake GUI 中，請選擇 Add Entry (新增輸入)，並在 Add Cache Entry (新增快取輸入) 視窗中，設定以下值：

名稱

AFR_BOARD

類型

STRING

值

pc.windows

描述

(選用)

3. 請選擇 Configure (設定)。如果 CMake 提示您建立建置目錄，請選擇 Yes (是)，然後選取 Specify the generator for this project (為此專案指定產生器) 下的產生器。我們建議您使用 Visual Studio 作為產生器，但也支援使用 Ninja。(請注意，使用 Visual Studio 2019 時，版本應設為 Win32，而非使用其預設設定。) 將其他產生器選項維持不變，然後選擇 Finish (完成)。
4. 產生並開啟 CMake 專案。

在您設定專案後，CMake GUI 會顯示所有產生的專案之可用選項。基於本教學課程的目的，您可以將選項保留為其預設值。

選擇 Generate (產生) 以建立 Visual Studio 解決方案，然後選擇 Open Project (開啟專案) 以在 Visual Studio 中開啟專案。

在 Visual Studio 中，請在 aws_demos 專案按一下滑鼠右鍵，然後選擇 Set as StartUp Project (設為啟動專案)。這可讓您建置並執行該專案。在初次執行時，您將需要選擇一個網路界面 (p. 113)。

如需搭配 FreeRTOS 使用 CMake 的詳細資訊，請參閱 [使用 CMake 配 FreeRTOS \(p. 20\)](#)。

設定網路界面

在初次執行示範專案時，您必須選擇要使用的網路界面。程式會列舉您的網路界面。找出您的有線乙太網路界面編號。輸出應如下所示：

```
0 0 [None] FreeRTOS_IPInit
1 0 [None] vTaskStartScheduler
1. rpcap://\Device\NPF_{AD01B877-A0C1-4F33-8256-EE1F4480B70D}
(Network adapter 'Intel(R) Ethernet Connection (4) I219-LM' on local host)

2. rpcap://\Device\NPF_{337F7AF9-2520-4667-8EFF-2B575A98B580}
(Network adapter 'Microsoft' on local host)

The interface that will be opened is set by "configNETWORK_INTERFACE_TO_USE", which
should be defined in FreeRTOSConfig.h

ERROR: configNETWORK_INTERFACE_TO_USE is set to 0, which is an invalid value.
Please set configNETWORK_INTERFACE_TO_USE to one of the interface numbers listed above,
then re-compile and re-start the application. Only Ethernet (as opposed to Wi-Fi)
interfaces are supported.
```

在您找出有線乙太網路界面編號之後，請關閉應用程式視窗。在上述範例中使用的編號為 1。

開啟 FreeRTOSConfig.h，並將 configNETWORK_INTERFACE_TO_USE 設定為您有線乙太網路界面的對應編號。

Important

僅支援乙太網路介面。不支援 Wi-Fi。如需詳細資訊，請參閱 WinPcap FAQ 項目 [Q-16: Which network adapters are supported by WinPcap?](#)。

監控雲端的 MQTT 訊息

您可以在 AWS IoT 主控台中使用 MQTT 用戶端來監控裝置傳送至 AWS 雲端的訊息。

使用 AWS IoT MQTT 用戶端訂閱 MQTT 主題

1. 登入 [AWS IoT 主控台](#)。
2. 在導覽窗格中，選擇 Test (測試) 以開啟 MQTT 用戶端。
3. 在 Subscription topic (訂閱主題) 中輸入 `iotdemo/#`，然後選擇 Subscribe to topic (訂閱主題)。

故障診斷

在 Windows 執行常見問題故障診斷

您可能會在嘗試透過 Visual Studio 建置示範專案時遭遇以下錯誤：

```
Error "The Windows SDK version X.Y was not found" when building the provided Visual Studio solution.
```

該專案必須定位到您電腦上呈現的 Windows 軟體開發套件版本。

如需 FreeRTOS 入門的一般疑難排解資訊，請參閱 [故障診斷入門 \(p. 19\)](#)。

Xilinx Avnet MicroZed Industrial IoT 套件入門

本教學課程提供 Xilinx Avnet MicroZed Industrial IoT 套件入門指示。如果您未擁有 Xilinx Avnet MicroZed Industrial IoT Kit，請前往 AWS Partner Device Catalog，向我們的[合作夥伴](#)購買。

開始之前，請您務必要設定 AWS IoT 和 FreeRTOS 下載目錄，才能將裝置連接至 AWS 雲端。如需說明，請參閱 [首要步驟 \(p. 14\)](#)。在本教學課程中，FreeRTOS 下載目錄的路徑會以 *freertos* 表示。

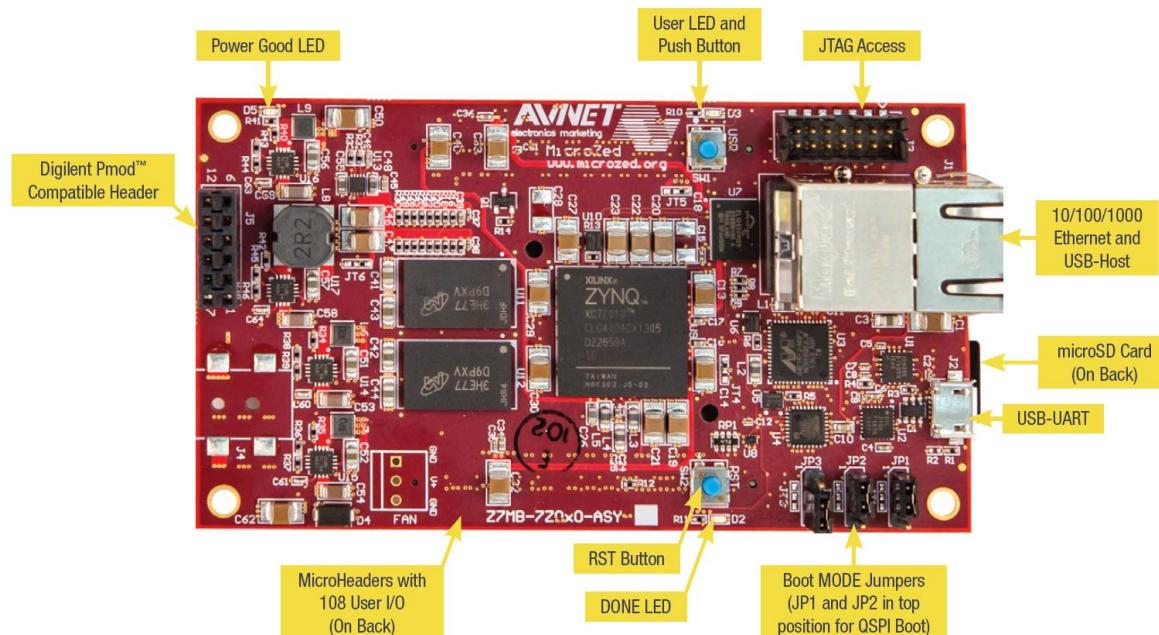
Overview

本教學課程包含以下入門步驟的指示：

1. 將主機板連線到主機機器。
2. 在主機機器上安裝軟體以對微控制器主機板的內嵌應用程式進行開發和除錯。
3. 將 FreeRTOS 示範應用程式跨編譯為二進位映像。
4. 將應用程式二進位映像載入主機板，然後執行應用程式。

設定 MicroZed 硬體

當您設定 MicroZed 硬體時，下圖可能有所助益：



設定 MicroZed 主機板

1. 將您的電腦連接到 MicroZed 電路板上的 USB-UART 連接埠。
2. 將您的電腦連接到 MicroZed 電路板的 JTAG Access 連接埠。
3. 將路由器或連接網際網路的乙太網路連接埠連接到您 MicroZed 主機板的乙太網路和 USB 主機連接埠。

設定開發環境

若要設定 FreeRTOS 套件的 MicroZed 組態，您必須使用 Xilinx 軟體開發套件（XSDK）。Windows 和 Linux 上支援 XSDK。

下載並安裝 XSDK

若要安裝 Xilinx 軟體，您需要免費的 Xilinx 帳戶。

下載 XSDK

1. 前往 [軟體開發套件獨立 WebInstall Client](#) 下載頁面。
2. 選擇適合您作業系統的選項。
3. 系統會將您導向至 Xilinx 登入頁面。

如果您有搭配 Xilinx 的帳戶，請輸入您的使用者名稱和密碼，然後選擇 Sign in (登入)。

如果您沒有帳戶，請選擇 Create your account (建立您的帳戶)。註冊後，您應該會收到一封電子郵件，其中包含可啟用您 Xilinx 帳戶的連結。

4. 在 Name and Address Verification (名稱與地址驗證) 頁面上，輸入您的資訊，然後選擇 Next (下一步)。下載應該可以準備開始了。
5. 儲存 `Xilinx_SDK_version_os` 檔案。

安裝 XSDK

1. 開啟 `Xilinx_SDK_version_os` 檔案。
2. 在 Select Edition to Install (選取要安裝的版本) 中，選擇 Xilinx Software Development Kit (XSDK) (Xilinx 軟體開發套件 (XSDK))，然後選擇 Next (下一步)。
3. 在安裝精靈的下列頁面上，於 Installation Options (安裝選項) 下，選取 Install Cable Drivers (安裝纜線驅動程式)，然後選擇 Next (下一步)。

如果您的電腦未偵測到 MicroZed 的 USB UART 連接，請手動安裝 CP210x USB 對 UART Bridge VCP 驅動程式。如需指示，請查看 [Silicon Labs CP210x USB 至 UART 安裝指南](#)。

如需 XSDK 的詳細資訊，請參閱 Xilinx 網站上的 [Getting Started with Xilinx SDK](#)。

建置並執行 FreeRTOS 示範專案

在 XSDK IDE 中開啟 FreeRTOS 示範

1. 啟動工作空間目錄設定為 `freertos/projects/xilinx/microzed/xsdk` 的 XSDK IDE。
2. 關閉歡迎頁面。從功能表中，選擇 Project (專案)，然後清除 Build Automatically (自動建立)。
3. 從功能表中，選擇 File (檔案)，然後選擇 Import (匯入)。
4. 在 Select (選取) 頁面上，展開 General (一般)、選擇 Existing Projects into Workspace (現有專案到工作空間)，然後選擇 Next (下一步)。

5. 在 Import Projects (匯入專案) 頁面上，選擇 Select root directory (選取根資料夾)，然後輸入示範專案的根資料夾：`freertos/projects/xilinx/microzed/xsdk/aws_demos -`。若要瀏覽此資料夾，請選擇 Browse (瀏覽)。

在您指定根目錄之後，該目錄中的專案會出現在 Import Projects (匯入專案) 頁面上。根據預設，會選取所有可用的專案。

Note

如果您在 Import Projects (匯入專案) 頁面頂部看到警告 (「有些專案無法匯入，因為它們已存在於工作區中。」)，則可以忽略它。

6. 在選取了所有專案後，請選擇 Finish (完成)。
7. 如果您在專案窗格中沒有看到 aws_bsp、fsbl 和 MicroZed_hw_platform_0 專案，請重複先前的步驟，請從第 3 步驟開始，但將根目錄設定為 `freertos/vendors/xilinx`，接著匯入 aws_bsp、fsbl 和 MicroZed_hw_platform_0。
8. 從功能表中，選擇 Window (視窗)，然後選擇 Preferences (偏好設定)。
9. 在導覽窗格中，展開 Run/Debug (執行/偵錯)、選擇 String Substitution (字串替換)，然後選擇 New (新增)。
10. 在 New String Substitution Variable (新增字串替換變數) 中，針對 Name (名稱)，輸入 **AFR_ROOT**。針對 Value (值)，輸入 `freertos/projects/xilinx/microzed/xsdk/aws_demos` 的根路徑。選擇 OK (確定)，然後選擇 OK (確定) 以儲存變數並關閉 Preferences (偏好設定)。

建置 FreeRTOS 示範專案

1. 在 XSDK IDE 中，從功能表中選擇 Project (專案)，然後選擇 Clean (清除)。
2. 在 Clean (清除) 中，保留選項的預設值，然後選擇 OK (確定)。XSDK 即會清除並建立所有專案，然後產生 .elf 檔案。

Note

若要建立所有專案，而不清除它們，請選擇 Project (專案)，然後選擇 Build All (全部建立)。

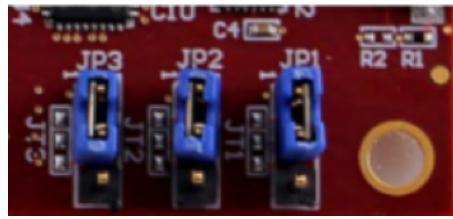
若要建立個別專案，請選取您要建置的專案、選擇 Project (專案)，然後選擇 Build Project (建置專案)。

產生 FreeRTOS 示範專案的開機映像

1. 在 XSDK IDE 中，於 aws_demos 上按一下滑鼠右鍵，然後選擇 Create Boot Image (建立開機映像)。
2. 在 Create Boot Image (建立開機映像) 中，選擇 Create new BIF file (建立新的 BIF 檔案)。
3. 在 Output BIF file path (輸出 BIF 檔案路徑) 旁邊，選擇 Browse (瀏覽)，然後選擇位於 `<freertos/vendors/xilinx/microzed/aws_demos/aws_demos.bif` 的 aws_demos.bif。
4. 選擇 Add (新增)。
5. 在 Add new boot image partition (新增開機映像分割區) 上，於 File path (檔案路徑) 旁邊，選擇 Browse (瀏覽)，然後選擇位於 `vendors/xilinx/fsbl/Debug/fsbl.elf` 的 fsbl.elf。
6. 對於 Partition type (分割區類型)，選擇 bootloader，然後選擇 OK (確定)。
7. 在 Create Boot Image (建立開機映像) 上，選擇 Create Image (建立映像)。在 Override Files (覆寫檔案) 上，選擇 OK (確定) 以覆寫現有的 aws_demos.bif，並在 `projects/xilinx/microzed/xsdk/aws_demos/BOOT.bin` 產生 BOOT.bin 檔案。

JTAG 偵錯

1. 將您的 MicroZed 主機板的開機模式跳接器設定為 JTAG 開機模式。

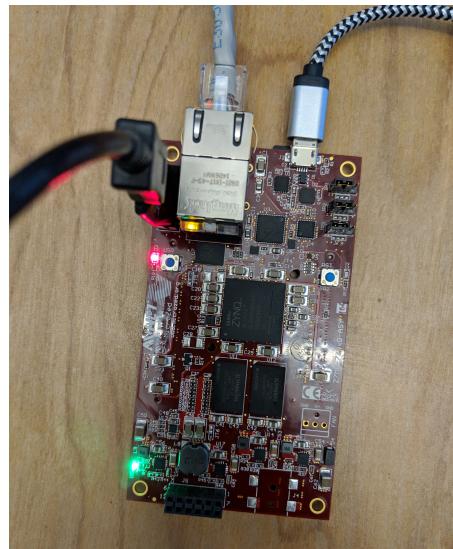


- 將您的 MicroSD 卡直接插入至位於 USB UART 連接埠下方的 MicroSD 卡槽。

Note

除錯之前，請務必備份 MicroSD 卡片上的任何檔案。

您的主機板看起來如下：



- 在 XSDK IDE 中，於 aws_demos 上按一下滑鼠右鍵、選擇 Debug As (偵錯工具)，然後選擇 1 Launch on System Hardware (System Debugger) (1 在系統硬體 (系統偵錯器) 上啟動)。
- 當偵錯器在 main() 的中斷點停止時，請從功能表中選擇 Run (執行)，然後選擇 Resume (繼續)。

Note

第一次執行應用程式時，新的憑證金鑰對會匯入非揮發性記憶體。對於後續執行，您可以註銷 main.c 檔案中的 vDevModeKeyProvisioning()，然後再重建映像和 BOOT.bin 檔案。這可防止每次執行時將憑證和金鑰複製到儲存體。

您可以選擇從 MicroZed 卡或 QSPI 刷新開機 MicroSD 主機板，以執行 FreeRTOS 示範專案。如需說明，請參閱 [產生 FreeRTOS 示範專案的開機映像 \(p. 116\)](#) 與 [執行 FreeRTOS 示範專案 \(p. 117\)](#)。

執行 FreeRTOS 示範專案

若要執行 FreeRTOS 示範專案，您可以從 MicroZed 卡或 QSPI 刷新來開機您的 MicroSD 主機板。

當您設定 MicroZed 主機板以執行 FreeRTOS 示範專案時，請參考 [設定 MicroZed 硬體 \(p. 114\)](#) 中的圖表。請確定您已將 MicroZed 主機板連接至電腦。

從 FreeRTOS 卡片開機 MicroSD 專案

格式化 Xilinx MicroSD Industrial MicroZed 套件隨附的 IoT 卡。

1. 將 BOOT.bin 檔案複製到 MicroSD 卡。
2. 直接將卡片插入 USB UART 連接埠下方的 MicroSD 卡槽。
3. 將 MicroZed 開機模式跳接器設定為 SD 開機模式。

SD Card



4. 按 RST 按鈕以重設裝置，並開始啟動應用程式。您也可以從 USB-UART 連接埠拔下 USB UART 纜線，然後重新插入纜線。

從 QSPI 快閃記憶體啟動 FreeRTOS 示範專案

1. 將您的 MicroZed 主機板的開機模式跳接器設定為 JTAG 開機模式。



2. 驗證您的電腦是否連接到 USB-UART 和 JTAG Access 連接埠。綠色電源正常 LED 燈應該亮起。
3. 在 XSDK IDE 中，從功能表中選擇 Xilinx，然後選擇 Program Flash (程式快閃)。
4. 在 Program Flash Memory (程式快閃記憶體) 中，應該自動填入硬體平台。對於 Connection (連接)，選擇您的 MicroZed 硬體伺服器，將您的主機板連接到您的主機電腦。

Note

如果您使用的是 Xilinx Smart Lync JTAG 纜線，則必須在 XSDK IDE 中建立一部硬體伺服器。
選擇 New (新增)，然後定義您的伺服器。

5. 在 Image File (映像檔) 中，輸入 BOOT.bin 映像檔的目錄路徑。改為選擇 Browse (瀏覽) 以瀏覽檔案。
6. 在 Offset (位移) 中，輸入 **0x0**。
7. 在 FSBL File (FSBL 檔案) 中，輸入 fsbl.elf 檔案的目錄路徑。改為選擇 Browse (瀏覽) 以瀏覽檔案。
8. 選擇 Program (程式)，對主機板進行程式設計。
9. 在 QSPI 程式設計完成之後，請拔下 USB-UART 纜線以關閉主機板的電源。
10. 將您的 MicroZed 主機板的開機模式跳接器設定為 QSPI 開機模式。
11. 將您的卡直接插入至位於 USB UART 連接埠下方的 MicroSD 卡槽。

Note

請務必備份您在 MicroSD 卡片上的任何檔案。

12. 按 RST 按鈕以重設裝置，並開始啟動應用程式。您也可以從 USB-UART 連接埠拔下 USB UART 纜線，然後重新插入纜線。

監控雲端的 MQTT 訊息

您可以在 AWS IoT 主控台中使用 MQTT 用戶端來監控裝置傳送至 AWS 雲端的訊息。

使用 AWS IoT MQTT 用戶端訂閱 MQTT 主題

1. 登入 [AWS IoT 主控台](#)。
2. 在導覽窗格中，選擇 Test (測試) 以開啟 MQTT 用戶端。
3. 在訂閱主題中輸入 `iotdemo/#`，然後選擇訂閱主題。

Troubleshooting

如果您遇到與路徑不正確相關的建立錯誤，請嘗試清除並重建專案，如[建置 FreeRTOS 示範專案 \(p. 116\)](#)中所述。

如果您使用的是 Windows，請確定您在 Windows XSDK IDE 中設定字串替換變數時使用的是正斜線。

如需 FreeRTOS 入門的一般疑難排解資訊，請參閱 [故障診斷入門 \(p. 19\)](#)。

FreeRTOS 無線更新

無線 (OTA) 更新可讓您將韌體更新部署到您叢集中的一或多個裝置。雖然 OTA 更新的設計旨在更新裝置韌體，您可以使用它們來將任何檔案傳送到向 AWS IoT 註冊的一或多個裝置。當您無線傳送更新時，我們建議您以數位方式簽署它們，以便接收檔案的裝置可以確認它們在途中未遭到竄改。

您可以使用[適用於 AWS IoT 的程式碼簽署](#)來簽署及加密您的檔案，或是您可以使用您自己的程式碼簽署工具來簽署檔案。

當您建立 OTA 更新時，[OTA Update Manager 服務 \(p. 162\)](#) 會建立一個 [AWS IoT 任務](#)來通知您的裝置有可用的更新。OTA 示範應用程式會在您的裝置上執行，並建立一個 FreeRTOS 任務，訂閱 AWS IoT 任務的通知主題並接聽更新訊息。有可用的更新時，OTA 代理程式會使用 HTTP 或 MQTT 通訊協定，將請求發佈至 AWS IoT 並接收更新，取決於您選擇的設定。OTA 代理程式 會檢查下載檔案的數位簽章，並且若檔案有效，就會安裝韌體更新。如果您並非使用 FreeRTOS OTA 更新示範應用程式，則必須將[OTA 代理程式程式庫 \(p. 196\)](#)整合到您自己的應用程式，以取得韌體更新功能。

FreeRTOS 無線更新可讓您：

- 在部署前，先以數位方式簽署韌體。
- 將新的韌體映像部署到單一裝置、裝置群組，或是您的整個機群。
- 在韌體新增到群組、重設或重新佈建時將韌體部署到裝置。
- 在韌體部署到裝置後驗證其真確性及完整性。
- 監控部署進度。
- 對失敗的部署進行除錯。

標記 OTA 資源

為協助您管理您的 OTA 資源，您可以選擇性的將您自己的中繼資料，以標籤的形式指派給更新與串流。標籤可讓您以不同的方式 (例如依據目的、擁有者或環境) 將 AWS IoT 資源分類。這在您擁有許多相同類型的資源時很有用。您可以根據指派給資源的標籤來快速識別資源。

如需詳細資訊，請參閱[標記您的 AWS IoT 資源](#)。

OTA 更新先決條件

若要使用無線 (OTA) 更新，請執行下列動作：

- 檢查[使用 HTTP 進行 OTA 更新的先決條件 \(p. 132\)](#)或[使用 MQTT 進行 OTA 更新的先決條件 \(p. 130\)](#)。
- 建立 Amazon S3 儲存貯體來存放您的更新 (p. 120).
- 建立 OTA 更新服務角色 (p. 121).
- 建立 OTA 使用者政策 (p. 122).
- 建立程式碼簽署憑證 (p. 124).
- 如果您使用適用於 AWS IoT 的程式碼簽章，[將存取權限授予適用於 AWS IoT 的程式碼簽署 \(p. 130\)](#)。
- 使用 OTA 程式庫下載 FreeRTOS (p. 130).

建立 Amazon S3 儲存貯體來存放您的更新

OTA 更新檔案會存放在 Amazon S3 儲存貯體中。

如果您是使用適用於 AWS IoT 的程式碼簽署，您用來建立程式碼簽署任務的命令會接收一個來源儲存貯體 (未簽署韌體映像所在的位置) 以及一個目的地儲存貯體 (簽署韌體映像的寫入位置)。您可以為來源及目標指定相同的儲存貯體。檔案名稱會變更為 GUIDs，因此原始檔案不會遭到覆寫。

建立 Amazon S3 儲存貯體

1. 前往 <https://console.aws.amazon.com/s3/>，並登入 Amazon S3 主控台。
2. 選擇 Create bucket (建立儲存貯體)。
3. 輸入儲存貯體名稱。
4. 在 Bucket settings for Block Public Access (封鎖公開存取的儲存貯體設定) 下，保持選取 Block all public access (封鎖所有公開存取) 以接受預設許可。
5. 在 Bucket Versioning (儲存貯體版本控制) 下，選取 Enable (允許)，以將所有版本保留在相同儲存貯體中。
6. 選擇 Create bucket (建立儲存貯體)。

如需 Amazon S3 的詳細資訊，請參閱 [Amazon Simple Storage Service 主控台使用者指南](#)。

建立 OTA 更新服務角色

OTA 更新服務會取得此角色，代您建立及管理 OTA 更新任務。

建立 OTA 服務角色

1. 登入 <https://console.aws.amazon.com/iam/>。
2. 從導覽窗格中，選擇 Roles (角色)。
3. 選擇 Create Role (建立角色)。
4. 在 Select type of trusted entity (選擇信任的實體類型) 下，選擇 AWS Service (AWS 服務)。
5. 從 AWS 服務清單中選擇 IoT。
6. 在 Select your use case (選取您的使用案例) 下，選擇 IoT。
7. 選擇 Next : (下一步：)。標籤。
8. 選擇 Next : (下一步：)。審核。
9. 輸入角色名稱和說明，然後選擇 Create role (建立角色)。

如需 IAM 角色的詳細資訊，請參閱 [IAM 角色](#)。

將 OTA 更新許可新增到您的 OTA 服務角色

1. 在 IAM 主控台頁面上的搜尋方塊中，輸入您的角色名稱，然後從清單中選擇它。
2. 選擇 Attach policies (連接政策)。
3. 在 Search (搜尋) 方塊中，輸入「AmazonFreeRTOSOTAUpdate」，從篩選政策清單中選取 AmazonFreeRTOSOTAUpdate，然後選擇 Attach policy (連接政策) 將該政策連接至您的服務角色。

將必要 IAM 許可新增到您的 OTA 服務角色

1. 在 IAM 主控台頁面上的搜尋方塊中，輸入您的角色名稱，然後從清單中選擇它。
2. 選擇 Add inline policy (新增內嵌政策)。
3. 請選擇 JSON 索引標籤。
4. 將下列政策文件複製並貼入文字方塊：

```
{  
    "Version": "2012-10-17",  
}
```

```
"Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "iam:GetRole",
            "iam:PassRole"
        ],
        "Resource": "arn:aws:iam::your_account_id:role/your_role_name"
    }
]
```

請務必取代 *your_account_id* 取代為您的 AWS 帳號 ID，以及 *your_role_name* 取代為 OTA 服務角色的名稱。

5. 選擇 Review policy (檢閱政策)。
6. 輸入政策名稱，然後選擇 Create policy (建立政策)。

Note

如果您的 Amazon S3 儲存貯體名稱以 "afr-ota" 開頭，則不需要下列程序。若是如此，則 AWS 管理政策 AmazonFreeRTOSOTAUpdate 已包含必要許可。

將必要 Amazon S3 許可新增到您的 OTA 服務角色

1. 在 IAM 主控台頁面上的搜尋方塊中，輸入您的角色名稱，然後從清單中選擇它。
2. 選擇 Add inline policy (新增內嵌政策)。
3. 請選擇 JSON 索引標籤。
4. 將下列政策文件複製並貼入方塊。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3.GetObjectVersion",
                "s3.GetObject",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::example-bucket/*"
            ]
        }
    ]
}
```

此政策會將讀取 Amazon S3 物件的許可授予 OTA 服務角色。請務必取代 *example-bucket* 取代為儲存貯體的名稱。

5. 選擇 Review policy (檢閱政策)。
6. 輸入政策名稱，然後選擇 Create policy (建立政策)。

建立 OTA 使用者政策

您必須授予 IAM 使用者許可來執行無線更新。您的 IAM 使用者必須擁有執行以下作業的許可：

- 存取存放您韌體更新的 S3 儲存貯體。

- 存取存放在 AWS Certificate Manager 中的憑證。
- 存取 AWS IoT 串流服務。
- 存取 FreeRTOS OTA 更新。
- 存取 AWS IoT 任務。
- 存取 IAM。
- 存取適用於 AWS IoT 的程式碼簽署。請參閱[將存取權限授予適用於 AWS IoT 的程式碼簽署 \(p. 130\)](#)。
- 列出 FreeRTOS 硬體平台。

若要將必要許可授予您的 IAM 使用者，請建立 OTA 使用者政策，然後將它連接到您的 IAM 使用者。如需詳細資訊，請參閱[IAM 政策](#)。

建立 OTA 使用者政策

1. 開啟 <https://console.aws.amazon.com/iam/> 主控台。
2. 在導覽窗格中，選擇 Users (使用者)。
3. 從清單中選擇您的 IAM 使用者。
4. 選擇新增許可。
5. 選擇 Attach existing policies directly (直接連接現有政策)。
6. 選擇 Create policy (建立政策)。
7. 選擇 JSON 標籤，將以下政策文件複製並貼入政策編輯器：

```
{  
    "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"  
        }  
    ]  
}
```

Replace `example-bucket` 取代為您 OTA 更新韌體映像存放的 Amazon S3 儲存貯體名稱。Replace `your-account-id` 取代為您的 AWS 帳號 ID。您可以在主控台的右上角找到您的 AWS 帳戶 ID。當您輸入帳戶 ID 時，請移除任何破折號 (-)。Replace `role-name` 使用您剛建立的 IAM 服務角色的名稱。

8. 選擇 Review policy (檢閱政策)。
9. 輸入您新的 OTA 使用者政策名稱，然後選擇 Create policy (建立政策)。

將 OTA 使用者政策連接到您的 IAM 使用者

1. 在 IAM 主控台的導覽窗格中，選擇 Users (使用者)，然後選擇您的使用者。
2. 選擇新增許可。
3. 選擇 Attach existing policies directly (直接連接現有政策)。
4. 搜尋您剛建立的 OTA 使用者政策，然後選取一旁的核取方塊。
5. 選擇 Next : (下一步：)。審查。
6. 選擇新增許可。

建立程式碼簽署憑證

若要數位簽署韌體映像，您需要程式碼簽署憑證及私有金鑰。針對測試用途，您可以建立自我-簽署憑證和私有金鑰。針對實際執行環境，請透過知名的憑證授權機構 (CA) 購買憑證。-

不同平台需要不同類型的程式碼簽署憑證。以下章節說明如何為符合 FreeRTOS 資格的各種平台建立程式碼簽署憑證。

主題

- 為 Texas Instruments CC3220SF-LAUNCHXL 建立程式碼簽署憑證 (p. 124)
- 為 Microchip Curiosity PIC32MZEF 建立程式碼簽署憑證 (p. 126)
- 為 Espressif ESP32 建立程式碼簽署憑證 (p. 127)
- 為 Nordic nrf52840-dk 建立程式碼簽署憑證 (p. 128)
- 為 FreeRTOS Windows 模擬器建立程式碼簽署憑證 (p. 128)
- 為自訂硬體建立程式碼簽署憑證 (p. 129)

為 Texas Instruments CC3220SF-LAUNCHXL 建立程式碼簽署憑證

Wi-Fi CC3220SF Wireless Microcontroller Launchpad Development Kit 支援兩個韌體程式碼簽署的憑證鏈：SimpleLink

- 生產 (certificate-catalog)

若要使用生產憑證鏈，您必須購買商業程式碼簽署憑證，並使用 [TI Uniflash 工具](#) 來將電路板設為生產模式。

- 測試及開發 (certificate-playground)

Playground 憑證鏈可讓您使用自我-簽署的程式碼簽署憑證來試用 OTA 更新。

使用 AWS Command Line Interface 來將程式碼簽署憑證、私密金鑰及憑證鏈匯入至 AWS Certificate Manager。如需詳細資訊，請前往 [AWS CLI 安裝](#) 中的 AWS Command Line Interface 使用者指南。

下載並安裝最新版本的 [CC3220 軟體開發套件 SimpleLink](#)。根據預設，您需要檔案所在的位置如下：

C:\ti\simplelink_cc32xx_sdk_version\tools\cc32xx_tools\certificate-playground
(Windows)

/Applications/Ti/simplelink_cc32xx_sdk_version/tools/cc32xx_tools/certificate-
playground (macOS)

CC3220 開發套件中的憑證格式為 DER 格式。SimpleLink 若要建立自我-簽署的程式碼簽署憑證，您必須將它們轉換成 PEM 格式。

遵循這些步驟來建立連結到 Texas Instruments 遊樂場憑證階層的程式碼簽署憑證，並符合 AWS Certificate Manager 及適用於 AWS IoT 程式碼簽署的條件。

Note

若要建立程式碼簽署憑證，請在您的機器上安裝 [OpenSSL](#)。安裝 OpenSSL 之後，請確定將 openssl 指派給命令提示字元或終端機環境中的 OpenSSL 可執行檔。

建立自我-簽署程式碼簽署憑證

1. 使用管理員許可來開啟命令提示字元或終端機。
2. 在工作資料夾中，使用下列文字來建立名為 cert_config.txt 的檔案。取代 test_signer@amazon.com 取代為您的電子郵件地址。

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

3. 建立私有金鑰及憑證簽署請求 (CSR) :

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

4. 將 Texas Instruments 遊樂場根 CA 私有金鑰從 DER 格式轉換成 PEM 格式。

TI 遊樂場根 CA 私有金鑰的所在位置如下：

C:\ti\simplelink_cc32xx_sdk_version\tools\cc32xx_tools\certificate-
playground\dummy-root-ca-cert-key (Windows)

/Applications/Ti/simplelink_cc32xx_sdk_version/tools/cc32xx_tools/
certificate-playground/dummy-root-ca-cert-key (macOS)

```
openssl rsa -inform DER -in dummy-root-ca-cert-key -out dummy-root-ca-cert-key.pem
```

5. 將 Texas Instruments 遊樂場根 CA 憑證從 DER 格式轉換成 PEM 格式。

TI 遊樂場根憑證的所在位置如下：

C:\ti\simplelink_cc32xx_sdk_version\tools\cc32xx_tools\certificate-
playground\dummy-root-ca-cert (Windows)

/Applications/Ti/simplelink_cc32xx_sdk_version/tools/cc32xx_tools/
certificate-playground/dummy-root-ca-cert (macOS)

```
openssl x509 -inform DER -in dummy-root-ca-cert -out dummy-root-ca-cert.pem
```

6. 使用 Texas Instruments 根 CA 簽署 CSR :

```
openssl x509 -extfile cert_config.txt -extensions my_exts -req -days 365 -in tisigner.csr -CA dummy-root-ca-cert.pem -CAkey dummy-root-ca-cert-key.pem -set_serial 01 -out tisigner.crt.pem -sha1
```

7. 將您的程式碼簽署憑證 (tisigner.crt.pem) 轉換成 DER 格式 :

```
openssl x509 -in tisigner.crt.pem -out tisigner.crt.der -outform DER
```

Note

您會在稍後將 tisigner.crt.der 憑證寫入 TI 開發電路板。

8. 將程式碼簽署憑證、私有金鑰及憑證鏈匯入 AWS Certificate Manager :

```
aws acm import-certificate --certificate fileb://tisigner.crt.pem --private-key fileb://tisigner.key --certificate-chain fileb://dummy-root-ca-cert.pem
```

此命令會顯示您憑證的 ARN。在建立 OTA 更新任務時，您將需要此 ARN。

Note

在此撰寫的步驟假設您要使用適用於 AWS IoT 的程式碼簽署來簽署您的韌體映像。雖然建議使用適用於 AWS IoT 的程式碼簽署，但您也可以手動簽署您的韌體映像。

為 Microchip Curiosity PIC32MZEF 建立程式碼簽署憑證

Microchip Curiosity PIC32MZEF 支援使用 ECDSA 程式碼簽署憑證的自我簽署 SHA256。

Note

若要建立程式碼簽署憑證，請在您的機器上安裝 OpenSSL。安裝 OpenSSL 之後，請確定將 openssl 指派給命令提示字元或終端機環境中的 OpenSSL 可執行檔。

使用 AWS Command Line Interface 來將程式碼簽署憑證、私密金鑰及憑證鏈匯入至 AWS Certificate Manager。如需安裝 AWS CLI 的相關資訊，請參閱[安裝 AWS CLI](#)。

1. 在工作資料夾中，使用下列文字來建立名為 cert_config.txt 的檔案。取代 `test_signer@amazon.com` 包含您的電子郵件地址：

```
[ req ]
prompt          = no
distinguished_name = my_dn

[ my_dn ]
commonName = test_signer@amazon.com

[ my_exts ]
keyUsage      = digitalSignature
extendedKeyUsage = codeSigning
```

2. 建立 ECDSA 程式碼簽署私有金鑰：

```
openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -pkeyopt
ec_param_enc:named_curve -outform PEM -out ecdsasigner.key
```

3. 建立 ECDSA 程式碼簽署憑證：

```
openssl req -new -x509 -config cert_config.txt -extensions my_exts -nodes -days 365 -key ecdsasigner.key -out ecdsasigner.crt
```

4. 將程式碼簽署憑證、私有金鑰及憑證鏈匯入 AWS Certificate Manager：

```
aws acm import-certificate --certificate fileb://ecdsasigner.crt --private-key fileb://ecdsasigner.key
```

此命令會顯示您憑證的 ARN。在建立 OTA 更新任務時，您將需要此 ARN。

Note

在此撰寫的步驟假設您要使用適用於 AWS IoT 的程式碼簽署來簽署您的韌體映像。雖然建議使用適用於 AWS IoT 的程式碼簽署，但您也可以手動簽署您的韌體映像。

為 Espressif ESP32 建立程式碼簽署憑證

Espressif ESP32 主機板支援使用 ECDSA 程式碼簽署憑證的自我簽署 SHA256。

Note

若要建立程式碼簽署憑證，請在您的機器上安裝 [OpenSSL](#)。安裝 OpenSSL 之後，請確定將 openssl 指派給命令提示字元或終端機環境中的 OpenSSL 可執行檔。

使用 AWS Command Line Interface 來將程式碼簽署憑證、私密金鑰及憑證鏈匯入至 AWS Certificate Manager。如需安裝 AWS CLI 的相關資訊，請參閱[安裝 AWS CLI](#)。

1. 在工作資料夾中，使用下列文字來建立名為 cert_config.txt 的檔案。取代 [test_signer@amazon.com](#) 包含您的電子郵件地址：

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

2. 建立 ECDSA 程式碼簽署私有金鑰：

```
openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -pkeyopt ec_param_enc:named_curve -outform PEM -out ecdsasigner.key
```

3. 建立 ECDSA 程式碼簽署憑證：

```
openssl req -new -x509 -config cert_config.txt -extensions my_exts -nodes -days 365 -key ecdsasigner.key -out ecdsasigner.crt
```

4. 將程式碼簽署憑證、私有金鑰及憑證鏈匯入 AWS Certificate Manager：

```
aws acm import-certificate --certificate fileb://ecdsasigner.crt --private-key fileb://ecdsasigner.key
```

此命令會顯示您憑證的 ARN。在建立 OTA 更新任務時，您將需要此 ARN。

Note

在此撰寫的步驟假設您要使用適用於 AWS IoT 的程式碼簽署來簽署您的韌體映像。雖然建議使用適用於 AWS IoT 的程式碼簽署，但您也可以手動簽署您的韌體映像。

為 Nordic nrf52840-dk 建立程式碼簽署憑證

Nordic nrf52840-dk 支援透過 ECDSA 程式碼簽署憑證進行的自我簽署 SHA256。

Note

若要建立程式碼簽署憑證，請在您的機器上安裝 OpenSSL。安裝 OpenSSL 之後，請確定將 openssl 指派給命令提示字元或終端機環境中的 OpenSSL 可執行檔。

使用 AWS Command Line Interface 來將程式碼簽署憑證、私密金鑰及憑證鏈匯入至 AWS Certificate Manager。如需安裝 AWS CLI 的相關資訊，請參閱[安裝 AWS CLI](#)。

1. 在工作資料夾中，使用下列文字來建立名為 cert_config.txt 的檔案。取代 **test_signer@amazon.com** 包含您的電子郵件地址：

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

2. 建立 ECDSA 程式碼簽署私有金鑰：

```
openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -pkeyopt  
ec_param_enc:named_curve -outform PEM -out ecdsasigner.key
```

3. 建立 ECDSA 程式碼簽署憑證：

```
openssl req -new -x509 -config cert_config.txt -extensions my_exts -nodes -days 365 -  
key ecdsasigner.key -out ecdsasigner.crt
```

4. 將程式碼簽署憑證、私有金鑰及憑證鏈匯入 AWS Certificate Manager：

```
aws acm import-certificate --certificate fileb://ecdsasigner.crt --private-key fileb://  
ecdsasigner.key
```

此命令會顯示您憑證的 ARN。在建立 OTA 更新任務時，您將需要此 ARN。

Note

在此撰寫的步驟假設您要使用適用於 AWS IoT 的程式碼簽署來簽署您的韌體映像。雖然建議使用適用於 AWS IoT 的程式碼簽署，但您也可以手動簽署您的韌體映像。

為 FreeRTOS Windows 模擬器建立程式碼簽署憑證

FreeRTOS Windows 模擬器需要使用 ECDSA P-256 金鑰的程式碼簽署憑證及 SHA-256 雜湊來執行 OTA 更新。若您沒有程式碼簽署憑證，請依照以下步驟來建立。

Note

若要建立程式碼簽署憑證，請在您的機器上安裝 [OpenSSL](#)。安裝 OpenSSL 之後，請確定將 openssl 指派給命令提示字元或終端機環境中的 OpenSSL 可執行檔。
使用 AWS Command Line Interface 來將程式碼簽署憑證、私密金鑰及憑證鏈匯入至 AWS Certificate Manager。如需安裝 AWS CLI 的相關資訊，請參閱[安裝 AWS CLI](#)。

1. 在工作資料夾中，使用下列文字來建立名為 cert_config.txt 的檔案。取代 [test_signer@amazon.com](#) 包含您的電子郵件地址：

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

2. 建立 ECDSA 程式碼簽署私有金鑰：

```
openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -pkeyopt  
ec_param_enc:named_curve -outform PEM -out ecdsasigner.key
```

3. 建立 ECDSA 程式碼簽署憑證：

```
openssl req -new -x509 -config cert_config.txt -extensions my_exts -nodes -days 365 -  
key ecdsasigner.key -out ecdsasigner.crt
```

4. 將程式碼簽署憑證、私有金鑰及憑證鏈匯入 AWS Certificate Manager：

```
aws acm import-certificate --certificate fileb://ecdsasigner.crt --private-key fileb://  
ecdsasigner.key
```

此命令會顯示您憑證的 ARN。在建立 OTA 更新任務時，您將需要此 ARN。

Note

在此撰寫的步驟假設您要使用適用於 AWS IoT 的程式碼簽署來簽署您的韌體映像。雖然建議使用適用於 AWS IoT 的程式碼簽署，但您也可以手動簽署您的韌體映像。

為自訂硬體建立程式碼簽署憑證

使用適當的工具組，為您的硬體建立自我簽署憑證及私有金鑰。

使用 AWS Command Line Interface 來將程式碼簽署憑證、私密金鑰及憑證鏈匯入至 AWS Certificate Manager。如需安裝 AWS CLI 的相關資訊，請參閱[安裝 AWS CLI](#)。

在您建立程式碼簽署憑證後，您可以使用 AWS CLI 將它匯入 ACM：

```
aws acm import-certificate --certificate fileb://code-sign.crt --private-key fileb://code-  
sign.key
```

此命令的輸出會顯示您憑證的 ARN。在建立 OTA 更新任務時，您將需要此 ARN。

ACM 需要憑證才能使用特定演算法及金鑰大小。如需詳細資訊，請參閱[匯入憑證的事前準備](#)。如需 ACM 詳細資訊，請參閱[將憑證匯入 AWS Certificate Manager](#)。

您必須複製您程式碼簽署憑證的內容，在格式化後貼入您稍後下載 FreeRTOS 程式碼一部分的 aws_ota_codesigner_certificate.h 檔案。

將存取權限授予適用於 AWS IoT 的程式碼簽署

在生產環境中，建議您數位簽署您的韌體更新，確保更新的真確性及完整性。您可以手動簽署您的更新，或是使用適用於 AWS IoT 的程式碼簽署來簽署您的程式碼。若要使用適用於 FreeRTOS 的程式碼簽署，您必須將您 IAM 使用者帳戶的存取權限授予適用於 FreeRTOS 的程式碼簽署。

將您的 IAM 使用者帳戶許可授予適用於 AWS IoT 的程式碼簽署

1. 登入 <https://console.aws.amazon.com/iam/>。
2. 在導覽窗格上選擇 Policies (政策)。
3. 選擇 Create Policy (建立政策)。
4. 在 JSON 標籤上，將以下 JSON 文件複製並貼入政策編輯器。此政策會允許 IAM 使用者存取所有程式碼簽署操作。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "signer:*"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

5. 選擇 Review policy (檢閱政策)。
6. 輸入政策名稱及描述，然後選擇 Create policy (建立政策)。
7. 在導覽窗格中，選擇 Users (使用者)。
8. 選擇您的 IAM 使用者帳戶。
9. 在 Permissions (許可) 標籤上，選擇 Add permissions (新增許可)。
10. 選擇 Attach existing policies directly (直接連接現有政策)，然後選取您剛建立程式碼簽署政策旁邊的核取方塊。
11. 選擇 Next : (下一步：)。審查。
12. 選擇新增許可。

使用 OTA 程式庫下載 FreeRTOS

您可以從 FreeRTOS 下載 FreeRTOS 主控台 (p. 9)，或可從 FreeRTOS 複製或下載 GitHub。請前往 README.md 檔案以獲得指示。

若要將 OTA 程式庫納入您從主控台下載的 FreeRTOS 組態，您可自訂預先定義的組態，或針對支援 OTA 功能的平台建立組態。在 Configure FreeRTOS Software (設定 RTOS 軟體) 組態屬性頁面的 Libraries (程式庫) 中，選擇 OTA Updates (OTA 更新)。您可在 Demo projects (示範專案) 中選擇啟用 OTA 示範。您也可以在稍後手動啟用示範。

如需設定和執行 OTA 示範應用程式的相關資訊，請參閱無線更新示範應用程式 (p. 248)。

使用 MQTT 進行 OTA 更新的先決條件

本節描述使用 MQTT 執行無線 (OTA) 更新的一般需求。

最低需求

- 裝置韌體必須包含必要的 FreeRTOS 程式庫 (coreMQTT、OTA 代理程式及其相依性)。
- 必須使用 FreeRTOS 1.4.0 版或更新版本。不過，我們建議您儘可能使用最新版本。

Configurations

從版本 201912.00 開始，FreeRTOS OTA 可以使用 HTTP 或 MQTT 通訊協定，將韌體更新映像從 AWS IoT 傳輸至裝置。如果您在 FreeRTOS 中建立 OTA 更新時同時指定這兩個通訊協定，每部裝置都會各自判斷要用來傳輸影像的通訊協定。如需詳細資訊，請參閱[使用 HTTP 進行 OTA 更新的先決條件 \(p. 132\)](#)。

根據預設，aws_ota_agent_config.h 中的 OTA 通訊協定組態會使用 MQTT 通訊協定：

```
/**  
 * @brief The protocol selected for OTA control operations.  
 * This configuration parameter sets the default protocol for all the OTA control  
 * operations like requesting OTA job, updating the job status etc.  
 *  
 * Note - Only MQTT is supported at this time for control operations.  
 */  
#define configENABLED_CONTROL_PROTOCOL      ( OTA_CONTROL_OVER_MQTT )  
/**  
 * @brief The protocol selected for OTA data operations.  
 * This configuration parameter sets the protocols selected for the data operations  
 * like requesting file blocks from the service.  
 *  
 * Note - Both MQTT and HTTP are supported for data transfer. This configuration parameter  
 * can be set to the following -  
 * Enable data over MQTT - ( OTA_DATA_OVER_MQTT )  
 * Enable data over HTTP - ( OTA_DATA_OVER_HTTP )  
 * Enable data over both MQTT & HTTP ( OTA_DATA_OVER_MQTT | OTA_DATA_OVER_HTTP )  
 */  
#define configENABLED_DATA_PROTOCOLS       ( OTA_DATA_OVER_MQTT )  
/**  
 * @brief The preferred protocol selected for OTA data operations.  
 *  
 * Primary data protocol will be the protocol used for downloading files if more than  
 * one protocol is selected while creating OTA job. Default primary data protocol is MQTT  
 * and the following update here switches to HTTP as primary.  
 *  
 * Note - use OTA_DATA_OVER_HTTP for HTTP as primary data protocol.  
 */  
#define configOTA_PRIMARY_DATA_PROTOCOL    ( OTA_DATA_OVER_MQTT )
```

裝置特定的組態

無。

記憶體用量

當 MQTT 用於資料傳輸時，MQTT 連線不需要額外的記憶體，因為它是在控制操作與資料操作之間共用。

裝置政策

每部使用 MQTT 接收 OTA 更新的裝置都必須註冊為 AWS IoT 中的物件，且該物件必須有連接的政策，如此處列出的政策所示。如需 "Action" 和 "Resource" 物件中的項目的詳細資訊，請參閱[AWS IoT 核心政策動作](#)和[AWS IoT 核心動作資源](#)。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "iot:Connect",  
            "Resource": "arn:partition:iot:region:account:client/  
${iot:Connection.Thing.ThingName}"  
        },  
        {  
            "Effect": "Allow",  
            "Action": "iot:Subscribe",  
            "Resource": [  
                "arn:partition:iot:region:account:topicfilter/$aws/things/  
${iot:Connection.Thing.ThingName}/streams/*",  
                "arn:partition:iot:region:account:topicfilter/$aws/things/  
${iot:Connection.Thing.ThingName}/jobs/*"  
            ]  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "iot:Publish",  
                "iot:Receive"  
            ],  
            "Resource": [  
                "arn:partition:iot:region:account:topic/$aws/things/  
${iot:Connection.Thing.ThingName}/streams/*",  
                "arn:partition:iot:region:account:topic/$aws/things/  
${iot:Connection.Thing.ThingName}/jobs/*"  
            ]  
        }  
    ]  
}
```

Notes

- `iot:Connect` 許可允許您的裝置透過 MQTT 連接至 AWS IoT。
- AWS IoT 工作 (`.../jobs/*`) 主題的 `iot:Subscribe` 和 `iot:Publish` 許可允許連接的裝置接收工作通知和工作文件，並發佈工作執行的完成狀態。
- AWS IoT OTA 串流 (`.../streams/*`) 主題的 `iot:Subscribe` 和 `iot:Publish` 許可允許連接的裝置從 AWS IoT 中擷取 OTA 更新資料。需要這些許可才能透過 MQTT 執行韌體更新。
- `iot:Receive` 許可允許 AWS IoT Core 將這些主題上的訊息發佈到連接的裝置。每次交付 MQTT 訊息時，都會檢查此許可。您可以使用此許可來撤銷目前訂閱主題之用戶端的存取權。

使用 HTTP 進行 OTA 更新的先決條件

本節描述使用 HTTP 執行無線 (OTA) 更新的一般需求。從版本 201912.00 開始，FreeRTOS OTA 可以使用 HTTP 或 MQTT 通訊協定，將韌體更新映像從 AWS IoT 傳輸至裝置。

Note

- 雖然 HTTP 通訊協定可以用來傳輸韌體映像，但仍然需要 coreMQTT 程式庫，因為與 AWS IoT Core 的其他互動會使用 coreMQTT 程式庫，包括傳送或接收任務執行通知、任務文件，以及執行狀態更新。
- 當您為 OTA 更新任務同時指定 MQTT 和 HTTP 通訊協定時，每個個別裝置上 OTA 代理程式軟體的設定都會決定用來傳輸韌體映像的通訊協定。若要將 OTA 代理程式從預設 MQTT 通訊協定方法變更為 HTTP 通訊協定，您可以修改用來編譯裝置的 FreeRTOS 原始碼的標頭檔案。

最低需求

- 裝置韌體必須包含必要的 FreeRTOS 程式庫 (coreMQTT、HTTP、OTA 代理程式及其相依性)。
- 必須使用 FreeRTOS 201912.00 版或更新版本，才能變更 OTA 通訊協定的組態，來啟用透過 HTTP 傳輸 OTA 資料的功能。

Configurations

請參閱 \vendors\boards\board\aws_demos\config_files\aws_ota_agent_config.h 檔案中 OTA 通訊協定的下列組態。

```
/**  
 * @brief The protocol selected for OTA control operations.  
 * This configuration parameter sets the default protocol for all the OTA control  
 * operations like requesting an OTA job, updating the job status, and so on.  
 *  
 * Note - Only MQTT is supported at this time for control operations.  
 */  
#define configENABLED_CONTROL_PROTOCOL      ( OTA_CONTROL_OVER_MQTT )  
/**  
 * @brief The protocol selected for OTA data operations.  
 * This configuration parameter sets the protocols selected for the data operations  
 * like requesting file blocks from the service.  
 *  
 * Note - Both MQTT and HTTP are supported for data transfer. This configuration parameter  
 * can be set to the following -  
 * Enable data over MQTT - ( OTA_DATA_OVER_MQTT )  
 * Enable data over HTTP - ( OTA_DATA_OVER_HTTP )  
 * Enable data over both MQTT & HTTP ( OTA_DATA_OVER_MQTT | OTA_DATA_OVER_HTTP )  
 */  
#define configENABLED_DATA_PROTOCOLS       ( OTA_DATA_OVER_MQTT )  
/**  
 * @brief The preferred protocol selected for OTA data operations.  
 *  
 * Primary data protocol will be the protocol used for downloading files if more than  
 * one protocol is selected while creating OTA job. Default primary data protocol is MQTT  
 * and the following update here switches to HTTP as primary.  
 *  
 * Note - use OTA_DATA_OVER_HTTP for HTTP as primary data protocol.  
 */  
#define configOTA_PRIMARY_DATA_PROTOCOL   ( OTA_DATA_OVER_MQTT )
```

透過 HTTP 啟用 OTA 資料傳輸

- 將 configENABLED_DATA_PROTOCOLS 變更為 OTA_DATA_OVER_HTTP。
- 當 OTA 更新時，您可以指定這兩個通訊協定，以便可以使用 MQTT 或 HTTP 通訊協定。您可以將 configOTA_PRIMARY_DATA_PROTOCOL 變更 OTA_DATA_OVER_HTTP，以將裝置使用的主要通訊協定設定為 HTTP。

Note

OTA 資料操作僅支援 HTTP。若為控制操作，您必須使用 MQTT。

裝置特定的組態

ESP32

由於 RAM 數量有限，當您啟用 HTTP 做為 OTA 資料通訊協定時，必須關閉 BLE。在 `vendors/espressif/boards/esp32/aws_demos/config_files/aws_iot_network_config.h` 檔案中，僅將 `configENABLED_NETWORKS` 變更為 `AWSIOT_NETWORK_TYPE_WIFI`。

```
/**  
 * @brief Configuration flag which is used to enable one or more network interfaces  
for a board.  
 *  
 * The configuration can be changed any time to keep one or more network enabled or  
disabled.  
 * More than one network interfaces can be enabled by using 'OR' operation with  
flags for  
* each network types supported. Flags for all supported network types can be found  
* in "aws_iot_network.h"  
*  
*/  
#define configENABLED_NETWORKS      ( AWSIOT_NETWORK_TYPE_WIFI )
```

記憶體用量

當 MQTT 用於資料傳輸時，MQTT 連線不需要額外的堆積記憶體，因為它是在控制操作與資料操作之間共用。不過，透過 HTTP 啟用資料傳輸需要額外的堆積記憶體。以下是所有支援平台的堆積記憶體用量資料，這是使用 FreeRTOS `xPortGetFreeHeapSize` API 計算得來的。您必須確定有足夠的 RAM 才能使用 OTA 程式庫。

Texas Instruments CC3220SF-LAUNCHXL

控制操作（MQTT）：12 KB

資料操作（HTTP）：10 KB

Note

TI 會使用相當少的 RAM，因為它在硬體上使用 SSL，所以它不會使用 mbedTLS 程式庫。

Microchip Curiosity PIC32MZEF

控制操作（MQTT）：65 KB

資料操作（HTTP）：43 KB

Espressif ESP32

控制操作（MQTT）：65 KB

資料操作（HTTP）：45 KB

Note

ESP32 上的 BLE 大約需要 87 KB RAM。沒有足夠的 RAM 來啟用所有的程式，如上述裝置特定的組態中所述。

Windows 模擬器

控制操作（MQTT）：82 KB

資料操作 (HTTP) : 63 KB
Nordic nrf52840-dk

不支援 HTTP。

裝置政策

此政策可讓您使用 MQTT 或 HTTP 來進行 OTA 更新。

每部使用 HTTP 接收 OTA 更新的裝置都必須註冊為 AWS IoT 中的物件，且該物件必須有連接的政策，如此處列出的政策所示。如需 "Action" 和 "Resource" 物件中的項目的詳細資訊，請參閱 [AWS IoT 核心政策動作](#) 和 [AWS IoT 核心動作資源](#)。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "iot:Connect",  
            "Resource": "arn:partition:iot:region:account:client/  
${iot:Connection.Thing.ThingName}"  
        },  
        {  
            "Effect": "Allow",  
            "Action": "iot:Subscribe",  
            "Resource": [  
                "arn:partition:iot:region:account:topicfilter/$aws/things/  
${iot:Connection.Thing.ThingName}/jobs/*"  
            ]  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "iot:Publish",  
                "iot:Receive"  
            ],  
            "Resource": [  
                "arn:partition:iot:region:account:topic/$aws/things/  
${iot:Connection.Thing.ThingName}/jobs/*"  
            ]  
        }  
    ]  
}
```

Notes

- `iot:Connect` 許可允許您的裝置透過 MQTT 連接至 AWS IoT。
- AWS IoT 工作 (`.../jobs/*`) 主題的 `iot:Subscribe` 和 `iot:Publish` 許可允許連接的裝置接收工作通知和工作文件，並發佈工作執行的完成狀態。
- `iot:Receive` 許可允許 AWS IoT Core 將這些主題上的訊息發佈到目前連接的裝置。每次交付 MQTT 訊息時，都會檢查此許可。您可以使用此許可來撤銷目前訂閱主題之用戶端的存取權。

OTA 教學

本節包含使用 OTA 更新，在執行 FreeRTOS 的裝置上更新韌體的教學。除了韌體映像之外，您還可以使用 OTA 更新，將任何類型的檔案傳送至連接至 AWS IoT 的裝置。

您可以使用 AWS IoT 主控台或 AWS CLI 來建立 OTA 更新。主控台是開始使用 OTA 最簡單的方式，因為它會為您完成許多工作。AWS CLI 在您自動化 OTA 更新任務、使用大量裝置，或是使用尚未符合 FreeRTOS 資格的裝置時很有用。如需有關讓裝置符合 FreeRTOS 資格的詳細資訊，請參閱 [FreeRTOS 合作夥伴網站](#)。

建立 OTA 更新

1. 將您韌體的初始版本部署到一或多個裝置。
2. 確認韌體已正常運作。
3. 當需要韌體更新時，對程式碼進行變更並建置新映像。
4. 若您要手動簽署您的韌體，請簽署並將簽署後的韌體映像上傳到您的 Amazon S3 儲存貯體。若您使用適用於 AWS IoT 的程式碼簽署，請將您未簽署的韌體映像上傳至 Amazon S3 儲存貯體。
5. 建立 OTA 更新。

當您建立 OTA 更新時，請指定映像傳遞通訊協定 (MQTT 或 HTTP)，或指定兩者以允許裝置選擇。裝置上的 FreeRTOS OTA 代理程式會接收更新後的韌體映像並驗證數位簽章、檢查總和，以及新映像的版本編號。若韌體更新通過驗證，裝置便會重設，並根據應用程式定義的邏輯遞交更新。若您的裝置沒有執行 FreeRTOS，您必須實作在您裝置上執行的 OTA 代理程式。

安裝初始韌體

若要更新韌體，您必須安裝使用 OTA 代理程式程式庫的初始版本韌體，接聽 OTA 更新任務。若您並非執行 FreeRTOS，請跳過此步驟。您必須將您的 OTA 代理程式實作改為複製到您的裝置。

主題

- 在 Texas Instruments CC3220SF-LAUNCHXL 上安裝初始版本韌體 (p. 136)
- 在 Microchip Curiosity PIC32MZEF 上安裝初始版本韌體 (p. 139)
- 在 Espressif ESP32 上安裝初始版本韌體 (p. 141)
- 在 Nordic nRF52840 DK 上安裝韌體的初始版本 (p. 143)
- Windows 模擬器上的初始韌體 (p. 144)
- 在自訂電路板上安裝初始版本韌體 (p. 144)

在 Texas Instruments CC3220SF-LAUNCHXL 上安裝初始版本韌體

這些步驟的撰寫是假設您已完成建置 aws_demos 專案，如在 [Texas Instruments CC3220SF-LAUNCHXL 下載、建置、更新並執行 FreeRTOS OTA 示範 \(p. 251\)](#) 中所述。

1. 在您的 Texas Instruments CC3220SF-LAUNCHXL 上，將 SOP 跳躍點置放在腳位的中間組上 (位置 = 1)，並重設電路板。
2. 下載並安裝 [TI Uniflash 工具](#)。
3. 啟動 Uniflash。從組態清單中選擇 CC3220SF-LAUNCHXL，然後選擇 Start Image Creator (啟動映像建立工具)。
4. 選擇 New Project (新專案)。
5. 在 Start new project (開始新專案) 頁面上，輸入您專案的名稱。針對 Device Type (裝置類型)，選擇 CC3220SF。針對 Device Mode (裝置模式)，選擇 Develop (開發)。選擇 Create Project (建立專案)。
6. 中斷您終端機模擬器的連線。
7. 在 Uniflash 應用程式視窗的右側，選擇 Connect (連線)。

8. 在 Advanced (進階) 下的 Files (檔案)，選擇 User Files (使用者檔案)。
9. 在 File (檔案) 選擇器窗格中，選擇 Add File (新增檔案) 圖示 .
10. 瀏覽至 /Applications/Ti/simplelink_cc32xx_sdk_version/tools/cc32xx_tools/certificate-playground 目錄，選取 dummy-root-ca-cert，選擇 Open (開啟)，然後選擇 Write (寫入)。
11. 在 File (檔案) 選擇器窗格中，選擇 Add File (新增檔案) 圖示 .
12. 瀏覽至您建立程式碼簽署憑證及私有金鑰的工作目錄，選擇 tisigner.crt.der，選擇 Open (開啟)，然後選擇 Write (寫入)。
13. 從 Action (動作) 下拉式清單中，選擇 Select MCU Image (選取 MCU 映像)，然後選擇 Browse (瀏覽) 來選擇要用於寫入您裝置的韌體映像 (aws_demos.bin)。這個檔案位於 *freertos/vendors/ti/boards/cc3220_launchpad/aws_demos/ccs/Debug* 目錄中。選擇 Open (開啟)。
 - a. 在檔案對話方塊中，確認檔案名稱已設為 mcuflashimg.bin。
 - b. 選取 Vendor (廠商) 核取方塊。
 - c. 在 File Token (檔案字符) 下方，輸入 **1952007250**。
 - d. 在 Private Key File Name (私有金鑰檔案名稱) 下方，選擇 Browse (瀏覽)，然後從您建立程式碼簽署憑證及私有金鑰的工作目錄中選擇 tisigner.key。
 - e. 在 Certification File Name (認證檔案名稱) 下方，選擇 tisigner.crt.der。
 - f. 選擇 Write (寫入)。
14. 在左側窗格中，於 Files (檔案) 下方，選擇 Service Pack (服務套件)。
15. 在 Service Pack File Name (服務套件檔案名稱) 下方，選擇 Browse (瀏覽)，瀏覽至 simplelink_cc32x_sdk_version/tools/cc32xx_tools/servicepack-cc3x20，選擇 sp_3.7.0.1_2.0.0.0_2.2.0.6.bin，然後選擇 Open (開啟)。
16. 在左側窗格中，於 Files (檔案) 下方，選擇 Trusted Root-Certificate Catalog (受信任根憑證目錄)。
17. 清除 Use default Trusted Root-Certificate Catalog (使用預設受信任根憑證目錄) 核取方塊。
18. 在 Source File (來源檔案) 下，選擇 Browse (瀏覽)，然後選擇 simplelink_cc32xx_sdk_version/tools/cc32xx_tools/certificate-playground/certcatalogPlayGround20160911.lst，然後選擇 Open (開放)。
19. 在 Signature Source File (簽章來源檔案) 下，選擇 Browse (瀏覽)，然後選擇 simplelink_cc32xx_sdk_version/tools/cc32xx_tools/certificate-playground/certcatalogPlayGround20160911.lst.signed_3220.bin，然後選擇 Open (開放)。
20. 選擇  以儲存專案。
21. 選擇  按鈕。
22. 選擇 Program Image (Create and Program) (程式映像 (建立及編寫程式))。
23. 在程式設計程序完成後，將 SOP 跳躍點置放在第一組腳位上 (位置 = 0)，重設電路板，然後重新連線您的終端機模擬器以確認輸出與您使用 Code Composer Studio 除錯示範時相同。記下終端機輸出中的應用程式版本編號。您會在稍後使用此版本編號驗證您的韌體已透過 OTA 更新。

終端機應會顯示與以下內容相似的輸出。

```
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 上安裝初始版本韌體

這些步驟的撰寫是假設您已完成建置 aws_demos 專案，如在 [Microchip Curiosity PIC32MZEF 下載、建置、更新並執行 FreeRTOS OTA 示範 \(p. 253\)](#) 中所述。

將示範應用程式燒入您的電路板

- 重新建置 aws_demos 專案，並確認它已在沒有任何錯誤的情況下順利編譯。

2.



在工具列上，選擇

- 在程式設計程序完成後，請中斷與 ICD 4 除錯器的連線，並重設電路板。重新連線您的終端機模擬器，確認輸出與您使用 MPLAB X IDE 除錯示範時相同。

終端機應會顯示與以下內容相似的輸出。

```
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
```

以下程序會建立一個統一 hex 檔案，或是由參考開機載入器及使用密碼編譯簽章應用程式組成的原廠映像。開機載入器在啟動時會驗證應用程式的密碼編譯簽章，並支援 OTA 更新。

建置及刷新原廠映像

- 確定您已安裝來自 SRecordSource Forge 的 工具。驗證包含 srec_cat 和 srec_info 程式的目錄位於您的系統路徑。
- 更新原廠映像的 OTA 序號及應用程式版本。
- 建置 aws_demos 專案。
- 執行 factory_image_generator.py 指令碼來產生原廠映像。

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

此命令會使用下列參數：

- mplab.production.bin：應用程式二進位。
- MCHP-Curiosity-PIC32MZEF：平台名稱。
- private_key.pem：程式碼簽署私有金鑰。
- aws_bootloader.X.production.hex：開機載入器 hex 檔案。

當您建置 aws_demos 專案時，應用程式二進位映像及開機載入器 hex 檔案也會做為程序的一部分進行建置。vendors/microchip/boards/curiosity_pic32mzef/aws_demos/ 目錄底下的每個專案都包含一個具備這些檔案的 dist/pic32mz_ef_curiosity/production/ 目錄。產生的統一 hex 檔案名為 mplab.production.unified.hex。

- 使用 MPLab IPE 工具將生成的 hex 檔案程式設計到裝置上。
- 您可以透過在上傳映像時觀察電路板的 UART 輸出，來檢查您的原廠映像是否正常運作。若一切設定正確，您應該會看到映像已成功開機：

```
[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. 若您的憑證設定不正確，或是若 OTA 映像並未適當簽署，則您可能會在晶片的開機載入器清除無效更新的資料前看到與以下內容相似的訊息。檢查您的程式碼簽署憑證是否一致，並小心檢閱先前的步驟。

```
[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 上安裝初始版本韌體

本指南是在您已執行 [Espressif ESP32-DevKitC 及 ESP-WROVER-KIT 入門](#)，及 [無線更新事前準備](#) 之步驟的前提下，撰寫而成。在嘗試更新 OTA 前，建議您先執行 [FreeRTOS 入門](#) 中說明的 MQTT 示範專案，藉此確保主機板和工具鏈皆已正確設定。

刷新主機板的初始原廠映像

- 開啟 `freertos/vendors/vendor/boards/board/aws_demos/config_files/aws_demo_config.h`、註解 `#define CONFIG_MQTT_DEMO_ENABLED`，然後定義 `CONFIG_OTA_UPDATE_DEMO_ENABLED`。
- 將您在 [OTA 更新先決條件 \(p. 120\)](#) 中所建立的 SHA-256/ECDSA PEM 格式程式碼簽署憑證複製到 `demos/include/aws_ota_codesigner_certificate.h`。它應以下列方式進行格式化。

```
static const char signingcredentialsIGNING_CERTIFICATE_PEM[] = "-----BEGIN
CERTIFICATE-----\n"
"...base64 data...\n"
"-----END CERTIFICATE-----\n";
```

- 在選取 OTA 更新示範的情況下，請遵循 [ESP32 入門](#) 中所概述的相同步驟來建置及刷新映像。若您先前已建置及刷新專案，您可能需要先執行 `make clean`。在您執行 `make flash monitor` 之後，您應該會看到與以下相似的內容。有些訊息的順序可能會不同，因為示範應用程式會一次執行多個任務。

```
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 00003EC0 (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->0x3ffbb4a8
25 1290 [OTA] [OTA_AgentInit] Ready.
26 1390 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
27 1490 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
28 1590 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
29 1690 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
[ ... ]
```

4. ESP32 電路板現在已正在接聽 OTA 更新。ESP-IDF 監控會由 make flash monitor 命令啟動。您可以按下 Ctrl+J 來結束。您也可以使用您偏好的 TTY 終端機程式（例如 PuTTY、Tera Term 或 GNU Screen）來接聽電路板的序列輸出。請注意，連線到主機板的序列埠可能會導致重新開機。

在 Nordic nRF52840 DK 上安裝韌體的初始版本

本指南是在您已執行 [Nordic nRF52840-DK 入門 \(p. 91\)](#)和[無線更新事前準備](#)所述步驟的前提下撰寫而成。在嘗試更新 OTA 前，建議您先執行 [FreeRTOS 入門](#)中說明的 MQTT 示範專案，藉此確保主機板和工具鏈皆已正確設定。

刷新主機板的初始原廠映像

1. Open `freertos/vendors/nordic/boards/nrf52840-dk/aws_demos/config_files/aws_demo_config.h`.
2. 以 `#define CONFIG_MQTT_DEMO_ENABLED` 取代 `#define democonfigOTA_UPDATE_DEMO_ENABLED`.
3. 在選取 OTA 更新示範的情況下，請遵循 [Nordic nRF52840-DK 入門 \(p. 91\)](#)中所概述的相同步驟來建置及刷新映像。

您應該會看到類似下列的輸出。

```
9 1285 [OTA Task] [prvSubscribeToJobNotificationTopics] OK: $aws/things/your-thing-name/jobs/notify-next
10 1286 [OTA Task] [OTA_CheckForUpdate] Request #0
11 1289 [OTA Task] [prvParseJSONbyModel] Extracted parameter [ clientToken: 0:your-thing-name ]
12 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: execution
13 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: jobId
14 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: jobDocument
15 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: afr_ota
16 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: streamname
17 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: files
18 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: filepath
19 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: filesize
20 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: fileid
21 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: certfile
22 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: sig-sha256-ecdsa
23 1289 [OTA Task] [prvParseJobDoc] Ignoring job without ID.
24 1289 [OTA Task] [prvOTA_Close] Context->0x3ffbb4a8
25 1290 [OTA] [OTA_AgentInit] Ready.
26 1390 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
27 1490 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
28 1590 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
```

```
29 1690 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
```

主機板正在接聽 OTA 更新。

Windows 模擬器上的初始韌體

當您使用 Windows 模擬器時，無須刷新初始版本的韌體。Windows 模擬器是 aws_demos 應用程式的一部份，其中也包含了韌體。

在自訂電路板上安裝初始版本韌體

使用您的 IDE，建置 aws_demos 專案，並確認包含 OTA 程式庫。如需 FreeRTOS 原始程式碼結構的詳細資訊，請參閱[FreeRTOS 示範 \(p. 210\)](#)。

請確認您在 FreeRTOS 專案中或您的裝置上包含您的程式碼簽署憑證、私有金鑰及憑證信任鏈。

使用適當的工具，將應用程式燒入您的電路板，並確保其正常執行。

更新您的韌體版本

隨著 FreeRTOS 的 OTA 代理程式會檢查任何更新的版本，並只會在更新版本比現有韌體版本更新時才會進行安裝。以下步驟會示範如何增加 OTA 示範應用程式的韌體版本。

1. 在您的 IDE 中開啟 aws_demos 專案。
2. 開啟 demos/include/aws_application_version.h 並增加 APP_VERSION_BUILD 字符值。
3. 如果您使用的是 Microchip Curiosity PIC32MZEF，請增加 vendors/microchip/boards/curiosity_pic32mzef/bootloader/bootloader/utility/user-config/ota-descriptor.config 中的 OTA 序號。每個新 OTA 映像建立的 OTA 序號都應該遞增。
4. 重新建置專案。

您必須將您的韌體更新複製到您先前根據[建立 Amazon S3 儲存貯體來存放您的更新 \(p. 120\)](#)中所述內容建立的 Amazon S3 儲存貯體。您需要複製到 Amazon S3 的檔案名稱取決於您使用的硬體平台：

- Texas Instruments CC3220SF-LAUNCHXL : vendors/ti/boards/cc3220_launchpad/aws_demos/ccs/debug/aws_demos.bin
- Microchip Curiosity PIC32MZEF : vendors/microchip/boards/curiosity_pic32mzef/aws_demos/mplab/dist/pic32mz_ef_curiosity/production/mplab.production.ota.bin
- Espressif ESP32 : vendors/espressif/boards/esp32/aws_demos/make/build/aws_demos.bin

建立 OTA 更新 (AWS IoT 主控台)

1. 在 AWS IoT 主控台的導覽窗格中，選擇 Manage (管理)，然後選擇 Jobs (任務)。
2. 選擇 Create (建立)。
3. 在 Create a FreeRTOS Over-the-Air (OTA) update job (建立遠端更新任務) 下，選擇 Create OTA update job (建立 OTA 更新任務)。
4. 您可以將 OTA 更新部署到單一裝置或一組裝置。在 Select devices to update (選取要更新的裝置) 下方，選擇 Select (選取)。若要更新單一裝置，請選擇 Things (實物) 標籤。若要更新一組裝置，請選擇 Thing Groups (物件群組) 標籤。
5. 若您要更新單一裝置，請選取與您裝置相關聯 IoT 實物旁的核取方塊。若您要更新一組裝置，請選取與您裝置相關聯物件群組旁的核取方塊。選擇 Next (下一步)。

6. 在 Select the protocol for firmware image transfer (選取通訊協定進行韌體映像傳輸) 下方，選擇 HTTP、MQTT，或選擇兩者以允許每個裝置決定要使用的通訊協定。
7. 在 Select and sign your firmware image (選取及簽署您的韌體映像) 下方，選擇 Sign a new firmware image for me (為我簽署新的韌體映像)。
8. 在 Code signing profile (程式碼簽署描述檔) 下方，選擇 Create (建立)。
9. 在 Create a code signing profile (建立程式碼簽署描述檔) 中，輸入您的程式碼簽署描述檔名稱。
 - a. 在 Device hardware platform (裝置硬體平台) 下方，選擇您的硬體平台。

Note

此清單中只會顯示符合 FreeRTOS 資格的硬體平台。如果您測試的是不符資格的平台，而且您使用 ECDSA P-256 SHA-256 密碼套件來進行簽署，您可以挑選 Windows 模擬器程式碼簽署設定檔以產生相容的簽章。如果您使用的是不符資格的平台，而且您使用 ECDSA P-256 SHA-256 以外的密碼套件來進行簽署，您可以使用適用於 AWS IoT 的程式碼簽署，也可以自行簽署韌體更新。如需詳細資訊，請參閱 [數位簽署您的韌體更新 \(p. 147\)](#)。

- b. 在 Code signing certificate (程式碼簽署憑證) 下方，選擇 Select (選取) 來選取先前匯入的憑證，或是 Import (匯入) 來匯入新憑證。
- c. 在 Pathname of code signing certificate on device (裝置上程式碼簽署憑證的路徑名稱) 下方，輸入指向您裝置上程式碼簽署憑證的完整路徑名稱。憑證的位置因平台而異。且憑證應位於您在依照 [安裝初始韌體 \(p. 136\)](#) 的指示時放置程式碼簽署憑證的位置。

Important

在 Texas Instruments CC3220SF-LAUNCHXL 上，若您的程式碼簽署憑證位於檔案系統的根，請不要在檔案名稱前端包含正斜線 (/)。否則，OTA 更新便會在身份驗證期間發生 file not found 錯誤而失敗。

10. 在 Select your firmware image in S3 or upload it (在 S3 中選取您的韌體映像或上傳它) 下方，選擇 Select (選取)。隨即顯示您的 Amazon S3 儲存貯體清單。選擇包含您韌體更新的儲存貯體，然後在儲存貯體中選擇您的韌體更新。

Note

Microchip Curiosity PIC32MZEF 示範專案會製作兩個預設名稱為 `mplab.production.bin` 和 `mplab.production.ota.bin` 的二進位映像。當您上傳 OTA 更新的映像時，請使用第二個檔案。

11. 在 Pathname of firmware image on device (裝置上韌體映像的路徑名稱) 下方，對 OTA 任務將複製韌體映像之裝置上的位置，輸入完全合格的路徑名稱。此位置因平台而異。

Important

在 Texas Instruments CC3220SF-LAUNCHXL 上，由於安全限制，韌體映像路徑名稱必須是 /sys/mcuflashimg.bin。

12. 在 File Type (檔案類型) 下，輸入範圍 0-255 中的整數值。您輸入的檔案類型會新增到傳送至 MCU 的任務文件。MCU 韌體/軟體開發人員具有有關此數值使用的完整權限。可能的案例包括 MCU，其具備次要處理器，且其韌體可獨立於主要處理器進行更新。當裝置收到 OTA 更新任務時，就可以使用檔案類型來找出更新對象的處理器。
13. 在 IAM role for OTA update job (OTA 更新任務的 IAM 角色) 下方，根據 [建立 OTA 更新服務角色 \(p. 121\)](#) 中的指示選擇角色。
14. 選擇 Next (下一步)。
15. 輸入您 OTA 更新任務的 ID 和說明。
16. 在 Job type (任務類型) 下方，選擇 Your job will complete after deploying to the selected devices/groups (snapshot) (您的任務將會在部署到選取的裝置/群組 (快照) 之後完成)。
17. 針對您的任務 (Job executions rollout (任務執行推展)、Job abort (任務中止)、Job executions timeout (任務執行逾時) 和 Tags (標籤)) 選擇任何適當的選用組態。

18. 選擇 Create (建立)。

使用先前簽署的韌體映像

1. 在 Select and sign your firmware image (選取及簽署您的韌體映像) 下方，選擇 Select a previously signed firmware image (選取先前簽署的韌體映像)。
2. 在 Pathname of firmware image on device (裝置上韌體映像的路徑名稱) 下方，對 OTA 任務將複製韌體映像之裝置上的位置，輸入完全合格的路徑名稱。此位置因平台而異。
3. 在 Previous code signing job (先前的程式簽署任務) 下方，選擇 Select (選取)，然後選擇先前用於簽署用於 OTA 更新韌體映像的程式碼簽署任務。

使用自訂簽署韌體映像

1. 在 Select and sign your firmware image (選取並簽署您的韌體映像) 下方，選擇 Use my custom signed firmware image (使用我的自訂簽署韌體映像)。
2. 在 Pathname of code signing certificate on device (裝置上程式碼簽署憑證的路徑名稱) 下方，輸入指向您裝置上程式碼簽署憑證的完整路徑名稱。此路徑名稱依平台而有所不同。
3. 在 Pathname of firmware image on device (裝置上韌體映像的路徑名稱) 下方，對 OTA 任務將複製韌體映像之裝置上的位置，輸入完全合格的路徑名稱。此位置因平台而異。
4. 在 Signature (簽章) 下方，貼上您的 PEM 格式簽章。
5. 在 Original hash algorithm (原始雜湊演算法) 下方，選擇您在建立檔案簽章時所使用的雜湊演算法。
6. 在 Original encryption algorithm (原始加密演算法) 下方，選擇您在建立檔案簽章時所使用的演算法。
7. 在 Select your firmware image in (在 Amazon S3 中選取您的韌體映像) 下方，選擇 Amazon S3 儲存貯體和 Amazon S3 儲存貯體中已簽署的韌體映像。

在您指定完程式碼簽署資訊後，請指定 OTA 更新任務類型、服務角色及您更新的 ID。

Note

請不要在您 OTA 更新的任務 ID 中使用任何個人識別資訊。個人識別資訊的範例包括：

- 名稱。
- IP 地址。
- 電子郵件地址。
- 位置
- 銀行詳細資訊。
- 醫療資訊。

1. 在 Job type (任務類型) 下方，選擇 Your job will complete after deploying to the selected devices/groups (snapshot) (您的任務將會在部署到選取的裝置/群組 (快照) 之後完成)。
2. 在 IAM role for OTA update job (OTA 更新任務的 IAM 角色) 下方，選擇您的 OTA 服務角色。
3. 輸入您任務的英數字元 ID，然後選擇 Create (建立)。

任務會出現在 AWS IoT 主控台中，且狀態為 IN PROGRESS (進行中)。

Note

- AWS IoT 主控台不會自動更新任務狀態。請重新整理您的瀏覽器，以查看更新。

將您的序列 UART 終端機連線到您的裝置。您應該會看到輸出，指出裝置正在下載更新韌體。

在裝置下載完更新韌體後，它便會重新啟動並安裝韌體。您可以在 UART 終端機中查看實際發生情況。

如需示範如何使用主控台建立 OTA 更新的教學課程，請參閱[無線更新示範應用程式 \(p. 248\)](#)。

使用 AWS CLI 建立 OTA 更新

當您使用 AWS CLI 建立 OTA 更新時，您必須執行以下操作：

1. 數位簽署您的韌體映像。
2. 建立您數位簽署韌體映像的串流。
3. 啟動 OTA 更新任務。

數位簽署您的韌體更新

當您使用 AWS CLI 執行 OTA 更新時，您可以使用適用於 AWS IoT 的程式碼簽署，或是可以自行簽署您的韌體更新。如需適用於 AWS IoT 的程式碼簽署所支援的密碼編譯簽署和井字號演算法清單，請前往[SigningConfigurationOverrides](#)。如果您想要使用適用於 AWS IoT 的程式碼簽署不支援的加密演算法，您必須先簽署韌體二進位檔案，再將其上傳到 Amazon S3。

使用適用於 AWS IoT 的程式碼簽署簽署您的韌體映像

若要使用適用於 AWS IoT 的程式碼簽署簽署您的韌體映像，您可使用[AWS SDKs](#)或命令列工具。如需適用於 AWS IoT 的程式碼簽署詳細資訊，請前往[適用於 AWS IoT 的程式碼簽署](#)。

在您安裝及設定程式碼簽署工具之後，將您尚未簽署的韌體映像複製到 Amazon S3 儲存貯體，並使用以下 CLI 命令啟動程式碼簽署任務。put-signing-profile 命令會建立可重複使用的程式碼簽署描述檔。start-signing-job 命令會啟動簽署任務。

```
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* 可以是相同的 Amazon S3 儲存貯體。

這些是 put-signing-profile 和 start-signing-job 命令的參數：

source

指定 S3 儲存貯體中未簽署韌體的位置。

- **bucketName**：S3 儲存貯體的名稱。
- **key**：您 S3 儲存貯體中韌體的金鑰（檔案名稱）。
- **version**：您 S3 儲存貯體中韌體的 S3 版本。這與您的韌體版本不同。您可以透過瀏覽至 Amazon S3 主控台，選擇您的儲存貯體，然後在頁面頂端的 Versions (版本) 旁邊，選擇 Show (顯示) 來取得它。

destination

裝置上將複製 S3 儲存貯體中已簽署韌體的目標。此參數的格式與 source 參數相同。

signing-material

您程式碼簽署憑證的 ARN。此 ARN 會在您將憑證匯入 ACM 時產生。

signing-parameters

用於簽署的鍵/值對映射。這些可包含任何您希望在簽署期間使用的資訊。

Note

當您建立簽署描述檔以簽署擁有適用於 AWS IoT 程式碼簽署的 OTA 更新時，必須採用此參數。

platform

您分發 OTA 更新之目標硬體平台的 platformId。

若要傳回可用平台及其 platformId 值的清單，請使用 aws signer list-signing-platforms 命令。

簽署任務會啟動並將簽署後的韌體映像寫入目標 Amazon S3 儲存貯體。已簽署韌體映像的檔案名稱是 GUID。當您建立串流時，會需要此檔案名稱。您可以透過瀏覽至 Amazon S3 主控台並選擇您的儲存貯體來尋找檔案名稱。若您沒有看到具有 GUID 檔案名稱的檔案，請重新整理您的瀏覽器。

此命令會顯示任務 ARN 及任務 ID。您稍後會需要這些值。如需適用於 AWS IoT 程式碼簽署的詳細資訊，請參閱[適用於 AWS IoT 的程式碼簽署](#)。

手動簽署您的韌體映像

數位簽署您的韌體映像並將已簽署的韌體映像上傳到您的 Amazon S3 儲存貯體。

建立您韌體更新的串流

串流是可以由裝置使用之資料的抽象介面。串流可以隱藏下列動作的複雜性：存取在不同位置或不同雲端型服務中存放的資料。OTA Update Manager 服務可讓您使用多個資料片段 (存放在 Amazon S3 的不同位置中) 來執行 OTA 更新。

建立 AWS IoT OTA 更新時，您也可以建立一個包含已簽署韌體更新的串流。建立可識別已簽署韌體映像的 JSON 檔案 (stream.json)。JSON 檔案應包含以下內容。

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

這些是 JSON 檔案中的屬性：

fileId

介於 0 – 255 之間的任意整數，可識別您的韌體映像。

s3Location

儲存貯體及用於串流的韌體鍵。

bucket

您未簽署韌體映像存放的 Amazon S3 儲存貯體。

key

Amazon S3 儲存貯體中您已簽署韌體映像的檔案名稱。您可以透過查看您儲存貯體的內容，來在 Amazon S3 主控台中尋找此值。

若您使用適用於 AWS IoT 的程式碼簽署，檔案名稱為適用於 AWS IoT 程式碼簽署所產生的 GUID。

使用 create-stream CLI 命令來建立串流。

```
aws iot create-stream \
--stream-id your_stream_id \
--description your_description \
--files file://stream.json \
--role-arn your_role_arn
```

這些是 create-stream CLI 命令的引數：

stream-id

用於識別串流的任意字串。

description

串流的選擇性說明。

files

一或多個指向 JSON 檔案的參考，JSON 檔案中包含用於串流的韌體映像相關資料。JSON 檔案必須包含以下屬性：

fileId

任意的檔案 ID。

s3Location

儲存貯體名稱，其中存放了已簽署的韌體映像及已簽署韌體映像的鍵 (檔案名稱)。

bucket

已簽署韌體映像存放的 Amazon S3 儲存貯體。

key

已簽署韌體映像的鍵 (檔案名稱)。

當您使用適用於 AWS IoT 的程式碼簽署時，此鍵為 GUID。

以下是範例 *stream.json* 檔案。

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

role-arn

OTA 服務角色 ([p. 121](#))，也授予對韌體映像儲存所在位置的 Amazon S3 儲存貯體的存取權。

若要尋找您已簽署韌體映像的 Amazon S3 物件鍵，請使用 aws signer describe-signing-job --job-id **my-job-id** 命令，其中 my-job-id 是 create-signing-job CLI 命令所顯示的任務 ID。describe-signing-job 命令的輸出包含已簽署韌體映像的鍵。

```
... text deleted for brevity ...
"signedObject": {
    "s3": {
        "bucketName": "ota-bucket",
        "key": "7309da2c-9111-48ac-8ee4-5a4262af4429"
    }
}
... text deleted for brevity ...
```

建立 OTA 更新

使用 create-ota-update CLI 命令建立 OTA 更新任務。

Note

請不要在您 OTA 更新的任務 ID 中使用任何個人識別資訊 (PII)。個人識別資訊的範例包括：

- 名稱。
- IP 地址。
- 電子郵件地址。
- 位置
- 銀行詳細資訊。
- 醫療資訊。

```
aws iot create-ota-update \
--ota-update-id value \
[--description value] \
--targets value \
[--protocols value] \
[--target-selection value] \
[--aws-job-executions-rollout-config value] \
[--aws-job-presigned-url-config value] \
[--aws-job-abort-config value] \
[--aws-job-timeout-config value] \
--files value \
--role-arn value \
[--additional-parameters value] \
[--tags value] \
[--cli-input-json value] \
[--generate-cli-skeleton]
```

cli-input-json 格式

```
{
    "otaUpdateId": "string",
    "description": "string",
    "targets": [
        "string"
    ],
    "protocols": [
        "string"
    ],
    "targetSelection": "string",
    "awsJobExecutionsRolloutConfig": {
        "maximumPerMinute": "integer",
```

```
"exponentialRate": {
    "baseRatePerMinute": "integer",
    "incrementFactor": "double",
    "rateIncreaseCriteria": {
        "numberOfNotifiedThings": "integer",
        "numberOfSucceededThings": "integer"
    }
},
"awsJobPresignedUrlConfig": {
    "expiresInSec": "long"
},
"awsJobAbortConfig": {
    "abortCriteriaList": [
        {
            "failureType": "string",
            "action": "string",
            "thresholdPercentage": "double",
            "minNumberOfExecutedThings": "integer"
        }
    ]
},
"awsJobTimeoutConfig": {
    "inProgressTimeoutInMinutes": "long"
},
"files": [
    {
        "fileName": "string",
        "fileType": "integer",
        "fileVersion": "string",
        "fileLocation": {
            "stream": {
                "streamId": "string",
                "fileId": "integer"
            },
            "s3Location": {
                "bucket": "string",
                "key": "string",
                "version": "string"
            }
        },
        "codeSigning": {
            "awsSignerJobId": "string",
            "startSigningJobParameter": {
                "signingProfileParameter": {
                    "certificateArn": "string",
                    "platform": "string",
                    "certificatePathOnDevice": "string"
                },
                "signingProfileName": "string",
                "destination": {
                    "s3Destination": {
                        "bucket": "string",
                        "prefix": "string"
                    }
                }
            },
            "customCodeSigning": {
                "signature": {
                    "inlineDocument": "blob"
                },
                "certificateChain": {
                    "certificateName": "string",
                    "inlineDocument": "string"
                },
                "hashAlgorithm": "string",
                "awsSignerJobId": "string"
            }
        }
    }
]
```

```
        "signatureAlgorithm": "string"
    },
},
"attributes": {
    "string": "string"
}
],
"roleArn": "string",
"additionalParameters": {
    "string": "string"
},
"tags": [
{
    "Key": "string",
    "Value": "string"
}
]
}
```

cli-input-json 欄位

名稱	類型	描述
otaUpdateId	string (上限：128；下限：1)	欲建立的 OTA 更新 ID。
description	string (上限：2028)	OTA 更新的描述。
targets	清單	將目標鎖定於接收 OTA 更新的裝置。
protocols	清單	用來傳輸 OTA 更新映像的通訊協定。有效值為 [HTTP]、[MQTT]、[HTTP, MQTT]。指定 HTTP 和 MQTT 時，目標裝置可以選擇通訊協定。
targetSelection	string	指定更新是否要持續運作 (CONTINUOUS)，或在指定為目標的所有物件完成更新後視為完成 (SNAPSHOT)。如果是 CONTINUOUS，則在目標中偵測到變更時，更新也可能會運作於物件上。例如，當物件新增至目標群組，更新就會在物件上運作，即使該更新已被原本就在群組中的所有物件完成。有效值：CONTINUOUS SNAPSHOT。 列舉：CONTINUOUS SNAPSHOT
awsJobExecutionsRolloutConfig		OTA 更新的發行組態。
maximumPerMinute	integer (上限：1000；下限：1)	每分鐘可啟動的 OTA 更新任務執行數上限。

名稱	類型	描述
<code>exponentialRate</code>		任務推展的增加率。此參數可讓您為任務推展定義指數增加率。
<code>baseRatePerMinute</code>	integer (上限：1000；下限：1)	在任務推展開始時，每分鐘通知待處理任務的最低物件數量。這是推展的初始率。
<code>rateIncreaseCriteria</code>		任務啟動增加推展速率的條件。 AWS IoT 最多支援小數點後一位數（例如，1.5，但不支援1.55）。
<code>numberOfNotifiedThings</code>	integer (下限：1)	收到此物件數量通知時，它會啟動推展率的增加。
<code>numberOfSucceededThings</code>	integer (下限：1)	當此物件數量已在其任務執行中成功時，它會啟動推展率的增加。
<code>awsJobPresignedUrlConfig</code>		預先簽章的組態資訊 URLs。
<code>expiresInSec</code>	長整數	預先簽章的 URLs 有效時間（以秒為單位）。有效值為 60 - 3600，預設值為 1800 秒。當收到工作文件的要求時，就會建立預先簽章的 URLs。
<code>awsJobAbortConfig</code>		決定任務中止發生之時間和方式的準則。
<code>abortCriteriaList</code>	清單	決定中止任務之時間和方式的準則清單。
<code>failureType</code>	string	可啟動任務中止的任務執行失敗類型。 列舉：FAILED REJECTED TIMED_OUT ALL
<code>action</code>	string	啟動任務中止所要採取的任務動作類型。 列舉：取消
<code>minNumberOfExecutedThings</code>	integer (下限：1)	在任務中止前，必須收到任務執行通知的物件數量下限。
<code>awsJobTimeoutConfig</code>		指定每個裝置必須完成其任務執行的時間。任務執行狀態設定為 IN_PROGRESS 時，計時器即會開始。在計時器到期之前，如果任務執行狀態未設定為其他結束狀態，就會自動設為 TIMED_OUT。

名稱	類型	描述
<code>inProgressTimeoutInMinutes</code>	長整數	指定此裝置必須完成這項任務執行的時間 (以分鐘為單位)。逾時間隔可介於 1 分鐘到 7 天之間 (1 到 10080 分鐘)。進行中的計時器無法更新，並會套用到任務的所有任務執行。每當任務執行維持在 IN_PROGRESS 狀態超過此間隔時，任務執行就會失敗，並切換到結束 TIMED_OUT 狀態。
<code>files</code>	清單	OTA 更新所串流的檔案。
<code>fileName</code>	string	檔案名稱。
<code>fileType</code>	integer 範圍 - 上限 : 255 ; 下限 : 0	您可以在任務文件中包含整數值，讓您的裝置可辨識從雲端收到的檔案類型。
<code>fileVersion</code>	string	檔案版本。
<code>fileLocation</code>		已更新韌體的位置。
<code>stream</code>		包含 OTA 更新的串流。
<code>streamId</code>	string (上限 : 128 ; 下限 : 1)	串流 ID。
<code>fileId</code>	integer (上限 : 255 ; 下限 : 0)	與串流建立關聯的檔案 ID。
<code>s3Location</code>		S3 中已更新韌體的位置。
<code>bucket</code>	string (下限 : 1)	S3 儲存貯體。
<code>key</code>	string (下限 : 1)	S3 金鑰。
<code>version</code>	string	S3 儲存貯體版本。
<code>codeSigning</code>		檔案的代碼簽署方式。
<code>awsSignerJobId</code>	string	建立來簽署檔案的 AWSSignerJob ID。
<code>startSigningJobParameter</code>		描述程式碼簽署任務。
<code>signingProfileParameter</code>		描述程式碼簽署描述檔。
<code>certificateArn</code>	string	憑證 ARN。
<code>platform</code>	string	您裝置的硬體平台。
<code>certificatePathOnDevice</code>	string	程式碼簽署憑證在您裝置上的位置。

名稱	類型	描述
<code>signingProfileName</code>	<code>string</code>	程式碼簽署描述檔名稱。
<code>destination</code>		撰寫程式碼簽章檔案的位置。
<code>s3Destination</code>		描述 S3 中已更新韌體的位置。
<code>bucket</code>	<code>string</code> (下限 : 1)	內含已更新韌體的 S3 儲存貯體。
<code>prefix</code>	<code>string</code>	S3 字首。
<code>customCodeSigning</code>		代碼簽署檔案的自訂方式。
<code>signature</code>		檔案的簽章。
<code>inlineDocument</code>	<code>blob</code>	以 base64 編碼呈現的二進位代碼簽署簽章。
<code>certificateChain</code>		憑證鏈。
<code>certificateName</code>	<code>string</code>	憑證名稱。
<code>inlineDocument</code>	<code>string</code>	以 base64 編碼呈現的二進位代碼簽署憑證鏈。
<code>hashAlgorithm</code>	<code>string</code>	用於以代碼簽署檔案的雜湊演算法。
<code>signatureAlgorithm</code>	<code>string</code>	用於以代碼簽署檔案的簽章演算法。
<code>attributes</code>	對應	名稱/屬性對清單。
<code>roleArn</code>	<code>string</code> (上限 : 2048 ; 下限 : 20)	授予 AWS IoT 對 Amazon S3、AWS IoT 任務和 AWS 程式碼簽署資源之存取權的 IAM 角色，可建立 OTA 更新任務。
<code>additionalParameters</code>	對應	其他 OTA 更新參數清單，以名稱/值對表示。
<code>tags</code>	清單	用於管理更新的中繼資料。
<code>Key</code>	<code>string</code> (上限 : 128 ; 下限 : 1)	標籤的金鑰。
<code>Value</code>	<code>string</code> (上限 : 256 ; 下限 : 1)	標籤的值。

輸出

```
{
  "otaUpdateId": "string",
  "awsIotJobId": "string",
  "otaUpdateArn": "string",
  "awsIotJobArn": "string",
```

```
    "otaUpdateStatus": "string"
}
```

CLI 輸出欄位

名稱	類型	描述
otaUpdateId	string (上限：128；下限：1)	OTA 更新 ID。
awsIotJobId	string	與 OTA 更新建立關聯的 AWS IoT 任務 ID。
otaUpdateArn	string	OTA 更新 ARN。
awsIotJobArn	string	與 OTA 更新建立關聯的 AWS IoT 任務 ARN。
otaUpdateStatus	string	OTA 更新狀態。 列舉：CREATE_PENDING CREATE_IN_PROGRESS CREATE_COMPLETE CREATE_FAILED

以下是 JSON 檔案的範例。該檔案會傳遞到使用適用於 AWS IoT 程式碼簽署的 create-ota-update 命令。

```
[  
  {  
    "fileName": "firmware.bin",  
    "fileType": 1,  
    "fileLocation": {  
      "stream": {  
        "streamId": "004",  
        "fileId": 123  
      }  
    },  
    "codeSigning": {  
      "awsSignerJobId": "48c67f3c-63bb-4f92-a98a-4ee0fbc2bef6"  
    }  
  }  
]
```

以下是 JSON 檔案的範例。該檔案會傳遞到使用內嵌檔案提供自訂程式碼簽署材料的 create-ota-update CLI 命令。

```
[  
  {  
    "fileName": "firmware.bin",  
    "fileType": 1,  
    "fileLocation": {  
      "stream": {  
        "streamId": "004",  
        "fileId": 123  
      }  
    },  
    "codeSigning": {  
      "customCodeSigning": {  
        "signature": {  
          "key": "signatureKey",  
          "value": "signatureValue"  
        }  
      }  
    }  
]
```

```
        "inlineDocument": "your_signature"
    },
    "certificateChain": {
        "certificateName": "your_certificate_name",
        "inlineDocument": "your_certificate_chain"
    },
    "hashAlgorithm": "your_hash_algorithm",
    "signatureAlgorithm": "your_signature_algorithm"
}
}
]
```

以下是 JSON 檔案的範例。該檔案會傳遞到允許 FreeRTOS OTA 啟動程式碼簽署任務並建立程式碼簽署描述檔與串流的 create-ota-update CLI 命令。

```
[
{
    "fileName": "your_firmware_path_on_device",
    "fileType": 1,
    "fileVersion": "1",
    "fileLocation": {
        "s3Location": {
            "bucket": "your_bucket_name",
            "key": "your_object_key",
            "version": "your_S3_object_version"
        }
    },
    "codeSigning": {
        "startSigningJobParameter": {
            "signingProfileName": "myTestProfile",
            "signingProfileParameter": {
                "certificateArn": "your_certificate_arn",
                "platform": "your_platform_id",
                "certificatePathOnDevice": "certificate_path"
            },
            "destination": {
                "s3Destination": {
                    "bucket": "your_destination_bucket"
                }
            }
        }
    }
}]
```

以下是 JSON 檔案的範例。該檔案會傳遞到建立 OTA 更新，以使用現有描述檔啟動程式碼簽署任務並使用指定串流的 create-ota-update CLI 命令。

```
[
{
    "fileName": "your_firmware_path_on_device",
    "fileType": 1,
    "fileVersion": "1",
    "fileLocation": {
        "s3Location": {
            "bucket": "your_s3_bucket_name",
            "key": "your_object_key",
            "version": "your_S3_object_version"
        }
    },
    "codeSigning": {
        "startSigningJobParameter": {
```

```
[  
    {  
        "signingProfileName": "your_unique_profile_name",  
        "destination": {  
            "s3Destination": {  
                "bucket": "your_destination_bucket"  
            }  
        }  
    }  
]
```

以下是 JSON 檔案的範例。該檔案會傳遞到允許 FreeRTOS OTA 使用現有程式碼簽署任務 ID 建立串流的 create-ota-update CLI 命令。

```
[  
    {  
        "fileName": "your_firmware_path_on_device",  
        "fileType": 1,  
        "fileVersion": "1",  
        "codeSigning": {  
            "awsSignerJobId": "your_signer_job_id"  
        }  
    }  
]
```

以下是 JSON 檔案的範例。該檔案會傳遞到建立 OTA 更新的 create-ota-update CLI 命令。更新會從指定的 S3 物件建立串流，並使用自訂程式碼簽署。

```
[  
    {  
        "fileName": "your_firmware_path_on_device",  
        "fileType": 1,  
        "fileVersion": "1",  
        "fileLocation": {  
            "s3Location": {  
                "bucket": "your_bucket_name",  
                "key": "your_object_key",  
                "version": "your_S3_object_version"  
            }  
        },  
        "codeSigning": {  
            "customCodeSigning": {  
                "signature": {  
                    "inlineDocument": "your_signature"  
                },  
                "certificateChain": {  
                    "inlineDocument": "your_certificate_chain",  
                    "certificateName": "your_certificate_path_on_device"  
                },  
                "hashAlgorithm": "your_hash_algorithm",  
                "signatureAlgorithm": "your_sig_algorithm"  
            }  
        }  
    }  
]
```

列出 OTA 更新

您可以使用 list-ota-updates CLI 命令，取得所有 OTA 更新的清單。

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

list-ota-updates 命令的輸出看起來與以下內容相似。

```
{  
    "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  
        }  
    ]  
}
```

取得 OTA 更新的相關資訊

您可以使用 get-ota-update CLI 命令以取得 OTA 更新的建立或刪除狀態。

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

get-ota-update 命令的輸出如下。

```
{  
    "otaUpdateInfo": {  
        "otaUpdateId": "ota-update-001",  
        "otaUpdateArn": "arn:aws:iot:us-west-2:123456789012:otaupdate/ota-update-001",  
        "creationDate": 1575414146.286,  
        "lastModifiedDate": 1575414149.091,  
        "targets": [  
            "arn:aws:iot:us-west-2:123456789012:thing/myDevice"  
        ],  
        "protocols": [ "HTTP" ],  
        "awsJobExecutionsRolloutConfig": {  
            "maximumPerMinute": 0  
        },  
        "awsJobPresignedUrlConfig": {  
            "expiresInSec": 1800  
        },  
        "targetSelection": "SNAPSHOT",  
        "otaUpdateFiles": [  
            {  
                "fileName": "my_firmware.bin",  
                "fileType": 1,  
                "fileLocation": {  
                    "s3Location": {  
                        "bucket": "my-bucket",  
                        "key": "my_firmware.bin",  
                        "version": "AvP3bfJC9gyqnwoxPHuTqM5GWENT4iiii"  
                    }  
                },  
                "codeSigning": {  
                    "awsSignerJobId": "b7a55a54-fae5-4d3a-b589-97ed103737c2",  
                    "startSigningJobParameter": {  
                        "signerArn": "arn:aws:iot:us-west-2:123456789012:signer/ota-signer-001",  
                        "signerJobName": "ota-signer-001",  
                        "signerJobArn": "arn:aws:iot:us-west-2:123456789012:signerJob/ota-signer-001",  
                        "signerJobStatus": "PENDING",  
                        "signerJobLastModified": 1575414149.091  
                    }  
                }  
            }  
        ]  
    }  
}
```

```
        "signingProfileParameter": {},
        "signingProfileName": "my-profile-name",
        "destination": {
            "s3Destination": {
                "bucket": "some-ota-bucket",
                "prefix": "SignedImages/"
            }
        },
        "customCodeSigning": {}
    }
],
"otaUpdateStatus": "CREATE_COMPLETE",
"awsIotJobId": "AFR_OTA-ota-update-001",
"awsIotJobArn": "arn:aws:iot:us-west-2:123456789012:job/AFR_OTA-ota-update-001"
}
}
```

為 otaUpdateStatus 傳回的值包括下列項目：

CREATE_PENDING

建立 OTA 更新正在擱置中。

CREATE_IN_PROGRESS

正在建立 OTA 更新。

CREATE_COMPLETE

已建立 OTA 更新。

CREATE_FAILED

建立 OTA 更新失敗。

DELETE_IN_PROGRESS

正在刪除 OTA 更新。

DELETE_FAILED

刪除 OTA 更新失敗。

Note

若要在 OTA 更新建立後取得其執行狀態，您需要使用 describe-job-execution 命令。如需詳細資訊，請參閱 [Describe Job Execution](#)。

刪除 OTA 相關資料

目前，您無法使用 AWS IoT 主控台刪除串流或 OTA 更新。您可以使用 AWS CLI 刪除串流、OTA 更新，以及在 OTA 更新期間建立的 AWS IoT 任務。

刪除 OTA 串流

建立一個使用 MQTT 的 OTA 更新時，您可以使用命令列或 AWS IoT 主控台建立串流，將韌體分成區塊，以便透過 MQTT 傳送。您可以使用 delete-stream CLI 命令刪除此串流，如下範例所示。

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

刪除 OTA 更新

當您建立 OTA 更新時，系統會建立以下項目：

- OTA 更新任務資料庫中的項目。
- 執行更新的 AWS IoT 任務。
- 每個更新裝置的 AWS IoT 任務執行。

delete-ota-update 命令只會刪除 OTA 更新任務資料庫中的項目。您必須使用 delete-job 命令來刪除 AWS IoT 任務。

使用 delete-ota-update 命令刪除 OTA 更新。

```
aws iot delete-ota-update --ota-update-id your_ota_update_id
```

ota-update-id

欲刪除的 OTA 更新 ID。

delete-stream

刪除與 OTA 更新相關聯的串流。

force-delete-aws-job

刪除與 OTA 更新相關聯的 AWS IoT 任務。若未設定此標記，並且任務正處於 In_Progress 狀態，便不會刪除任務。

棄用為 OTA 更新建立的 IoT 任務

FreeRTOS 會在您建立 OTA 更新時建立 AWS IoT 任務。也會為每個裝置建立任務執行，處理任務。您可以使用 delete-job CLI 命令來刪除任務及其相關聯的任務執行。

```
aws iot delete-job --job-id your-job-id --no-force
```

no-force 參數會指定僅刪除處於終止狀態 (COMPLETED 或 CANCELLED) 的任務。您可以透過傳遞 **force** 參數，來刪除並非處於終止狀態的任務。如需詳細資訊，請前往 [DeleteJob API](#)。

Note

刪除任何處於 IN_PROGRESS 狀態的任務會插斷任何您裝置上同樣處於 IN_PROGRESS 狀態的任務執行，並且可能導致裝置處於不具確定性的狀態。請確保每個執行遭刪除任務的裝置都能復原至已知狀態。

視為任務建立的任務執行數及其他因素而定，刪除任務可能需要數分鐘。刪除任務時，其狀態會是 DELETION_IN_PROGRESS。嘗試刪除或取消狀態已為 DELETION_IN_PROGRESS 的任務將會導致錯誤。

您可以利用 delete-job-execution 來刪除任務執行。您可能會想要在一小部分的裝置無法處理任務時刪除任務執行。這會刪除單一裝置的任務執行，如下列範例所示。

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

與 delete-job CLI 命令相同，您可以將 --force 參數傳遞給 delete-job-execution 來強制刪除任務執行。如需詳細資訊，請前往 [DeleteJobExecution API](#)。

Note

刪除任何處於 IN_PROGRESS 狀態的任務執行會插斷任何您裝置上同樣處於 IN_PROGRESS 狀態的任務執行，並且可能導致裝置處於不具確定性的狀態。請確保每個執行遭刪除任務的裝置都能復原至已知狀態。

關於使用 OTA 更新示範應用程式的詳細資訊，請參閱無線更新示範應用程式 (p. 248)。

OTA Update Manager 服務

無線 (OTA) Update Manager 服務提供了一種方式來執行下列動：

- 建立 OTA 更新及其使用的資源，包括 AWS IoT 任務、AWS IoT 串流，以及程式碼簽署。
- 取得 OTA 更新的相關資訊。
- 列出與 AWS 帳戶相關聯的所有 OTA 更新。
- 刪除 OTA 更新。

OTA 更新是一種由 OTA Update Manager 服務維護的資料結構。它包含以下內容：

- OTA 更新 ID。
- 選擇性的 OTA 更新說明。
- 待更新的裝置清單（「目標」）。
- OTA更新的類型：繼續或暫停。查看 [職位 章節 AWS IoT 開發人員指南](#) 討論您需要的更新類型。
- 用於進行OTA更新的研究方案：[MQTT]、[TTP]或[MQTT,HTTPS]。當您指定 MQTT 和 HTTP 時，裝置設定會判斷已使用的通訊協定。
- 要傳送到目標裝置的檔案清單。
- 授予 AWS IoT 對 Amazon S3、AWS IoT 任務和 AWS 程式碼簽署資源之存取權限的 IAM 角色，可建立 OTA 更新任務。
- 選擇性的使用者定義名稱/值對清單。

OTA 更新的設計旨在更新裝置韌體，但您可以使用它們來將任何您想傳送的檔案傳送到向 AWS IoT 註冊的一或多個裝置。當您無線傳送韌體更新時，我們建議您以數位方式簽署它們，以便接收它們的裝置可以確認它們在途中未遭到竄改。

您可以使用 HTTP 或 MQTT 通訊協定來傳送更新的韌體映像，取決於您選擇的設定。您可以使用[適用於 FreeRTOS 的程式碼簽署](#)來簽署您的韌體更新，或是使用您自己的程式碼簽署工具。

若要進一步控制程序，您可以使用 [CreateStream API](#)，在透過 MQTT 傳送更新時建立串流。在某些情況下，您可以修改 FreeRTOS 代理程式碼，以調整您傳送和接收的區塊大小。

當您建立 OTA 更新時，OTA Manager 服務會建立一個 [AWS IoT 任務](#) 來通知您的裝置有可用的新。FreeRTOS OTA 代理程式會在您的裝置上執行，並接聽更新訊息。當更新可用時，它會透過 HTTP 或 MQTT 請求韌體更新映像，並將檔案存放在本機上。它會檢查下載檔案的數位簽章，並且在檔案有效時安裝韌體更新。如果您不是使用 FreeRTOS，則必須實作您自己的 OTA 代理程式，以接聽下載更新及執行任何安裝操作。

將 OTA 代理程式整合到您的應用程式

無線 (OTA) 代理程式旨在簡化為了將 OTA 更新功能新增到您的產品，所需要撰寫的程式碼數量。整合的負荷主要由 OTA 代理程式初始化及選擇性的建立自訂回撥函數，以回應 OTA 完成事件訊息組成。

Note

雖然將 OTA 更新功能整合到您的應用程式相對簡單，但 OTA 更新系統需要您對裝置程式碼整合之外的知識有更進一步的了解。若要熟悉如何搭配 AWS IoT 物件、憑證、程式碼簽署憑證、佈建裝置及 OTA 更新任務設定您的 AWS 帳戶，請參閱 [FreeRTOS 事前準備](#)。

連線管理

OTA 代理程式會將 MQTT 通訊協定用於所有涉及 AWS IoT 服務的控制通訊操作，但它不會管理 MQTT 連線。為了確保 OTA 代理程式不會影響您應用程式的連線管理政策，MQTT 連線（包含中斷連線及任何重新連線功能）必須由主要使用者應用程式處理。檔案可透過 MQTT 或 HTTP 通訊協定下載。您可以選擇建立 OTA 任務時的通訊協定。如果您選擇 MQTT，則 OTA 代理程式會使用相同的連線，進行控制操作和下載檔案。如果您選擇 HTTP，則 OTA 代理程式會處理 HTTP 連線。

簡單OTA演示

以下是簡易 OTA 示範的摘要，向您展示代理程式連線到 MQTT 中介裝置及初始化 OTA 代理程式的方式。在本例中，我們將演示配置為使用默認的 OTA 完成回調，並返回一些統計數據，每秒鐘一次。為求簡化，我們會從此示範刪掉一下詳細資訊。

OTA 演示還通過監控斷開連接的回撥並重新建立連接來演示 MQTT 連接管理。當發生斷開連接時，演示首先暫停 OTA 代理操作，然後嘗試重新建立 MQTT 連接。MQTT 重連嘗試被延遲一個時間，該時間以指數方式增加，達到最大值，並且還增加了一個振動。如果重新建立連接，OTA 代理將繼續操作。

如需使用 AWS IoT MQTT 中介裝置的運作範例，請參閱 `demos/ota` 目錄中的 OTA 示範程式碼。

因為 OTA 代理程式位於自己的任務中，此範例中刻意造成的一秒鐘延遲只會影響此應用程式。它不會影響代理程式的效能。

```
void vRunOTAUpdateDemo( bool awsIotMqttMode,
                        const char * pIdentifier,
                        void * pNetworkServerInfo,
                        void * pNetworkCredentialInfo,
                        const IotNetworkInterface_t * pNetworkInterface )
{
    OTA_State_t eState;
    static OTA_ConnectionContext_t xOTAConnectionCtx;

    IotLogInfo( "OTA demo version %u.%u.%u\r\n",
                xAppFirmwareVersion.u.x.ucMajor,
                xAppFirmwareVersion.u.x.ucMinor,
                xAppFirmwareVersion.u.x.usBuild );

    for( ; ; )
    {
        IotLogInfo( "Connecting to broker...\r\n" );

        /* Establish a new MQTT connection. */
        if( _establishMqttConnection( awsIotMqttMode,
                                      pIdentifier,
                                      pNetworkServerInfo,
                                      pNetworkCredentialInfo,
                                      pNetworkInterface,
                                      &_mqttConnection ) == EXIT_SUCCESS )
        {
            /* Update the connection context shared with OTA Agent.*/
            xOTAConnectionCtx.pxNetworkInterface = ( void * ) pNetworkInterface;
            xOTAConnectionCtx.pvNetworkCredentials = pNetworkCredentialInfo;
            xOTAConnectionCtx.pvControlClient = _mqttConnection;

            /* Set the base interval for connection retry.*/
            _retryInterval = OTA_DEMO_CONN_RETRY_BASE_INTERVAL_SECONDS;
        }
    }
}
```

```
/* Update the connection available flag.*/
_networkConnected = true;

/* Check if OTA Agent is suspended and resume.*/
if( ( eState = OTA_GetAgentState() ) == eOTA_AgentState_Suspended )
{
    OTA_Resume( &xOTAConnectionCtx );
}

/* Initialize the OTA Agent , if it is resuming the OTA statistics will be
cleared for new connection.*/
OTA_AgentInit( ( void * ) ( &xOTAConnectionCtx ),
               ( const uint8_t * ) ( clientcredentialIOT_THING_NAME ),
               App_OTACompleteCallback,
               ( TickType_t ) ~0 );

while( ( ( eState = OTA_GetAgentState() ) != eOTA_AgentState_Stopped ) &&
      _networkConnected )
{
    /* Wait forever for OTA traffic but allow other tasks to run and output
statistics only once per second. */
    IotClock_SleepMs( OTA_DEMO_TASK_DELAY_SECONDS * 1000 );

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

/* Check if we got network disconnect callback and suspend OTA Agent.*/
if( _networkConnected == false )
{
    /* Suspend OTA agent.*/
    if( OTA_Suspend() == kOTA_Err_None )
    {
        while( ( eState = OTA_GetAgentState() ) != eOTA_AgentState_Suspended )
        {
            /* Wait for OTA Agent to process the suspend event. */
            IotClock_SleepMs( OTA_DEMO_TASK_DELAY_SECONDS * 1000 );
        }
    }
    else
    {
        /* Try to close the MQTT connection. */
        if( _mqttConnection != NULL )
        {
            IotMqtt_Disconnect( _mqttConnection, 0 );
        }
    }
}
else
{
    IotLogError( "ERROR: MQTT_AGENT_Connect() Failed.\r\n" );
}

/* After failure to connect or a disconnect, delay for retrying connection.*/
_connectionRetryDelay();
}
```

以下是此示範應用程式的高層級流程：

- 建立 MQTT 代理程式內容。

- 連線到您的 AWS IoT 端點。
- 初始化 OTA 代理程式。
- 允許OTA更新作業的循環,並輸出統計結果,每次一秒。
- 如果MQTT斷開連接,則暫停OTA代理操作。
- 請嘗試使用指數延遲和振動再次連接。
- 如果重新連接,則恢復OTA代理操作。
- 如果代理停止,請延遲一秒鐘,然後嘗試重新連接。

針對 OTA 完成事件使用自訂回撥

之前使用的示例 App_OТАCompleteCallback 作為OTA完成事件的回調處理者。(參見 OTA_AgentInit API 調用。) App_OТАCompleteCallback 定義見 [freertos/demos/ota/aws_iot_ota_update_demo.c](#)。如果您想要對完成事件執行自定義處理,您必須將回撥處理程序的功能地址傳遞給 OTA_AgentInit API。在 OTA 程序期間 , 代理程式可傳送以下其中一種事件列舉給回撥處理常式。如何及何時處理這些事件 , 則由應用程式開發人員決定。

```
/**  
 * @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 代理程式可在主要應用程式的作用中處理期間 , 於背景接收更新。交付這些事件的目的是讓應用程式決定是否可立即採取動作 , 或是否應進行延遲 , 直到完成某些其他應用程式限定處理為止。這個防止您的裝置在作用中處理期間因韌體更新後重設而發生未預期的插斷 (例如真空)。以下是回撥處理常式會接收到的任務事件 :

eOTA_JobEvent_Activate event

當回撥處理常式接收到此事件時 , 您可以立即重設裝置 , 或是排程呼叫於稍後重設裝置。這可讓您延期裝置重設及自我測試階段 (若需要的話)。

eOTA_JobEvent_Fail event

當回撥處理常式接收到此事件時 , 表示更新已失敗。在這種情況下 , 您無須採取任何行動。您可能會希望輸出日誌訊息或執行某些應用程式限定操作。

eOTA_JobEvent_StartTest event

自我測試階段旨在讓剛更新的韌體執行並自我測試 , 以判斷其是否正常運作並遞交為最新的永久應用程式映像。接收到新的更新並完成驗證 , 且重設裝置後 , OTA 代理程式會在準備好進行測試時傳送 eOTA_JobEvent_StartTest 事件給回撥函數。開發人員可以新增必要測試 , 以判斷裝置韌體在更新之後是否正常運作。當裝置韌體已由自我測試視為可靠時 , 程式碼必須透過呼叫 OTA_SetImageState(eOTA_ImageState_Accepted) 函數來將韌體遞交為新的永久映像。

若您的裝置沒有任何需要測試的特殊硬體或機制，您可以使用預設回撥處理常式。在接收到 `eOTA_JobEvent_Activate` 事件時，預設處理常式會立即重設裝置。

OTA 安全性

以下是無線 (OTA) 安全性的三個方面：

連線安全

OTA Update Manager 服務倚賴 AWS IoT 使用的現有安全機制，例如 Transport Layer Security (TLS) 交互身分驗證。OTA 更新流量會透過 AWS IoT 裝置閘道傳遞，並使用 AWS IoT 安全機制。每個透過裝置閘道傳入及傳出的 HTTP 或 MQTT 訊息都會經過身分驗證及授權。

OTA 更新的真確性及完整性

韌體可在 OTA 更新之前進行數位簽署，確保其來自可靠的來源並且並未遭到竄改。

FreeRTOS OTA Update Manager 服務使用適用於 AWS IoT 的程式碼簽署來自動簽署您的韌體。如需詳細資訊，請參閱[適用於 AWS IoT 的程式碼簽署](#)。

在您裝置上執行的 OTA 代理程式會在韌體抵達裝置時對其進行完整性檢查。

操作員安全

每個透過控制面 API 進行的 API 呼叫都會經過標準 IAM 簽章版本 4 身份驗證及授權。若要建立部署，您必須擁有呼叫 `CreateDeployment`、`CreateJob` 及 `CreateStream` API 的許可。此外，在您的 Amazon S3 儲存貯體政策或 ACL 中，您必須為 AWS IoT 服務委託人提供讀取許可，讓存放在 Amazon S3 中的韌體更新可在串流期間受到存取。

適用於 AWS IoT 的程式碼簽署

AWS IoT 主控台會使用[適用於 AWS IoT 的程式碼簽署](#)為任何 AWS IoT 支援的裝置自動簽署您的韌體映像。

適用於 AWS IoT 的程式碼簽署會使用您匯入 ACM 的憑證及私有金鑰。您可以使用自我 - 簽署憑證進行測試，但我們建議您從知名的 – 商業憑證授權單位 (CA) 取得憑證。

使用 X.509 第 3 版 Key Usage 和 Extended Key Usage 延伸模組的程式碼 – 簽署憑證。的 Key Usage 分機設置為 Digital Signature 以及 Extended Key Usage 分機設置為 Code Signing。有關簽署代碼圖像的更多信息，請參閱[代碼簽名 AWS IoT 開發者指南](#) 以及[代碼簽名 AWS IoT API 參考](#)。

Note

您可以從適用於 Amazon Web Services 的工具下載適用於 AWS IoT 開發套件的程式碼簽署。

OTA 故障診斷

以下章節包含有助於您故障診斷 OTA 更新問題的資訊。

主題

- [設定 CloudWatch Logs 以進行 OTA 更新 \(p. 167\)](#)
- [使用 AWS CloudTrail 記錄 AWS IoT OTA API 呼叫 \(p. 170\)](#)
- [Get CreateOTAUpdate failure details using the AWS CLI \(p. 172\)](#)
- [使用 AWS CLI 取得 OTA 失敗代碼 \(p. 173\)](#)

- 故障診斷多個裝置的 OTA 更新 (p. 174)
- 故障診斷 Texas Instruments CC3220SF Launchpad 的 OTA 更新 (p. 174)

設定 CloudWatch Logs 以進行 OTA 更新

OTA 更新服務支援使用 Amazon CloudWatch 記錄日誌。您可以使用 AWS IoT 主控台啟用並設定 OTA 更新的 Amazon CloudWatch 記錄日誌。如需詳細資訊，請參閱 [Cloudwatch Logs](#)。

若要啟用記錄日誌，您必須建立 IAM 角色並設定 OTA 更新記錄日誌。

Note

在啟用 OTA 更新的記錄日誌功能前，請確認您對 CloudWatch Logs 存取許可有所了解。具有 CloudWatch Logs 存取權限的使用者可查看您的除錯資訊。For information, see [Authentication and Access Control for Amazon CloudWatch Logs](#).

建立記錄角色及啟用記錄日誌

請使用 [AWS IoT 主控台](#) 來建立記錄角色及啟用記錄日誌。

1. 從導覽窗格中，選擇 Settings (設定)。
2. 在 Logs (日誌) 下方，選擇 Edit (編輯)。
3. 在 Level of verbosity (詳細層級) 下方，選擇 Debug (除錯)。
4. 在 Set role (設定角色) 下方，選擇 Create new (新增) 來建立記錄日誌的 IAM 角色。
5. 在 Name (名稱) 下方，輸入您角色的唯一名稱。隨即建立您的角色，並包含所有必要許可。
6. 選擇 Update (更新)。

OTA 更新日誌

OTA 更新服務會在發生下列其中一項情況時，將日誌發佈到您的帳戶：

- 已建立 OTA 更新。
- 已完成 OTA 更新。
- 已建立程式碼簽署任務。
- 已完成程式碼簽署任務。
- 已建立 AWS IoT 任務。
- 已完成 AWS IoT 任務。
- 已建立串流。

您可以在 [CloudWatch 主控台](#) 中檢視您的日誌。

在 CloudWatch Logs 中檢視 OTA 更新

1. 從導覽窗格中，選擇 Logs (日誌)。
2. In Log Groups, choose AWSIoTLogsV2.

OTA 更新日誌可包含以下屬性：

accountId

於其中產生日誌的 AWS 帳戶 ID。

actionType

產生日誌的動作。這可以設為下列其中一個值：

- CreateOTAUpdate: An OTA update was created.
- DeleteOTAUpdate: An OTA update was deleted.
- StartCodeSigning: A code-signing job was started.
- CreateAWSJob: An AWS IoT job was created.
- CreateStream: A stream was created.
- GetStream: A request for a stream was sent to the AWS IoT Streaming service.
- DescribeStream: A request for information about a stream was sent to the AWS IoT Streaming service.

awsJobId

產生日誌的 AWS IoT 任務 ID。

clientId

發出產生日誌請求的 MQTT 用戶端 ID。

clientToken

與產生日誌請求相關聯的用戶端字符。

details

用於產生日誌之操作的詳細資訊。

logLevel

日誌的記錄層級。針對 OTA 更新日誌，這一律設為 DEBUG。

otaUpdateId

產生日誌的 OTA 更新 ID。

protocol

用於發出產生日誌請求的通訊協定。

status

產生日誌操作的狀態。有效值為：

- Success (成功)
- Failure (失敗)

streamId

產生日誌的 AWS IoT 串流 ID。

timestamp

日誌產生的時間。

topicName

用於發出產生日誌請求的 MQTT 主題。

範例日誌

以下是啟動程式碼簽署任務時產生的範例日誌：

```
{  
    "timestamp": "2018-07-23 22:59:44.955",  
    "logLevel": "DEBUG",
```

```
"accountId": "123456789012",
"status": "Success",
"actionType": "StartCodeSigning",
"otaUpdateId": "08957b03-eea3-448a-87fe-743e6891ca3a",
"details": "Start code signing job. The request status is SUCCESS."
}
```

以下是建立 AWS IoT 任務時產生的範例日誌：

```
{
  "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."
}
```

以下是建立 OTA 更新時產生的範例日誌：

```
{
  "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."
}
```

以下是建立串流時產生的範例日誌：

```
{
  "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."
}
```

以下是刪除 OTA 更新時產生的範例日誌：

```
{
  "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."
}
```

以下是裝置從串流服務請求串流時產生的範例日誌：

```
{
```

```
"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."
}
```

以下是裝置呼叫 `DescribeStream` API 時產生的範例日誌：

```
{
  "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 記錄 AWS IoT OTA API 呼叫

FreeRTOS is integrated with CloudTrail, a service that captures AWS IoT OTA API calls and delivers the log files to an Amazon S3 bucket that you specify. CloudTrail captures API calls from your code to the AWS IoT OTA APIs. Using the information collected by CloudTrail, you can determine the request that was made to AWS IoT OTA, the source IP address from which the request was made, who made the request, when it was made, and so on.

如需 CloudTrail 的詳細資訊，包括如何設定及啟用，請參閱 [AWS CloudTrail User Guide](#)。

CloudTrail 中的 FreeRTOS 資訊

在 AWS 帳戶中啟用 CloudTrail 記錄後，系統即會在 CloudTrail 日誌檔案中追蹤對 AWS IoT OTA 動作發出的 API 呼叫，並在此處將這些呼叫與其他 AWS 服務記錄一起寫入。CloudTrail 會根據時間週期和檔案大小來決定建立和寫入新檔案的時機。

CloudTrail 會記錄下列 AWS IoT OTA 控制平面動作：

- [CreateStream](#)
- [DescribeStream](#)
- [ListStreams](#)
- [UpdateStream](#)
- [DeleteStream](#)
- [CreateOTAUpdate](#)
- [GetOTAUpdate](#)
- [ListOTAUpdates](#)
- [DeleteOTAUpdate](#)

Note

CloudTrail 不會記錄 AWS IoT OTA 資料平面動作（裝置端），因此您需使用 CloudWatch 來監控這些動作。

每個日誌項目都會包含產生請求對象的資訊。日誌記錄中的使用者身分資訊，可協助您判斷下列事項：

- 該請求是否使用根或 IAM 使用者登入資料提出。
 - 提出該要求時，是否使用了特定角色或聯合身分使用者的暫時安全登入資料。
 - 該請求是否由另一項 AWS 服務提出。

For more information, see the [CloudTrail userIdentity Element](#). AWS IoT OTA actions are documented in the [AWS IoT OTA API Reference](#).

您可以視需要在 Amazon S3 儲存貯體存放您的日誌檔，但也可以定義 Amazon S3 生命週期規則，自動封存或刪除日誌檔案。根據預設，您的日誌檔案使用 Amazon S3 伺服器端加密 (SSE) 進行加密。

若想要在日誌檔案送達時收到通知，您可以將 CloudTrail 設定為發佈 Amazon SNS 通知。For more information, see [Configuring Amazon SNS Notifications for CloudTrail](#).

您也可以將多個 AWS 區域與多個 AWS 帳戶的 AWS IoT OTA 日誌檔彙整至單一 Amazon S3 儲存貯體。

For more information, see [Receiving CloudTrail Log Files from Multiple Regions](#) and [Receiving CloudTrail Log Files from Multiple Accounts](#).

了解 FreeRTOS 目誌檔項目

CloudTrail 日誌檔案可包含一個或多個日誌項目。每一項目均列出多個 JSON 格式的事件。一個日誌項目為任何來源提出的單一請求，並包含請求動作、請求的日期和時間、請求參數等資訊。日誌項目並非公有 API 呼叫的有序堆疊追蹤，因此不會以任何特定順序顯示。

以下範例顯示 CloudTrail 日誌項目，示範呼叫 CreateOTAUpdate 動作的日誌。

```
{  
    "eventVersion": "1.05",  
    "userIdentity": {  
        "type": "IAMUser",  
        "principalId": "EXAMPLE",  
        "arn": "arn:aws:iam::your_aws_account:user/your_user_id",  
        "accountId": "your_aws_account",  
        "accessKeyId": "your_access_key_id",  
        "userName": "your_username",  
        "sessionContext": {  
            "attributes": {  
                "mfaAuthenticated": "false",  
                "creationDate": "2018-08-23T17:27:08Z"  
            }  
        },  
        "invokedBy": "apigateway.amazonaws.com"  
    },  
    "eventTime": "2018-08-23T17:27:19Z",  
    "eventSource": "iot.amazonaws.com",  
    "eventName": "CreateOTAUpdate",  
    "awsRegion": "your_aws_region",  
    "sourceIPAddress": "apigateway.amazonaws.com",  
    "userAgent": "apigateway.amazonaws.com",  
    "requestParameters": {  
        "targets": [  
            "arn:aws:iot:your_aws_region:your_aws_account:thing/Thing",  
        ],  
        "roleArn": "arn:aws:iam::your_aws_account:role/Role_FreeRTOS"  
    }  
}
```

```
"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"
}
```

Get CreateOTAUpdate failure details using the AWS CLI

If the process of creating an OTA update job fails, there may be actions you can take to remedy the problem. When you create an OTA update job, the OTA manager service creates an IoT job and schedules it for the target devices, and this process also creates or uses other types of AWS resources in your account (a code-signing job, an AWS IoT stream, an Amazon S3 object). Any error encountered may cause the process to fail without creating an AWS IoT job. In this troubleshooting section we give instructions on how to retrieve the details of the failure.

1. 安裝及設定 AWS CLI。
2. Run `aws configure` and enter the following information.

```
$ aws configure
AWS Access Key ID [None]: AccessID
AWS Secret Access Key [None]: AccessKey
Default region name [None]: Region
Default output format [None]: json
```

For more information, see [Quick configuration with aws configure](#).

3. 執行：

```
aws iot get-ota-update --ota-update-id ota_update_job_001
```

Where `ota_update_job_001` is the ID you gave the OTA update when you created it.

4. 輸出看起來像這樣：

```
{
    "otaUpdateInfo": {
        "otaUpdateId": "ota_update_job_001",
```

```
"otaUpdateArn": "arn:aws:iot:us-west-2:account_id:otaupdate/ota_update_job_001",
  "creationDate": 1584646864.534,
  "lastModifiedDate": 1584646865.913,
  "targets": [
    "arn:aws:iot:us-west-2:account_id:thing/thing_001"
  ],
  "protocols": [
    "MQTT"
  ],
  "awsJobExecutionsRolloutConfig": {},
  "awsJobPresignedUrlConfig": {},
  "targetSelection": "SNAPSHOT",
  "otaUpdateFiles": [
    {
      "fileName": "/12ds",
      "fileLocation": {
        "s3Location": {
          "bucket": "bucket_name",
          "key": "demo.bin",
          "version": "Z7X.TWSAS7JSi4rybc02nMdcE41W1tV3"
        }
      },
      "codeSigning": {
        "startSigningJobParameter": {
          "signingProfileParameter": {},
          "signingProfileName": "signing_profile_name",
          "destination": {
            "s3Destination": {
              "bucket": "bucket_name",
              "prefix": "SignedImages/"
            }
          }
        },
        "customCodeSigning": {}
      }
    }
  ],
  "otaUpdateStatus": "CREATE_FAILED",
  "errorInfo": {
    "code": "AccessDeniedException",
    "message": "S3 object demo.bin not accessible. Please check your permissions (Service: AWSSigner; Status Code: 403; Error Code: AccessDeniedException; Request ID: 01d8e7a1-8c7c-4d85-9fd7-dcde975fdd2d)"
  }
}
```

If the create failed, the otaUpdateStatus field in the command output will contain CREATE_FAILED and the errorInfo field will contain the details of the failure.

使用 AWS CLI 取得 OTA 失敗代碼

1. 安裝及設定 AWS CLI。
2. Run aws configure and enter following information.

```
$ aws configure
AWS Access Key ID [None]: AccessID
AWS Secret Access Key [None]: AccessKey
Default region name [None]: Region
Default output format [None]: json
```

For more information, see [Quick configuration with aws configure](#).

3. 執行：

```
aws iot describe-job-execution --job-id JobID --thing-name ThingName
```

Where *JobID* is the complete job ID string for the job whose status we want to get (it was associated with the OTA update job when it was created) and *ThingName* is the AWS IoT thing name that the device is registered as in AWS IoT

4. 輸出看起來像這樣：

```
{
    "execution": {
        "jobId": "AFR_OTA-*****",
        "status": "FAILED",
        "statusDetails": {
            "detailsMap": {
                "reason": "0xEEEEEEEE: 0xffffffff"
            }
        },
        "thingArn": "arn:aws:iot:Region:AccountID:thing/ThingName",
        "queuedAt": 1569519049.9,
        "startedAt": 1569519052.226,
        "lastUpdatedAt": 1569519052.226,
        "executionNumber": 1,
        "versionNumber": 2
    }
}
```

在此範例輸出中，「detailsmap」中的「reason」有兩個欄位：顯示為「0xEEEEEEEE」的欄位包含 OTA 代理程式的一般錯誤碼；顯示為「0xffffffff」的欄位則包含子代碼。通用錯誤碼列在 https://docs.aws.amazon.com/freertos/latest/lib-ref/html1/aws_ota_agent_8h.html 中。請參閱前綴為 "kOTA_Err_" 的錯誤代碼。子代碼可以是平台專用代碼，也可以提供一般錯誤的詳細資訊。

故障診斷多個裝置的 OTA 更新

To perform OTAs on multiple devices (things) using the same firmware image, implement a function (for example `getThingName()`) that retrieves `clientcredentialIOT_THING_NAME` from non-volatile memory. 請確定此函數是從非揮發性記憶體中 OTA 不覆寫的部分讀取物件名稱，而且物件名稱是在執行第一個任務之前佈建的。如果您使用 JITP 流程，則可以從裝置憑證的通用名稱中讀取物件名稱。

故障診斷 Texas Instruments CC3220SF Launchpad 的 OTA 更新

CC3220SF Launchpad 平台提供軟體竄改偵測機制。這項機制使用安全提醒計數器，每當完整性違規時就會遞增。裝置會在安全提醒計數器到達預先定義的閾值 (預設值為 15) 時鎖定，並且主機會接收到非同步事件 `SL_ERROR_DEVICE_LOCKED_SECURITY_ALERT`。鎖定後的裝置接著便僅有受限的存取性。若要復原裝置，您可以重新編寫它的程式，或是使用「還原至原廠」程序來回復至原廠映像。建議您透過在 `network_if.c` 中更新非同步事件處理常式，來編寫所需要的行為。

FreeRTOS 程式庫

FreeRTOS 程式庫為 FreeRTOS 核心及其外部程式庫提供額外的功能。您可以使用 FreeRTOS 程式庫在嵌入式應用程式中進行聯網和安全。FreeRTOS 程式庫也可讓您的應用程式與 AWS IoT 服務互動。FreeRTOS 包含程式庫，可讓您：

- 使用 MQTT 和裝置影子安全地將裝置連線至 AWS IoT 雲端。
- 搜尋並連線到 AWS IoT Greengrass 核心。
- 管理 Wi-Fi 連線。
- 聆聽和處理 [FreeRTOS 無線更新 \(p. 120\)](#)。

`libraries` 目錄包含 FreeRTOS 程式庫的來源碼。helper 函數可協助您實作程式庫功能。我們不建議您變更這些 helper 函數。

FreeRTOS 移植程式庫

以下移植程式庫會併入在 FreeRTOS 的組態中，而您可在 FreeRTOS 主控台下載這些組態。這些程式庫與平台相依。其內容會根據您的硬體平台而變更。如需有關將這些程式庫移植到裝置的資訊，請參閱 [FreeRTOS 移植指南](#)。

FreeRTOS 移植程式庫

程式庫	API 參考	描述
低功耗藍牙	低功耗藍牙 API 參考	使用 FreeRTOS 低功耗藍牙程式庫，您的微型控制器可以透過閘道裝置與 AWS IoT MQTT 中介裝置通訊。如需詳細資訊，請參閱 低功耗藍牙程式庫 (p. 182) 。
無線更新	OTA 代理程式 API 參考	FreeRTOS AWS IoT 無線 (OTA) 代理程式程式庫可將您的 FreeRTOS 裝置連接至 AWS IoT OTA 代理程式。 如需詳細資訊，請參閱 OTA 代理程式程式庫 (p. 196) 。
FreeRTOS +POSIX	FreeRTOS+POSIX API 參考	您可以使用 FreeRTOS+POSIX 程式庫，將 POSIX 相容應用程式移植到 FreeRTOS 生態系統。 如需詳細資訊，請造訪 +POSIXFreeRTOS 。
Secure Sockets	Secure Sockets API 參考	如需詳細資訊，請參閱 Secure Sockets 程式庫 (p. 200) 。
FreeRTOS +TCP	FreeRTOS+TCP API 參考	FreeRTOS +TCP 是一種適用於 FreeRTOS 的可擴展、開放原始碼和執行緒安全 TCP/IP 堆疊。

程式庫	API 參考	描述
		如需詳細資訊，請造訪 +TCPFreeRTOS 。
Wi-Fi	Wi-Fi API 參考	FreeRTOS Wi-Fi 程式庫可讓您透過界面與微型控制器的更低階無線堆疊互動。 如需詳細資訊，請參閱 Wi-Fi 程式庫 (p. 205) 。
corePKCS11		程式庫是公開金鑰加密標準 #11 的參考實作，支援配置和 TLS 使用者身份驗證。corePKCS11 如需詳細資訊，請參閱 corePKCS11 程式庫 (p. 198) 。
TLS		如需詳細資訊，請參閱 Transport Layer Security (p. 205) 。
通用 I/O	通用 I/O API 參考	如需詳細資訊，請參閱 通用 I/O (p. 191) 。

FreeRTOS 應用程式庫

您可以選擇將下列獨立應用程式庫併入您的 FreeRTOS 組態中，以與雲端上的 AWS IoT 服務互動。

Note

有些應用程式程式庫與適用於 Embedded 的 APIs 裝置開發套件 AWS IoT 中的程式庫具有相同的 。如需這些程式庫，請參見 [裝置開發套件 C API 參考 AWS IoT](#)。如需適用於 Embedded AWS IoT 的 裝置 SDK 之詳細資訊，請前往 [AWS IoT 適用於 Embedded 的 裝置 SDK C \(p. 13\)](#)。

FreeRTOS 應用程式庫

程式庫	API 參考	描述
AWS IoT Device Defender	Device Defender C SDK API 參考	FreeRTOS AWS IoT Device Defender 程式庫可將您的 FreeRTOS 裝置連接至 AWS IoT Device Defender。 如需詳細資訊，請參閱 AWS IoT Device Defender 程式庫 (p. 192) 。
AWS IoT Greengrass	Greengrass API 參考	FreeRTOS AWS IoT Greengrass 程式庫可將您的 FreeRTOS 裝置連接至 AWS IoT Greengrass。 如需詳細資訊，請參閱 AWS IoT Greengrass Discovery 程式庫 (p. 192) 。
MQTT	MQTT (v1.x.x) 程式庫 API 參考 MQTT (v1) 代理程式 API 參考	此 coreMQTT 程式庫可讓您的 FreeRTOS 裝置發行和訂用 MQTT 主

程式庫	API 參考	描述
	MQTT (v2.x.x) C SDK API 參考	題。MQTT 是裝置用來與 AWS IoT 互動的通訊協定。 如需 coreMQTT 程式庫 3.0.0 版的詳細資訊，請前往 coreMQTT 程式庫 (p. 195) 。
AWS IoT Device Shadow	Device Shadow C SDK API 參考	AWS IoT 裝置陰影程式庫可讓您的 FreeRTOS 裝置與 AWS IoT 裝置陰影進行互動。 如需詳細資訊，請參閱 AWS IoT 裝置影子程式庫 (p. 204) 。

FreeRTOS 常見的程式庫

以下常見的程式庫會搭配額外的資料結構，和內嵌應用程式開發的函式來擴展核心功能。這些程式庫通常是 FreeRTOS 移植和應用程式庫的依存項目。

Note

適用於 Embedded 的 AWS IoT 裝置 SDK C 包含 APIs 和 功能相同的通用程式庫。如需 API 參考資料，請參閱 [AWS IoT 裝置開發套件 C API 參考資料](#)。

FreeRTOS 常見的程式庫

程式庫	API 參考	描述
Atomic Operations	Atomic Operations C SDK API 參考	如需詳細資訊，請參閱 不可部分完成操作 (p. 178) 。
Linear Containers	Linear Containers C SDK API 參考	如需詳細資訊，請參閱 Linear Containers 程式庫 (p. 178) 。
日誌	Logging C SDK API 參考	如需詳細資訊，請參閱 記錄程式庫 (p. 179) 。
Static Memory	Static Memory C SDK API 參考	如需詳細資訊，請參閱 Static Memory 程式庫 (p. 179) 。
Task Pool	Task Pool C SDK API 參考	如需詳細資訊，請參閱 任務集區程式庫 (p. 179) 。

設定 FreeRTOS 程式庫

適用於 FreeRTOS 和適用於 Embedded AWS IoT 的 裝置 SDK 組態設定定義為 C 預先處理器常數。您可以透過全域組態檔案或使用編譯器選項（例如 -D 中的 gcc）來設定組態設定。由於組態設定是定義為編譯時間常數，因此若組態設定變更，便必須重建程式庫。

如果您想要使用全域組態檔案來設定組態選項，請使用名稱 iot_config.h 建立並儲存檔案，再將它放在包含路徑中。在該檔案中，請使用 #define 指令以設定 FreeRTOS 程式庫、示範和測試。

如需支援的全球組態選項詳細資訊，請參閱 [全域組態檔案參考](#)。

常見的程式庫

FreeRTOS 包含一些常見的程式庫，這些程式庫會搭配額外的資料結構，和內嵌應用程式開發的函式來擴展核心功能。這些程式庫通常是其他 FreeRTOS 程式庫的依存項目。

主題

- [不可部分完成操作 \(p. 178\)](#)
- [Linear Containers 程式庫 \(p. 178\)](#)
- [記錄程式庫 \(p. 179\)](#)
- [Static Memory 程式庫 \(p. 179\)](#)
- [任務集區程式庫 \(p. 179\)](#)

不可部分完成操作

Overview

[不可部分完成操作](#)可確保並行程式設計為非封鎖同步。當共用記憶體上的非同步操作導致效能問題時，您就能使用不可部分完成操作加以解決。FreeRTOS 可支援 `iot_atomic.h` 標頭檔案中實作的不可部分完成操作。

`iot_atomic.h` 標頭檔案包含兩種不可部分完成操作的實作：

- 全域停用中斷點。
此實作適用於所有 FreeRTOS 平台。
- ISA 原生不可部分完成操作支援。

此實作僅適用以 GCC 4.7.0 版和更新版本編譯，且具備 ISA 不可部分完成操作支援的平台。如需 GCC 內建函數的相關資訊，請參閱 [Built-in Functions for Memory Model Aware Atomic Operations](#) (記憶體模型感知不可部分完成操作的內建函數)。

Initialization

使用 FreeRTOS 不可部分完成操作前，您需要選擇要使用的不可部分完成操作實作。

1. 開啟 [FreeRTOSConfig.h \(p. 8\)](#) 組態檔案以進行編輯。
2. 如果選擇 ISA 原生不可部分完成操作支援實作，請定義 `configUSE_ATOMIC_INSTRUCTION` 變數，並將其設為 1。

如果選擇全域停用中斷點實作，則請取消 `configUSE_ATOMIC_INSTRUCTION` 的定義或予以清除。

API 參考

如需完整的 API 參考，請參閱 [Atomic Operations C SDK API 參考](#)。

Linear Containers 程式庫

Linear Containers library 定義了在開發嵌入式應用程式時要使用的資料結構，包括清單和佇列。FreeRTOS API 參考

如需完整的 API 參考，請參閱 [Linear Containers C SDK API 參考](#)。

記錄程式庫

LoggingFreeRTOS 程式庫可讓 程式庫將訊息列印到 log 以進行偵錯。FreeRTOS API 參考

如需完整的 API 參考，請查看 [Logging C SDK API 參考](#)。

Static Memory 程式庫

FreeRTOS Static Memory (靜態記憶體) 程式庫可定義部分用於管理靜態緩衝區的函數。這個程式庫能讓您使用 Static Memory 元件來提供靜態配置緩衝區，而無需使用動態記憶體配置。

API 參考

如需完整的 API 參考，請參閱 [Static Memory C SDK API 參考](#)。

任務集區程式庫

Overview

FreeRTOS 透過 FreeRTOSTask Pool (任務集區) 程式庫支援任務管理。任務集區程式庫可讓您排程背景任務，也可進行或取消安全、非同步任務的排程。使用任務集區 APIs，您可以設定應用程式的任務，以最佳化效能和記憶體使用量之間的權衡取捨。

任務集區程式庫建置在兩個主要資料結構上：任務集區和任務集區任務。

任務集區 (`IotTaskPool_t`)

任務集區包含了分派佇列，可管理要執行的工作佇列，以及執行工作的工作者執行續。

任務集區工作 (`IotTaskPoolJob_t`)

任務集區工作可做為背景工作或定時的背景工作來執行。背景工作是以先進先出的順序開始進行的，且沒有時間限制。定時工作是根據計時器來排定以在背景執行。

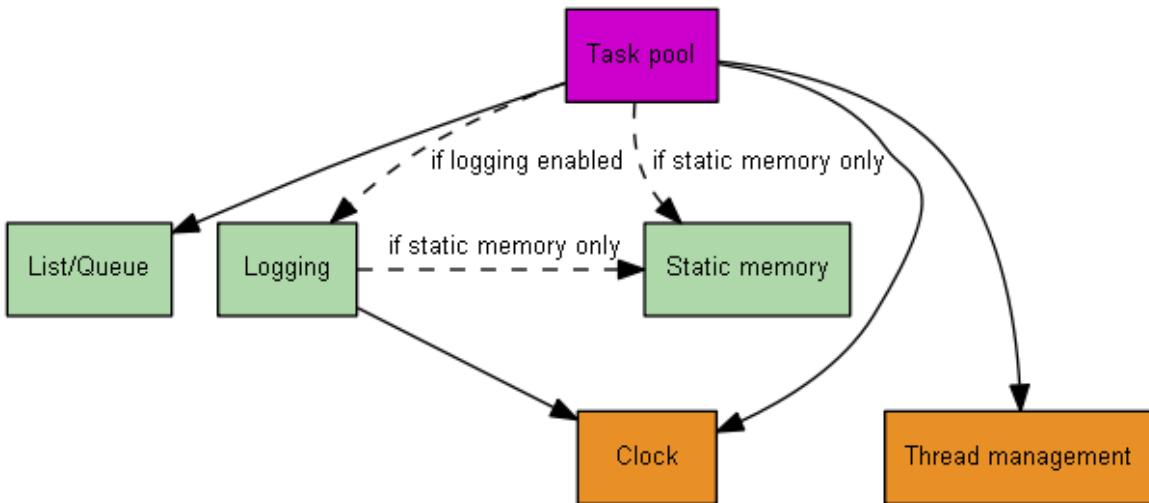
Note

任務集區只能保證定時工作會在逾時時間之後執行，而不會在特定時段之內的時間執行。

相依性和要求

任務集區程式庫有如下的相依性：

- 線性容器 (清單/佇列) 程式庫，是用於維護已排程和正在進行的任務集區操作的資料結構。
- 記錄程式庫 (如果 `IOT_LOG_LEVEL_TASKPOOL` 組態設定不是 `IOT_LOG_NONE`)。
- 向作業系統提供界面的平台層，適用於執行緒管理、計時器、時鐘函數等。



Features

使用任務集區程式庫APIs，您可以執行下列作業：

- 使用程式庫的非封鎖 API 函式來排程立即和延遲工作。
- 建立靜態和動態分配工作。
- 依您的系統資源設定程式庫設定以擴展效能和使用量。
- 在動態建立工作時自訂低記憶體負擔的快取。

Troubleshooting

任務集區程式庫函式會以 `IotTaskPoolError_t` 列舉值傳回錯誤代碼。如需每個錯誤碼的詳細資訊，請查看 `IotTaskPoolError_t` 任務集區 C 開發套件 API 參考中 列舉資料類型的參考文件。

使用限制

任務集區程式庫無法由中斷服務常式 (ISR) 使用。

我們強烈建議執行阻擋式操作 (尤其是無限期阻擋式操作) 的任務集區使用者不要使用回呼。長期的阻擋式操作將有效地奪取任務集區執行緒，並造成潛在的死鎖或資源耗盡。

Initialization

應用程式需要呼叫 `IotTaskPool_CreateSystemTaskPool`，以在使用任務集區之前初始化系統任務集區的值行個體。應用程式需要確保任何程式庫使用任務集區之前，以及在任何應用程式碼將工作發佈到任務集區之前，在啟動序列中儘早初始化系統層級的任務集區。啟動之後不久，系統將隨即初始化單一、系統層級的任務集區，以供所有程式庫進行共享。在初始化後，便可擷取任務集區處理常式以和 `IOT_SYSTEM_TASKPOOL` API 一起使用。

Note

呼叫 `IotTaskPool_CreateSystemTaskPool` 不會分配記憶體來保留任務集區資料結構和狀態，但可能分配記憶體來保留相依實體和資料結構 (例如任務集區執行緒)。

API 參考

如需完整的 API 參考，請參閱 [任務集區 C SDK API 參考](#)。

範例使用方式

假設您需要排程重複的 AWS IoT Device Defender 指標集合，而您想要使用計時器來排程對 MQTT 連接、訂用和發布 APIs 的呼叫集合。下列程式碼定義回呼函數，用來在 MQTT 間接受 AWS IoT Device Defender 指標，與 MQTT 連接中斷連接的中斷回呼。

```
/* An example of a user context to pass to a callback through a task pool thread. */
typedef struct JobUserContext
{
    uint32_t counter;
} JobUserContext_t;

/* An example of a user callback to invoke through a task pool thread. */
static void ExecutionCb( IotTaskPool_t * pTaskPool, IotTaskPoolJob_t * pJob, void * context )
{
    ( void )pTaskPool;
    ( void )pJob;
    JobUserContext_t * pUserContext = ( JobUserContext_t * )context;
    pUserContext->counter++;
}

void TaskPoolExample( )
{
    JobUserContext_t userContext = { 0 };
    IotTaskPoolJob_t job;
    IotTaskPool_t * pTaskPool;
    IotTaskPoolError_t errorSchedule;

    /* Configure the task pool to hold at least two threads and three at the maximum. */
    /* Provide proper stack size and priority per the application needs. */
    const IotTaskPoolInfo_t tpInfo = { .minThreads = 2, .maxThreads = 3, .stackSize =
512, .priority = 0 };

    /* Create a task pool. */
    IotTaskPool_Create( &tpInfo, &pTaskPool );

    /* Statically allocate one job, then schedule it. */
    IotTaskPool_CreateJob( &ExecutionCb, &userContext, &job );
    errorSchedule = IotTaskPool_Schedule( pTaskPool, &job, 0 );

    switch ( errorSchedule )
    {
        case IOT_TASKPOOL_SUCCESS:
            break;
        case IOT_TASKPOOL_BAD_PARAMETER:           // Invalid parameters, such as a NULL handle,
can trigger this error.
        case IOT_TASKPOOL_ILLEGAL_OPERATION:       // Scheduling a job that was previously
scheduled or destroyed could trigger this error.
        case IOT_TASKPOOL_NO_MEMORY:               // Scheduling a with flag
#IOT_TASKPOOL_JOB_HIGH_PRIORITY could trigger this error.
        case IOT_TASKPOOL_SHUTDOWN_IN_PROGRESS:   // Scheduling a job after trying to destroy
the task pool could trigger this error.
            // ASSERT
            break;
        default:
            // ASSERT*/
    }

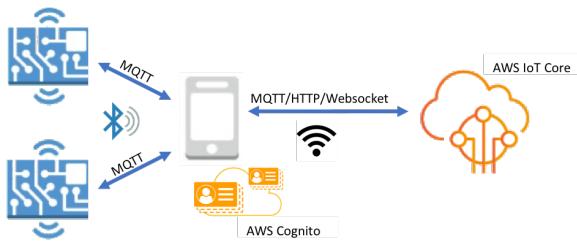
    /* ..... */
    /* ... Perform other operations ... */
    /* ..... */

    IotTaskPool_Destroy( pTaskPool );
}
```

低功耗藍牙程式庫

Overview

FreeRTOS 支援透過代理裝置 (例如行動電話) 經由低功耗藍牙發佈和訂閱 MQTT 主題。使用 FreeRTOS Bluetous Low Energy (低功耗藍牙) 程式庫，您的微型控制器可以安全地與 AWS IoT MQTT 中介裝置通訊。



使用 SDKs 藍牙裝置的 Mobile FreeRTOS，您可以編寫原生行動應用程式，以透過低功耗藍牙與微型控制器上的嵌入式應用程式通訊。如需 SDKs 行動裝置的詳細資訊，請前往 [藍牙裝置適用的 SDKs 行動 FreeRTOS \(p. 191\)](#)。

FreeRTOS 低功耗藍牙程式庫包含設定 Wi-Fi 網路、傳送大量資料，以及透過低功耗藍牙提供網路抽象層的服務。低功耗藍牙程式庫也包含中介軟體和更低階的 FreeRTOS，以更能直接控制您的低功耗藍牙堆疊。APIs

Architecture

FreeRTOS 低功耗藍牙程式庫是由三個層組成：服務、中介軟體和低階包裝函式。

Services

低功耗藍牙服務層包含四個利用中介軟體的通用屬性 (GATT) 服務：FreeRTOS APIs 裝置資訊、Wi-Fi Provisioning、Network Abstraction 和 Large Object Transfer。

裝置資訊

裝置資訊服務會收集微型控制器的相關資訊，包括：

- 您的裝置正在使用的 FreeRTOS 版本。
- 用於註冊裝置之帳戶的 AWS IoT 端點。
- 低功耗藍牙最大傳輸單位 (MTU)。

Wi-Fi 佈建

Wi-Fi 佈建服務可讓微型控制器具有 Wi-Fi 功能，以執行下列動作：

- 列出範圍內的網路。
- 將網路和網路登入資料儲存至快閃記憶體。
- 設定網路優先順序。

- 從快閃記憶體中刪除網路和網路登入資料。

網路抽象

網路抽象服務可為應用程式擷取網路連線類型。常見的 API 會與您裝置的 Wi-Fi、乙太網路和低功耗藍牙硬體堆疊進行互動，讓應用程式相容於多種連線類型。

大型物件傳輸

大型物件傳輸服務可為用戶端傳送及接收資料。其他例如 Wi-Fi 佈建和網路抽象等服務，也會使用大型物件傳輸服務來傳送和接收資料。您也可以使用大型物件傳輸 API 直接與服務互動。

Middleware

FreeRTOS 低功耗藍牙中介軟體是一種來自較低層級 APIs 的抽象。中介軟體 APIs 組成了與低功耗藍牙堆疊更方便使用者使用的界面。

使用中介軟體 APIs，您可以對多個 layer 將多個回呼登記至單一事件。初始化低功耗藍牙中介軟體也會初始化服務，並開始進行廣告。

彈性回呼訂閱

假設您的低功耗藍牙硬體中斷連線，而且透過 MQTT 的低功耗藍牙服務需要偵測連線中斷。您撰寫的應用程式也可能需要偵測相同的連線中斷事件。低功耗藍牙中介軟體可以將事件路由到程式碼的不同部分，而您已在其中已註冊回呼，這使得更高層級無需爭用更低階的資源。

低階包裝函式

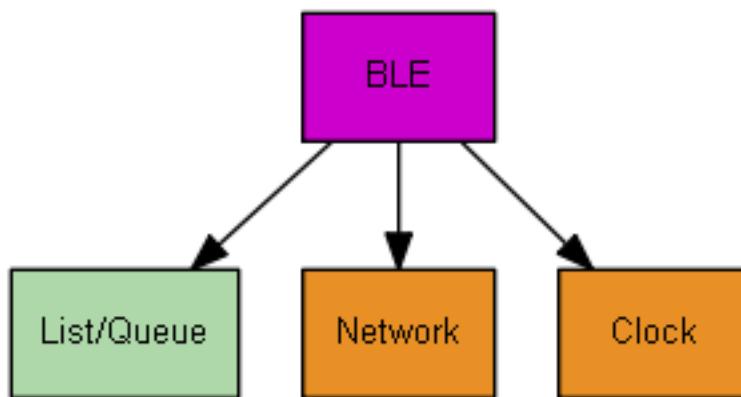
低階 FreeRTOS 低功耗藍牙包裝函式是一種製造商的低功耗藍牙堆疊抽象。低階包裝函式提供一組常用的 APIs 以直接控制硬體。低階 APIs 會最佳化 RAM 用量，但限於功能方面。

使用低功耗藍牙服務 APIs 來與低功耗藍牙服務進行互動。服務需要的資源比低階 APIs 更多。APIs

相依性和要求

低功耗藍牙程式庫有以下直接相依性：

- [Linear Containers 程式庫 \(p. 178\)](#)
- 與作業系統接觸的平台層，適用於執行緒管理、計時器、時鐘函數和網路存取。



只有 Wi-Fi 佈建服務有 FreeRTOS 程式庫相依性：

GATT 服務	相依性
Wi-Fi 佈建	Wi-Fi 程式庫 (p. 205)

若要與 AWS IoT MQTT 中介裝置通訊，您必須具有一個 AWS 帳戶，而且必須將您的裝置註冊為 AWS IoT 物件。如需設定的詳細資訊，請參閱 [AWS IoT 開發人員指南](#)。

FreeRTOS 低功耗藍牙會在您的行動裝置上使用 Amazon Cognito 進行使用者驗證。若要使用 MQTT 代理服務，您必須建立一個 Amazon Cognito 身分和使用者集區。每個 Amazon Cognito 身分須有適當的政策連接至其中。如需詳細資訊，請參閱 [Amazon Cognito 開發人員指南](#)。

程式庫組態檔案

應用程式若使用透過低功耗藍牙服務的 FreeRTOS MQTT，則必須提供 `iot_ble_config.h` 標頭檔案，其中即為組態參數的定義所在。未定義的組態參數會採用 `iot_ble_config_defaults.h` 中指定的預設值。

有些重要的組態參數包括：

`IOT_BLE_ADD_CUSTOM_SERVICES`

允許使用者建立自己的服務。

`IOT_BLE_SET_CUSTOM_ADVERTISEMENT_MSG`

允許使用者自訂廣告和掃描回應訊息。

如需詳細資訊，請參閱 [低功耗藍牙 API 參考](#)。

Optimization

當最佳化主機板的效能時，請考慮以下情況：

- 低階 APIs 使用較少的 RAM，但提供的功能有限。
- 您可以將 `iot_ble_config.h` 標頭檔案中的 `bleconfigMAX_NETWORK` 參數設定為較低的值，以減少堆疊的消耗量。
- 您可以將 MTU 大小增加至其最大值，以限制訊息緩衝，並使程式碼執行更快且耗用更少的 RAM。

使用限制

根據預設，FreeRTOS 低功耗藍牙程式庫會將 `eBTpropertySecureConnectionOnly` 屬性設定為 `TRUE`，表示將裝置置於僅限安全連線模式下。如同 [Bluetooth Core Specification](#) 5.0 版第 3 冊 C 篇 10.2.4 中所指定，當裝置處於僅限安全連線模式時，需有最高 LE 安全模式 1 層級 4，才能存取其具有的許可高於最低 LE 安全模式 1 層級 1 的任何屬性。在 LE 安全模式 1 層級 4 中，裝置必須具有輸入和輸出功能，才能進行數字比較。

以下是支援的模式，及其關聯的屬性：

模式 1、第 1 級 (不安全)

```
/* Disable numeric comparison */
#define IOT_BLE_ENABLE_NUMERIC_COMPARISON ( 0 )
```

```
#define IOT_BLE_ENABLE_SECURE_CONNECTION      ( 0 )
#define IOT_BLE_INPUT_OUTPUT                  ( eBTIONone )
#define IOT_BLE_ENCRYPTION_REQUIRED           ( 0 )
```

模式 1、第 2 級 (未驗證配對與加密)

```
#define IOT_BLE_ENABLE_NUMERIC_COMPARISON    ( 0 )
#define IOT_BLE_ENABLE_SECURE_CONNECTION        ( 0 )
#define IOT_BLE_INPUT_OUTPUT                  ( eBTIONone )
```

模式 1、第 2 級 (已驗證配對與加密)

不支援此模式。

模式 1、第 4 級 (已驗證的 LE 安全連線配對與加密)

依預設支援此模式。

如需 LE 安全模式的詳細資訊，請參閱 [Bluetooth Core Specification 5.0 版第 3 冊 C 篇 10.2.1](#)。

Initialization

如果您的應用程式透過中介軟體與低功耗藍牙堆疊互動，則您只需要初始化中介軟體。中介軟體負責初始化更低階的堆疊。

Middleware

初始化中介軟體

1. 在呼叫低功耗藍牙中介軟體 API 之前，請初始化任何低功耗藍牙硬體驅動程式。
2. 啟用低功耗藍牙。
3. 以 `IotBLE_Init()` 初始化中介軟體。

Note

如果您正在執行 AWS 示範，則您不需要執行此初始化步驟。示範初始化是由 `freertos/demos/network_manager` 中的網路管理員進行處理。

低階 APIs

如果您不想使用 FreeRTOS 低功耗藍牙 GATT 服務，則可以略過中介軟體，並直接與低階 APIs 互動，以節省資源。

初始化低階 APIs

1. 在呼叫 APIs 之前初始化任何低功耗藍牙硬體驅動程式。驅動程式初始化不屬於低功耗藍牙低階 APIs。
2. 低功耗藍牙低階 API 提供啟用/停用呼叫低功耗藍牙堆疊的功能，以最佳化電源和資源。呼叫 APIs 之前，您必須支援低功耗藍牙。

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

3. 藍牙管理員包含低功耗藍牙和藍牙傳統常用的 APIs。接著必須初始化常用管理員的回呼。

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

4. 低功耗藍牙轉接器放在常用 API 之上。您必須初始化其回呼，如同您初始化常用 API 一般。

```
xBTInterface.pxBTLeAdapterInterface = ( BTBleAdapter_t * ) xBTInterface.pxBTInterface->pxGetLeAdapter();
xStatus = xBTInterface.pxBTLeAdapterInterface->pxBleAdapterInit( &xBTBleAdapterCb );
```

5. 註冊新的使用者應用程式。

```
xBTInterface.pxBTLeAdapterInterface->pxRegisterBleApp( pxAppUuid );
```

6. 初始化 GATT 伺服器的回呼。

```
xBTInterface.pxGattServerInterface = ( BTGattServerInterface_t * )
xBTInterface.pxBTLeAdapterInterface->ppvGetGattServerInterface();
xBTInterface.pxGattServerInterface->pxGattServerInit( &xBTGattServerCb );
```

在初始化低功耗藍牙轉接器之後，您可以新增 GATT 伺服器。您一次只能註冊一部 GATT 伺服器。

```
xStatus = xBTInterface.pxGattServerInterface->pxRegisterServer( pxAppUuid );
```

7. 設定應用程式屬性，像是僅限安全連線和 MTU 大小。

```
xStatus = xBTInterface.pxBTInterface->pxSetDeviceProperty( &pxProperty[ usIndex ] );
```

API 參考

如需完整的 API 參考，請參閱[低功耗藍牙 API 參考](#)。

範例使用方式

以下範例示範如何使用低功耗藍牙程式庫進行廣告並建立新的服務。如需完整的 FreeRTOS 低功耗藍牙示範應用程式，請參閱[低功耗藍牙示範應用程式](#)。

Advertising

1. 在您的應用程式中設定廣告 UUID：

```
static const BTUuid_t _advUUID =
{
    .uu.uu128 = IOT_BLE_ADVERTISING_UUID,
    .ucType   = eBtUuidType128
};
```

2. 然後定義 IotBle_SetCustomAdvCb 回呼函式：

```
void IotBle_SetCustomAdvCb( IotBleAdvertisementParams_t * pAdvParams,
                            IotBleAdvertisementParams_t * pScanParams)
{
```

```
memset(pAdvParams, 0, sizeof(IotBleAdvertisementParams_t));
memset(pScanParams, 0, sizeof(IotBleAdvertisementParams_t));

/* Set advertisement message */
pAdvParams->pUUID1 = &_advUUID;
pAdvParams->nameType = BTGattAdvNameNone;

/* This is the scan response, set it back to true. */
pScanParams->setScanRsp = true;
pScanParams->nameType = BTGattAdvNameComplete;
}
```

此回呼會在廣告訊息中傳送 UUID 訊息，並在掃描回應中傳送完整名稱。

- 開放 `vendors/vendor/boards/board/aws_demos/config_files/iot_ble_config.h`，並將 `IOT_BLE_SET_CUSTOMADVERTISEMENT_MSG` 設定為 1。這會觸發 `IotBle_SetCustomAdvCb` 回呼。

新增服務

如需完整服務範例，請參閱 [freertos/.../ble/services](#)。

- 為服務的特性和描述項建立 UUIDs：

```
#define xServiceUUID_TYPE \
{\
    .uu.uu128 = gattDemoSVC_UUID, \
    .ucType   = eBTuuidType128 \
}
#define xCharCounterUUID_TYPE \
{\
    .uu.uu128 = gattDemoCHAR_COUNTER_UUID, \
    .ucType   = eBTuuidType128 \
}
#define xCharControlUUID_TYPE \
{\
    .uu.uu128 = gattDemoCHAR_CONTROL_UUID, \
    .ucType   = eBTuuidType128 \
}
#define xClientCharCfgUUID_TYPE \
{\
    .uu.uu16 = gattDemoCLIENT_CHAR_CFG_UUID, \
    .ucType   = eBTuuidType16 \
}
```

- 建立緩衝以註冊特性和描述項的處理常式：

```
static uint16_t usHandlesBuffer[egattDemoNbAttributes];
```

- 建立屬性表格。若要節省一些 RAM，請將表格定義為 const。

Important

請一律按順序建立屬性，並將服務做為第一個屬性。

```
static const BTAttribute_t pxAttributeTable[] = {
{
    .xServiceUUID = xServiceUUID_TYPE
},
{
    .xAttributeType = eBTDbCharacteristic,
```

```

.xCharacteristic =
{
    .xUuid = xCharCounterUUID_TYPE,
    .xPermissions = ( IOT_BLE_CHAR_READ_PERM ),
    .xProperties = ( eBTPropRead | eBTPropNotify )
},
{
    .xAttributeType = eBTDbDescriptor,
    .xCharacteristicDescr =
    {
        .xUuid = xClientCharCfgUUID_TYPE,
        .xPermissions = ( IOT_BLE_CHAR_READ_PERM | IOT_BLE_CHAR_WRITE_PERM )
    }
},
{
    .xAttributeType = eBTDbCharacteristic,
    .xCharacteristic =
    {
        .xUuid = xCharControlUUID_TYPE,
        .xPermissions = ( IOT_BLE_CHAR_READ_PERM | IOT_BLE_CHAR_WRITE_PERM ),
        .xProperties = ( eBTPropRead | eBTPropWrite )
    }
}
);

```

4. 建立一系列的回呼。此回呼陣列必須遵循上述表格陣列所定義的相同順序。

例如，如果存取 xCharCounterUUID_TYPE 時 vReadCounter 被觸發了，而存取 xCharControlUUID_TYPE 時 vWriteCommand 被觸發了，則依下列內容定義陣列：

```

static const IotBleAttributeEventCallback_t pxCallBackArray[egattDemoNbAttributes] =
{
    NULL,
    vReadCounter,
    vEnableNotification,
    vWriteCommand
};

```

5. 建立服務：

```

static const BTService_t xGattDemoService =
{
    .xNumberOfAttributes = egattDemoNbAttributes,
    .ucInstId = 0,
    .xType = eBTServiceTypePrimary,
    .pusHandlesBuffer = usHandlesBuffer,
    .pxBLEAttributes = (BTAttribute_t *)pxAttributeTable
};

```

6. 使用您在上一步驟建立的結構來呼叫 API IotBle_CreateService。中介軟體會同步所有服務的建立，所以在 IotBle_AddCustomServicesCb 回呼觸發時，任何新的服務都必須已完成定義。

- 在 IOT_BLE_ADD_CUSTOM_SERVICES 中將 1 設為 vendors/*vendor*/boards/*board*/aws_demos/config_files/iot_ble_config.h。
- 在您的應用程式中建立 IotBle_AddCustomServicesCb：

```

void IotBle_AddCustomServicesCb(void)
{
    BTStatus_t xStatus;
    /* Select the handle buffer. */
    xStatus = IotBle_CreateService( (BTService_t *)&xGattDemoService,
                                    (IotBleAttributeEventCallback_t *)pxCallBackArray );

```

}

Porting

使用者輸入和輸出周邊

安全連線需要輸入和輸出兩者，才能進行數字比較。您可以使用事件管理員註冊 eBLENumericComparisonCallback 事件：

```
xEventCb.pxNumericComparisonCb = &prvNumericComparisonCb;  
xStatus = BLE_RegisterEventCb( eBLENumericComparisonCallback, xEventCb );
```

周邊必須顯示數字密碼，並採取比較結果做為輸入。

移植 API 實作

若要將 FreeRTOS 移植到新的目標，您必須實作一些 APIs 來使用 Wi-Fi Provisioning 服務和低功耗藍牙功能。

低功耗藍牙APIs

若要使用 FreeRTOS 低功耗藍牙中介軟體，您必須實作一些 APIs。

GAP for Bluetooth Classic 和 GAP for Bluetooth Low Energy 之間常用的APIs

- pxBtManagerInit
- pxEnable
- pxDisable
- pxGetDeviceProperty
- pxSetDeviceProperty (除 eBTpropertyRemoteRssi 和 eBTpropertyRemoteVersionInfo 以外的所有選項都是必要選項)
- pxPair
- pxRemoveBond
- pxGetConnectionState
- pxPinReply
- pxSspReply
- pxGetTxpower
- pxGetLeAdapter
- pxDeviceStateChangedCb
- pxAdapterPropertiesCb
- pxSspRequestCb
- pxPairingStateChangedCb
- pxTxPowerCb

APIs 專用於低功耗藍牙的 GAP

- pxRegisterBleApp
- pxUnregisterBleApp
- pxBleAdapterInit

- pxStartAdv
- pxStopAdv
- pxSetAdvData
- pxConnParameterUpdateRequest
- pxRegisterBleAdapterCb
- pxAdvStartCb
- pxSetAdvDataCb
- pxConnParameterUpdateRequestCb
- pxCongestionCb

GATT 伺服器

- 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

如需將 FreeRTOS 低功耗藍牙程式庫移植到您的平台的詳細資訊，請見 [移植指南中的 移植低功耗藍牙程式庫FreeRTOS。](#)

藍牙裝置適用的 SDKs 行動FreeRTOS

您可以使用 SDKs 藍牙裝置的 Mobile FreeRTOS 來建立行動應用程式，其透過低功耗藍牙與您的微型控制器互動。行動 SDKs 也可使用 AWS 進行使用者身份驗證與 Amazon Cognito 服務通訊。

FreeRTOS 藍牙裝置的 Android 軟體開發套件

使用 FreeRTOS 藍牙裝置的 Android 行動軟體開發套件來建置 Android 行動應用程式，該應用程式透過低功耗藍牙與您的微型控制器互動。您可以在 [GitHub 取得此開發套件](#)。

若要安裝適用於 FreeRTOS 藍牙裝置的 Android 軟體開發套件，請遵循專案 [README.md](#) 檔案中的「設定軟體開發套件」指示。

如需設定和執行軟體開發套件隨附之示範行動應用程式的詳細資訊，請參閱[Prerequisites \(p. 211\)](#)和[FreeRTOS 低功耗藍牙 Mobile SDK 示範應用程式 \(p. 213\)](#)。

FreeRTOS 藍牙裝置的 iOS 軟體開發套件

使用 FreeRTOS 藍牙裝置的 iOS 行動軟體開發套件來建置 iOS 行動應用程式，該應用程式透過低功耗藍牙與您的微型控制器互動。您可以在 [GitHub 取得此開發套件](#)。

安裝 iOS 軟體開發套件

1. 安裝 CocoaPods：

```
$ gem install cocoapods  
$ pod setup
```

Note

您可能需要使用 `sudo` 來安裝 CocoaPods。

2. 使用 CocoaPods 安裝軟體開發套件（將其新增到您的 `podfile`）：

```
$ pod 'FreeRTOS', :git => 'https://github.com/aws/amazon-freertos-ble-ios-sdk.git'
```

如需設定和執行軟體開發套件隨附之示範行動應用程式的詳細資訊，請參閱[Prerequisites \(p. 211\)](#)和[FreeRTOS 低功耗藍牙 Mobile SDK 示範應用程式 \(p. 213\)](#)。

通用 I/O

概觀

通常，裝置驅動程式與基礎作業系統無關，而且是特定於指定的硬體組態。硬體抽象層 (HAL) 可提供驅動程式與更高階應用程式碼之間的通用介面。HAL 抽取了特定驅動程式運作方式的詳細資訊，並提供統一的 API 來控制這類裝置。您可以使用相同的 APIs，跨多個以微型控制器 (MCU) 為基礎的參考電路板存取各種裝置驅動程式。

FreeRTOS [通用 I/O](#) 作為此硬體抽象層。它提供了一組標準 APIs，用於在支援的參考電路板上存取通用序列裝置。這些通用 APIs 會與這些周邊裝置進行通訊和互動，並讓您的程式碼可跨平台運作。若沒有通用 I/O，編寫程式碼來使用低階裝置是矽晶廠商所特定的。

支援的周邊裝置

- UART

- SPI
- I2C

支援的功能

- 同步讀取/寫入 – 此功能在傳輸請求的資料量之前不會傳回資料。
- 非同步讀取/寫入 – 此功能會立即傳回資料，而且資料傳輸會非同步發生。當動作完成時，即會叫用已註冊的用戶回呼。

周邊裝置特定

- I2C – 將多個操作合併成一個交易。用來在一個交易上執行先寫入後讀取動作。
- SPI – 在主要裝置與次要裝置之間傳輸資料，這表示寫入和讀取同時發生。

移植

如需詳細資訊，請參閱 [FreeRTOS 移植指南](#)。

AWS IoT Device Defender 程式庫

Introduction

您可以使用 AWS IoT Device Defender 程式庫，從您的 IoT 裝置傳送安全指標至 AWS IoT Device Defender。您可以使用 AWS IoT Device Defender 從裝置持續監控這些安全指標，偏離您為每個裝置定義為適當行為的行為。如果某些項目看起來不正確，AWS IoT Device Defender 會傳送提醒，讓您可以執行操作來修正問題。與 AWS IoT Device Defender 的互動使用 [MQTT](#)，這是輕量型發行訂用協定。此程式庫提供 API 來編寫和辨識 AWS IoT Device Defender 使用的 MQTT 主題字串。

如需更多詳細資訊，請參閱 AWS IoT 開發人員指南中的 [AWS IoT Device Defender](#)。

此程式庫是以 C 撰寫，並設計為符合 [ISO C90](#) 和 [MISRA C : 2012](#)。此程式庫與標準 C 程式庫以外的任何其他程式庫並無相依性。它還不具備任何平台相依性，例如執行緒或同步。它可與任何 MQTT 程式庫及任何 [JSON](#) 或 [CBOR](#) 程式庫搭配使用。此程式庫具有 [可確保](#) 顯示安全記憶體使用和無堆積配置，適合 IoT 微型控制器，但也可完全轉移至其他平台。

程式庫可以自由使用，並根據 AWS IoT Device Defender MIT 開放原始碼授權 [分發](#)。

AWS IoT Greengrass Discovery 程式庫

Overview

您的微型控制器裝置會使用 [Discovery AWS IoT Greengrass 程式庫](#)來探索您網路上的 Greengrass 核心。使用 AWS IoT Greengrass Discovery APIs，您的裝置可在找到核心的端點後傳送訊息到 Greengrass 核心。

相依性和要求

若要使用 Greengrass Discovery 程式庫，您必須在 AWS IoT 中建立物件，包含憑證與政策。如需詳細資訊，請參閱 [AWS IoT 入門](#)。

您必須在 `freertos/demos/include/aws_clientcredential.h` 檔案中設定以下常數的值：

`clientcredentialMQTT_BROKER_ENDPOINT`

您的 AWS IoT 端點。

`clientcredentialIOT_THING_NAME`

您的 IoT 實物名稱。

`clientcredentialWIFI_SSID`

您的 Wi-Fi 網路 SSID。

`clientcredentialWIFI_PASSWORD`

您的 Wi-Fi 密碼。

`clientcredentialWIFI_SECURITY`

您 Wi-Fi 網路所使用的安全類型。

您必須同時在 `freertos/demos/include/aws_clientcredential_keys.h` 檔案中設定以下常數的值：

`keyCLIENT_CERTIFICATE_PEM`

與您物件相關聯的憑證 PEM。

`keyCLIENT_PRIVATE_KEY_PEM`

與您物件相關聯的私有金鑰 PEM。

您必須在主控台中設定一個 Greengrass 群組及核心裝置。如需詳細資訊，請參閱 [AWS IoT Greengrass 入門](#)。

雖然 Greengrass 連接不需要 coreMQTT 程式庫，我們極力建議您進行安裝。在搜尋到之後，程式庫便可用來和 Greengrass 核心通訊。

API 參考

如需完整的 API 參考，請參閱 [Greengrass API 參考](#)。

範例使用方式

Greengrass 工作流程

MCU 裝置會透過從 AWS IoT 請求包含 Greengrass 核心連線能力參數的 JSON 檔案來初始化搜尋程序。有兩種從 JSON 檔案擷取 Greengrass 核心連線能力參數的方法：

- 自動選取會逐一查看 JSON 檔案中列出的所有 Greengrass 核心，並連線到第一個可用的核心。
- 手動選取會使用 `aws_ggd_config.h` 中的資訊來連線到指定的 Greengrass 核心。

如何使用 Greengrass API

所有 Greengrass API 的預設組態選項都定義在 `aws_ggd_config_defaults.h` 中。

若只有一個 Greengrass 核心，請呼叫 `GGD_GetGGCIPandCertificate` 來請求附帶 Greengrass 核心連線能力資訊的 JSON 檔案。當傳回 `GGD_GetGGCIPandCertificate` 時，`pcBuffer` 參數會包含 JSON 檔案的文字。`pxHostAddressData` 參數則包含您可以連線的 Greengrass 核心 IP 地址及連接埠。

如需更多自訂選項（例如動態配置憑證），您必須呼叫下列 APIs：

`GGD_JSONRequestStart`

向 AWS IoT 提出 HTTP GET 請求，初始化探索 Greengrass 核心的探索請求。`GD_SecureConnect_Send` 用以將請求傳送至 AWS IoT。

`GGD_JSONRequestGetSize`

從 HTTP 回應取得 JSON 檔案的大小。

`GGD_JSONRequestGetFile`

取得 JSON 物件字串。`GGD_JSONRequestGetSize` 和 `GGD_JSONRequestGetFile` 使用 `GGD_SecureConnect_Read` 從通訊端取得 JSON 資料。必須呼叫 `GGD_JSONRequestStart`、`GGD_SecureConnect_Send`、`GGD_JSONRequestGetSize` 以接收來自 AWS IoT 的 JSON 資料。

`GGD_GetIPandCertificateFromJSON`

從 JSON 資料擷取 IP 地址和 Greengrass 核心憑證。您可以將 `xAutoSelectFlag` 設定為 `True` 以打開自動選擇。自動選取可找出 FreeRTOS 裝置可以連接的第一個核心裝置。若要連線到 Greengrass 核心，請呼叫 `GGD_SecureConnect_Connect` 函數、傳遞 IP 地址、連接埠及核心裝置的憑證。若要使用手動選取，請設定 `HostParameters_t` 參數的以下欄位：

`pcGroupName`

核心所屬的 Greengrass 群組 ID。您可以使用 `aws greengrass list-groups` CLI 命令來尋找您 Greengrass 群組的 ID。

`pcCoreAddress`

您正在連線的 Greengrass 核心 ARN。

coreHTTP 程式庫

適用於小型 IoT 裝置（MCU 或小型 MPU）的 HTTP C Client Library

Introduction

程式庫是 coreHTTPHTTP/1.1 標準子集的使用者實作。HTTP 標準提供無狀態通訊協定，可在 TCP/IP 上執行，且通常用於分散式、協作的超文字資訊系統。

程式庫會實作 coreHTTPHTTP/1.1 通訊協定標準的子集。此程式庫已針對低記憶體使用量進行最佳化。此程式庫提供完全同步的 API，讓應用程式可以完全管理並行。它僅使用固定緩衝區，因此應用程式可以完全控制其記憶體配置策略。

此程式庫是以 C 撰寫，並設計為符合 ISO C90 和 MISRA C : 2012。程式庫的唯一相依性是 Node.js 的 `http-parser` 的標準 C 程式庫和 LTS 版本（v12.19.1）。此程式庫具有可確保顯示安全記憶體使用和無堆積配置，適合 IoT 微型控制器，但也可完全轉移至其他平台。

在 IoT 應用程式中使用 HTTP 連接時，我們建議您使用安全的傳輸界面，例如使用 TLS 通訊協定的傳輸界面，如 [HTTP TLS](#) 示範中所示。

此程式庫可以自由使用，並根據 [MIT 開放原始碼授權](#) 分發。

| Code Size of coreHTTP
| (example generated with [GCC for ARM Cortex-M](#)) |

File	With -O1 Optimisation	With -Os Optimisation
core-http_client.c	3.0K	2.4K
http_parser.c (third-party utility)	15.7K	13.0K
Total estimate	18.7K	15.4K

coreJSON 程式庫

Introduction

JSON (JavaScript 物件標記法) 是一種人類可讀取的資料序列化格式。它廣泛用於交換資料，例如與 [Device Shadow 服務](#) [AWS IoT](#) 搭配使用，屬於許多 的一部分，例如 APIs REST API。GitHubEcma 國際將 JSON 維護為標準。

此 coreJSON 程式庫提供一個剖析器，可支援金鑰查詢，同時嚴格執行 [ECMA-404 標準 JSON 資料交換語法](#)。此程式庫是以 C 撰寫，並設計為符合 ISO C90 和 MISTRAC : 2012。它具有可確保顯示安全記憶體使用和無堆積配置，適合 IoT 微控制器，但也可完全轉移至其他平台。

記憶體用量

程式庫使用 nent 堆疊在 JSON 文件中追蹤巢狀結構。coreJSON 在單一函數呼叫期間，堆疊會存在，不會保留。透過定義巨集 (JSON_MAX_DEPTH) 即可指定堆疊大小，其預設為 32 個層級。每個關卡都會消耗一個位元組。

Code Size of coreJSON (example generated with GCC for ARM Cortex-M)		
File	With -O1 Optimisation	With -Os Optimisation
core_json.c	2.5K	1.9K
Total estimate	2.5K	1.9K

coreMQTT 程式庫

Introduction

程式庫是 coreMQTTMQTT (訊息併列 Telemetry Transport) 標準的實作。MQTT 標準提供可在 TCP/IP 上執行的輕量型發行/訂名 (或 PubSub) 簡訊通訊協定，而且通常用於機器對機器 (M2M) 和物聯網 (IoT) 使用案例。

程式庫符合 coreMQTTMQTT 3.1.1 通訊協定標準。此程式庫已針對低記憶體使用量進行最佳化。此程式庫的設計包含不同的使用案例，範圍僅包括僅使用 QoS 0 MQTT PUBLISH 訊息的資源限制平台，到使用 QoS 2 MQTT PUBLISH over TLS (Transport Layer Security) 連接的資源豐富的平台。此程式庫提供了可組合功能的選單，選擇和結合這些功能表可以完全符合特定使用案例的需求。

此程式庫是以 C 編寫，並設計為符合 ISO C90 和 MISRA C : 2012。此 MQTT 程式庫與任何其他程式庫並無相依性，除了：

- 標準 C 程式庫
- 消費者實作的網路傳輸界面
- (選用) 使用者實作的平台時間函數

透過提供簡單的傳送和接收傳輸界面規格，程式庫與基礎網路驅動程式分離。應用程式撰寫者可以選取現有的傳輸界面，或實作自己的適當應用程式。

此程式庫提供高階 API 來連接到 MQTT 中介裝置、訂用/取消訂用到主題、將訊息推送到主題，以及接收傳入的訊息。此 API 會以參數方式取得上述的傳輸界面，並使用該界面來傳送和接收與 MQTT 中介裝置之間的訊息。

此程式庫也會公開低階序列化程式/還原序列化程式 API。此 API 可用來建置簡單的 IoT 應用程式，其中包含只需要 MQTT 功能的子集，無需任何其他額外負荷。序列化程式/還原序列化程式 API 可與任何可用的傳輸層 API (如通訊端) 搭配使用，以傳送和接收與中介之間的訊息。

在 IoT 應用程式中使用 MQTT 連接時，我們建議您使用安全的傳輸界面，例如使用 TLS 通訊協定的傳輸界面。

此 MQTT 程式庫無平台相依性，例如執行緒或同步。此程式庫確實具有可示範安全記憶體使用和無堆積配置的可確保其適用於 IoT 微型控制器，但也可完全轉移至其他平台。它可以自由使用，並根據 MIT 開放原始碼授權分發。

Code Size of coreMQTT (example generated with GCC for ARM Cortex-M)		
File	With -O1 Optimisation	With -Os Optimisation
core_mqtt.c	3.0K	2.6K
core_mqtt_state.c	1.4K	1.1K
core_mqtt_serializer.c	2.5K	2.0K
Total estimate	6.9K	5.7K

OTA 代理程式程式庫

Overview

Over-The-Air (OTA) Agent (遠端 (OTA) 代理程式) 可讓您使用 HTTP 或 MQTT 做為通訊協定來管理 裝置的通知、下載和韌體更新驗證。FreeRTOS 透過使用 OTA 代理程式程式庫，您可以邏輯式的區分韌體更新及在您裝置上執行的應用程式。OTA 代理程式可和應用程式共享網路連線。透過共享網路連線，您可能可以節省大量的 RAM。此外，OTA 代理程式程式庫可讓您定義應用程式特定的邏輯，用於測試、遞交或轉返韌體更新。

如需使用 FreeRTOS 設定 OAT 更新的詳細資訊，請參閱 [FreeRTOS 無線更新 \(p. 120\)](#)。

Features

以下是完整的 OTA 代理程式界面：

OTA_AgentInit

初始化 OTA 代理程式。發起人會提供訊息簡訊通訊協定內容、選擇性的回撥，以及逾時。

OTA_AgentShutdown

在使用 OTA 代理程式之後清理資源。

OTA_GetAgentState

取得 OTA 代理程式的目前狀態。

OTA_ActivateNewImage

啟用透過 OTA 接收到的最新微控制器韌體映像。(詳細任務狀態現在應該會處於自我測試。)

OTA_SetImageState

設定目前執行中微控制器韌體映像的驗證狀態(測試中、已接受或已拒絕)。

OTA_GetImageState

取得目前執行中微控制器韌體映像的狀態(測試中、已接受或已拒絕)。

OTA_CheckForUpdate

從 OTA 更新服務請求下一個可用的 OTA 更新。

OTA_Suspend

暫停所有 OTA 代理程式操作。

OTA_Resume

繼續 OTA 代理程式操作。

API 參考

如需詳細資訊，請查看 [OTA 代理程式 API 參考](#)。

範例使用方式

典型的具 OTA 功能裝置應用程式會使用以下一系列 API 呼叫，使用 MQTT 通訊協定驅動程式。

1. 連線到 AWS IoT MQTT 經紀人。如需詳細資訊，請參閱 [coreMQTT 程式庫 \(p. 195\)](#)。
2. 透過呼叫 `OTA_AgentInit` 來初始化 OTA 代理程式。您的應用程式可以定義自訂 OTA 回撥函數，或透過指定 NULL 回撥函數指標來使用預設回撥。您也必須提供初始化逾時。
- 回撥會實作應用程式限定邏輯，在完成 OTA 更新任務後執行。逾時會定義等待初始化完成的時間長度。
3. 若 `OTA_AgentInit` 在代理程式準備就緒前逾時，您可以呼叫 `OTA_GetAgentState` 來確認代理程式已初始化完畢並照預期的方式運作。
4. OTA 更新完成時，FreeRTOS 會使用下列其中一個事件呼叫任務完成回呼：`accepted`、`rejected` 或 `self test`。
5. 若新的韌體映像遭到拒絕(例如因為發生驗證錯誤)，則應用程式通常可以忽略通知並等待下一次的更新。
6. 若更新有效並已標記為「已接受」，請呼叫 `OTA_ActivateNewImage` 來重設裝置並啟動新的韌體映像。

Porting

如需將 OTA 功能移植到您的平台，請前往 [移植指南中的移植 OTA 程式庫FreeRTOS](#)。

corePKCS11 程式庫

Overview

公有金鑰密碼編譯標準 #11 (PKCS#11) 是一種密碼編譯 API，可抽象化金鑰儲存、取得/設定密碼編譯物件屬性及工作階段語意。請參閱 FreeRTOS 來源碼儲存庫中的 `pkcs11.h` (從 OASIS 標準主體取得)。在 FreeRTOS 參考實作中，[PKCS#11 API](#) 呼叫是由 TLS 協助程式界面所發出，以便於在 `SOCKETS_Connect` 期間執行 TLS 使用者身份驗證。PKCS#11 API 呼叫也是由我們一次性的開發人員配置工作流程發出，以匯入 TLS client 憑證和私有金鑰，以便對 AWS IoT MQTT 中介裝置進行身份驗證。這兩個使用案例 (佈建及 TLS 用戶端身份驗證) 僅需要實作一小部分的 PKCS#11 界面標準。

Features

其會使用以下 PKCS#11 子集。此清單的順序大約是支援佈建、TLS 用戶端身份驗證及清理時呼叫常式的順序。如需函數的詳細說明，請參閱標準委員會提供的 PKCS#11 文件。

一般設定和卸除 API

- `C_Initialize`
- `C_Finalize`
- `C_GetFunctionList`
- `C_GetSlotList`
- `C_GetTokenInfo`
- `C_OpenSession`
- `C_CloseSession`
- `C_Login`

佈建 API

- `C_CreateObject CKO_PRIVATE_KEY` (適用於裝置私有金鑰)
- `C_CreateObject CKO_CERTIFICATE` (適用於裝置憑證和程式碼驗證憑證)
- `C_GenerateKeyPair`
- `C_DestroyObject`

用戶端身分驗證

- `C_GetAttributeValue`
- `C_FindObjectsInit`
- `C_FindObjects`
- `C_FindObjectsFinal`
- `C_GenerateRandom`
- `C_SignInit`
- `C_Sign`
- `C_VerifyInit`
- `C_Verify`
- `C_DigestInit`
- `C_DigestUpdate`

- C_DigestFinal

非對稱加密系統支援

FreeRTOS PKCS#11 參考實作支援 2048 位元的 RSA (僅簽署) 及使用 NIST P-256 曲線的 ECDSA。以下說明解釋如何建立基於 P-256 用戶端憑證的 AWS IoT 物件。

請確定您使用的以下版本 (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
```

下列程序假設您已使用 aws configure 命令來設定 AWS CLI。如需詳細資訊，請前往 <https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html#cli-configure-quickstart-config> 中的使用 aws configure 的快速組態AWS Command Line Interface 使用者指南。

根據 P-256 使用者憑證建立 AWS IoT 物件

1. 建立 AWS IoT 物件。

```
aws iot create-thing --thing-name thing-name
```

2. 使用 OpenSSL 建立 P-256 金鑰。

```
openssl genkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -pkeyopt
ec_param_enc:named_curve -outform PEM -out thing-name.key
```

3. 建立由步驟 2 所建立金鑰簽署的憑證註冊請求。

```
openssl req -new -nodes -days 365 -key thing-name.key -out thing-name.req
```

4. 將憑證註冊請求提交給 AWS IoT。

```
aws iot create-certificate-from-csr \
--certificate-signing-request file://thing-name.req --set-as-active \
--certificate-pem-outfile thing-name.crt
```

5. 將憑證 (由先前命令的 ARN 輸出參考) 連接到物件。

```
aws iot attach-thing-principal --thing-name thing-name \
--principal "arn:aws:iot:us-east-1:123456789012:cert/86e41339a6d1bbc67abf31faf455092cdebfb8f21ffbc67c4d238d1326c7de729"
```

6. 建立政策 (此政策太過寬鬆。應僅用於開發用途。)

```
aws iot create-policy --policy-name FullControl --policy-document file://policy.json
```

以下是 create-policy 命令中所指定 policy.json 檔案的清單。若您不希望執行 Greengrass 連線能力及搜索的 FreeRTOS 示範，您可以省略 greengrass:* 動作。

```
{  
    "Version": "2012-10-17",
```

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": "iot:/*",
    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Action": "greengrass:/*",
    "Resource": "*"
  }
]
```

- 將委託人 (憑證) 及政策連接至物件。

```
aws iot attach-principal-policy --policy-name FullControl \
--principal "arn:aws:iot:us-
east-1:123456789012:cert/
86e41339a6d1bbc67abf31faf455092cdebf8f21ffbc67c4d238d1326c7de729"
```

現在，遵循本指南中 [AWS IoT 入門](#)一節內的步驟。請記得將您建立的憑證及私有金鑰複製到您的 `aws_clientcredential_keys.h` 檔案。將您的物件名稱複製到 `aws_clientcredential.h`。

Note

將憑證和私有金鑰硬式編碼，僅作示範用途。生產層級應用程式必須將這些檔案存放在安全的位置。

Porting

如需將 `corePKCS11` 程式庫移植到您的平台之相關資訊，請見 [移植指南中的corePKCS11移植](#) 程式庫 FreeRTOS。

Secure Sockets 程式庫

Overview

You can use the FreeRTOS [Secure Sockets](#) library to create embedded applications that communicate securely. 此程式庫的設計旨在讓來自各種網路程式設計背景的軟體開發人員都能夠輕鬆上線。

FreeRTOS Secure Sockets 程式庫是以 Berkeley 通訊端界面為基礎，附有透過 TLS 通訊協定的額外安全通訊選項。For information about the differences between the FreeRTOS Secure Sockets library and the Berkeley sockets interface, see `SOCKETS_SetSockOpt` in the [Secure Sockets API Reference](#).

Note

Currently, only client APIs, plus a [lightweight IP \(lwIP\)](#) implementation of the server side `Bind` API, are supported for FreeRTOS Secure Sockets.

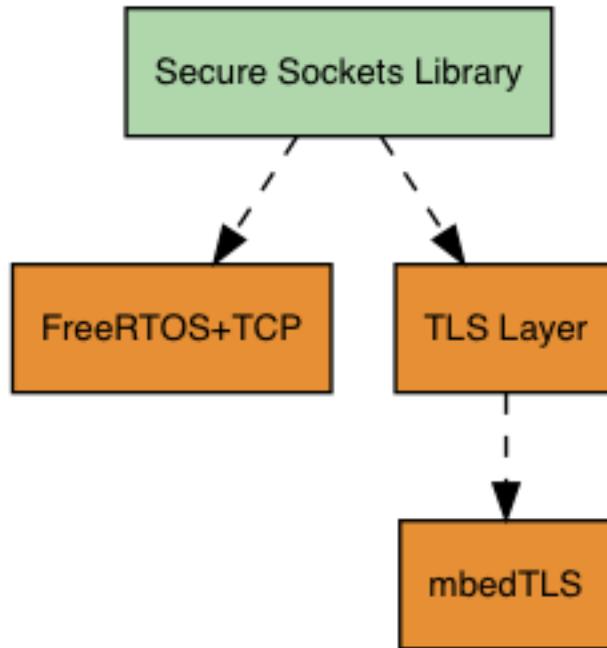
相依性和要求

FreeRTOS Secure Sockets 程式庫取決於 TCP/IP 堆疊和 TLS 實作。FreeRTOS 的連接埠以下列三種方式之一符合這些相依性：

- 同時自訂實作 TCP/IP 及 TLS

- A custom implementation of TCP/IP, and the FreeRTOS TLS layer with [mbedTLS](#)
- [FreeRTOS+TCP](#) and the FreeRTOS TLS layer with [mbedTLS](#)

The dependency diagram below shows the reference implementation included with the FreeRTOS Secure Sockets library. This reference implementation supports TLS and TCP/IP over Ethernet and Wi-Fi with FreeRTOS+TCP and mbedTLS as dependencies. 如需 FreeRTOS TLS 層的詳細資訊，請參閱 [Transport Layer Security \(p. 205\)](#)。



Features

FreeRTOS Secure Sockets 程式庫功能包括：

- 標準 Berkeley 通訊端型界面
- Thread-safe APIs for sending and receiving data
- 方便啟用 TLS

Troubleshooting

錯誤代碼

FreeRTOS Secure Sockets 程式庫傳回的錯誤碼為負值。如需每個錯誤碼的詳細資訊，請參閱 [Secure Sockets API 參考](#) 中的 Secure Sockets 錯誤碼。

Note

如果 FreeRTOS Secure Sockets API 傳回錯誤碼，則取決於 FreeRTOS Secure Sockets 程式庫的 [coreMQTT 程式庫 \(p. 195\)](#) 會傳回錯誤碼 `AWS_IOT_MQTT_SEND_ERROR`。

開發人員支援

FreeRTOS Secure Sockets 程式庫包含兩個處理 IP 地址的 helper 巨集：

SOCKETS_inet_addr_quick

此巨集會依網路位元組順序將以四個單獨數字八位元表示的 IP 地址轉換為以 32 位元數字表示的 IP 地址。

SOCKETS_inet_ntoa

此巨集會依網路位元組順序將以 32 位元數字表示的 IP 地址轉換為以小數點表示法表示的字串。

使用限制

FreeRTOS Secure Sockets 程式庫僅支援 TCP 通訊端。不支援 UDP 通訊端。

Server APIs are not supported by the FreeRTOS Secure Sockets library, except for a [lightweight IP \(lwIP\)](#) implementation of the server side Bind API. Client APIs are supported.

Initialization

若要使用 FreeRTOS Secure Sockets 程式庫，您需要初始化此程式庫及其相依性。若要初始化 Secure Sockets 程式庫，請在您的應用程式中使用下列程式碼：

```
BaseType_t xResult = pdPASS;  
xResult = SOCKETS_Init();
```

必須個別初始化相依程式庫。For example, if FreeRTOS+TCP is a dependency, you need to invoke [FreeRTOS_IPInit](#) in your application as well.

API 參考

如需完整的 API 參考，請參閱 [Secure Sockets API 參考](#)。

範例使用方式

以下程式碼會將用戶端連接到伺服器。

```
#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/Owua2eiedgPySjAKBggqhkJOPQQDAjA5\n"
"MQswCQYDVQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwFwYDVQODExBBbWF6b24g\n"
```

```
"Um9vdCBDQSAzMB4XDTE1MDUyNjAwMDAwMFoXDTQwMDUyNjAwMDAwMFowOTELMAkG\n"
"A1UEBhMCVVMxDzANBgNVBAoTBkftYXpvbjEZMBcGA1UEAxMQQW1hem9uIFJvb3Qg\n"
"Q0EgMzBZMBMGBYqGSM49AgEGCCqGSM49AwEHA0IABCmXp8ZBf8ANm+gBG1bG81K1\n"
"ui2yEujSLtf6ycXYqm0fc4E7O5hrOXwzpcVoho6AF2hiRVd9RFgdssf1ZwjrZt6j\n"
"QjBAMA8GA1UdEwEB/wQFMAMBAF8wDgYDVROPAQH/BAQDAgGGMB0GA1UdDgQWBBSr\n"
"ttvXBp43rDCGB5Fwx5zEGbF4wDAKBggqhkJOPQODAgNJADBGAIeA4IWSoxe3jfkr\n"
"BqWTrBqYAGFy+uGh0PsceGCmQ5nFuMQCIQCcAu/xlJyz1vnrxir4tiz+OpAUFeM\n"
"YyRIHN8wfdfVoOw==\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. */ 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 );
}
```

}

如需完整範例，請參閱 [Secure Sockets echo 用戶端示範 \(p. 288\)](#)。

Porting

FreeRTOS Secure Sockets 程式庫取決於 TCP/IP 堆疊和 TLS 實作。根據您的堆疊，若要移植 Secure Sockets 程式庫，您可能需要移植下列一些項目：

- The [FreeRTOS+TCP TCP/IP stack](#)
- [corePKCS11 程式庫 \(p. 198\)](#)
- [Transport Layer Security \(p. 205\)](#)

For more information about porting, see [Porting the Secure Sockets Library](#) in the FreeRTOS Porting Guide.

AWS IoT 裝置影子程式庫

Introduction

您可以使用 AWS IoT Device Shadow 程式庫來儲存和擷取每個已登記裝置的目前狀態（陰影）。裝置的影子是您的裝置的持久性、語音呈現，您可以在 Web 應用程式中與它們互動，即使該裝置不同。裝置狀態是在 [JSON](#) 文件中以影子形式擷取。您可以透過 MQTT 或 HTTP 將命令傳送到 AWS IoT Device Shadow 服務，以查詢最新已知的裝置狀態，或變更狀態。每個裝置的影子藉由對應物件的名稱（AWS 雲端上特定裝置或邏輯實體的表示方式）來唯一辨識。如需詳細資訊，請前往 [使用 AWS IoT 管理裝置](#)。您可以在 [文件 AWS IoT 中找到影子的詳細資訊](#)。

除了標準 C 程式庫之外，AWS IoT Device Shadow 程式庫不會相依於其他程式庫。它還不具備任何平台相依性，例如執行緒或同步。它可與任何 MQTT 程式庫及任何 JSON 程式庫搭配使用。

此程式庫可以自由使用，並根據 [MIT 開放原始碼授權分發](#)。

Code Size of AWS IoT Device Shadow (example generated with GCC for ARM Cortex-M)		
File	With -O1 Optimisation	With -Os Optimisation
shadow.c	1.2K	0.7K
Total estimate	1.2K	0.7K

AWS IoT 任務程式庫

Introduction

AWS IoT 任務是一種服務，可通知一或多個連接的裝置待定任務。您可以使用任務來管理您的裝置叢集、更新裝置上的韌體與安全憑證，或執行管理任務，例如重新開機裝置和執行診斷。如需詳細資訊，請前往 <https://docs.aws.amazon.com/iot/latest/developerguide/iot-jobs.html> 中的 Jobs（任務）AWS IoT 開發人員

指南。與 AWS IoT 任務服務的互動使用 [MQTT](#)，這是輕量型發行訂用通訊協定。此程式庫提供 API 來編寫和辨識 AWS IoT 任務服務所使用的 MQTT 主題字串。

Jobs 程式庫是以 C 編寫，並設計為符合 AWS IoT ISO C90 和 MISRA C : 2012。除了標準 C 程式庫之外，程式庫的並無相依性任何其他程式庫。它可與任何 MQTT 程式庫及任何 JSON 程式庫搭配使用。此程式庫具有[可確保](#)顯示安全記憶體使用和無堆積配置，適合 IoT 微型控制器，但也可完全轉移至其他平台。

此程式庫可以自由使用，並根據 [MIT 開放原始碼授權](#) 分發。

Code Size of AWS IoT Jobs (example generated with GCC for ARM Cortex-M)		
File	With -O1 Optimisation	With -Os Optimisation
jobs.c	1.7K	1.4K
Total estimate	1.7K	1.4K

Transport Layer Security

Transport Layer Security (TLS) 界面是一種薄型的選擇性包裝函數，可用來從通訊協定堆疊中位於其上方的 FreeRTOS Secure Sockets Layer ([SSL](#) 界面抽象化密碼編譯實作的詳細資訊)。TLS 界面的目的是讓使用 TLS 通訊協定交涉的替代實作及密碼編譯基本功能取代目前的軟體加密程式庫 (mbed TLS) 更為容易。TLS 界面可切換出去，無須變更 SSL 界面。請參閱 FreeRTOS 來源碼儲存庫中的 `iot_tls.h`。

TLS 界面是選擇性的項目，因為您可以選擇直接從 SSL 建立與加密程式庫之間的界面。此界面不會用於包含 TLS 及網路傳輸完整堆疊卸載實作的 MCU 解決方案。

如需移植 TLS 界面的詳細資訊，請前往 [移植指南](#) 中的移植 TLS 程式庫FreeRTOS。

Wi-Fi 程式庫

Overview

The FreeRTOS [Wi-Fi](#) library abstracts port-specific Wi-Fi implementations into a common API that simplifies application development and porting for all FreeRTOS-qualified boards with Wi-Fi capabilities. 只要使用此常用 API，應用程式即可透過常用界面與更低階的無線堆疊通訊。

相依性和要求

The FreeRTOS Wi-Fi library requires the [FreeRTOS+TCP](#) core.

Features

Wi-Fi 程式庫包含下列功能：

- Support for WEP, WPA, WPA2, and WPA3 authentication
- 存取點掃描
- 電源管理

- 網路分析

如需 Wi-Fi 程式庫功能的詳細資訊，請參閱下面資訊。

Wi-Fi 模式

Wi-Fi devices can be in one of three modes: Station, Access Point, or P2P. You can get the current mode of a Wi-Fi device by calling `WIFI_GetMode`. You can set a device's wi-fi mode by calling `WIFI_SetMode`. Switching modes by calling `WIFI_SetMode` disconnects the device, if it is already connected to a network.

站台模式

將您的裝置設定為站台模式，以將主機板連接到現有的存取點。

存取點 (AP) 模式

將您的裝置設定為 AP 模式，讓裝置成為其他裝置連接到其中的存取點。當您的裝置處於 AP 模式時，您可以將另一個裝置連接到 FreeRTOS 裝置，並設定新的 Wi-Fi 登入資料。To configure AP mode, call `WIFI_ConfigureAP`. To put your device into AP mode, call `WIFI_StartAP`. To turn off AP mode, call `WIFI_StopAP`.

Note

FreeRTOS 程式庫不會在 AP 模式下提供 Wi-Fi 佈建。您必須提供額外的功能，包括 DHCP 和 HTTP 伺服器功能，才能完全支援 AP 模式。

P2P 模式

將您的裝置設定為 P2P 模式，以允許多個裝置無需存取點即可彼此直接連接。

Security

The Wi-Fi API supports WEP, WPA, WPA2, and WPA3 security types. 當裝置處於站台模式時，您必須在呼叫 `WIFI_ConnectAP` 函數時指定網路安全類型。當裝置處於 AP 模式時，裝置可以設定為使用任何支援的安全類型：

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

掃描與連接

若要掃描附近的存取點，請將您的裝置設定為站台模式，並呼叫 `WIFI_Scan` 函數。若您在掃描中找到所需要的網路，您可以透過呼叫 `WIFI_ConnectAP` 並提供網路登入資料來連接到網路。You can disconnect a Wi-Fi device from the network by calling `WIFI_Disconnect`. For more information about scanning and connecting, see [範例使用方式 \(p. 208\)](#) and [API 參考 \(p. 207\)](#).

電源管理

不同的 Wi-Fi 裝置具有不同的電力需求，取決於應用程式及可用的電力來源。裝置可能需要持續接續電源以減少延遲，或是間歇性連接電源，並在不需要 Wi-Fi 時切換到低電力模式。界面 API 支援各種電源管理模式，例如持續開啟、低電力及標準模式。您可以使用 `WIFI_SetPMMode` 函數設定裝置的電源模式。您可以透過呼叫 `WIFI_GetPMMode` 函數來取得裝置目前的電源模式。

網路描述檔

Wi-Fi 程式庫可讓您在裝置的非揮發性記憶體中儲存網路描述檔。This allows you to save network settings so they can be retrieved when a device reconnects to a Wi-Fi network, removing the need to provision devices again after they have been connected to a network. `WIFI_NetworkAdd` adds a network profile. `WIFI_NetworkGet` retrieves a network profile. `WIFI_NetworkDel` deletes a network profile. 您可以儲存的描述檔數量需視平台而定。

Configuration

若要使用 Wi-Fi 程式庫，您需要在組態檔案中定義數個識別碼。如需這些識別碼的相關資訊，請參閱 [API 參考 \(p. 207\)](#)。

Note

此程式庫不包含所需的組態檔案。您必須建立一個。在建立您的組態檔案時，請務必包含您主機板需要的任何主機板特定的組態識別碼。

Initialization

Before you use the Wi-Fi library, you need to initialize some board-specific components, in addition to the FreeRTOS components. 使用 `vendors/vendor/boards/board/aws_demos/application_code/main.c` 檔案做為初始化範本時，請執行下列動作：

1. 如果您的應用程式會處理 Wi-Fi 連線，請移除 `main.c` 中的範例 Wi-Fi 連線邏輯。接著，請取代以下 `DEMO_RUNNER_RunDemos()` 函數呼叫：

```
if( SYSTEM_Init() == pdPASS )
{
...
    DEMO_RUNNER_RunDemos();
...
}
```

改為呼叫您自己的應用程式：

```
if( SYSTEM_Init() == pdPASS )
{
...
    // This function should create any tasks
    // that your application requires to run.
    YOUR_APP_FUNCTION();
...
}
```

2. 呼叫 `WIFI_On()` 來初始化您的 Wi-Fi 晶片，並開啟其電源。

Note

有些主機板可能需要額外的硬體初始化。

3. 將設定的 `WIFINetworkParams_t` 架構傳遞至 `WIFI_ConnectAP()`，以將您的主機板連接到可用的 Wi-Fi 網路。如需 `WIFINetworkParams_t` 架構的詳細資訊，請參閱 [範例使用方式 \(p. 208\)](#) 和 [API 參考 \(p. 207\)](#)。

API 參考

如需完整的 API 參考，請參閱 [Wi-Fi API 參考](#)。

範例使用方式

連接到已知 AP

```
#define clientcredentialWIFI_SSID      "MyNetwork"
#define clientcredentialWIFI_PASSWORD   "hunter2"

WIFINetworkParams_t xNetworkParams;
WIFIReturnCode_t xWifiStatus;

xWifiStatus = WIFI_On(); // Turn on Wi-Fi module

// Check that Wi-Fi initialization was successful
if( xWifiStatus == eWiFiSuccess )
{
    configPRINT( ( "WiFi library initialized.\n" ) );
}
else
{
    configPRINT( ( "WiFi library failed to initialize.\n" ) );
    // Handle module init failure
}

/* Setup parameters. */
xNetworkParams.pcSSID = clientcredentialWIFI_SSID;
xNetworkParams.ucSSIDLength = sizeof( clientcredentialWIFI_SSID );
xNetworkParams.pcPassword = clientcredentialWIFI_PASSWORD;
xNetworkParams.ucPasswordLength = sizeof( clientcredentialWIFI_PASSWORD );
xNetworkParams.xSecurity = eWiFiSecurityWPA2;

// Connect!
xWifiStatus = WIFI_ConnectAP( &( xNetworkParams ) );

if( xWifiStatus == eWiFiSuccess )
{
    configPRINT( ( "WiFi Connected to AP.\n" ) );
    // IP Stack will receive a network-up event on success
}
else
{
    configPRINT( ( "WiFi failed to connect to AP.\n" ) );
    // Handle connection failure
}
```

Scanning for nearby APs

```
WIFINetworkParams_t xNetworkParams;
WIFIReturnCode_t xWifiStatus;

configPRINT( ("Turning on wifi...\n" ) );
xWifiStatus = WIFI_On();

configPRINT( ("Checking status...\n" ) );
if( xWifiStatus == eWiFiSuccess )
{
    configPRINT( ("WiFi module initialized.\n" ) );
}
else
{
    configPRINTF( ("WiFi module failed to initialize.\n" ) );
    // Handle module init failure
```

```
}

WIFI_SetMode(eWiFiModeStation);

/* Some boards might require additional initialization steps to use the Wi-Fi library. */

while (1)
{
    configPRINT( ("Starting scan\n") );
    const uint8_t ucNumNetworks = 12; //Get 12 scan results
    WiFiScanResult_t xScanResults[ ucNumNetworks ];
    xWifiStatus = WIFI_Scan( xScanResults, ucNumNetworks ); // Initiate scan

    configPRINT( ("Scan started\n") );

    // For each scan result, print out the SSID and RSSI
    if ( xWifiStatus == eWiFiSuccess )
    {
        configPRINT( ("Scan success\n") );
        for ( uint8_t i=0; i<ucNumNetworks; i++ )
        {
            configPRINTF( ("%s : %d \n", xScanResults[i].cSSID, xScanResults[i].cRSSI) );
        }
    } else {
        configPRINTF( ("Scan failed, status code: %d\n", (int)xWifiStatus) );
    }

    vTaskDelay(200);
}
```

Porting

The `iot_wifi.c` implementation needs to implement the functions defined in `iot_wifi.h`. At the very least, the implementation needs to return `eWiFiNotSupported` for any non-essential or unsupported functions.

For more information about porting the Wi-Fi library, see [Porting the Wi-Fi Library](#) in the FreeRTOS Porting Guide.

FreeRTOS 示範

FreeRTOS 包含主要 FreeRTOS 目錄下 demos 資料夾中的部分示範應用程式。可以由 FreeRTOS 執行的所有範例會顯示在 common 下的 demos 資料夾中。資料夾下也有每個 FreeRTOS 合格平台的資料夾。demos如果您是使用 [FreeRTOS 主控台](#)，則 demos 下只會有您選擇的目標平台子目錄。

嘗試使用示範應用程式前，我們建議您先完成 [FreeRTOS 入門 \(p. 14\)](#) 中的教學。它顯示如何設定和執行 coreMQTT 交互身份驗證示範。

執行 FreeRTOS 示範

下列主題說明如何設定和執行 FreeRTOS 示範：

- [低功耗藍牙示範應用程式 \(p. 210\)](#)
- [Microchip Curiosity PIC32MZEF 的示範開機載入器 \(p. 221\)](#)
- [AWS IoT Device Defender 示範 \(p. 226\)](#)
- [AWS IoT Greengrass 探索示範應用程式 \(p. 229\)](#)
- [coreHTTP 示範 \(p. 232\)](#)
- [AWS IoT 任務程式庫示範 \(p. 239\)](#)
- [coreMQTT 交互身份驗證示範 \(p. 241\)](#)
- [無線更新示範應用程式 \(p. 248\)](#)
- [Secure Sockets echo 用戶端示範 \(p. 288\)](#)
- [AWS IoT 裝置影子示範應用程式 \(p. 277\)](#)

位於 DEMO_RUNNER_RunDemos 檔案的 `freertos/demos/demo_runner/iot_demo_runner.c` 函數會將執行單一示範應用程式的分離執行緒初始化。根據預設，DEMO_RUNNER_RunDemos 只會呼叫並開始 coreMQTT 交互身份驗證示範。根據您下載 FreeRTOS 時選取的組態，以及您下載 FreeRTOS 的位置，其他範例執行器函數可能根據預設開始。若要讓示範應用程式開始執行，請打開 `freertos/vendors/vendor/boards/board/aws_demos/config_files/aws_demo_config.h` 檔案，並定義要執行的示範。

Note

並非所有範例組合都可搭配運作。根據組合不同，由於記憶體限制，可能無法在所選目標上執行軟體。我們建議您一次執行一個示範。

設定示範

示範已設定為可讓您快速開始使用。建議您變更專案的某些組態，以建立在您平台上執行的版本。您可以在 `vendors/vendor/boards/board/aws_demos/config_files` 中找到組態檔案。

低功耗藍牙示範應用程式

Overview

FreeRTOS 低功耗藍牙包含三個示範應用程式：

- 透過低功耗藍牙執行的 MQTT (p. 217) 示範

這個應用程式會示範如何透過低功耗藍牙服務使用 MQTT。

- Wi-Fi 佈建 (p. 218) 示範

這個應用程式會示範如何使用低功耗藍牙 Wi-Fi 佈建服務。

- 一般屬性伺服器 (p. 220) 示範

這個應用程式示範如何使用 FreeRTOS 低功耗藍牙中介軟體 APIs 來建立簡單的 GATT 伺服器。

Note

若要設定和執行 FreeRTOS 示範，請遵循[FreeRTOS 入門 \(p. 14\)](#)中的步驟。

Prerequisites

若要遵循這些示範，您需要使用微型控制器搭配低功耗藍牙功能。您還需具備 [FreeRTOS 藍牙裝置的 iOS 軟體開發套件 \(p. 191\)](#)或 [FreeRTOS 藍牙裝置的 Android 軟體開發套件 \(p. 191\)](#)。

為 FreeRTOS 低功耗藍牙設定 AWS IoT 和 Amazon Cognito

您需要設定 AWS IoT 和 Amazon Cognito，才能將裝置跨 MQTT 連接至 AWS IoT。

設定 AWS IoT

- 在 AWS 上設定 <https://aws.amazon.com/> 帳號。
- 開啟 [AWS IoT 主控台](#)，然後從導覽窗格中，選擇 Manage (管理)，然後選擇 Things (實物)。
- 選擇 Create (建立)，然後選擇 Create a single thing (建立單一實物)。
- 輸入您裝置的名稱，然後選擇 Next (下一步)。
- 如果您是透過行動裝置將微型控制器連接到雲端，請選擇 Create thing without certificate (建立沒有憑證的物件)。由於行動裝置 SDKs 使用 Amazon Cognito 進行裝置身份驗證，您不需要為使用低功耗藍牙的示範建立裝置憑證。

如果您是透過 Wi-Fi 直接將微型控制器連接到雲端，則請選擇 Create certificate (建立憑證)，再選擇 Activate (啟動)，然後下載物件的憑證、公開金鑰和私密金鑰。

- 從已註冊的實物清單中選擇您剛建立的實物，然後從實物的頁面中選擇 Interact (互動)。記下 AWS IoT REST API 端點。

如需設定的詳細資訊，請參閱 [AWS IoT 入門](#)。

建立 Amazon Cognito 使用者集區

- 開啟 Amazon Cognito 主控台，然後選擇 Manage User Pools (管理使用者集區)。
- 選擇 Create a user pool (建立使用者集區)。
- 為使用者集區提供一個名稱，然後選擇 Review defaults (檢閱預設值)。
- 從導覽窗格中選擇 App clients (應用程式用戶端)，然後選擇 Add an app client (新增應用程式用戶端)。
- 輸入應用程式用戶端的名稱，然後選擇 Create app client (建立應用程式用戶端)。
- 從導覽窗格中選擇 Review (檢閱)，然後選擇 Create pool (建立集區)。

記下使用者集區之 General Settings (一般設定) 頁面上出現的集區 ID。

- 從導覽窗格中選擇 App clients (應用程式用戶端)，然後選擇 Show details (顯示詳細資訊)。記下應用程式用戶端 ID 和應用程式用戶端密碼。

建立 Amazon Cognito 身分集區

1. 開啟 Amazon Cognito 主控台，然後選擇 Manage Identity Pools (管理身分集區)。
2. 輸入身分集區的名稱。
3. 展開 Authentication providers (驗證供應商)、選擇 Cognito 標籤，然後輸入您的使用者集區 ID 和應用程式用戶端 ID。
4. 選擇 Create Pool (建立集區)。
5. 展開 View Details (檢視詳細資訊)，並記下兩個 IAM 角色名稱。選擇 Allow (允許)，為已驗證和未驗證的身分建立 IAM 角色，以存取 Amazon Cognito。
6. 選擇 Edit identity pool (編輯身分集區)。記下身分集區 ID。其格式應為 us-west-2:12345678-1234-1234-1234-123456789012。

如需設定 Amazon Cognito 的詳細資訊，請參閱 [Amazon Cognito 入門](#)。

建立 IAM 政策並將其連接到已驗證的身分

1. 開啟 IAM 主控台，然後從導覽窗格中選擇 Roles (角色)。
2. 尋找並選擇您已驗證身分的角色、選擇 Attach policies (連接政策)，然後選擇 Add inline policy (新增內嵌政策)。
3. 選擇 JSON 標籤，並貼上下列 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. 選擇 Review policy (檢閱政策)、輸入政策的名稱，然後選擇 Create policy (建立政策)。

將 AWS IoT 和 Amazon Cognito 資訊放在手邊。您需要 端點和 IDs，才能向 AWS 雲端驗證行動應用程式。

設定適用於低功耗藍牙的 FreeRTOS 環境

若要設定環境，您需要在微型控制器上下載 FreeRTOS 與 [低功耗藍牙程式庫 \(p. 182\)](#)，並在行動裝置上下載和設定適用於 FreeRTOS 藍牙裝置的 Mobile SDK (行動軟體開發套件)。

使用 FreeRTOS 低功耗藍牙設定微型控制器的環境

1. 從 FreeRTOS [GitHub 下載或複製](#)。請前往 [README.md](#) 檔案以獲得指示。
2. 在微型控制器上設定 FreeRTOS。

如需在符合 FreeRTOS 資格的微型控制器上開始使用 FreeRTOS 的相關資訊，請參閱 [FreeRTOS 入門](#) 中的主機板指南。

Note

您能夠在任何已啟用低功耗藍牙的微型控制器上使用 FreeRTOS 和移植的 FreeRTOS 低功耗藍牙程式庫來執行示範。目前，FreeRTOS 透過低功耗藍牙執行的 MQTT (p. 217) 示範專案已完全移植至下列已啟用低功耗藍牙的裝置：

- Espressif ESP32-DevKitC 和 ESP-WROVER-KIT
- Nordic nRF52840-DK

常見元件

FreeRTOS 示範應用程式有兩個常見的元件：

- 網路管理員
- 低功耗藍牙 Mobile SDK 示範應用程式

網路管理員

網路管理員可管理您的微型控制器的網路連線。它位於您的 FreeRTOS，位置在 `demos/network_manager/aws_iot_network_manager.c`。如果網路管理員同時支援 Wi-Fi 和低功耗藍牙，則示範預設會從低功耗藍牙開始。如果低功耗藍牙連線中斷，且主機板已啟用 Wi-Fi，則網路管理員會切換到可用的 Wi-Fi 連線，以防網路連線中斷。

若要使用網路管理員建立網路連接類型，請將網路連接類型新增至 `configENABLED_NETWORKS` 中的 `vendors/vendor/boards/board/aws_demos/config_files/aws_iot_network_config.h` 參數（其中 `vendor` 是廠商的名稱和 `board` 是您用來執行示範的電路板名稱）。

例如，如果您同時啟用了低功耗藍牙和 Wi-Fi，`aws_iot_network_config.h` 中開頭為 `#define configENABLED_NETWORKS` 的該行會顯示為：

```
#define configENABLED_NETWORKS ( AWSIOT_NETWORK_TYPE_BLE | AWSIOT_NETWORK_TYPE_WIFI )
```

若要取得目前支援的網路連線類型清單，請查看 `aws_iot_network.h` 中開頭為 `#define AWSIOT_NETWORK_TYPE` 的那幾行。

FreeRTOS 低功耗藍牙 Mobile SDK 示範應用程式

低功耗藍牙 Mobile SDK 示範應用程式位於 FreeRTOS 的 GitHub 上，即 [下的 FreeRTOS 藍牙裝置的 Android 開發套件](#) 和 [amazon-freertos-ble-android-sdk/app](#) 下的 [藍牙裝置的 iOS 開發套件 FreeRTOS](#)。[amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo](#) 在這個範例中，我們使用 iOS 版示範行動應用程式的螢幕擷取畫面。

Note

如果您使用 iOS 裝置，您需要 Xcode 來建置示範行動應用程式。如果您使用 Android 裝置，您可以使用 Android Studio 來建置示範行動應用程式。

設定 iOS SDK 示範應用程式

當您定義組態變數時，請使用組態檔案中提供之預留位置值的格式。

1. 確認已安裝 [FreeRTOS 藍牙裝置的 iOS 軟體開發套件 \(p. 191\)](#)。
2. 從 [amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/](#) 發出下列命令：

```
$ pod install
```

3. 使用 Xcode 開啟 amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/AmazonFreeRTOSDemo.xcworkspace 專案，並將簽署開發人員帳戶變更為您的帳戶。
4. 在您的區域建立 AWS IoT 政策 (如果尚未建立的話)。

Note

此政策與針對 Cognito 驗證身分所建立的 IAM 政策不同。

- a. 開啟 AWS IoT 主控台。
- b. 在導覽窗格中，選擇 Secure (安全)、Policies (政策)，然後選擇 Create (建立)。輸入可識別政策的名稱。在 Add statements (新增陳述式) 區段中，選擇 Advanced mode (進階模式)。將下列 JSON 複製並貼入政策編輯器視窗。Replace **aws-region** 和 **aws-account** 取代為您的 AWS 區域和帳號 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:"*"  
        }  
    ]  
}
```

- c. 選擇 Create (建立)。
5. 開啟 amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/AmazonFreeRTOSDemo/Amazon/AmazonConstants.swift 並重新定義以下變數：
 - region：您的 AWS 區域。
 - iotPolicyName：您的 AWS IoT 政策名稱。
 - mqttCustomTopic：您要發行的 MQTT 主題。
6. Open amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/AmazonFreeRTOSDemo/Support/awsconfiguration.json.

在 CognitoIdentity 下，重新定義以下變數：

- PoolId：您的 Amazon Cognito 身分集區 ID。
- Region：您的 AWS 區域。

在 CognitoUserPool 下，重新定義以下變數：

- PoolId：您的 Amazon Cognito 使用者集區 ID。
- AppClientId：您的應用程式使用者 ID。
- AppClientSecret：您的應用程式使用者秘密。
- Region：您的 AWS 區域。

設定 Android SDK 示範應用程式

當您定義組態變數時，請使用組態檔案中提供之預留位置值的格式。

1. 確認已安裝 FreeRTOS 藍牙裝置的 Android 軟體開發套件 (p. 191)。
2. 在您的區域建立 AWS IoT 政策 (如果尚未建立的話)。

Note

此政策與針對 Cognito 驗證身分所建立的 IAM 政策不同。

- a. 開啟 AWS IoT 主控台。
- b. 在導覽窗格中，選擇 Secure (安全)、Policies (政策)，然後選擇 Create (建立)。輸入可識別政策的名稱。在 Add statements (新增陳述式) 區段中，選擇 Advanced mode (進階模式)。將下列 JSON 複製並貼入政策編輯器視窗。Replace `aws-region` 和 `aws-account` 取代為您的 AWS 區域和帳號 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:"*"  
        }  
    ]  
}
```

- c. 選擇 Create (建立)。
3. 開啟 <https://github.com/aws/amazon-freertos-ble-android-sdk/blob/master/app/src/main/java/software/amazon/freertos/demo/DemoConstants.java> 並重新定義下列變數：
 - AWS_IOT_POLICY_NAME：您的 AWS IoT 政策名稱。
 - AWS_IOT_REGION：您的 AWS 區域。
4. 開啟 <https://github.com/aws/amazon-freertos-ble-android-sdk/blob/master/app/src/main/res/raw/awsconfiguration.json>。

在 CognitoIdentity 下，重新定義以下變數：

- PoolId：您的 Amazon Cognito 身分集區 ID。
- Region：您的 AWS 區域。

在 CognitoUserPool 下，重新定義以下變數：

- PoolId：您的 Amazon Cognito 使用者集區 ID。
- AppClientId：您的應用程式使用者 ID。
- AppClientSecret：您的應用程式使用者秘密。
- Region：您的 AWS 區域。

透過低功耗藍牙探索並建立微型控制器的安全連線

1. 為了安全地將您的微型控制器與行動裝置進行配對（步驟 6），您需要同時具有輸入和輸出功能的序列終端機模擬器（例如 TeraTerm）。依照[安裝終端機模擬器 \(p. 20\)](#)中的指示，將終端機設定為透過序列連線連接到您的電路板。
2. 在微型控制器上執行低功耗藍牙示範專案。
3. 在行動裝置上執行低功耗藍牙 Mobile SDK 示範應用程式。

若要從命令列啟動 Android SDK 中的示範應用程式，請執行下列命令：

```
$ ./gradlew installDebug
```

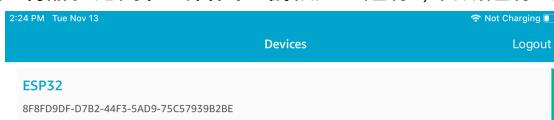
4. 確認低功耗藍牙 Mobile SDK 示範應用程式的 Devices (裝置) 底下有顯示您的微型控制器。



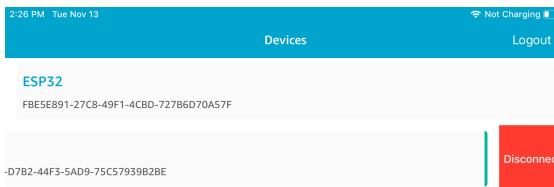
Note

清單中會出現範圍內具備 FreeRTOS 和裝置資訊服務 (*freertos/.../device_information*) 的所有裝置。

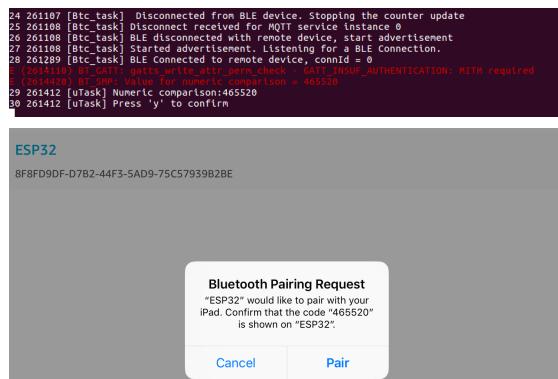
5. 從裝置清單選擇您的微型控制器。應用程式會與主機板建立連線，而所連線的裝置旁會出現綠色線條。



您可以透過將該線條拖曳至左側來與微型控制器中斷連線。



6. 如果出現提示，請將您的微型控制器與行動裝置進行配對。



如果兩個裝置上用來比較數字的程式碼相同，系統即會配對裝置。

Note

低功耗藍牙 Mobile SDK 示範應用程式會使用 Amazon Cognito 進行使用者身分驗證。確定您已設定 Amazon Cognito 使用者和身分集區，且已將 IAM 政策連接至經過驗證的身分。

透過低功耗藍牙執行的 MQTT

在透過低功耗藍牙執行的 MQTT 示範中，微型控制器會經由 MQTT Proxy 將訊息發佈到 AWS 雲端。

訂閱示範 MQTT 主題

1. 登入 AWS IoT 主控台。
2. 在導覽窗格中，選擇 Test (測試) 以開啟 MQTT 用戶端。
3. 在訂閱主題中輸入 **thing-name/example/topic1**，然後選擇訂閱主題。

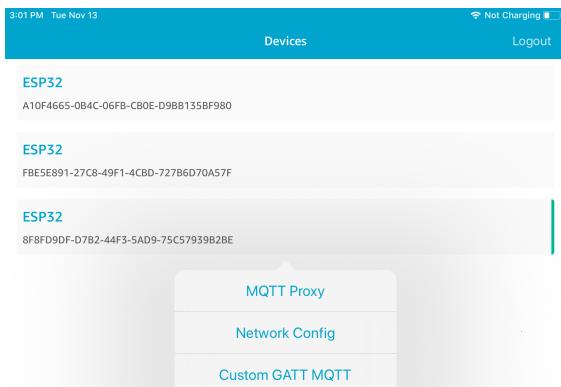
如果您是使用低功耗藍牙來配對微型控制器與行動裝置，則系統會在行動裝置上透過低功耗藍牙 Mobile SDK 示範應用程式路由 MQTT 訊息。

透過低功耗藍牙進行示範

- 開啟 `vendors/vendor/boards/board/aws_demos/config_files/aws_demo_config.h`，然後定義 `CONFIG_MQTT_BLE_TRANSPORT_DEMO_ENABLED`。

執行示範

1. 在微型控制器上建置和執行示範專案。
2. 確保已使用 [FreeRTOS 低功耗藍牙 Mobile SDK 示範應用程式 \(p. 213\)](#) 將您的主機板與行動裝置配對。
3. 從示範行動應用程式中的 Devices (裝置) 清單選擇微型控制器，接著選擇 MQTT Proxy 以開啟 MQTT Proxy 設定。



4. 啟用 MQTT Proxy 後，MQTT 訊息即會顯示在 `thing-name/example/topic1` 主題上，且系統會將資料列印到 UART 終端機。

Wi-Fi 佈建

Wi-Fi 佈建是一種 FreeRTOS 低功耗藍牙服務，可讓您安全地透過低功耗藍牙從行動裝置傳送 Wi-Fi 網路登入資料至微型控制器。您可以在 `freertos/.../wifi_provisioning` 中找到 Wi-Fi 佈建服務的原始程式碼。

Note

目前在 Espressif ESP32-DevKitC 上支援 Wi-Fi 佈建示範。

啟用示範

1. 啟用 Wi-Fi 佈建服務。開放 `vendors/vendor/boards/board/aws_demos/config_files/iot_ble_config.h`，並將 `#define IOT_BLE_ENABLE_WIFI_PROVISIONING` 設定為 1 (即 `vendor` 是廠商的名稱和 `board` 是您用來執行示範的電路板名稱)。

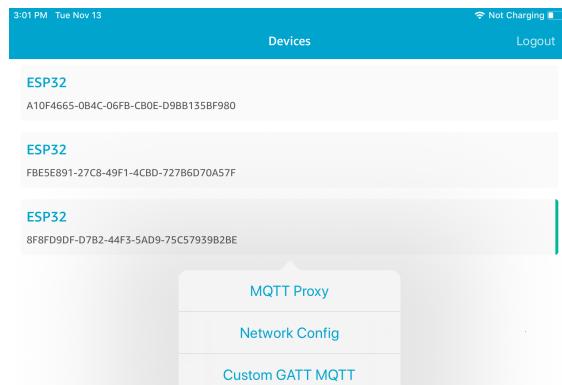
Note

在預設情況下，Wi-Fi 佈建服務會處於停用狀態。

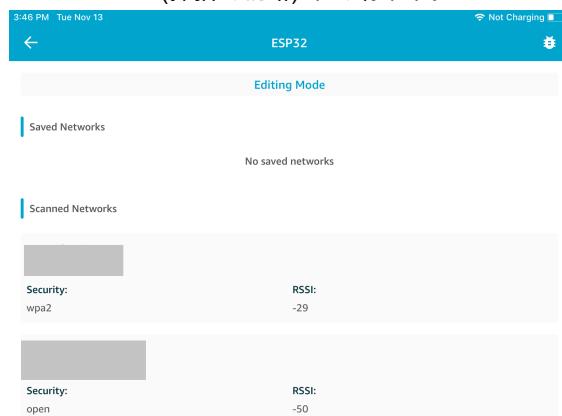
2. 設定 [網路管理員 \(p. 213\)](#)，即可同時啟用低功耗藍牙和 Wi-Fi。

執行示範

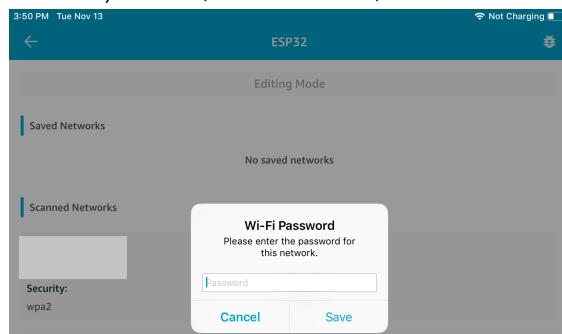
1. 在微型控制器上建置和執行示範專案。
2. 確保已使用 [FreeRTOS 低功耗藍牙 Mobile SDK 示範應用程式 \(p. 213\)](#) 將您的微型控制器與行動裝置配對。
3. 從示範行動應用程式的 Devices (裝置) 清單中選擇您的微型控制器，然後選擇 Network Config (網路組態) 以開啟網路組態設定。



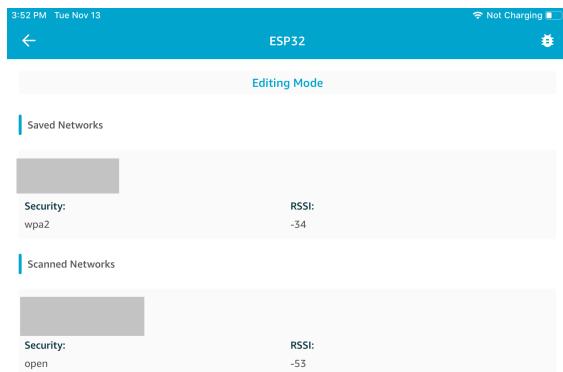
4. 選擇主機板的 Network Config (網路組態) 後，微型控制器會傳送鄰近的網路清單至行動裝置。可用的 Wi-Fi 網路會出現在 Scanned Networks (掃描的網路) 下的清單中。



從 Scanned Networks (掃描的網路) 清單中，選擇您的網路，然後輸入 SSID 和密碼 (如果必要)。

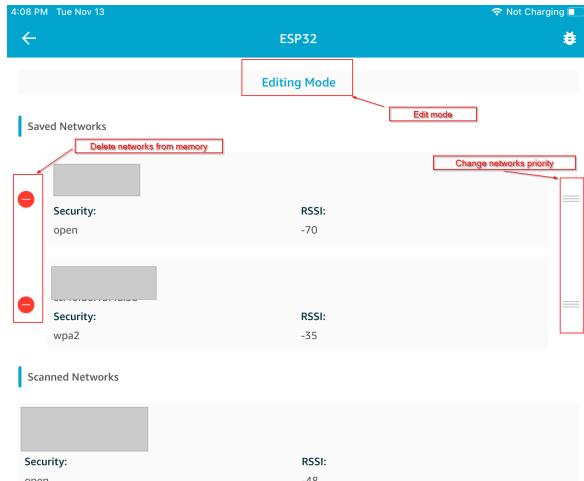


微型控制器會連接至並儲存網路。網路會出現在 Saved Networks (儲存的網路) 下。



您可以將數個網路儲存在示範行動應用程式中。重新啟動應用程式和示範時，微型控制器會連接到第一個可用的已儲存網路，從 Saved Networks (儲存的網路) 清單的頂端開始。

若要變更網路優先順序或刪除網路，請在 Network Configuration (網路組態) 頁面上，選擇 Editing Mode (編輯模式)。若要變更網路的優先順序，請選擇您要重新設定優先順序的網路右側，並將網路往上或往下拖曳。若要刪除網路，請選擇您要刪除的網路左側的紅色按鈕。



一般屬性伺服器

在本範例中，微型控制器上的示範一般屬性 (GATT) 伺服器應用程式會傳送一個簡單的計數器值至 FreeRTOS 低功耗藍牙 Mobile SDK 示範應用程式 (p. 213)。

使用低功耗藍牙行動SDKs，您可以為連接到微型控制器上 GATT 伺服器的行動裝置建立自己的 GATT Client，並與示範行動應用程式平行執行。

啟用示範

1. 啟用低功耗藍牙 GATT 示範。在 `vendors/vendor/boards/board/aws_demos/config_files/iot_ble_config.h` 中（其中的 `vendor` 是廠商的名稱和 `board` 是您用來執行示範的電路板名稱，請將 `#define IOT_BLE_ADD_CUSTOM_SERVICES (1)` 新增至 define 陳述式的清單）。

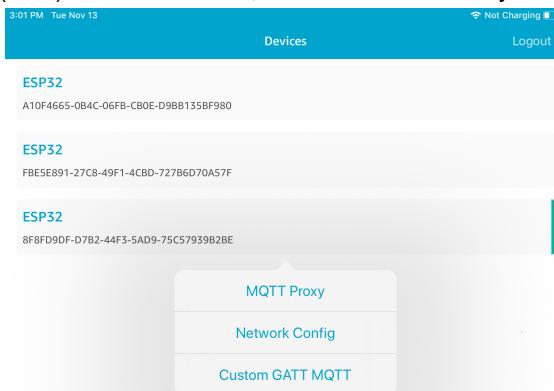
Note

在預設情況下，低功耗藍牙 GATT 示範會處於停用狀態。

2. 開啟 `freertos/vendors/vendor/boards/board/aws_demos/config_files/aws_demo_config.h`、註解 `#define CONFIG_MQTT_DEMO_ENABLED`，然後定義 `CONFIG_BLE_GATT_SERVER_DEMO_ENABLED`。

執行示範

1. 在微型控制器上建置和執行示範專案。
2. 確保已使用 [FreeRTOS 低功耗藍牙 Mobile SDK](#) 示範應用程式 (p. 213) 將您的主機板與行動裝置配對。
3. 從應用程式中的 Devices (裝置) 清單選擇主機板，然後選擇 MQTT Proxy 以開啟 MQTT Proxy 選項。



4. 返回 Devices (裝置) 清單並選擇主機板，然後選擇 Custom GATT MQTT (自訂 GATT MQTT) 以開啟自訂 GATT 服務選項。
5. 選擇 Start Counter (啟動計數器)，即可開始發佈資料到 `iotdemo/#` MQTT 主題。

啟用 MQTT Proxy 之後，Hello World 和遞增的計數器訊息會顯示在 `iotdemo/#` 主題上。

Microchip Curiosity PIC32MZEF 的示範開機載入器

此示範開機載入器實作韌體版本檢查、加密簽章驗證和應用程式自主測試。這些功能支援 FreeRTOS 的遠端 (OTA) 韌體更新。

韌體驗證包含驗證遠端接收之新韌體的可靠性和完整性。開機載入器在啟動之前會驗證應用程式的加密簽章。示範會使用橢圓曲線數位簽章演算法 (ECDSA) 而不是 SHA-256。提供的公用程式，可用於產生可在裝置上刷新的已簽署應用程式。

開機載入器支援下列 OTA 所需的功能：

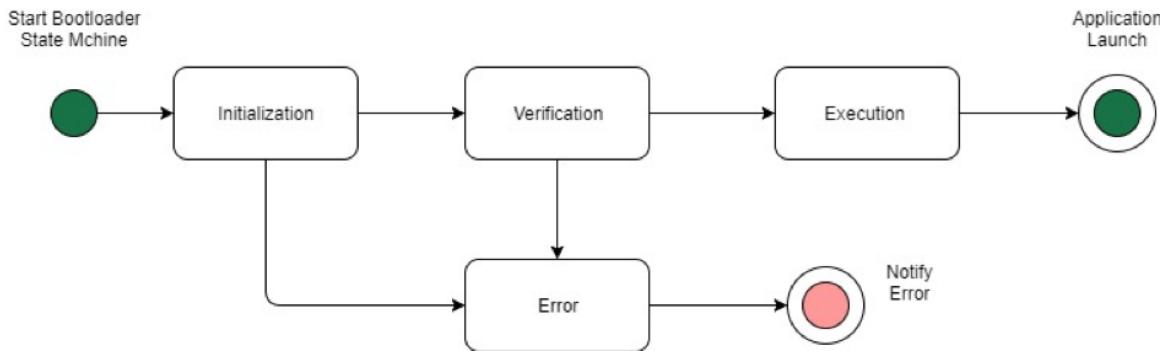
- 在裝置上維護應用程式映像，並在其間切換。
- 允許自主測試執行收到的 OTA 映像並在失敗時復原。
- 檢查 OTA 更新映像的簽章和版本。

Note

若要設定和執行 FreeRTOS 示範，請遵循[FreeRTOS 入門 \(p. 14\)](#)中的步驟。

開機載入器狀態

以下狀態機器中會顯示開機載入器程序。



下表描述開機載入器狀態。

開機載入器狀態	描述
初始化	開機載入器處於初始化狀態。
驗證	開機載入器正在驗證裝置上出現的映像。
執行映像	開機載入器正在啟動選取的映像。
執行預設	開機載入器正在啟動預設映像。
錯誤	開機載入器處於錯誤狀態。

在先前的圖表中，Execute Image 和 Execute Default 皆會顯示為 Execution 狀態。

開機載入器執行狀態

開機載入器處於 Execution 狀態，並準備好啟動所選的已驗證映像。如果要啟動的映像位於上層區塊中，則在執行映像之前會切換區塊，因為應用程式一律針對較低層區塊建置。

開機載入器預設執行狀態

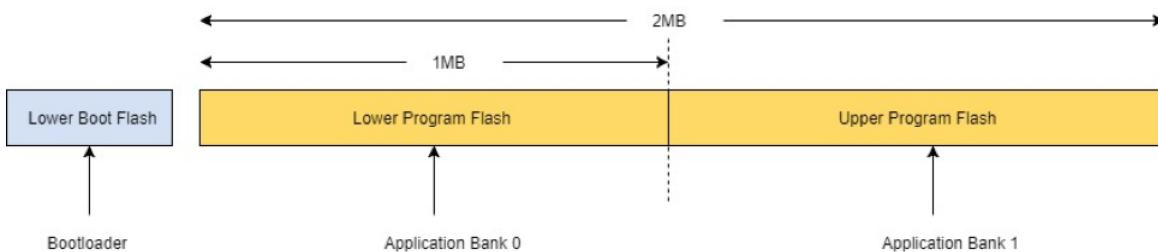
如果啟用了啟動預設映像的組態選項，則開機載入器將從預設執行地址啟動應用程式。除了偵錯外，此選項必須停用。

開機載入器錯誤狀態

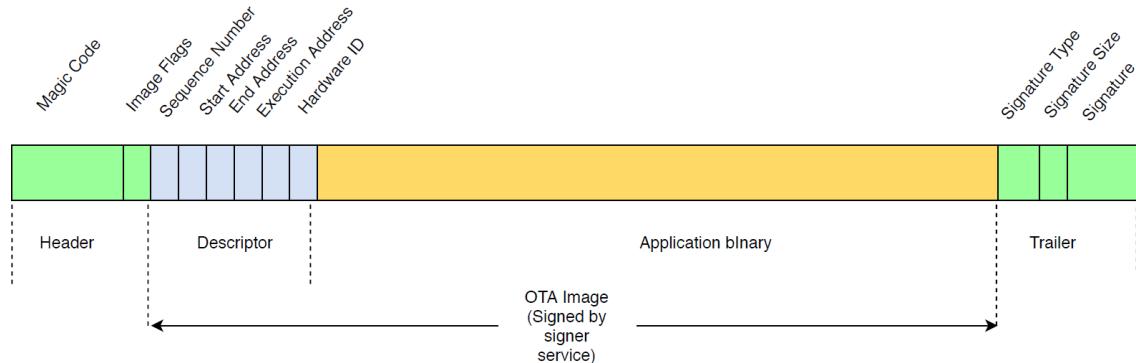
開機載入器處於錯誤狀態，且裝置上不存在有效映像。開機載入器必須通知使用者。預設實作會向主控制傳送日誌訊息，並無限期快速閃爍電路板上的 LED。

快閃裝置

Microchip Curiosity PIC32MZEF 平台包含 2 MB 的內部程式快閃，分為兩個區塊。此快閃支援這兩個區塊之間的記憶體映射切換與即時更新。示範開機載入器在個別的較低啟動快閃區域中設定。



應用程式映像結構



此圖顯示存放在裝置每個區塊中應用程式映像的主要元件。

元件	大小 (位元組)
映像標頭	8 位元組
映像描述項	24 位元組
應用程式二進位	< 1 MB - (324)
預告片	292 位元組

映像標頭

裝置上的應用程式映像，必須以包含神奇代碼和映像旗標的標頭做為開頭。

標頭欄位	大小 (位元組)
神奇代碼	7 位元組
映像旗標	1 位元組

神奇代碼

裝置上的映像必須以神奇代碼開頭。預設神奇代碼為 @AFRTOS。開機載入器會檢查在開機映像之前是否有有效的神奇代碼。這是驗證的第一步。

映像旗標

影像旗標用於存放應用程式映像的狀態。旗標用於 OTA 程序。兩個區塊的映像旗標會判斷裝置的狀態。如果執行映像標記為遞交等待中，則表示裝置處於 OTA 自主測試階段。即使裝置上的映像標記為有效，也會在每次啟動時執行相同的驗證步驟。如果映像標記為新映像，則開機載入器會將其標記為遞交等待中，並在驗證後將其啟動以進行自主測試。開機載入器也會初始化並啟動監視程式計時器，在新的 OTA 映像未通過自主測試時，裝置會重新啟動並且開機載入器會透過清除映像來拒絕映像，並執行先前有效的映像。

裝置只能具有一個有效的映像。另一個映像可以是新的 OTA 映像，或遞交等待中（自主測試）。OTA 更新成功之後，舊映像會從裝置中清除。

Status	數值	描述
新映像	0xFF	應用程式映像是新的且從未執行過。
遞交等待中	0xFE	應用程式映像標示為測試執行。
有效	0xFC	應用程式映像標示為有效和已遞交。
無效	0xF8	應用程式映像標示為無效。

映像描述項

快閃裝置上的應用程式映像，必須包含映像標頭後面的映像描述項。映像描述項由建置後公用程式產生，該公用程式會使用組態檔案 (`ota-descriptor.config`) 來產生適當的描述項，並將其附加至應用程式二進位中。此建構後步驟的輸出是可用於 OTA 的二進位映像。

描述項欄位	大小 (位元組)
序號	4 位元組
起始地址	4 位元組
結束地址	4 位元組
執行地址	4 位元組
硬體 ID	4 位元組
已預留	4 位元組

序號

建置新 OTA 映像之前，必須遞增新序號。請參閱 `ota-descriptor.config` 檔案。開機載入器使用此數量來決定要啟動的映像。有效值為 1 到 4294967295。

起始地址

裝置上應用程式映像的起始地址。由於映像描項已附加至應用程式二進位，所以此地址是映像描述項的起始。

結束地址

裝置上應用程式映像的結束地址，不含映像尾部。

執行地址

映像的執行地址。

硬體 ID

開機載入器用於驗證 OTA 映像的唯一硬體 ID，針對正確的平台所建置。

已預留

已預留供日後使用。

映像尾部

映像尾部會附加至應用程式二進位。包含簽章類型字串、簽章大小和映像的簽章。

尾部欄位	大小 (位元組)
簽章類型	32 位元組
簽章大小	4 位元組
簽章	256 位元組

簽章類型

簽章類型是表示正在使用之加密演算法的字串，並用做尾部的標記。開機載入器支援橢圓曲線數位簽章演算法 (ECDSA)。預設為 sig-sha256-ecdsa。

簽章大小

加密簽章的大小 (以位元組為單位)。

簽章

以映像描述項前綴的應用程式二進位檔案加密簽章。

開機載入器組態

基本的開機載入器組態選項會在 `freertos/vendors/microchip/boards/curiosity_pic32mzef/bootloader/config_files/aws_boot_config.h` 中提供。某些選項僅供偵錯之用。

啟用預設啟動

允許從預設地址執行應用程式，並且必須僅針對偵錯啟用。映像會從預設地址執行，且不需任何驗證。

啟用加密簽章驗證

啟動時啟用加密簽章驗證。從裝置清除失敗的映像。此選項僅供偵錯之用，且必須在生產環境中保持啟用。

清除無效的映像

如果該區塊的映像驗證失敗，則啟用完整區塊清除。此選項供偵錯之用，且必須在生產環境中保持啟用。

啟用硬體 ID 驗證

允許驗證 OTA 映像描述項中的硬體 ID，以及在開機載入器中設定的硬體 ID。此為選用，如果不需要硬體 ID 驗證，則可以將其停用。

啟用地址驗證

允許驗證 OTA 映像描述項中的起始、結束和執行地址。建議您保持啟用此選項。

建置開機載入器

示範開機載入器會作為可載入專案，包含在位於 FreeRTOS 來源碼儲存庫中 `freertos/vendors/microchip/boards/curiosity_pic32mzef/aws_demos/mplab/` 的 `aws_demos` 專案中。當

aws_demos 專案建置時，會先建置開機載入器，接著建置應用程式。最終輸出是統一的十六進位映像，包含開機載入器和應用程式。會提供 factory_image_generator.py 公用程式，以產生具有加密簽章的統一十六進位映像。開機載入器公用程式指令碼位於 [freertos/demos/ota/bootloader/utility/](#)。

開機載入器預先建置步驟

此預先建置步驟會執行稱為 codesigner_cert_utility.py 的公有程式指令碼，從程式碼簽署憑證中擷取公有金鑰，並產生包含 Abstract Syntax Notation One (ASN.1) 編碼格式公有金鑰的 C 標頭檔案。此標頭會編譯至開機載入器專案。產生的標頭包含兩個常數：公有金鑰的陣列和金鑰長度。開機載入器專案也可以在沒有 aws_demos 的情況下建置，並可以做為一般應用程式進行偵錯。

AWS IoT Device Defender 示範

Introduction

此示範示範如何使用 AWS IoT 裝置影子程式庫來連接到 AWS IoT Device Defender。示範使用 coreMQTT 程式庫建立與 AWS IoT MQTT 中介裝置和 coreJSON 程式庫的 MQTT 連接，以驗證及剖析從 AWS IoT Device Defender 接收的回應。此示範示範如何使用從裝置收集的指標，以及將報告提交至 AWS IoT Device Defender 的方式，建立 JSON 格式的報告。示範也示範如何使用 coreMQTT 程式庫來登記回呼函數，以處理 AWS IoT Device Defender 傳送的回應，以確認報告是否接受或遭拒。

Note

若要設定和執行 FreeRTOS 示範，請遵循[FreeRTOS 入門 \(p. 14\)](#)中的步驟。

Functionality

此示範會建立單一應用程式任務，示範如何收集指標、建構 JSON 格式的裝置防禦者報告，並透過與 AWS IoT Device Defender MQTT 中介裝置的安全 MQTT 連接，將它提交到 AWS IoT。

收集指標的方式視所使用的 TCP/IP 堆疊而定。對於 FreeRTOS+TCP 和支援的 lwIP 組態，我們提供收集裝置實際指標的指標收集實作，並在 AWS IoT Device Defender 報告中提交這些實作。您可以在 [上找到 FreeRTOS+TCP 和 lwIP 的實作](#)。GitHub

對於使用任何其他 TCP/IP 堆疊的電路板，會提供所有指標傳回零之指標集合函數的 stub 定義。若要使用此 stub 實作在電路板上傳送真實指標，請為您的網路堆疊實作 [freertos/demos/device_defender_for_aws/metrics_collector/stub/metrics_collector.c](#) 中的函數。檔案也可以在 [GitHub](#) 網站上取得。

對於 ESP32，預設 lwIP 組態不使用核心鎖定，因此示範將使用 stubed 指標。如果您想要使用參考 lwIP 指標集合實作，請在 lwiopts.h 中定義以下巨集：

```
#define LINK_SPEED_OF_YOUR_NETIF_IN_BPS 0
#define LWIP_TCPIP_CORE_LOCKING 1
#define LWIP_STATS 1
#define MIB2_STATS 1
```

以下是您執行示範時的範例輸出。

```
15 2026 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:632] MQTT connection successfully established with broker.  
16 2026 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:660] A clean MQTT connection is established. Cleaning up all the stored outgoing publishes.  
17 2026 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:751] SUBSCRIBE topic $aws/things/MyThingName/defender/metrics/json/accepted to broker.  
18 2086 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=3.  
19 2086 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:463] MQTT_PACKET_TYPE_SUBACK.  
20 2686 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:751] SUBSCRIBE topic $aws/things/MyThingName/defender/metrics/json/rejected to broker.  
21 2746 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=3.  
22 2746 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:463] MQTT_PACKET_TYPE_SUBACK.  
23 3346 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:864] the published payload:{"header": {"report_id": 1499, "version": "1.0"}, "metrics": {"listening_tcp_ports": {"ports": [{"port": 26023}], "total": 1}, "listening_udp_ports": {"ports": [], "total": 0}, "network_stats": {"bytes_in": 0, "bytes_out": 0, "packets_in": 0, "packets_out": 0}}, "topic": "$aws/things/MyThingName/defender/metrics/json", "packet_id": 3}.  
24 3346 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:892] PUBLISH sent for topic $aws/things/MyThingName/defender/metrics/json to broker with packet ID 3.  
25 3506 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=2.  
26 3506 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1132] Ack packet deserialized with result: MQTTSuccess.  
27 3506 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1145] State record updated. New state=MQTTPublishDone.  
28 3506 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:484] PUBACK received for packet id 3.  
29 3506 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:448] Cleaned up outgoing publish packet with packet id 3.  
30 3506 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=145.  
31 3506 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1015] De-serialized incoming PUBLISH packet: DeserializerResult=MQTTSuccess.  
32 3506 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1028] State record updated. New state=MQTTPublishDone.  
33 3906 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:814] UNSUBSCRIBE sent topic $aws/things/MyThingName/defender/metrics/json/accepted to broker.  
34 3986 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=2.  
35 3986 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:469] MQTT_PACKET_TYPE_UNSUBACK.  
36 4586 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:814] UNSUBSCRIBE sent topic $aws/things/MyThingName/defender/metrics/json/rejected to broker.  
37 4646 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=2.  
38 4646 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:469] MQTT_PACKET_TYPE_UNSUBACK.  
39 5246 [iot_thread] [INFO] [MQTT] [core_mqtt.c:2118] Disconnected from the broker.  
40 5246 [iot_thread] [INFO] [DEMO][5246] memory_metrics::freertos_heap::before::bytes::2088152  
[INFO ][DEMO][5246] memory_metrics::freertos_heap::after::bytes::1986576  
41 5246 [iot_thread] [INFO] [DEMO][5246] memory_metrics::demo_task_stack::before::bytes::1908  
[INFO ][DEMO][5246] memory_metrics::demo_task_stack::after::bytes::1908  
42 6246 [iot_thread] [INFO] [DEMO][6246] Demo completed successfully.  
[INFO ][INIT][6248] SDK cleanup done.  
43 6248 [iot_thread] [INFO] [DEMO][6248] -----DEMO FINISHED-----
```

如果您的主機板未使用 FreeRTOS+TCP 或支援的 lwIP 組態，輸出將如下所示。

```
15 1528 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:632] MQTT connection successfully established with broker.
16 1528 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:660] A clean MQTT connection is established. Cleaning up all the
stored outgoing publishes.
17 1528 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:751] SUBSCRIBE topic $aws/things/MyThingName/defender/metrics/json/accepted to broker.
18 1568 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=3.
19 1568 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:463] MQTT_PACKET_TYPE_SUBACK.
20 2168 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:751] SUBSCRIBE topic $aws/things/MyThingName/defender/metrics/json/rejected to broker.
21 2248 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=3.
22 2248 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:463] MQTT_PACKET_TYPE_SUBACK.
23 2848 [iot_thread] [ERROR] [Device_Defender_Demo] [metrics_collector.c:82] Using stub definition of GetNetworkStats! Please implement for your network stack to get correct metrics.
24 2848 [iot_thread] [ERROR] [Device_Defender_Demo] [metrics_collector.c:110] Using stub definition of GetOpenTcpPorts! Please implement for your network stack to get correct metrics.
25 2848 [iot_thread] [ERROR] [Device_Defender_Demo] [metrics_collector.c:146] Using stub definition of GetOpenUdpPorts! Please implement for your network stack to get correct metrics.
26 2848 [iot_thread] [ERROR] [Device_Defender_Demo] [metrics_collector.c:183] Using stub definition of GetEstablishedConnections! Please implement for your network stack to get correct metrics.
27 2848 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:864] The published payload:{"header": {"report_id": 1093 "version": "1.0"}, "metrics": [{"listening_tcp_ports": {"ports": [], "total": 0}, "listening_udp_ports": {"ports": [], "total": 0}, "network_stats": {"bytes_in": 0, "bytes_out": 0, "packets_in": 0, "packets_out": 0}}]
28 2848 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:892] PUBLISH sent for topic $aws/things/MyThingName/defender/metrics/json to broker with packet ID 3.
29 2968 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=2.
30 2968 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1132] Ack packet deserialized with result: MQTTSuccess.
31 2968 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1145] State record updated. New state=MQTTPublishDone.
32 2968 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:484] PUBACK received for packet id 3.
33 2968 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:448] Cleaned up outgoing publish packet with packet id 3.
34 2968 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=145.
35 2968 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1015] De-serialized incoming PUBLISH packet: DeserializerResult=MQTTSuccess.
36 2968 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1028] State record updated. New state=MQTTPublishDone.
37 3368 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:814] UNSUBSCRIBE sent topic $aws/things/MyThingName/defender/metrics/json/accepted to broker.
38 3408 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=2.
39 3408 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:469] MQTT_PACKET_TYPE_UNSUBACK.
40 4008 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:814] UNSUBSCRIBE sent topic $aws/things/MyThingName/defender/metrics/json/rejected to broker.
41 4088 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=2.
42 4088 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:469] MQTT_PACKET_TYPE_UNSUBACK.
43 4688 [iot_thread] [INFO] [MQTT] [core_mqtt.c:2118] Disconnected from the broker.
44 4688 [iot_thread] [INFO] [DEMO][4688] memory_metrics::freertos_heap::before::bytes::2088152
[INFO ][DEMO][4688] memory_metrics::freertos_heap::after::bytes::1986760
45 4688 [iot_thread] [INFO] [DEMO][4688] memory_metrics::demo_task_stack::before::bytes::1908
[INFO ][DEMO][4688] memory_metrics::demo_task_stack::after::bytes::1908
46 5688 [iot_thread] [INFO] [DEMO][5688] Demo completed successfully.
[INFO ][INIT][5690] SDK cleanup done.
47 5690 [iot_thread] [INFO] [DEMO][5690] -----DEMO FINISHED-----
```

示範的來源碼位於 [freertos/demos/device_defender_for_aws/](#) 或 GitHub 網站上的下載。

描述 AWS IoT Device Defender 主題

函數 `subscribeToDefenderTopics` 會訂用 MQTT 主題，在該主題中，將會收到已發布 Device Defender 報告的回應。它使用巨集 `DEFENDER_API_JSON_ACCEPTED` 來建構主題字串，以在主題字串上接收接受的裝置保護器報告的回應。它使用巨集 `DEFENDER_API_JSON_REJECTED` 建構主題字串，用於接收遭拒裝置保護器報告的回應。

收集裝置指標

`collectDeviceMetrics` 函數使用 中定義的函數來收集聯網指標。`metrics_collector.h` 收集的指標包括傳送和接收的位元組數和封包數、開放的 TCP 連接埠，開放的 UDP 連接埠和已建立的 TCP 連接。

生成 AWS IoT Device Defender 報告

`generateDeviceMetricsReport` 函數使用 中定義的函數製作裝置保護器報告。`report_builder.h` 該函數會擷取聯網指標和緩衝區，並如預期般以 AWS IoT Device Defender 的格式建立 JSON 文件，並將其寫入所提供的緩衝區。預期的 JSON 文件格式在 AWS IoT Device Defender 的 [裝置端指標](#) 中已指定。AWS IoT 開發人員指南

發布 AWS IoT Device Defender 報告

報告會發布在 MQTT 主題上以發布 JSON AWS IoT Device Defender 報告。AWS IoT Device Defender 此報告是使用巨集 `DEFENDER_API_JSON_PUBLISH` 建構，如 [網站上的](#) 程式碼片段 GitHub 所示。

處理回應的回呼

`publishCallback` 函數會處理傳入的 MQTT 發布訊息。它會使用來自 `Defender_MatchTopic` 程式庫的 AWS IoT Device Defender API 來檢查傳入的 MQTT 訊息是否來自 AWS IoT Device Defender 服務。如果訊息來自 AWS IoT Device Defender，它會剖析收到的 JSON 回應並在回應中擷取報告 ID。然後，驗證報告 ID 與報告中傳送的報告相同。

AWS IoT Greengrass 探索示範應用程式

在執行適用於 FreeRTOS 的 AWS IoT Greengrass 探索示範之前，您需要設定 AWS、AWS IoT Greengrass 和 AWS IoT。若要設定 AWS，請依照 [設定 AWS 帳戶和許可 \(p. 15\)](#) 中的指示。若要設定 AWS IoT Greengrass，您需要建立 Greengrass 群組，然後新增 Greengrass 核心。如需設定 AWS IoT Greengrass 的詳細資訊，請參閱 [AWS IoT Greengrass 入門](#)。

在您設定 AWS 和 AWS IoT Greengrass 之後，您需要為 AWS IoT Greengrass 設定一些額外的許可。

若要設定 AWS IoT Greengrass 許可

1. 瀏覽至 [IAM 主控台](#)。
2. 請選擇導覽窗格上的 Roles (角色)，然後尋找並選擇 `Greengrass_ServiceRole` (`Greengrass_ServiceRole`)。
3. 選擇 Attach policies (連接政策)，選取 `AmazonS3FullAccess` 和 `AWSIoTFullAccessAttach policy` (連接政策)。
4. 瀏覽至 [AWS IoT 主控台](#)。
5. 在導覽窗格中，依序選擇 Greengrass (Greengrass)、Groups (群組)，然後選擇您先前建立的 Greengrass 群組。
6. 選擇 Settings (設定)，然後選擇 Add role (新增角色)。
7. 選擇 `Greengrass_ServiceRole` (`Greengrass_ServiceRole`)，然後選擇 Save (儲存)。

您可以使用 主控台中的 [Quick Connect \(快速連接\)](#) FreeRTOS 工作流程來快速將主機板連接到 並執行示範。目前下列主機板無法使用 組態：AWS IoT FreeRTOS

- Cypress CYW943907AEVAL1F 開發套件
- Cypress CYW954907AEVAL1F 開發套件
- Espressif ESP-WROVER-KIT
- Espressif ESP32-DevKitC
- Nordic nRF52840-DK

您也可以將主機板連接至 AWS IoT 並手動設定 FreeRTOS 示範。

1. 向 AWS IoT 註冊您的 MCU 電路板 (p. 15)

註冊主機板後，您需要建立新的 Greengrass 政策並將其與裝置憑證連接。

建立新 AWS IoT Greengrass 政策

1. [瀏覽至 AWS IoT 主控台](#)。
2. 在導覽窗格中，選擇 Secure (安全)、Policies (政策)，然後選擇 Create (建立)。
3. 輸入可識別政策的名稱。
4. 在 Add statements (新增陳述式) 區段中，選擇 Advanced mode (進階模式)。將下列 JSON 複製並貼入政策編輯器視窗：

```
{  
    "Effect": "Allow",  
    "Action": [  
        "greengrass:*"  
    ],  
    "Resource": "*"  
}
```

此政策授予 AWS IoT Greengrass 對所有資源的許可。

5. 選擇 Create (建立)。

將 AWS IoT Greengrass 政策連接至裝置憑證

1. [瀏覽至 AWS IoT 主控台](#)。
2. 在導覽窗格中，依序選擇 Manage (管理)、Things (實物)，然後選擇您之前建立的實物。
3. 選擇 Security (安全)，然後選擇您裝置所連接的憑證。
4. 依序選擇 Policies (政策)、Actions (動作)，然後選擇 Attach Policy (連接政策)。
5. 尋找並選擇您之前建立的 Greengrass 政策，然後選擇 Attach (連接)。

2. 下載 FreeRTOS (p. 17)

Note

如果您是從 FreeRTOS 主控台下載 FreeRTOS，請選擇 Connect to AWS IoT Greengrass- (連接到 -) **Platform**而不是 Connect to -AWS IoT **Platform**.

3. 設定 FreeRTOS 示範 (p. 18).

開啟 `freertos/vendors/vendor/boards/board/aws_demos/config_files/`
`aws_demo_config.h`、註解 `#define CONFIG_MQTT_DEMO_ENABLED`，然後定義
`CONFIG_GREENGRASS_DISCOVERY_DEMO_ENABLED`。

在您完成 AWS IoT 和 AWS IoT Greengrass 的設定，並下載與配置 FreeRTOS 後，就可以在裝置中建立、刷新及執行 Greengrass 示範。若要設定主機板的硬體和軟體開發環境，請依照[主機板特定的入門指南 \(p. 28\)](#)中的指示操作。

Greengrass 示範會將一系列的訊息發佈到 Greengrass 核心以及 AWS IoT MQTT 用戶端。要查看 AWS IoT MQTT 用戶端中的訊息，請開啟 [AWS IoT 主控台](#)，選擇 Test (測試)，然後將訂閱新增至 `freertos/demos/ggd`。

在 MQTT 用戶端中，您應該會看到下列字串：

```
Message from Thing to Greengrass Core: Hello world msg #1!
```

```
Message from Thing to Greengrass Core: Hello world msg #0!
Message from Thing to Greengrass Core: Address of Greengrass Core found! 123456789012.us-
west-2.compute.amazonaws.com
```

使用 Amazon EC2 執行個體

如果您使用的是 Amazon EC2 執行個體

- 找出與 Amazon EC2 執行個體關聯的公有 DNS (IPv4) — 前往 Amazon EC2 主控台，然後在左側導覽面板中選擇 Instances (執行個體)。選擇您的 Amazon EC2 執行個體，然後選擇 Description (描述) 面板。尋找 Public DNS (IPv4) (公有 DNS (IPv4)) 的項目，並記下該項目。
- 尋找 Security groups (安全群組) 的項目，然後選擇連接到 Amazon EC2 執行個體的安全群組。
- 選擇 Inbound rules (傳入規則) 標籤，然後選擇 Edit inbound rules (編輯傳入規則) 並新增下列規則。

傳入規則

類型	通訊協定	連接埠範圍	來源	描述 - 選用
HTTP	TCP	80	0.0.0.0/0	-
HTTP	TCP	80	::/0	-
SSH	TCP	22	0.0.0.0/0	-
自訂 TCP	TCP	8883	0.0.0.0/0	MQTT 通訊
自訂 TCP	TCP	8883	::/0	MQTT 通訊
HTTPS	TCP	443	0.0.0.0/0	-
HTTPS	TCP	443	::/0/0	-
所有 ICMP - IPv4	ICMP	全部	0.0.0.0/0	-
所有 ICMP - IPv4	ICMP	全部	::/0/0	-

- 在 AWS IoT 主控台中選擇 Greengrass，然後選擇 Groups (群組)，然後選擇您先前建立的 Greengrass 群組。選擇 Settings (設定)。將 Local connection detection (區域連線偵測) 變更為 Manually manage connection information (手動管理連線資訊)。
- 在導覽窗格中，選擇 Cores (核心)，然後選取群組核心。
- 選擇 Connectivity (連線)，並確定您只有一個核心端點 (刪除所有其他端點)，而且它不是 IP 地址 (因為它可能會變更)。最好的選擇是使用您在第一個步驟中記下的公有 DNS (IPv4)。
- 將您建立的 FreeRTOS IoT 實物新增到 GG 群組。
 - 選擇向後箭頭以返回 AWS IoT Greengrass 群組頁面。在導覽窗格中，選擇 Devices (裝置)，然後選擇 Add Device (新增裝置)。
 - 選擇 Select an IoT Thing (選取實物)。選擇您的裝置，然後選擇 Finish (完成)。
- 新增必要的訂閱— 在 Greengrass Group (Greengrass 群組) 頁面中，選擇 Subscriptions (訂閱)，然後選擇 Add Subscription (新增訂閱) 並輸入如下所示的資訊。

Subscriptions

來源	目標	主題
TIGG1	IoT 雲端	freertos/demos/ggd

- 啟動 AWS IoT Greengrass 群組的部署，並確定部署成功。您現在應該可以成功地執行 AWS IoT Greengrass 探索示範。

coreHTTP 示範

主題

- coreHTTP 交互身份驗證示範 (p. 232)
- 基本 coreHTTP 上傳示範Amazon S3 (p. 236)
- coreHTTP 基本 S3 下載示範 (p. 237)

coreHTTP 交互身份驗證示範

Introduction

(相互身份驗證) 示範專案會示範如何使用 TLS 建立 HTTP 伺服器，並在使用者與伺服器之間進行交互身份驗證。coreHTTP此示範使用以 mbedTLS 為基礎的傳輸界面實作來建立伺服器和使用者驗證的 TLS 連接，並示範 HTTP 中的請求回應工作流程。

Note

若要設定和執行 FreeRTOS 示範，請遵循[FreeRTOS 入門 \(p. 14\)](#)中的步驟。

Functionality

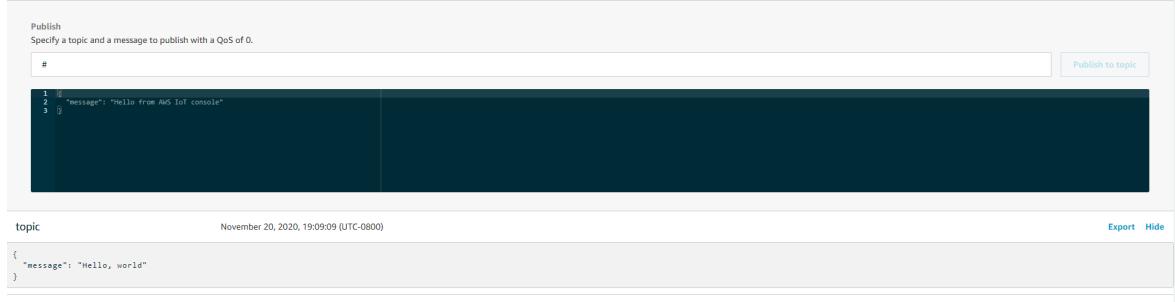
此示範會建立單一應用程式任務，其中包含示範如何完成下列項目的範例：

- 連接至 AWS IoT 端點上的 HTTP 伺服器
- 傳送 POST 請求
- 接收回應
- 中斷與伺服器的連接

完成這些步驟後，示範會建立類似以下螢幕擷取畫面的輸出。

```
9 1565 [iot_thread] [INFO ][DEMO][1565] -----STARTING DEMO-----
10 1566 [iot_thread] [INFO ][INIT][1566] SDK successfully initialized.
11 1566 [iot_thread] [INFO ][DEMO][1566] Successfully initialized the demo. Network type for the demo: 4
12 1566 [iot_thread] [INFO ][HTTPDemo] [http_demo_mutual_auth.c:319] 13 1566 [iot_thread] Establishing a TLS session to a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com:8443.14 1566 [iot_thread]
15 1622 [iot_thread] DNS[0x6BF5]: The answer to "a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com" (35.161.102.88) will be stored
16 1622 [iot_thread] DNS[0x6BF5]: The answer to "a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com" (35.161.102.88) will be stored
17 1622 [iot_thread] DNS[0x6BF5]: The answer to "a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com" (44.239.101.16) will be stored
18 1622 [iot_thread] DNS[0x6BF5]: The answer to "a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com" (44.239.97.33) will be stored
19 1622 [iot_thread] DNS[0x6BF5]: The answer to "a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com" (44.238.43.70) will be stored
20 1622 [iot_thread] DNS[0x6BF5]: The answer to "a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com" (52.32.144.134) will be stored
21 2082 [iot_thread] [INFO ][HTTPDemo] [http_demo_mutual_auth.c:393] 22 2082 [iot_thread] Sending HTTP POST request to a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com/topics/topic?qos=1...
24 2082 [iot_thread] [INFO ][HTTPDemo] [http_demo_mutual_auth.c:418] 25 2082 [iot_thread] Received HTTP response from a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com/topics/topic?qos=1...
26 2082 [iot_thread]
27 2082 [iot_thread] [INFO ][HTTPDemo] [http_demo_mutual_auth.c:283] 28 2082 [iot_thread] Demo completed successfully.29 2082 [iot_thread]
30 2082 [iot_thread] [INFO ][DEMO][2082] memory metrics::freerete_heap::before::bytes::2088152
31 2082 [iot_thread] [INFO ][DEMO][2082] memory metrics::freerete_heap::after::bytes::2088152
32 2082 [iot_thread] [INFO ][DEMO][2082] memory metrics::task_executefree::bytes::1908
33 2082 [iot_thread] [INFO ][DEMO][2082] memory metrics::demo_task_stack::after::bytes::1908
34 2082 [iot_thread] [INFO ][DEMO][2082] Demo completed successfully.
35 3084 [iot_thread] [INFO ][INIT][3084] SDK cleanup done.
36 3084 [iot_thread] [INFO ][DEMO][3084] -----DEMO FINISHED-----
```

主控台會建立類似以下螢幕擷取畫面的輸出。AWS IoT



以下範例是示範的結構。

```
int RunCoreHttpMutualAuthDemo( bool awsIotMqttMode,
                               const char * pIdentifier,
                               void * pNetworkServerInfo,
                               void * pNetworkCredentialInfo,
                               const IotNetworkInterface_t * pNetworkInterface )
{
    /* The transport layer interface used by the HTTP Client library. */
    TransportInterface_t xTransportInterface;
    /* The network context for the transport layer interface. */
    NetworkContext_t xNetworkContext = { 0 };
    TransportSocketStatus_t xNetworkStatus;
    BaseType_t xIsConnectionEstablished = pdFALSE;

    /* Upon return, pdPASS will indicate a successful demo execution.
     * pdFAIL will indicate some failures occurred during execution. The
     * user of this demo must check the logs for any failure codes. */
    BaseType_t xDemoStatus = pdPASS;

    /* Remove compiler warnings about unused parameters. */
    ( void ) awsIotMqttMode;
    ( void ) pIdentifier;
    ( void ) pNetworkServerInfo;
    ( void ) pNetworkCredentialInfo;
    ( void ) pNetworkInterface;

    /***** Connect. *****/
    /* Attempt to connect to the HTTP server. If connection fails, retry after a
     * timeout. The timeout value will be exponentially increased until either
     * the maximum number of attempts or the maximum timeout value is reached.
     * The function returns pdFAIL if the TCP connection cannot be established
     * with the broker after configured number of attempts. */
    xDemoStatus = connectToServerWithBackoffRetries( prvConnectToServer,
                                                    &xNetworkContext );

    if( xDemoStatus == pdPASS )
    {
        /* Set a flag indicating that a TLS connection exists. */
        xIsConnectionEstablished = pdTRUE;

        /* Define the transport interface. */
        xTransportInterface.pNetworkContext = &xNetworkContext;
        xTransportInterface.send = SecureSocketsTransport_Send;
        xTransportInterface.recv = SecureSocketsTransport_Recv;
    }
    else
    {
        /* Log error to indicate connection failure after all
         * reconnect attempts are over. */
        LogError( ( "Failed to connect to HTTP server %.*s.",
                    ( int32_t ) httpexampleAWS_IOT_ENDPOINT_LENGTH,
                    democonfigAWS_IOT_ENDPOINT ) );
    }

    /***** Send HTTP request. *****/
    if( xDemoStatus == pdPASS )
    {
        xDemoStatus = prvSendHttpRequest( &xTransportInterface,
                                         HTTP_METHOD_POST,
                                         httpexampleHTTP_METHOD_POST_LENGTH,
                                         democonfigPOST_PATH,
                                         democonfigPOST_PAYLOAD );
    }
}
```

```
        httpexamplePOST_PATH_LENGTH );
}

/********************* Disconnect. *****/
/* Close the network connection to clean up any system resources that the
 * demo may have consumed. */
if( xIsConnectionEstablished == pdTRUE )
{
    /* Close the network connection. */
    xNetworkStatus = SecureSocketsTransport_Disconnect( &xNetworkContext );

    if( xNetworkStatus != TRANSPORT_SOCKET_STATUS_SUCCESS )
    {
        xDemoStatus = pdFAIL;
        LogError( ( "SecureSocketsTransport_Disconnect() failed to close the network
connection. "
                    "StatusCode=%d.", ( int ) xNetworkStatus ) );
    }
}

if( xDemoStatus == pdPASS )
{
    LogInfo( ( "Demo completed successfully." ) );
}

return ( xDemoStatus == pdPASS ) ? EXIT_SUCCESS : EXIT_FAILURE;
}
```

連接到 AWS IoT HTTP 伺服器

函數 [會嘗試對 connectToServerWithBackoffRetries](#) HTTP 伺服器進行相互驗證的 TLS 連接。AWS IoT如果連接失敗，則會在逾時後重試。逾時值會以指數增加，直到達到嘗試次數上限或達到逾時值上限為止。函數提供指數增加的逾時值，並在達到最大嘗試次數時傳回 `RetryUtils_BackoffAndSleep`。`RetryUtilsRetriesExhausted`如果無法在設定的嘗試次數後建立代理程式的 TLS 連接，則 `connectToServerWithBackoffRetries` 函數會傳回失敗狀態。

傳送 HTTP 請求和接收回應

函數示範如何傳送 POST 請求到 `prvSendHttpRequest` HTTP 伺服器。AWS IoT如需在 AWS IoT 中對 REST API 提出請求的詳細資訊，請查看[裝置通訊協定 - HTTPS](#)。使用相同的 coreHTTP API 呼叫 (`HTTPClient_Send`) 收到回應。

```
static BaseType_t prvSendHttpRequest( const TransportInterface_t * pxTransportInterface,
                                       const char * pcMethod,
                                       size_t xMethodLen,
                                       const char * pcPath,
                                       size_t xPathLen )
{
    /* Return value of this method. */
    BaseType_t xStatus = pdPASS;

    /* Configurations of the initial request headers that are passed to
     * #HTTPClient_InitializeRequestHeaders. */
    HTTPRequestInfo_t xRequestInfo;
    /* Represents a response returned from an HTTP server. */
    HTTPResponse_t xResponse;
    /* Represents header data that will be sent in an HTTP request. */
    HTTPRequestHeaders_t xRequestHeaders;

    /* Return value of all methods from the HTTP Client library API. */
```

```
HTTPStatus_t xHTTPStatus = HTTPSuccess;

configASSERT( pcMethod != NULL );
configASSERT( pcPath != NULL );

/* Initialize all HTTP Client library API structs to 0. */
( void ) memset( &xRequestInfo, 0, sizeof( xRequestInfo ) );
( void ) memset( &xResponse, 0, sizeof( xResponse ) );
( void ) memset( &xRequestHeaders, 0, sizeof( xRequestHeaders ) );

/* Initialize the request object. */
xRequestInfo.pHost = democonfigAWS_IOT_ENDPOINT;
xRequestInfo.hostLen = httpexampleAWS_IOT_ENDPOINT_LENGTH;
xRequestInfo.pMethod = pcMethod;
xRequestInfo.methodLen = xMethodLen;
xRequestInfo.pPath = pcPath;
xRequestInfo.pathLen = xPathLen;

/* Set "Connection" HTTP header to "keep-alive" so that multiple requests
 * can be sent over the same established TCP connection. */
xRequestInfo.reqFlags = HTTP_REQUEST_KEEP_ALIVE_FLAG;

/* Set the buffer used for storing request headers. */
xRequestHeaders.pBuffer = ucUserBuffer;
xRequestHeaders.bufferLen = democonfigUSER_BUFFER_LENGTH;

xHTTPStatus = HTTPClient_InitializeRequestHeaders( &xRequestHeaders,
                                                &xRequestInfo );

if( xHTTPStatus == HTTPSuccess )
{
    /* Initialize the response object. The same buffer used for storing
     * request headers is reused here. */
    xResponse.pBuffer = ucUserBuffer;
    xResponse.bufferLen = democonfigUSER_BUFFER_LENGTH;

    LogInfo( ( "Sending HTTP %.*s request to %.*s%.*s...",
               ( int32_t ) xRequestInfo.methodLen, xRequestInfo.pMethod,
               ( int32_t ) httpexampleAWS_IOT_ENDPOINT_LENGTH,
               democonfigAWS_IOT_ENDPOINT,
               ( int32_t ) xRequestInfo.pathLen, xRequestInfo.pPath ) );
    LogDebug( ( "Request Headers:\n%.*s\n",
               "Request Body:\n%.*s\n",
               ( int32_t ) xRequestHeaders.headersLen,
               ( char * ) xRequestHeaders.pBuffer,
               ( int32_t ) httpexampleREQUEST_BODY_LENGTH, democonfigREQUEST_BODY ) );

    /* Send the request and receive the response. */
    xHTTPStatus = HTTPClient_Send( pxTransportInterface,
                                 &xRequestHeaders,
                                 ( uint8_t * ) democonfigREQUEST_BODY,
                                 httpexampleREQUEST_BODY_LENGTH,
                                 &xResponse,
                                 0 );
}
else
{
    LogError( ( "Failed to initialize HTTP request headers: Error=%s.",
                HTTPClient_strerror( xHTTPStatus ) ) );
}

if( xHTTPStatus == HTTPSuccess )
{
    LogInfo( ( "Received HTTP response from %.*s%.*s...\n",
               democonfigAWS_IOT_ENDPOINT,
               ( int32_t ) xResponse.statusCode,
               ( int32_t ) xResponse.statusTextLen, xResponse.pStatusText ) );
}
```

```
( int32_t ) httpexampleAWS_IOT_ENDPOINT_LENGTH,
democonfigAWS_IOT_ENDPOINT,
( int32_t ) xRequestInfo.pathLen, xRequestInfo.pPath );
LogDebug( ( "Response Headers:\n%.*s\n",
( int32_t ) xResponse.headersLen, xResponse.pHeaders ) );
LogDebug( ( "Status Code:\n%u\n",
xResponse.statusCode ) );
LogDebug( ( "Response Body:\n%.*s\n",
( int32_t ) xResponse.bodyLen, xResponse.pBody ) );
}
else
{
    LogError( ( "Failed to send HTTP %.*s request to %.*s%.*s: Error=%s.",
( int32_t ) xRequestInfo.methodLen, xRequestInfo.pMethod,
( int32_t ) httpexampleAWS_IOT_ENDPOINT_LENGTH,
democonfigAWS_IOT_ENDPOINT,
( int32_t ) xRequestInfo.pathLen, xRequestInfo.pPath,
HTTPClient_strerror( xHttpStatus ) ) );
}
if( xHttpStatus != HTTPSuccess )
{
    xStatus = pdFAIL;
}
return xStatus;
}
```

基本 coreHTTP 上傳示範Amazon S3

Introduction

此範例示範如何傳送 PUT 請求至 Amazon Simple Storage Service (Amazon S3) HTTP 伺服器並上傳小型檔案。它還會執行 GET 請求，以在上傳之後驗證檔案大小。此範例使用[網路傳輸界面](#)，其使用 mbedTLS 在執行 IoT 和 coreHTTP HTTP 伺服器之 Amazon S3 裝置使用者端之間建立相互驗證的連接。

Note

若要設定和執行 FreeRTOS 示範，請遵循[FreeRTOS 入門 \(p. 14\)](#)中的步驟。

單一執行緒與多執行緒

使用模型有兩種：coreHTTP單一執行緒和多執行緒（多任務）。雖然本節中的示範會在執行緒中執行 HTTP 程式庫，但它其實會示範如何在單一執行緒環境中使用 coreHTTP。此示範中只有一個任務使用 HTTP API。雖然單一執行緒應用程式必須重複呼叫 HTTP 程式庫，但多執行緒應用程式可以在代理程式（或精靈）任務的背景下傳送 HTTP 請求。

來源碼組織

示範來源檔案名為 `http_demo_s3_upload.c`，可在 [freertos/demos/coreHTTP/](#) 及 [GitHub](#) 網站中找到。

設定 Amazon S3 HTTP 伺服器連接

此示範使用預先簽章的 URL 連接到 Amazon S3 HTTP 伺服器，並授權物件下載存取權。HTTP 伺服器的 TLS 連接僅使用伺服器身份驗證。Amazon S3在應用程式層級，物件的存取會使用預先簽章的 URL 查詢中的參數進行驗證。請依照下列步驟來設定您的 AWS 連接。

1. 設定 AWS 帳號：

- 如果您尚未建立 帳號，請建立 AWS。

- b. 使用 AWS Identity and Access Management () 設定帳號和許可。IAM 您可以使用 IAM 來管理您帳號中每個使用者的許可。根據預設，在根擁有者授予之前，使用者不具許可。
 - i. 若要新增 IAM 使用者到您的 AWS 帳號，請查看 [IAM 使用者指南](#)。
 - ii. 新增此政策，以授予 AWS 存取 FreeRTOS 和 AWS IoT 的許可：
 - AmazonS3FullAccess
2. 依照 Amazon S3 中的 [如何建立 S3 儲存貯體？](#) 中的步驟，在 中建立儲存貯體。Amazon Simple Storage Service 主控台使用者指南
3. 依照 [Amazon S3 如何將檔案與資料夾上傳至 S3 儲存貯體？](#) 中的步驟，[將檔案上傳至](#)。
4. 使用位於 `FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/presigned_urls_gen.py` 檔案的指令碼建立預先簽章的 URL。如需用法指示，請查看 `FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/README.md` 檔案。

Functionality

示範會先使用 TLS 伺服器身份驗證連接至 Amazon S3 HTTP 伺服器。然後，它會建立 HTTP 請求以上傳 `democonfigDEMO_HTTP_UPLOAD_DATA` 中指定的資料。檔案上傳後，它會檢查檔案是否已成功上傳，其請求檔案大小。您可以在 [GitHub 網站上找到示範的來源碼](#)。

連接到 Amazon S3 HTTP 伺服器

函數 [會嘗試將 TCP 連接 HTTP 伺服器](#)。`connectToServerWithBackoffRetries` 如果連接失敗，則會在逾時後重試。逾時值會以指數增加，直到達到嘗試次數上限或達到逾時值上限為止。如果無法在設定的嘗試次數後建立連到伺服器的 TCP，則 `connectToServerWithBackoffRetries` 函數會傳回失敗狀態。

函數示範如何僅使用伺服器驗證來建立 `prvConnectToServer` HTTP 伺服器的連接。Amazon S3 它使用以 mbedTLS 為基礎的傳輸界面，此界面是在 `FreeRTOS-Plus/Source/Application-Protocols/network_transport/freertos_plus_tcp/using_mbedtls/using_mbedtls.c` 檔案中實作的。您可以在 [prvConnectToServer 網站上找到 GitHub 的定義](#)。

上傳資料

函數示範如何建立 PUT 要求，並指定要上傳的檔案。`prvUploadS3ObjectFile` 已在預先簽章的 URL 中指定檔案上傳的 Amazon S3 儲存貯體以及要上傳的檔案名稱。為了節省記憶體，相同的緩衝區會同時用於請求標頭及接收回應。使用 `HTTPClient_Send` API 函數以同步方式收到回應。回應狀態碼預期來自 200 OK HTTP 伺服器。Amazon S3 任何其他狀態碼都是錯誤。

您可以在 `prvUploadS3ObjectFile()` 網站上找到 [GitHub 的原始程式碼](#)。

驗證上傳

函數會呼叫 `prvVerifyS3ObjectFileSize` 以擷取 S3 儲存貯體中的物件大小。`prvGetS3ObjectFileSizeHTTP` 伺服器目前不支援使用預先簽章的 URL 提出 HEAD 請求，因此會請求第 0 個位元組。Amazon S3 該檔案大小會包含在回應的 Content-Range 標頭欄位中。預期從伺服器回應 206 Partial Content。Any 其他回應狀態碼是錯誤。

您可以在 `prvGetS3ObjectFileSize()` 網站上找到 [GitHub 的原始程式碼](#)。

coreHTTP 基本 S3 下載示範

Introduction

此示範示範如何使用 [範圍請求](#)，從 Amazon S3 HTTP 伺服器下載檔案。當您使用 coreHTTP 建立 HTTP 請求時，`HTTPClient_AddRangeHeader` API 原生支援範圍請求。針對微控制器環境，高度鼓勵範圍請求。

藉由以不同範圍下載大型檔案，而不是在單一請求中，處理檔案的每個區段時，可以不封鎖網路通訊端。範圍請求可降低封包捨棄的風險，這需要在 TCP 連接上重新傳輸，因而改善裝置的耗電量。

此範例使用 [網路傳輸界面](#)，其使用 mbedTLS 在執行 IoT 和 coreHTTP HTTP 伺服器的 Amazon S3 裝置使用者端之間建立相互驗證的連接。

Note

若要設定和執行 FreeRTOS 示範，請遵循[FreeRTOS 入門 \(p. 14\)](#)中的步驟。

單一執行緒與多執行緒

使用模型有兩種：coreHTTP單一執行緒和多執行緒（多任務）。雖然本節中的示範是在執行緒中執行 HTTP 程式庫，但它其實會示範如何在單一執行緒環境中使用 coreHTTP（只有一個任務在示範中使用 HTTP API）。雖然單一執行緒應用程式必須重複呼叫 HTTP 程式庫，但多執行緒應用程式可以在代理程式（或精靈）任務的背景下傳送 HTTP 請求。

來源碼組織

示範專案名為 `http_demo_s3_download.c`，可在 [freertos/demos/coreHTTP/](#) 及 [GitHub](#) 網站上找到。

設定 Amazon S3 HTTP 伺服器連接

此示範使用預先簽章的 URL 連接到 Amazon S3 HTTP 伺服器，並授權物件下載存取權。HTTP 伺服器的 TLS 連接僅使用伺服器身份驗證。Amazon S3在應用程式層級，物件的存取會使用預先簽章的 URL 查詢中的參數進行驗證。請遵循下列步驟來設定您的 AWS 連接。

1. 設定 Amazon Web Services (AWS) 帳號：
 - a. 如果您尚未建立和執行 [帳號AWS](#)，請現在建立。
 - b. 使用 AWS Identity and Access Management (IAM) 設定帳號和許可。IAM 可讓您管理帳號中每個使用者的許可。根據預設，在根擁有者授予之前，使用者不具許可。
 - i. 若要新增 IAM 使用者到您的 AWS 帳號，請查看 [IAM 使用者指南](#)。
 - ii. 新增這些政策，以授予 AWS 帳號存取 FreeRTOS 和 AWS IoT 的許可：
 - `AmazonS3FullAccess`
2. 依照《Amazon Simple Storage Service 主控台使用者指南》[中](#)如何建立 S3 儲存貯體的步驟，在 S3 中建立儲存貯體。
3. 依照[如何將檔案與資料夾上傳至 S3 儲存貯體？](#)中的步驟，將檔案上傳至 S3。
4. 使用位於 `FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/presigned_urls_gen.py` 的指令碼建立預先簽章的 URL。如需用法指示，請查看 `FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/README.md`。

Functionality

示範會先擷取檔案的大小。然後，它會在迴圈中以範圍大小為 `democonfigRANGE_REQUEST_LENGTH` 的順序請求每個位元組範圍。

您可以在 [GitHub](#) 網站上找到示範的來源碼。

連接到 Amazon S3 HTTP 伺服器

函數 `() connectToServerWithBackoffRetries` 會嘗試將 TCP 連接 HTTP 伺服器。如果連接失敗，則會在逾時後重試。逾時值會以指數增加，直到達到嘗試次數上限或達到逾時值上限為止。如果無法在設定的嘗試次數後建立連到伺服器的 TCP，`connectToServerWithBackoffRetries()` 會傳回失敗狀態。

函數 `prvConnectToServer()` 示範如何使用伺服器驗證建立 Amazon S3 HTTP 伺服器的連接。它使用以 mbedTLS 為基礎的傳輸界面，該界面以 [-Plus/Source/Application-Protocols/network_transport/freertos_plus_tcp/using_mbedtls/using_mbedtls.c](#)FreeRTOS 檔案實作。

您可以在 `prvConnectToServer()` GitHub 上找到 的原始程式碼。

建立範圍請求

API 函數 `HTTPClient_AddRangeHeader()` 支援將位元組範圍序列化到 HTTP 請求標頭，以形成範圍請求。此示範使用範圍請求來擷取檔案大小，以及請求檔案的每個區段。

函數 `prvGetS3ObjectFileSize()` 會擷取 S3 儲存貯體中檔案大小。此第一個請求會新增 `Connection: keep-alive` 標頭，以便在傳送回應之後讓連接保持開放。Amazon S3S3 HTTP 伺服器目前不支援使用預先簽章的 URL 提出 HEAD 請求，因此會請求第 0 個位元組。該檔案大小會包含在回應的 `Content-Range` 標頭欄位中。預期從伺服器回應；收到的任何其他回應狀態碼都是錯誤。206 Partial Content

您可以在 `prvGetS3ObjectFileSize()` GitHub 上找到 的原始程式碼。

擷取檔案大小之後，此示範會為要下載的檔案的每個位元組範圍建立新的範圍請求。它會使用 `HTTPClient_AddRangeHeader()` 來存放檔案的每個區段。

傳送範圍請求和接收回應

函數 `prvDownloadS3ObjectFile()` 會以迴圈傳送範圍請求，直到整個檔案下載完成。API 函數 `HTTPClient_Send()` 傳送請求並同步接收回應。當函式傳回時，`xResponse` 收到回應。接著，狀態碼會驗證為 206 Partial Content，而目前已下載的位元組數由 `Content-Length` 標頭值遞增。

您可以在 `prvDownloadS3ObjectFile()` GitHub 上找到 的原始程式碼。

AWS IoT 任務程式庫示範

Introduction

Jobs 程式庫示範示範如何透過 MQTT 連接連接至 AWS IoT Jobs 服務、從 AWS IoT 擷取任務，以及在裝置上處理它。AWS IoT 任務示範專案使用 AWS IoT Windows 連接埠FreeRTOS，因此可以使用 Windows 上的 Visual Studio Community 版本來建置和評估。不需要微型控制器硬體。示範使用 TLS 建立與 AWS IoTMQTT 交互身份驗證示範相同的 MQTT 中介裝置安全連接。

Note

若要設定和執行 FreeRTOS 示範，請遵循[FreeRTOS 入門 \(p. 14\)](#)中的步驟。

原始程式碼組織

示範程式碼位於 `jobs_demo.c` 檔案中，並可在 網站上或 GitHub 中找到。[freertos/demos/jobs_for_aws/](#)

設定 AWS IoT MQTT 中介裝置連接

在此示範中，您會使用 AWS IoT MQTT 中介裝置的 MQTT 連接。此連接的設定方式與 [MQTT 交互身份驗證示範](#)中相同。

Functionality

示範顯示從 AWS IoT 接收任務並在裝置上處理它們的工作流程。示範是互動式的，需要您使用 AWS IoT 主控台或 AWS Command Line Interface (AWS CLI) 建立任務。如需建立任務的詳細資訊，請前往

<https://docs.aws.amazon.com/cli/latest/reference/iot/create-job.html> 中的 create-job AWS CLI Command Reference。示範需要任務文件將 action 金鑰設定為 print，以列印訊息至主控台。

查看此任務文件的以下格式。

```
{  
    "action": "print",  
    "message": "ADD_MESSAGE_HERE"  
}
```

您可以使用 AWS CLI 來建立任務，如下列範例命令。

```
aws iot create-job \  
    --job-id t12 \  
    --targets arn:aws:iot:us-east-1:123456789012:thing/device1 \  
    --document '{"action":"print","message":"hello world!"}'
```

示範也會使用 action 金鑰設定為 publish 的任務文件，以重新發布訊息至主題。查看此任務文件的以下格式。

```
{  
    "action": "publish",  
    "message": "ADD_MESSAGE_HERE",  
    "topic": "topic/name/here"  
}
```

示範迴圈，直到收到 action 金鑰設定為 exit 的任務文件來結束示範為止。任務文件的格式如下。

```
{  
    "action": "exit"  
}
```

任務示範的進入點

您可以在 [GitHub](#) 上找到任務示範進入點函數的來源碼。此函數會執行下列操作：

1. 在 `mqtt_demo_helpers.c` 中使用協助程式函數建立 MQTT 連接。
2. 使用 `NextJobExecutionChanged` 中的協助程式函數，來獲得 `mqtt_demo_helpers.c` API 的 MQTT 主題。主題字串是先前彙總，使用由 AWS IoT 任務程式庫定義的巨集。
3. 使用 `StartNextPendingJobExecution` 中的協助程式函數，發布到 `mqtt_demo_helpers.c` API 的 MQTT 主題。主題字串之前彙總，使用由 AWS IoT 任務程式庫定義的巨集。
4. 重複呼叫 `MQTT_ProcessLoop` 來接收交給 `prvEventCallback` 處理的傳入訊息。
5. 示範接收結束動作後，請從 MQTT 主題取消描述，並使用 `mqtt_demo_helpers.c` 檔案中的協助程式函數中斷連結。

收到的 MQTT 訊息回呼

此函數會從 `Jobs_MatchTopic` 任務程式庫呼叫 AWS IoT，以分類傳入的 MQTT 訊息。如果訊息類型對應到新的任務，則呼叫 `prvNextJobHandler()`。

函數以及其呼叫的函數會從 JSON 格式訊息剖析任務文件，並執行任務指定的動作。`prvNextJobHandler` 尤其是 `prvSendUpdateForJob` 函數。

您可以在 [GitHub](#) 上找到傳入訊息的此回呼函數來源碼。

傳送執行中任務的更新

函數 `prvSendUpdateForJob()` 會從任務程式庫呼叫 `Jobs_Update()`，以立即填入 MQTT 發布操作中所使用的主題字串。

您可以在 `prvSendUpdateForJob()` 上找到 [GitHub](#) 函數的來源碼。

coreMQTT 交互身份驗證示範

Introduction

(相互身份驗證) 示範專案會示範如何使用 TLS 建立與 MQTT 中介裝置的連接，並在使用者與伺服器之間進行交互身份驗證。coreMQTT此示範使用以 mbedTLS 為基礎的傳輸界面實作來建立伺服器和由使用者驗證的 TLS 連接，並示範在 QoS 1 層級的 MQTT 訂用封裝工作流程。在它訂用單一主題篩選條件後，它會發布到相同主題，並等待從 QoS 1 層伺服器接收該訊息。無限期地重複發布至中介裝置以及從中介裝置接收相同訊息的這個循環。此示範中的訊息是傳送到 QoS 1，它可根據 MQTT 規格保證至少一次交付。

Note

若要設定和執行 FreeRTOS 示範，請遵循[FreeRTOS 入門 \(p. 14\)](#)中的步驟。

Functionality

示範會建立單一應用程式任務，循環執行一組範例，示範如何連接到中介裝置、訂用中介裝置上的主題、中介裝置上的主題，最後中斷與中介裝置的連接。示範應用程式都訂用及發布到相同主題。每次示範發布訊息至 MQTT 中介裝置時，中介裝置都會將相同的訊息傳回至示範應用程式。

成功完成示範後，將會得到類似下圖的輸出。

```
01 1548 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1798] 40 1548 [iot_thread] MQTT connection established with the broker.41 1548 [iot_thread]
02 1548 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:675] 43 1548 [iot_thread] An MQTT connection is established with a3c4bx1snco1p8-ats.iot.us-west-2.amazonaws.com:1548 [iot_thread]
05 1548 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:747] 46 1548 [iot_thread] Attempt to subscribe to the MQTT topic MyIOTThingTest5/example/topic.47
1548 [iot_thread]
08 1548 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:761] 49 1548 [iot_thread] SUBSCRIBE sent for topic MyIOTThingTest5/example/topic to broker.50 1548 [iot_thread]
09 1548 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:855] 52 1588 [iot_thread] Packet received. ReceivedBytes=3.53 1588 [iot_thread]
04 1588 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:907] 55 1588 [iot_thread] Subscribed to the topic MyIOTThingTest5/example/topic with maximum QoS 1.56 1588 [iot_thread]
07 2188 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:458] 58 2188 [iot_thread] Publish to the MQTT topic MyIOTThingTest5/example/topic.59 2188 [iot_thread]
08 2188 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:465] 61 2188 [iot_thread] Attempt to receive publish message from broker.62 2188 [iot_thread]
03 2248 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] 64 2248 [iot_thread] Packet received. ReceivedBytes=2.65 2248 [iot_thread]
06 2248 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1132] 67 2248 [iot_thread] Ack packet serialized with result: MQTTSuccess.68 2248 [iot_thread]
09 2248 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1145] 70 2248 [iot_thread] State record updated. New state=MQTTPublishDone.71 2248 [iot_thread]
07 2248 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:888] 73 2248 [iot_thread] PUBACK received for packet Id 2.74 2248 [iot_thread]
05 2248 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] 76 2248 [iot_thread] Packet received. ReceivedBytes=45.77 2248 [iot_thread]
08 2248 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1015] 79 2248 [iot_thread] De-serialized incoming PUBLISH packet. DeserializerResult=MQTTSuccess.80 2248 [iot_thread]
01 2248 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1028] 82 2248 [iot_thread] State record updated. New state=MQTTPubAckSend.83 2248 [iot_thread]
04 2248 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:958] 85 2248 [iot_thread] Incoming QoS : 1
06 2248 [iot_thread]
07 2248 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:969] 88 2248 [iot_thread] Incoming Publish Topic Name: MyIOTThingTest5/example/topic matches subscribed topic.Incoming Publish Message : Hello World!89 2248 [iot_thread]
08 2848 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:478] 91 2848 [iot_thread] Keeping Connection Idle...92 2848 [iot_thread]
03 4848 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:498] 94 4848 [iot_thread] Publish to the MQTT topic MyIOTThingTest5/example/topic.95 4848 [iot_thread]
06 4848 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:465] 97 4848 [iot_thread] Attempt to receive publish message from broker.98 4848 [iot_thread]
09 4888 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] 100 4888 [iot_thread] Packet received. ReceivedBytes=1.101 4888 [iot_thread]
02 4888 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1132] 103 4888 [iot_thread] Ack packet serialized with result: MQTTSuccess.104 4888 [iot_thread]
05 4888 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1145] 106 4888 [iot_thread] State record updated. New state=MQTTPublishDone.107 4888 [iot_thread]
08 4888 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:888] 109 4888 [iot_thread] PUBACK received for packet Id 3.110 4888 [iot_thread]
11 4928 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] 112 4928 [iot_thread] Packet received. ReceivedBytes=45.113 4928 [iot_thread]
14 4928 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1015] 115 4928 [iot_thread] De-serialized incoming PUBLISH packet. DeserializerResult=MQTTSuccess.116 4928 [iot_thread]
17 4928 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1028] 118 4928 [iot_thread] State record updated. New state=MQTTPubAckSend.119 4928 [iot_thread]
10 4928 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:958] 121 4928 [iot_thread] Incoming QoS : 1
02 4928 [iot_thread]
03 4928 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:969] 124 4928 [iot_thread] Incoming Publish Topic Name: MyIOTThingTest5/example/topic matches subscribed topic.Incoming Publish Message : Hello World!125 4928 [iot_thread]
16 5528 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:478] 127 5528 [iot_thread] Keeping Connection Idle...128 5528 [iot_thread]
```

主控台會建立類似下圖的輸出。AWS IoT

The screenshot shows the AWS IoT console interface. At the top, there's a 'Publish' section where a message is being sent to the topic '/example/topic'. The message content is a JSON string: "Hello World!". Below this, three published messages are listed:

- MyIOTThingTest5/example/topic** - Published at November 03, 2020, 13:03:57 (UTC-0800). The message content is "Hello World!".
- MyIOTThingTest5/example/topic** - Published at November 03, 2020, 13:03:52 (UTC-0800). The message content is "Hello World!".
- MyIOTThingTest5/example/topic** - Published at November 03, 2020, 13:03:47 (UTC-0800). The message content is "Hello World!".

Each message entry includes 'Export' and 'Hide' buttons.

以下是示範的結構。

```
static void prvMQTTDemoTask( void * pvParameters )
{
    uint32_t ulPublishCount = 0U, ulTopicCount = 0U;
    const uint32_t ulMaxPublishCount = 5UL;
    NetworkContext_t xNetworkContext = { 0 };
    NetworkCredentials_t xNetworkCredentials = { 0 };
    MQTTContext_t xMQTTContext = { 0 };
    MQTTStatus_t xMQTTStatus;
    TlsTransportStatus_t xNetworkStatus;

    /* Remove compiler warnings about unused parameters. */
    ( void ) pvParameters;

    /* Set the entry time of the demo application. This entry time will be used
     * to calculate relative time elapsed in the execution of the demo application,
     * by the timer utility function that is provided to the MQTT library.
     */
    ulGlobalEntryTimeMs = prvGetTimeMs();

    for( ; ; )
    {
        /***** Connect. *****/
        /* Attempt to establish TLS session with MQTT broker. If connection fails,
         * retry after a timeout. Timeout value will be exponentially increased
         * until the maximum number of attempts are reached or the maximum timeout
         * value is reached. The function returns a failure status if the TCP
         * connection cannot be established to the broker after the configured
         * number of attempts. */
        xNetworkStatus = prvConnectToServerWithBackoffRetries( &xNetworkCredentials,
                                                               &xNetworkContext );
        configASSERT( xNetworkStatus == TLS_TRANSPORT_SUCCESS );
    }
}
```

```
/* Sends an MQTT Connect packet over the already established TLS connection,
 * and waits for connection acknowledgment (CONNACK) packet. */
LogInfo( ( "Creating an MQTT connection to %s.\r\n",
democonfigMQTT_BROKER_ENDPOINT ) );
prvCreateMQTTConnectionWithBroker( &xMQTTContext, &xNetworkContext );

/********************* Subscribe. *********************/
/* If server rejected the subscription request, attempt to resubscribe to
 * topic. Attempts are made according to the exponential backoff retry
 * strategy implemented in retryUtils. */
prvMQTTSubscribeWithBackoffRetries( &xMQTTContext );

/********************* Publish and Keep Alive Loop. ********************/
/* Publish messages with QoS1, send and process Keep alive messages. */
for( ulPublishCount = 0; ulPublishCount < ulMaxPublishCount; ulPublishCount++ )
{
    LogInfo( ( "Publish to the MQTT topic %s.\r\n", mqttxampleTOPIC ) );
    prvMQTTPublishToTopic( &xMQTTContext );

    /* Process incoming publish echo, since application subscribed to the
     * same topic, the broker will send publish message back to the
     * application. */
    LogInfo( ( "Attempt to receive publish message from broker.\r\n" ) );
    xMQTTStatus = MQTT_ProcessLoop( &xMQTTContext,
mqtxamplePROCESS_LOOP_TIMEOUT_MS );
    configASSERT( xMQTTStatus == MQTTSuccess );

    /* Leave Connection Idle for some time. */
    LogInfo( ( "Keeping Connection Idle...\r\n\r\n" ) );
    vTaskDelay( mqttxampleDELAY_BETWEEN_PUBLISHES_TICKS );
}

/********************* Unsubscribe from the topic. ********************/
LogInfo( ( "Unsubscribe from the MQTT topic %s.\r\n", mqttxampleTOPIC ) );
prvMQTTUnsubscribeFromTopic( &xMQTTContext );

/* Process incoming UNSUBACK packet from the broker. */
xMQTTStatus = MQTT_ProcessLoop( &xMQTTContext,
mqtxamplePROCESS_LOOP_TIMEOUT_MS );
configASSERT( xMQTTStatus == MQTTSuccess );

/********************* Disconnect. *********************/
/* Send an MQTT Disconnect packet over the already connected TLS over
 * TCP connection. There is no corresponding response for the disconnect
 * packet. After sending disconnect, client must close the network
 * connection. */
LogInfo( ( "Disconnecting the MQTT connection with %s.\r\n",
democonfigMQTT_BROKER_ENDPOINT ) );
xMQTTStatus = MQTT_Disconnect( &xMQTTContext );
configASSERT( xMQTTStatus == MQTTSuccess );

/* Close the network connection. */
TLS_FreeRTOS_Disconnect( &xNetworkContext );

/* Reset SUBACK status for each topic filter after completion of
 * subscription request cycle. */
for( ulTopicCount = 0; ulTopicCount < mqttxampleTOPIC_COUNT; ulTopicCount++ )
{
    xTopicFilterContext[ ulTopicCount ].xSubAckStatus = MQTTSUBACKFailure;
}

/* Wait for some time between two iterations to ensure that we do not
 * bombard the broker. */
LogInfo( ( "prvMQTTDemoTask() completed an iteration successfully. " )
```

```
        "Total free heap is %u.\r\n",
        xPortGetFreeHeapSize() ) );
LogInfo( ( "Demo completed successfully.\r\n" ) );
LogInfo( ( "Short delay before starting the next iteration.... \r\n\r\n" ) );
vTaskDelay( mqttexampleDELAY_BETWEEN_DEMO_ITERATIONS_TICKS );
}
}
```

以指數退避與抖動重試邏輯

`prvBackoffForRetry` 函數顯示如何使用指數退避和抖動重試伺服器失敗的網路操作，例如，TLS 連接或 MQTT 訂用請求。該函數會計算下次重試嘗試的輪詢期間，如果重試嘗試尚未用盡，則會執行輪詢延遲。由於輪詢期間的計算需要生成隨機數字，因此函數使用 PKCS11 模組生成隨機數字。使用 PKCS11 模組允許存取 True Random Number Generator (TRNG) (如果廠商平台支援的話)。我們建議您將隨機數字發生器植入裝置特定的節流來源，以減少裝置在連接重試期間發生碰撞的機率。

連接到 MQTT 中介裝置

函數會嘗試對 MQTT 中介裝置進行相互驗證的 TLS 連接。`prvConnectToServerWithBackoffRetries` 如果連接失敗，它會在輪詢期間之後重試。輪詢期間會以指數增加，直到達到嘗試次數上限或達到輪詢期間上限為止。函數提供指數增加的輪詢值，並在達到最大嘗試次數時傳回 `BackoffAlgorithm_GetNextBackoff`。`RetryUtilsRetriesExhausted` 如果無法在設定的嘗試次數後建立代理程式的 TLS 連接，則 `prvConnectToServerWithBackoffRetries` 函數會傳回失敗狀態。

函數示範如何使用全新的工作階段建立 MQTT 連接到 MQTT 中介裝置。`prvCreateMQTTConnectionWithBroker` 它使用 TLS 傳輸界面，這會在 FreeRTOS-Plus/Source/Application-Protocols/platform/freertos/transport/src/tls_freertos.c 檔案中實作。函數的定義如下所示。`prvCreateMQTTConnectionWithBroker` 請記住，我們在 `xConnectInfo` 為中介裝置設定持續作用秒數。

下一個函數顯示如何使用 `MQTT_Init` 函數在 MQTT 細節中設定 TLS 傳輸界面和時間函數。它也會顯示事件回呼函數指標 (`prvEventCallback`) 如何設定。這個回呼會用來報告傳入的訊息。

```
static void prvCreateMQTTConnectionWithBroker( MQTTContext_t * pxMQTTContext,
                                              NetworkContext_t * pxNetworkContext )
{
    MQTTStatus_t xResult;
    MQTTConnectInfo_t xConnectInfo;
    bool xSessionPresent;
    TransportInterface_t xTransport;

    /**
     * For readability, error handling in this function is restricted to the use of
     * asserts().
     **/

    /* Fill in Transport Interface send and receive function pointers. */
    xTransport.pNetworkContext = pxNetworkContext;
    xTransport.send = TLS_FreeRTOS_send;
    xTransport.recv = TLS_FreeRTOS_recv;

    /* Initialize MQTT library. */
    xResult = MQTT_Init( pxMQTTContext,
                         &xTransport,
                         prvGetTimeMs,
                         prvEventCallback,
                         &xBuffer );
    configASSERT( xResult == MQTTSuccess );
```

```
/* Some fields are not used in this demo so start with everything at 0. */
( void ) memset( ( void * ) &xConnectInfo, 0x00, sizeof( xConnectInfo ) );

/* Start with a clean session i.e. direct the MQTT broker to discard any
 * previous session data. Also, establishing a connection with clean session
 * will ensure that the broker does not store any data when this client
 * gets disconnected. */
xConnectInfo.cleanSession = true;

/* The client identifier is used to uniquely identify this MQTT client to
 * the MQTT broker. In a production device the identifier can be something
 * unique, such as a device serial number. */
xConnectInfo.pClientIdentifier = democonfigCLIENT_IDENTIFIER;
xConnectInfo.clientIdentifierLength = ( uint16_t )
strlen( democonfigCLIENT_IDENTIFIER );

/* Set MQTT keep-alive period. If the application does not send packets at
 * an interval less than the keep-alive period, the MQTT library will send
 * PINGREQ packets. */
xConnectInfo.keepAliveSeconds = mqttexampleKEEP_ALIVE_TIMEOUT_SECONDS;

/* Append metrics when connecting to the AWS IoT Core broker. */
#ifndef democonfigUSE_AWS_IOT_CORE_BROKER
    #ifdef democonfigCLIENT_USERNAME
        xConnectInfo.pUserName = CLIENT_USERNAME_WITH_METRICS;
        xConnectInfo.userNameLength = ( uint16_t )
strlen( CLIENT_USERNAME_WITH_METRICS );
        xConnectInfo.pPassword = democonfigCLIENT_PASSWORD;
        xConnectInfo.passwordLength = ( uint16_t ) strlen( democonfigCLIENT_PASSWORD );
    #else
        xConnectInfo.pUserName = AWS_IOT_METRICS_STRING;
        xConnectInfo.userNameLength = AWS_IOT_METRICS_STRING_LENGTH;
        /* Password for authentication is not used. */
        xConnectInfo.pPassword = NULL;
        xConnectInfo.passwordLength = 0U;
    #endif
#else /* ifdef democonfigUSE_AWS_IOT_CORE_BROKER */
    #ifdef democonfigCLIENT_USERNAME
        xConnectInfo.pUserName = democonfigCLIENT_USERNAME;
        xConnectInfo.userNameLength = ( uint16_t ) strlen( democonfigCLIENT_USERNAME );
        xConnectInfo.pPassword = democonfigCLIENT_PASSWORD;
        xConnectInfo.passwordLength = ( uint16_t ) strlen( democonfigCLIENT_PASSWORD );
    #endif /* ifdef democonfigCLIENT_USERNAME */
#endif /* ifdef democonfigUSE_AWS_IOT_CORE_BROKER */

/* Send MQTT CONNECT packet to broker. LWT is not used in this demo, so it
 * is passed as NULL. */
xResult = MQTT_Connect(
    pxMQTTContext,
    &xConnectInfo,
    NULL,
    mqttexampleCONNACK_RECV_TIMEOUT_MS,
    &xSessionPresent );
configASSERT( xResult == MQTTSuccess );

/* Successfully established and MQTT connection with the broker. */
LogInfo( ( "An MQTT connection is established with %s.",
democonfigMQTT_BROKER_ENDPOINT ) );
}
```

描述 MQTT 主題

函數 [prvMQTTSubscribeWithBackoffRetries](#) 函數示範如何訂用 MQTT 中介裝置上的主題篩選條件。此範例示範如何訂用一個主題篩選條件，但可以在相同的訂用 API 中傳遞主題篩選條件清單，以訂用多個主題篩選條件。此外，如果 MQTT 中介裝置拒收訂用請求，將重試（使用指數退避）RETRY_MAX_ATTEMPTS。

發布到主題

函數示範如何在 MQTT 中介裝置上發布到主題篩選條件。prvMQTTPublishToTopic函數的定義如下所示。

```
static void prvMQTTPublishToTopic( MQTTContext_t * pxMQTTContext )
{
    MQTTStatus_t xResult;
    MQTTPublishInfo_t xMQTTPublishInfo;

    /**
     * For readability, error handling in this function is restricted to the use of
     * asserts().
     */

    /* Some fields are not used by this demo so start with everything at 0. */
    ( void ) memset( ( void * ) &xMQTTPublishInfo, 0x00, sizeof( xMQTTPublishInfo ) );

    /* This demo uses QoS1. */
    xMQTTPublishInfo.qos = MQTTQoS1;
    xMQTTPublishInfo.retain = false;
    xMQTTPublishInfo.pTopicName = mqttexampleTOPIC;
    xMQTTPublishInfo.topicNameLength = ( uint16_t ) strlen( mqttexampleTOPIC );
    xMQTTPublishInfo.pPayload = mqttexampleMESSAGE;
    xMQTTPublishInfo.payloadLength = strlen( mqttexampleMESSAGE );

    /* Get a unique packet id. */
    usPublishPacketIdentifier = MQTT_GetPacketId( pxMQTTContext );

    /* Send PUBLISH packet. Packet ID is not used for a QoS1 publish. */
    xResult = MQTT_Publish( pxMQTTContext, &xMQTTPublishInfo, usPublishPacketIdentifier );

    configASSERT( xResult == MQTTSuccess );
}
```

接收傳入的訊息

應用程式會在連接到中介裝置之前登記事件回呼函數，如先前所述。函數會呼叫 prvMQTTDemoTask 函數來接收傳入的訊息。MQTT_ProcessLoop收到傳入 MQTT 訊息時，它會呼叫應用程式登記的事件回呼函數。函數是此類事件回呼函數的範例。prvEventCallbackprvEventCallback 會檢查傳入封包類型並呼叫適當的處理常式。在以下範例中，函數會呼叫 prvMQTTProcessIncomingPublish() 來處理傳入的發布訊息，或呼叫 prvMQTTProcessResponse() 來處理確認 (ACK)。

```
static void prvEventCallback( MQTTContext_t * pxMQTTContext,
                             MQTTPacketInfo_t * pxPacketInfo,
                             MQTTDeserializedInfo_t * pxDeserializedInfo )
{
    /* The MQTT context is not used for this demo. */
    ( void ) pxMQTTContext;

    if( ( pxPacketInfo->type & 0xF0U ) == MQTT_PACKET_TYPE_PUBLISH )
    {
        prvMQTTProcessIncomingPublish( pxDeserializedInfo->pPublishInfo );
    }
    else
    {
        prvMQTTProcessResponse( pxPacketInfo, pxDeserializedInfo->packetIdentifier );
    }
}
```

處理傳入的 MQTT 發布封包

函數示範如何處理來自 MQTT 中介裝置的發布封包。prvMQTTProcessIncomingPublish函數的定義如下所示。

```
static void prvMQTTProcessIncomingPublish( MQTTPublishInfo_t * pxPublishInfo )
{
    configASSERT( pxPublishInfo != NULL );

    /* Process incoming Publish. */
    LogInfo( ( "Incoming QoS : %d\n", pxPublishInfo->qos ) );

    /* Verify the received publish is for the we have subscribed to. */
    if( ( pxPublishInfo->topicNameLength == strlen( mqttxampleTOPIC ) ) &&
        ( 0 == strncmp( mqttxampleTOPIC,
                        pxPublishInfo->pTopicName,
                        pxPublishInfo->topicNameLength ) ) )
    {
        LogInfo( ( "\r\nIncoming Publish Topic Name: %.s matches subscribed topic.\r\n"
                   "Incoming Publish Message : %.s\r\n",
                   pxPublishInfo->topicNameLength,
                   pxPublishInfo->pTopicName,
                   pxPublishInfo->payloadLength,
                   pxPublishInfo->pPayload ) );
    }
    else
    {
        LogInfo( ( "Incoming Publish Topic Name: %.s does not match subscribed topic.\r
\n",
                   pxPublishInfo->topicNameLength,
                   pxPublishInfo->pTopicName ) );
    }
}
```

取消訂用主題

工作流程的最後一個步驟是取消 mqttxampleTOPIC 主題中的描述，讓中介裝置不會從傳送任何已發布的訊息。函數的定義如下所示。

```
static void prvMQTTUnsubscribeFromTopic( MQTTContext_t * pxMQTTContext )
{
    MQTTStatus_t xResult;
    MQTTSubscribeInfo_t xMQTTSubscription[ mqttxampleTOPIC_COUNT ];

    /* Some fields not used by this demo so start with everything at 0. */
    ( void ) memset( ( void * ) &xMQTTSubscription, 0x00, sizeof( xMQTTSubscription ) );

    /* Get a unique packet id. */
    usSubscribePacketIdentifier = MQTT_GetPacketId( pxMQTTContext );

    /* Subscribe to the mqttxampleTOPIC topic filter. This example subscribes to
     * only one topic and uses QoS1. */
    xMQTTSubscription[ 0 ].qos = MQTTQoS1;
    xMQTTSubscription[ 0 ].pTopicFilter = mqttxampleTOPIC;
    xMQTTSubscription[ 0 ].topicFilterLength = ( uint16_t ) strlen( mqttxampleTOPIC );

    /* Get next unique packet identifier. */
    usUnsubscribePacketIdentifier = MQTT_GetPacketId( pxMQTTContext );
```

```
/* Send UNSUBSCRIBE packet. */
xResult = MQTT_Unsubscribe( pxMQTTContext,
                            xMQTTSubscription,
                            sizeof( xMQTTSubscription ) /
sizeof( MQTTSubscribeInfo_t ),
                            usUnsubscribePacketIdentifier );

configASSERT( xResult == MQTTSuccess );
}
```

無線更新示範應用程式

FreeRTOS 包含示範 無線 (OTA) 程式庫功能的示範應用程式。OTA 示範應用程式位於 [freertos/demos/ota/aws_iot_ota_update_demo.c](#) 檔案中。

OTA 示範應用程式可執行下列作業：

1. 初始化 FreeRTOS 網路堆疊和 MQTT 緩衝集區。
2. 建立任務以使用 `vRunOTAUpdateDemo()` 練習 OTA 程式庫。
3. 使用 `_establishMqttConnection()` 建立 MQTT 用戶端。
4. 使用 AWS IoT 連接到 `IotMqtt_Connect()` MQTT 中介裝置，並登記 MQTT 中斷回呼：`prvNetworkDisconnectCallback -`。
5. 呼叫 `OTA_AgentInit()` 來建立 OTA 任務，並註冊在 OTA 任務完成時使用的回呼。
6. 使用 `xOTAConnectionCtx.pvControlClient = _mqttConnection;` 重複使用 MQTT 連接
7. 如果 MQTT 中斷，應用程式會暫停 OTA 代理程式、嘗試使用指數延遲與抖動重新連接，然後繼續 OTA 代理程式。

您必須先完成 [FreeRTOS 無線更新 \(p. 120\)](#)中的所有必要條件，才能才能使用 OTA 更新。

在您完成 OTA 更新的設定後，請在支援 OTA 功能的平台下載、建置、更新並執行 FreeRTOS OTA 示範。下列 FreeRTOS 合格裝置提供特定裝置的示範說明：

- [Texas Instruments CC3220SF-LAUNCHXL \(p. 251\)](#)
- [Microchip Curiosity PIC32MZEF \(p. 253\)](#)
- [Espressif ESP32 \(p. 257\)](#)
- [Download, build, flash and run the FreeRTOS OTA demo on the Renesas RX65N \(p. 257\)](#)

在您的裝置上建置、更新並執行 OTA 示範應用程式之後，您可以使用 AWS IoT 主控台或 AWS CLI 建立 OTA 更新任務。在您建立 OTA 更新任務後，請連線終端機模擬器，以查看 OTA 的進度更新。請記下程序中產生的任何錯誤。

成功的 OTA 更新任務，會顯示如下所示的輸出。為簡潔起見，此範例中的某些行已從清單中移除。

```
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-shal-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
```

空中示範組態

OTA 示範組態是 `aws_iot_ota_update_demo.c` 中提供的示範特定組態選項。這些組態與 OTA 程式庫組態檔案所提供的 OTA 程式庫組態不同。

OTA_DEMO_KEP_ALIVE_SECONDS

對於 MQTT 使用者，此組態是完成一個控制封包傳輸並開始傳送之間的最大時間間隔。缺少控制封包就會傳送 PINGREQ。中介裝置必須中斷不以 1 次和一半時間將未傳送訊息或 PINGREQ 封包的使用者，保持作用間隔。此組態應依據應用程式的需求調整。

OTA_DEMO_CONN_RETRY_BASE_INTERVAL_SECONDS

重試網路連接之前的基底間隔（以秒為單位）。OTA 示範會在此基本時間間隔之後嘗試重新連接。間隔會在每次失敗的嘗試後加倍。隨機延遲（最多此基本延遲的上限）也會新增至間隔。

OTA_DEMO_CONN_RETRY_MAX_INTERVAL_SECONDS

在重試網路連接之前的間隔上限（以秒為單位）。在每次嘗試失敗時，重新連接延遲都會加倍，但最多只能達到這個最大值，加上相同間隔的抖動。

在 Texas Instruments CC3220SF-LAUNCHXL 下載、建置、更新並執行 FreeRTOS OTA 示範

下載 FreeRTOS 及 OTA 示範程式碼

1. 瀏覽至 AWS IoT 主控台，並從導覽窗格中，選擇 Software (軟體)。
2. 在 FreeRTOS Device Software (RTOS 裝置軟體) 下方，選擇 Configure download (設定下載)。
3. 從軟體組態清單中，選擇 Connect to AWS IoT - TI (連接到 - TI)。選擇組態名稱，而非 Download (下載) 連結。

4. 在 Libraries (程式庫) 下方，選擇 Add another library (新增另一個程式庫)，然後選擇 OTA Updates (OTA 更新)。
5. 在 Demo Projects (示範專案) 下方，選擇 OTA Updates (OTA 更新)。
6. 在 Name required (必要名稱) 下方，輸入 **Connect-to-IoT-OTA-TI**，然後選擇 Create and download (建立並下載)。

將包含 FreeRTOS 的 zip 檔案及 OTA 示範程式碼儲存到您的電腦。

建置示範應用程式

1. 解壓縮 .zip 檔案。
2. 遵循 [FreeRTOS 入門 \(p. 14\)](#) 中的說明，將 aws_demos 專案匯入 Code Composer Studio，並設定 AWS IoT 端點、Wi-Fi SSID 和密碼，以及主機板的私有金鑰和憑證。
3. 開啟 *freertos/vendors/vendor/boards/board/aws_demos/config_files/aws_demo_config.h*，註解 `#define CONFIG_MQTT_DEMO_ENABLED`，然後定義 `CONFIG_OTA_UPDATE_DEMO_ENABLED`。
4. 建置解決方案並確認過程中沒有發生任何錯誤。
5. 啟動終端機模擬器，並使用以下設定來連線到您的電路板：
 - 傳輸速率：115200
 - 資料位元：8
 - 同位：無
 - 停止位元：1
6. 在您的主機板上執行專案，確認它可以連線到 Wi-Fi 及 AWS IoT MQTT 訊息中介裝置。

執行時，終端機模擬器應會顯示與以下內容相似的文字：

```
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
```

在 Microchip Curiosity PIC32MZEF 下載、建置、更新並執行 FreeRTOS OTA 示範

下載 FreeRTOS OTA 示範程式碼

1. 瀏覽至 AWS IoT 主控台，並從導覽窗格中，選擇 Software (軟體)。
2. 在 FreeRTOS Device Software (RTOS 裝置軟體) 下方，選擇 Configure download (設定下載)。
3. 從軟體組態清單中，選擇 Connect to AWS IoT - Microchip (連接到 - Microchip)。選擇組態名稱，而非 Download (下載) 連結。
4. 在 Libraries (程式庫) 下方，選擇 Add another library (新增另一個程式庫)，然後選擇 OTA Updates (OTA 更新)。
5. 在 Demo projects (示範專案) 下方，選擇 OTA Update (OTA 更新)。
6. 在 Name required (必要名稱) 下方，輸入您自訂 FreeRTOS 軟體組態的名稱。
7. 選擇 Create and download (建立並下載)。

建置 OTA 更新示範應用程式

1. 將您剛下載的 .zip 檔案解壓縮。
2. 遵循[FreeRTOS 入門 \(p. 14\)](#) 中的說明，將 aws_demos 專案匯入 MPLAB X IDE，設定您的 AWS IoT 端點、您的 Wi-Fi SSID 及密碼，以及您電路板的私有金鑰和憑證。
3. Open demos/include/aws_ota_codesigner_certificate.h.
4. 將您程式碼簽署憑證的內容貼入 static const char signingcredentialSIGNING_CERTIFICATE_PEM 變數。遵循與 aws_clientcredential_keys.h 相同的格式，每一行的結尾都必須是新行字元 ('\n') 並包圍在引號中。

例如，您的憑證應看起來與下列類似：

```
"-----BEGIN CERTIFICATE-----\n"
"MIIBXTCCAQoAwIBAgIJAM4DeybZctTwKMAoGCCqGSM49BAMCMCExHzAdBgNVBAMM\n"
"FnRlc3Rf621nbmVyQGFtYXpvbi5jb20wHhcNMTcxMTAzMTkxODM1WhcNMTgxMTAz\n"
"MTkxODM2WjAhMR8wHQYDVQBZZOZXNOX3NpZ25lckBhbWF6b24uY29tMFkwEwYH\n"
"KoZIzj0CAQYIKoZIzj0DAQcDQgAEravZfvwL1X+E4dIF7dbkVMUu4IrJ1CasFkc8\n"
"gzxPzn683H40XMk1tDZPEwr9ng78w9+QYQg7ygnr2stz8yh06MkMCiWcWYDVR0P\n"
"BAQDAgeAMBGA1UdJQQMMAoGCCsGAQUFBwMDMAoGCCqGSM49BAMCA0gAMEUCIFOR\n"
"r5cb7rEUNtWoVgd05MacrgOABfSoVYvBOK9fp63WAqt5h3BaS123coKSGg84twlq\n"
"TkO/pV/xEmyZmZdV+HxV/OM=\n"
```

```
"-----END CERTIFICATE-----\n";
```

5. 安裝 [Python 3](#) 或更新版本。
6. 透過執行 `pip install pyopenssl` 來安裝 pyOpenSSL。
7. 複製 `demos/ota/bootloader/utility/codesigner_cert_utility/` 路徑中格式為 .pem 的程式碼簽署憑證。重新命名憑證檔案 `aws_ota_codesigner_certificate.pem`。
8. 開啟 `freertos/vendors/vendor/boards/board/aws_demos/config_files/` `aws_demo_config.h`、註解 `#define CONFIG_MQTT_DEMO_ENABLED`，然後定義 `CONFIG_OTA_UPDATE_DEMO_ENABLED`。
9. 建置解決方案並確認過程中沒有發生任何錯誤。
10. 啟動終端機模擬器，並使用以下設定來連線到您的電路板：
 - 傳輸速率：115200
 - 資料位元：8
 - 同位：無
 - 停止位元：1
11. 從您的主機板拔除除錯器，並在您的主機板上執行專案，確認它可以連線到 Wi-Fi 及 AWS IoT MQTT 訊息中介裝置。

當您執行專案時，MPLAB X IDE 應會開啟輸出視窗。確認已選取 ICD4 索引標籤。您應該會看到下列輸出。

```
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
```

終端機模擬器應會顯示與以下內容相似的文字：

```
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 #0
38 10336 [OTA Task] Sending command to MQTT task.
39 10336 [MQTT] Received message 40000 from queue.
40 10366 [IP-task] Socket sending wakeup to MQTT task.
41 10366 [MQTT] Received message 0 from queue.
42 10366 [MQTT] MQTT Publish was successful.
43 10366 [MQTT] Notifying task.
44 10366 [OTA Task] Command sent to MQTT task passed.
45 10376 [IP-task] Socket sending wakeup to MQTT task.
46 10376 [MQTT] Received message 0 from queue.
47 10376 [OTA Task] [OTA] Set job doc parameter [ clientToken: 0:Microchip ]
48 10376 [OTA Task] [OTA] Missing job parameter: execution
49 10376 [OTA Task] [OTA] Missing job parameter: jobId
50 10376 [OTA Task] [OTA] Missing job parameter: jobDocument
51 10378 [OTA Task] [OTA] Missing job parameter: ts_ota
52 10378 [OTA Task] [OTA] Missing job parameter: files
53 10378 [OTA Task] [OTA] Missing job parameter: streamname
54 10378 [OTA Task] [OTA] Missing job parameter: certfile
55 10378 [OTA Task] [OTA] Missing job parameter: filepath
56 10378 [OTA Task] [OTA] Missing job parameter: filesize
57 10378 [OTA Task] [OTA] Missing job parameter: sig-sha256-ecdsa
58 10378 [OTA Task] [OTA] Missing job parameter: fileid
59 10378 [OTA Task] [OTA] Missing job parameter: attr
60 10378 [OTA Task] [OTA] Returned buffer to MQTT Client.
61 11367 [OTA] [OTA] Queued: 1 Processed: 1 Dropped: 0
62 12367 [OTA] [OTA] Queued: 1 Processed: 1 Dropped: 0
63 13367 [OTA] [OTA] Queued: 1 Processed: 1 Dropped: 0
64 14367 [OTA] [OTA] Queued: 1 Processed: 1 Dropped: 0
65 15367 [OTA] [OTA] Queued: 1 Processed: 1 Dropped: 0
66 16367 [OTA] [OTA] Queued: 1 Processed: 1 Dropped: 0
```

此輸出顯示 Microchip Curiosity PIC32MZE 可以連線到 AWS IoT，並且訂閱 OTA 更新所需要的 MQTT 主題。Missing job parameter 訊息是在預期中的內容，因為沒有任何擋置中的 OTA 更新任務。

在 Espressif ESP32 下載、建置、更新並執行 FreeRTOS OTA 示範

1. 從 FreeRTOS [GitHub 下載](#) 來源。請查看 [README.md](#) 檔案以獲得指示。在您的 IDE 中建立專案，並包含所有必要的來源及程式庫。
2. 遵循 [Espressif 入門](#) 中的說明，設定必要的 GCC 型工具鏈。
3. 開啟 `freertos/vendors/vendor/boards/board/aws_demos/config_files/aws_demo_config.h`、註解 `#define CONFIG_MQTT_DEMO_ENABLED`，然後定義 `CONFIG_OTA_UPDATE_DEMO_ENABLED`。
4. 在 `vendors/espressif/boards/esp32/aws_demos` 目錄中執行 `make`，以便建置示範專案。您可以透過執行 `make flash monitor` 刷新示範程式並驗證其輸出，如 [Espressif 入門](#) 中所述。
5. 執行 OTA 更新示範前，請留意下列事項：
 - 開啟 `freertos/vendors/vendor/boards/board/aws_demos/config_files/aws_demo_config.h`、註解 `#define CONFIG_MQTT_DEMO_ENABLED`，然後定義 `CONFIG_OTA_UPDATE_DEMO_ENABLED`。
 - 確認 SHA-256/ECDSA 程式碼簽署憑證已複製到 `demos/include/aws_ota_codesigner_certificate.h`。

Download, build, flash and run the FreeRTOS OTA demo on the Renesas RX65N

This chapter shows you how download, build, flash and run the FreeRTOS OTA demo applications on the Renesas RX65N.

主題

- [Set up your operating environment \(p. 257\)](#)
- [Set up your AWS resources \(p. 258\)](#)
- [Import, configure the header file and build aws_demos and boot_loader \(p. 260\)](#)

Set up your operating environment

The procedures in this section use the following environments:

- IDE: e² studio 7.8.0, e² studio 2020-07
- Toolchains: CCRX Compiler v3.0.1
- Target devices: RSKRX65N-2MB
- Debuggers: E², E² Lite emulator
- Software: Renesas Flash Programmer, Renesas Secure Flash Programmer.exe, Tera Term

To set up your hardware

1. Connect the E² Lite emulator and USB serial port to your RX65N board and PC.
2. Connect the power source to the RX65N.

Set up your AWS resources

1. To run the FreeRTOS demos, you must have an AWS account with an AWS Identity and Access Management (IAM) user that has permission to access AWS IoT services. If you haven't already, follow the steps in [設定 AWS 帳戶和許可 \(p. 15\)](#).
2. To set up for OTA updates, follow the steps in [OTA 更新先決條件 \(p. 120\)](#). In particular, follow the steps in [使用 MQTT 進行 OTA 更新的先決條件 \(p. 130\)](#).
3. 開啟 [AWS IoT 主控台](#)。
4. In the left navigation pane, choose Manage, then choose Things to create a thing.

A thing is a representation of a device or logical entity in AWS IoT. 它可以是實體裝置或感應器 (例如燈泡或牆上的開關)。It can also be a logical entity like an instance of an application or physical entity that doesn't connect to AWS IoT, but is related to devices that do (for example, a car that has engine sensors or a control panel). AWS IoT provides a thing registry that helps you manage your things.

- a. Choose Create, then choose Create a single thing.
- b. Enter a Name for your thing, then choose Next.
- c. 選擇 Create certificate (建立憑證)。
- d. Download the three files that are created and then choose Activate.
- e. 選擇 Attach a policy (連接政策)。

Download these files and save them in a safe place. Certificates can be retrieved at any time, but the private and public keys cannot be retrieved after you close this page.

In order to connect a device, you need to download the following:

A certificate for this thing	9dba40d984.cert.pem	Download
A public key	9dba40d984.public.key	Download
A private key	9dba40d984.private.key	Download

You also need to download a root CA for AWS IoT:
A root CA for AWS IoT [Download](#)

[Activate](#)

[Cancel](#)

[Done](#)

[Attach a policy](#)

- f. Select the policy that you created in [裝置政策 \(p. 131\)](#).

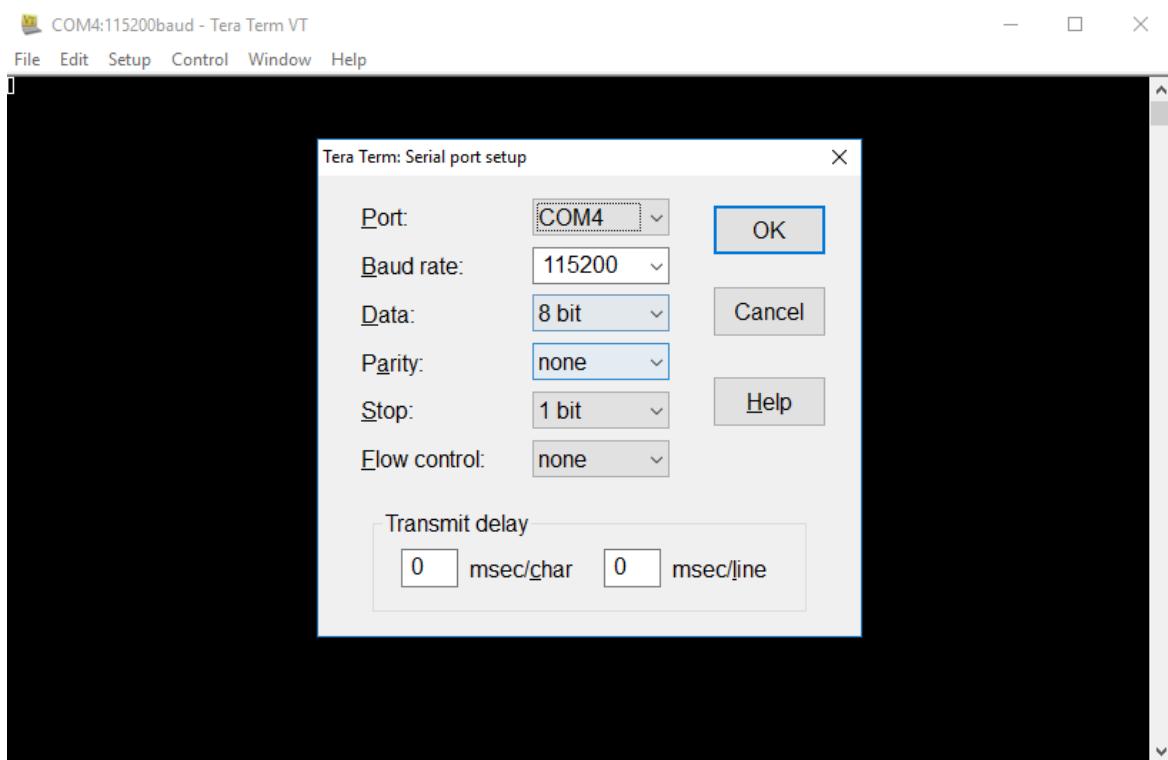
Each device that receives an OTA update using MQTT must be registered as a thing in AWS IoT and the thing must have an attached policy like the one listed. 如需 "Action" 和 "Resource" 物件中的項目的詳細資訊，請參閱 [AWS IoT 核心政策動作](#) 和 [AWS IoT 核心動作資源](#)。

Notes

- `iot:Connect` 許可允許您的裝置透過 MQTT 連接至 AWS IoT。
- AWS IoT 工作 (`.../jobs/*`) 主題的 `iot:Subscribe` 和 `iot:Publish` 許可允許連接的裝置接收工作通知和工作文件，並發佈工作執行的完成狀態。
- AWS IoT OTA 串流 (`.../streams/*`) 主題的 `iot:Subscribe` 和 `iot:Publish` 許可允許連接的裝置從 AWS IoT 中擷取 OTA 更新資料。需要這些許可才能透過 MQTT 執行韌體更新。
- `iot:Receive` 許可允許 AWS IoT Core 將這些主題上的訊息發佈到連接的裝置。每次交付 MQTT 訊息時，都會檢查此許可。您可以使用此許可來撤銷目前訂閱主題之用戶端的存取權。

5. To create a code-signing profile and register a code-signing certificate on AWS.
 - a. To create the keys and certification, see section 7.3 "Generating ECDSA-SHA256 Key Pairs with OpenSSL" in [Renesas MCU Firmware Update Design Policy](#).
 - b. 開啟 [AWS IoT 主控台](#)。In the left navigation pane, select Manage, then Jobs. Select Create a job then Create OTA update Job.
 - c. Under Select devices to update choose Select then choose the thing you created previously. 選取 Next (下一步)。
 - d. On the Create a FreeRTOS OTA update job page, do the following:
 - i. For Select the protocol for firmware image transfer, choose MQTT.
 - ii. For Select and sign your firmware image, choose Sign a new firmware image for me.
 - iii. For Code signing profile, choose Create.
 - iv. In the Create a code signing profile window, enter a Profile name. For the Device hardware platform select Windows Simulator. For the Code signing certificate choose Import.
 - v. Browse to select the certificate (secp256r1.crt), the certificate private key (secp256r1.key), and the certificate chain (ca.crt).
 - vi. Enter a Pathname of code signing certificate on device. 然後選擇 Create (建立)。
6. To grant access to code signing for AWS IoT, follow the steps in [將存取權限授予適用於 AWS IoT 的程式碼簽署 \(p. 130\)](#).

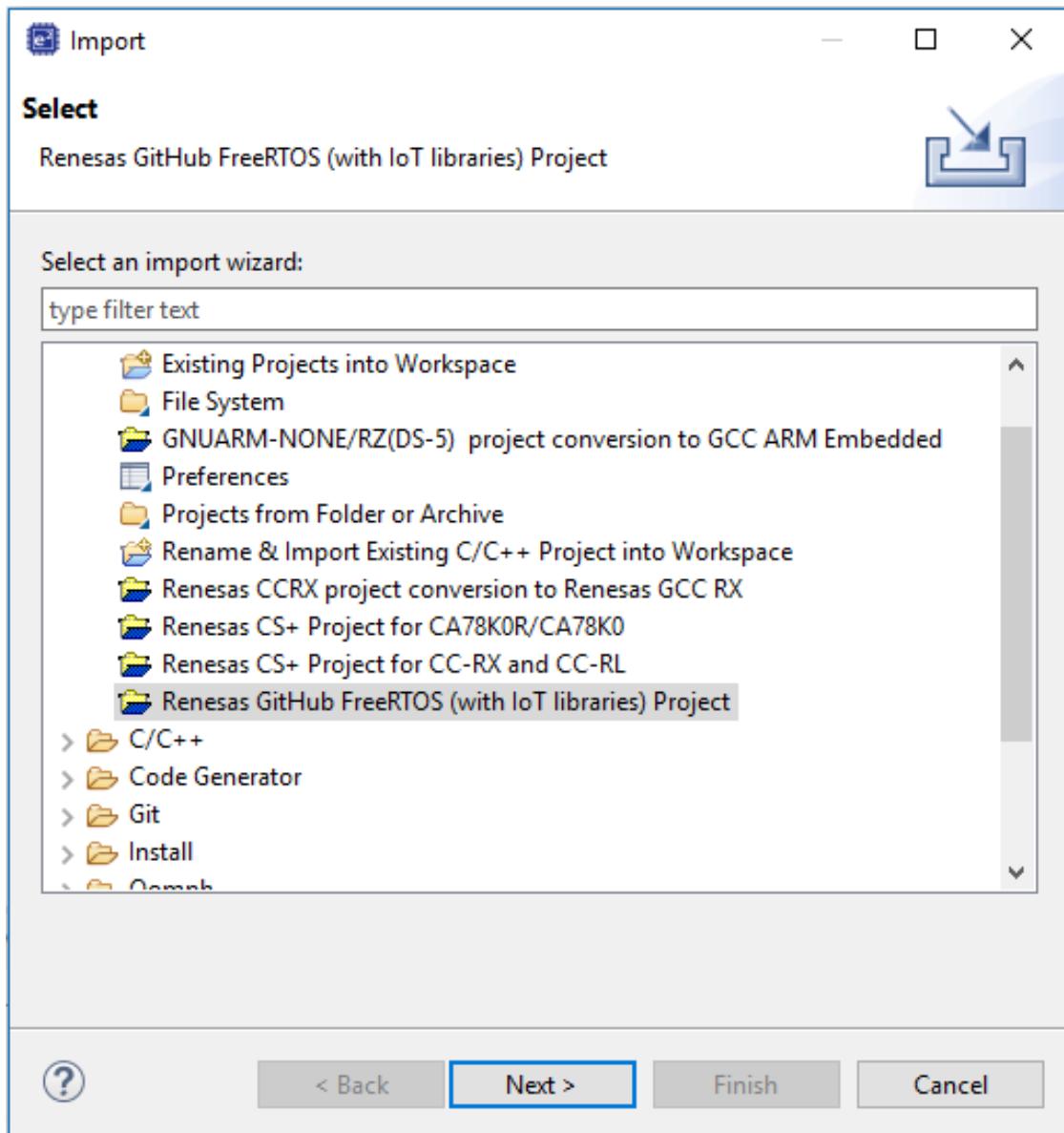
If don't have Tera Term installed on your PC, you can download it from <https://ttssh2.osdn.jp/index.html.en> and set it up as shown here. Make sure that you plug in the USB Serial port from your device to your PC.



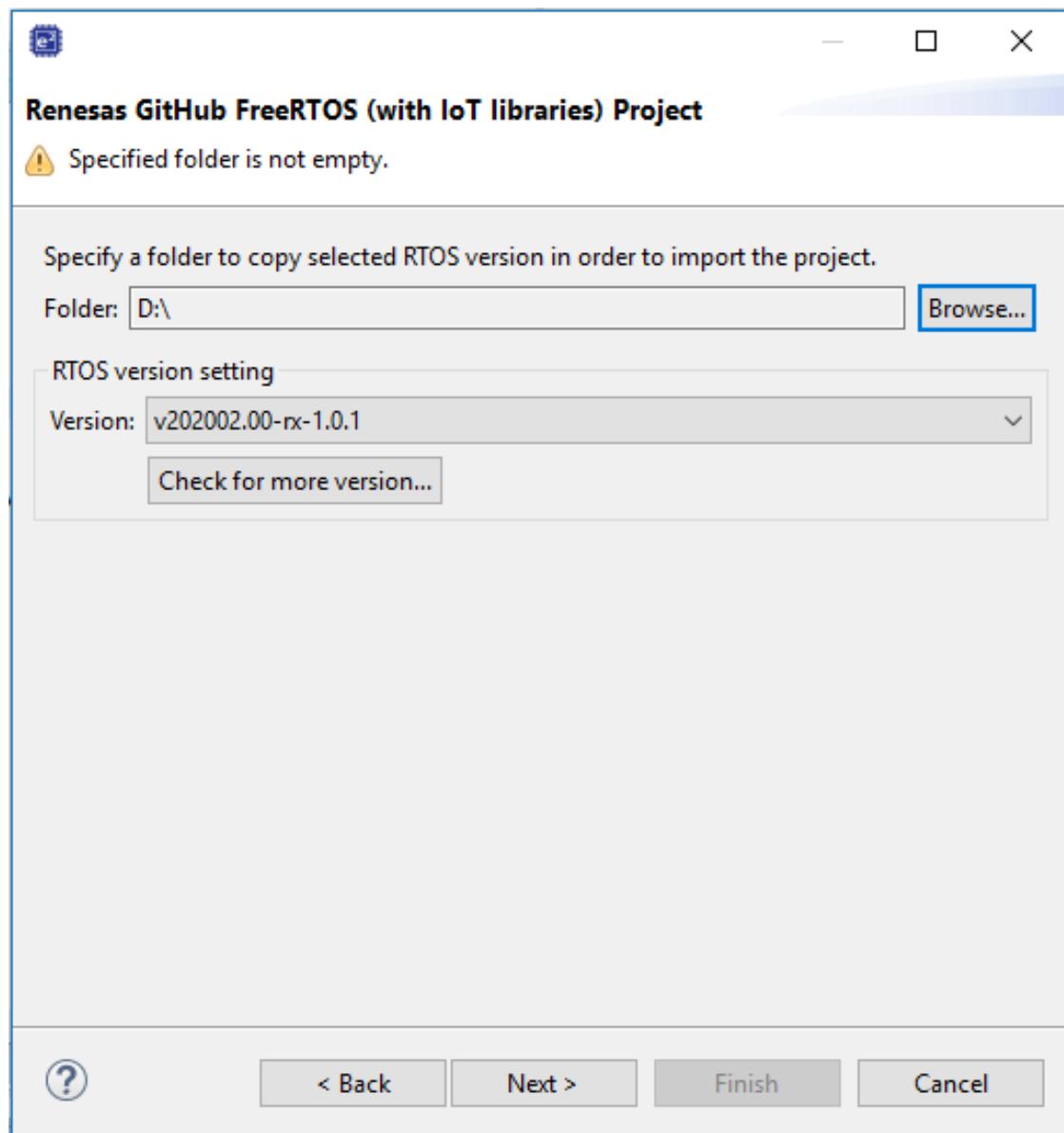
Import, configure the header file and build aws_demos and boot_loader

To begin, you select the latest version of the FreeRTOS package, and this will be downloaded from GitHub and imported automatically into the project. This way you can focus on the configuring FreeRTOS and writing application code.

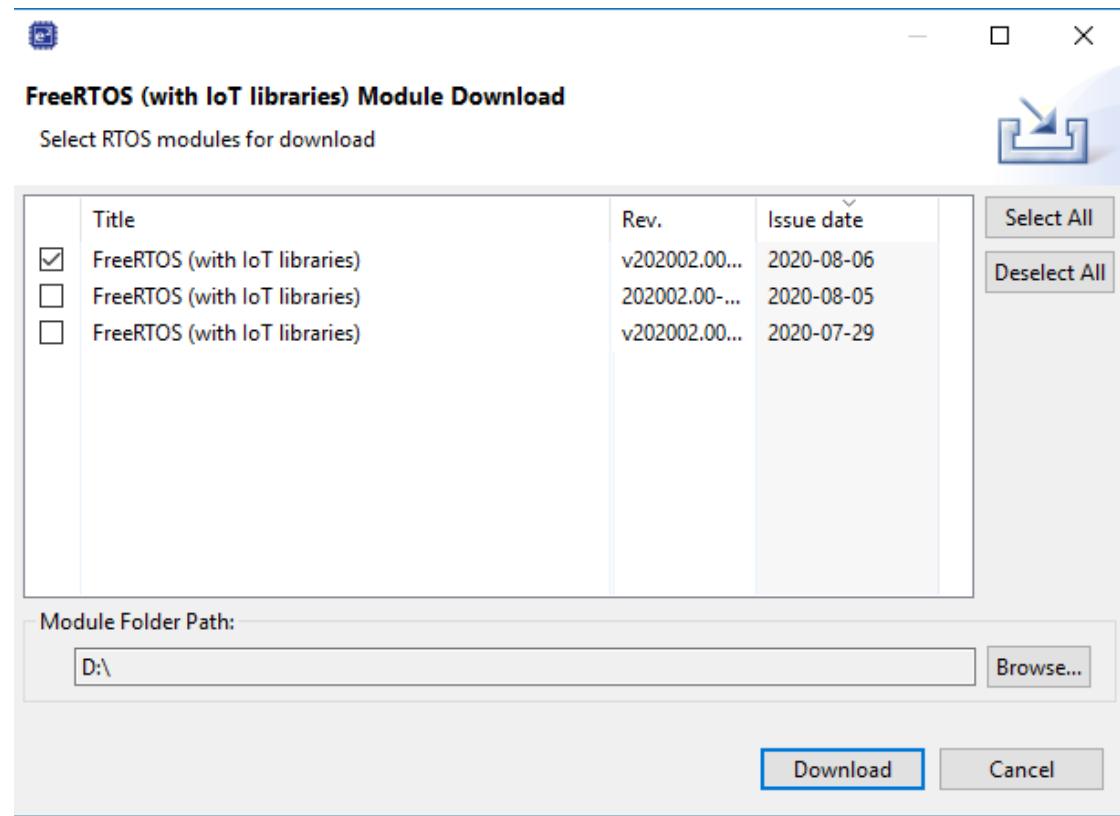
1. Launch e² studio.
2. Choose File, and then choose Import....
3. Select Renesas GitHub FreeRTOS (with IoT libraries) Project.



4. Choose Check for more version... to show the download dialog box.



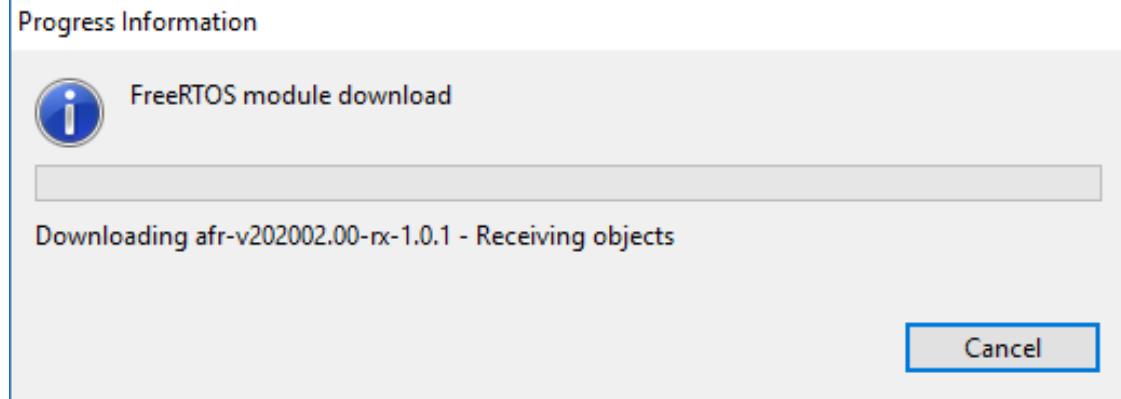
5. Select the lastest package.



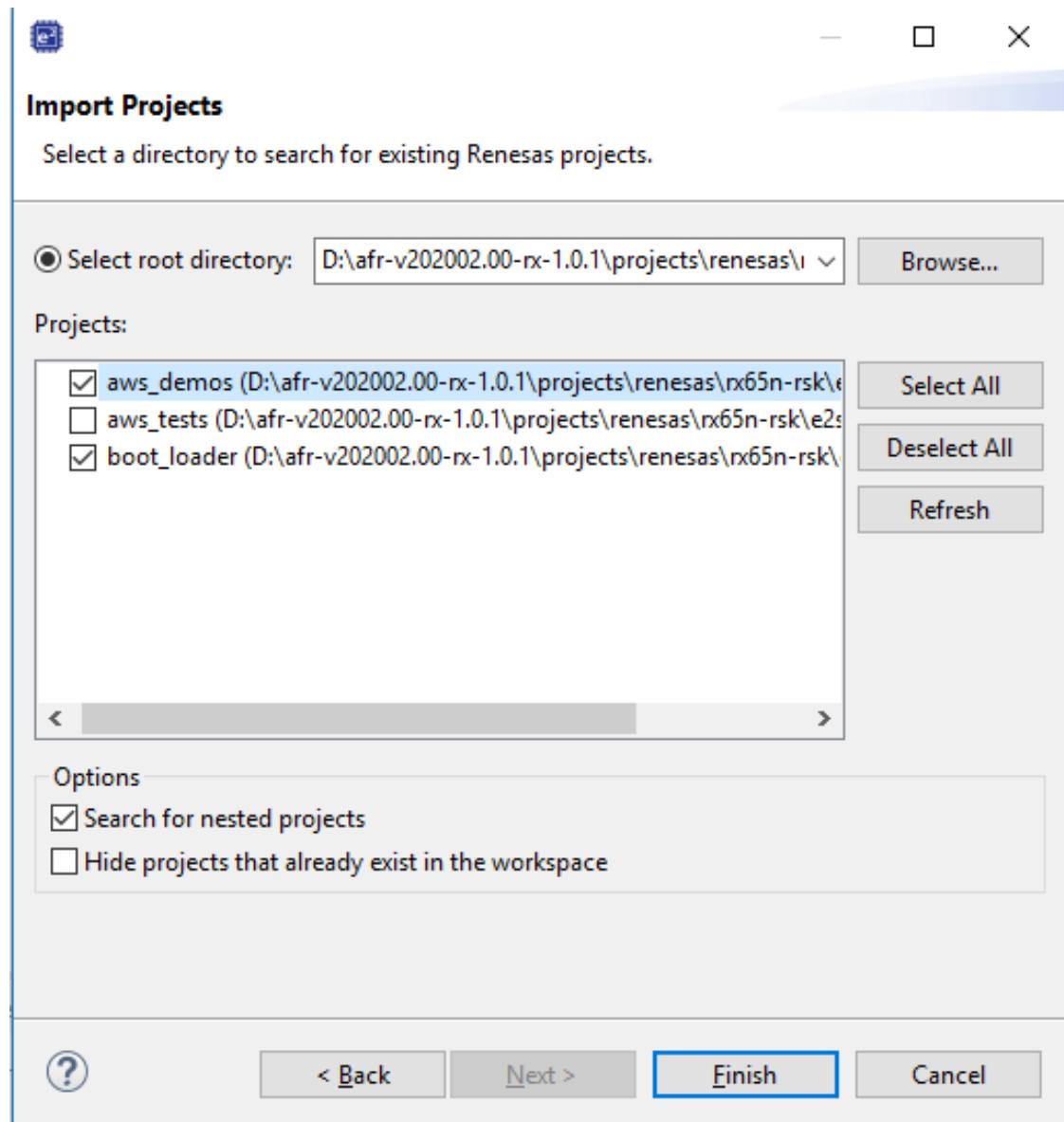
6. Choose Agree to accept the end user license agreement.



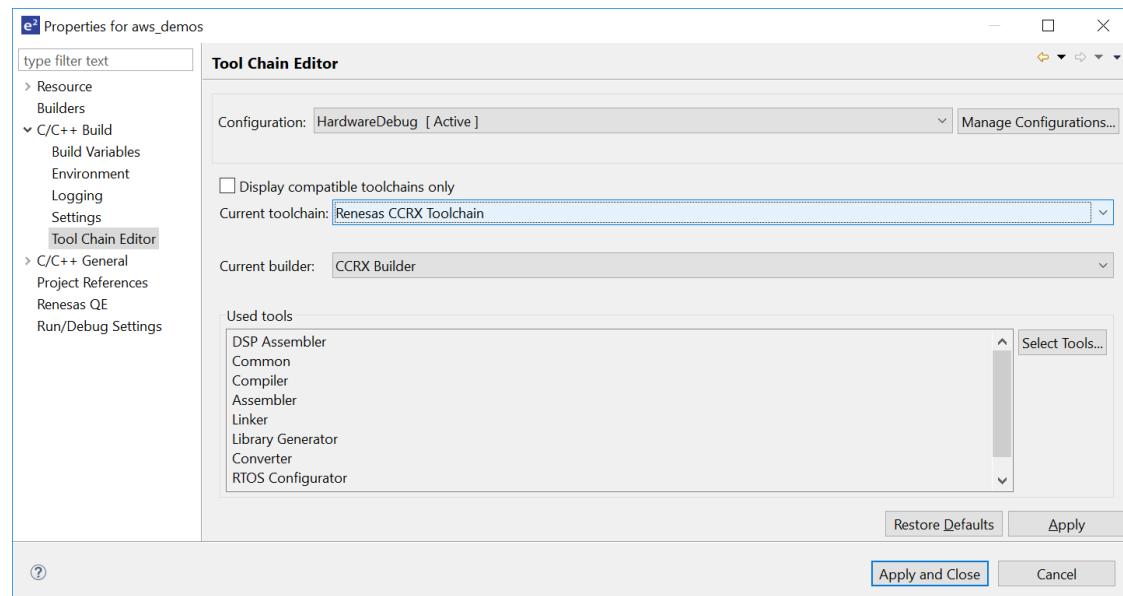
7. Wait for the download to complete.



8. Select the aws_demos and boot_loader projects, then choose Finish to import them.

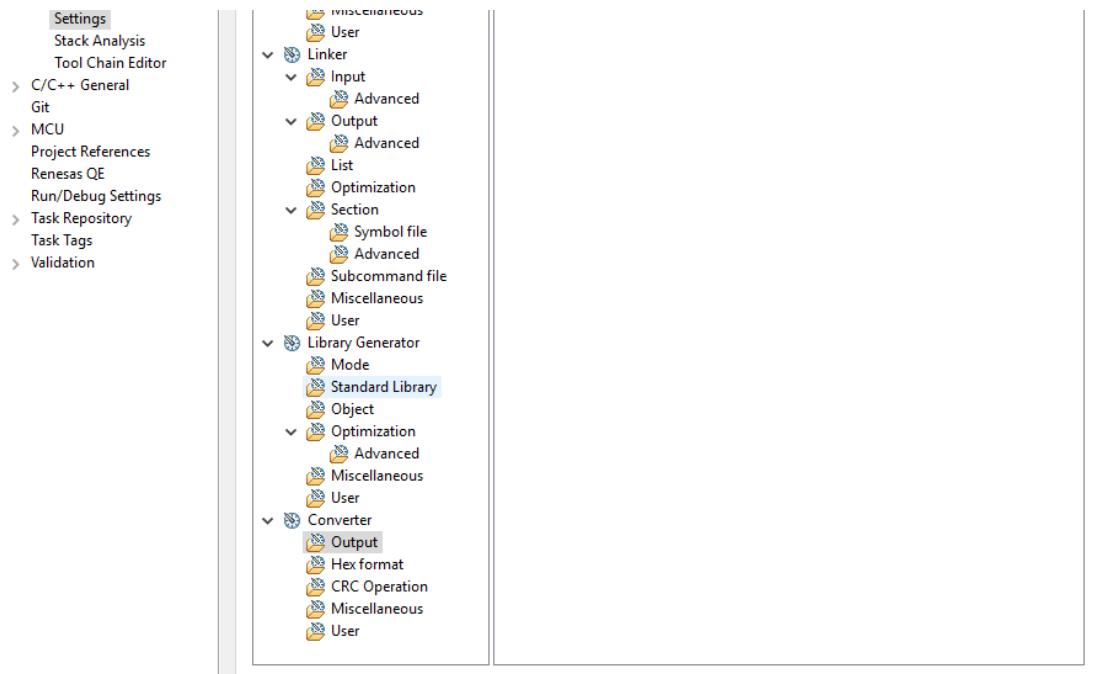


9. For both projects, open the project properties. In the navigation pane, choose Tool Chain Editor.
 - a. Choose the Current toolchain.
 - b. Choose the Current builder.



10. 在導覽窗格中選擇 Settings (設定)。Choose the Toolchain tab, and then choose the toolchain Version.

Choose the Tool Settings tab, expand Converter and then choose Output. In the main window, make sure Output hex file is selected, and then choose the Output file type.



- In the bootloader project, open `projects\renesas\rx65n-rsk\e2studio\boot_loader\src\key\code_signer_public_key.h` and input the public key. For information on how to create a public key, see [How to implement FreeRTOS OTA by using Amazon Web Services on RX65N](#) and section 7.3 "Generating ECDSA-SHA256 Key Pairs with OpenSSL" in [Renesas MCU Firmware Update Design Policy](#).

The screenshot shows the 'Project Explorer' and 'code_signer_public_key.h' code editor side-by-side. The Project Explorer lists projects like 'aws_demos' and 'boot_loader'. The code editor shows the following content:

```

2  /* DISCLAIMER */
20 * File Name : code_signer_public_key.h
24 * History : DD.MM.YYYY Version Description
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50

```

CODE SIGNER PUBLIC KEY PEM

```

@* DISCLAMER*
@* File Name : code_signer_public_key.h
@* History : DD.MM.YYYY Version Description

#ifndef CODE_SIGNER_PUBLIC_KEY_H_
#define CODE_SIGNER_PUBLIC_KEY_H_

/*
 * PEM-encoded code signer public key.
 *
 * Must include the PEM header and footer:
 * -----BEGIN CERTIFICATE-----\n"
 * ...base64 data...\n"
 * -----END CERTIFICATE-----"
 */
#define CODE_SIGNER_PUBLIC_KEY_PEM "Paste code signer public key here."
#define CODE_SIGNER_PUBLIC_KEY_PEM \
"-----BEGIN PUBLIC KEY-----"\n"MFkwEwYHKoZIzj0AQIKoZIzj0AQCDgAEhVxq1tUJZ5LXmrur1mTTQzijLtQ"\n"sz9cjj31BZl89nyhmB13UkaolV4/aEWa6fTuBPVeaiyEwJeQ7YBpYGC9iA=="\n"-----END PUBLIC KEY-----"

extern const uint8_t code_signer_public_key[];
extern const uint32_t code_signer_public_key_length;

#endif /* CODE_SIGNER_PUBLIC_KEY_H_ */

```

Then build the project to create `boot_loader.mot`.

- Open the `aws_demos` project.
 - 開啟 AWS IoT 主控台。
 - 在左側的導覽窗格中，選擇 Settings (設定)。Make a note of your custom Endpoint.
 - Choose Manage, and then choose Things. Make a note of the AWS IoT thing name of your board.
 - In the `aws_demos` project, open `/demos/include/aws_clientcredential.h` and specify the following values.

```
#define clientcredentialMQTT_BROKER_ENDPOINT[] = "Your AWS IoT endpoint";
#define clientcredentialIOT_THING_NAME "The AWS IoT thing name of your board"
```

```
28
29
30     */
31     * @brief MQTT Broker endpoint.
32     *
33     * @todo Set this to the fully-qualified DNS name of your MQTT broker.
34     */
35     #define clientcredentialMQTT_BROKER_ENDPOINT      "xxxxx-ats.iot.ap-northeast-1.amazonaws.com"
36
37     */
38     * @brief Host name.
39     *
40     * @todo Set this to the unique name of your IoT Thing.
41     */
42     #define clientcredentialIOT_THING_NAME      "thingname"
```

- e. 開啟 tools/certificate_configuration/CertificateConfigurator.html 檔案。
- f. Import the certificate PEM file and Private Key PEM file that you downloaded earlier.
- g. Choose Generate and save aws_clientcredential_keys.h and replace this file in the /demos/include/ directory.

Certificate Configuration Tool

FreeRTOS Developer Demos

Provide client certificate and private key PEM files downloaded from the AWS IoT Console.

Certificate PEM file:

No file chosen

Private Key PEM file:

No file chosen



 Save the generated header file to the demos/common/include folder of the demo project.

Copyright (C) 2017 Amazon.com, Inc. or its affiliates. All Rights Reserved.

- h. Open the /demos/include/aws_ota_codesigner_certificate.h file, and specify these values.

```
#define signingcredentialSIGNING_CERTIFICATE_PEM [] = "your-certificate-key";
```

Where **your-certificate-key** is the value from the file secp256r1.crt. Remember to add "\n" after each line in the certification. For more information on creating the secp256r1.crt file, see [How to implement FreeRTOS OTA by using Amazon Web Services on RX65N](#) and section 7.3 "Generating ECDSA-SHA256 Key Pairs with OpenSSL" in [Renesas MCU Firmware Update Design Policy](#).

The screenshot shows the FreeRTOS Project Explorer and a code editor window. The Project Explorer lists several subfolders under 'aws_demos' such as 'Includes', 'application_code', 'config_files', 'demos', 'defender', 'demo_runner', 'dev_mode_key_provisioning', 'greengrass_connectivity', 'https', and 'include'. The 'include' folder contains files like 'aws_application_version.h', 'aws_clientcredential_keys.h', 'aws_clientcredential.h', 'aws_demo.h', 'aws_iot_demo_network.h', 'aws_ota_codesigner_certificate.h', 'iot_config_common.h', 'iot_demo_logging.h', 'iot_demo_runner.h', 'mqtt.h', 'network_manager.h', 'ota.h', 'shadow.h', and 'tcp.h'. The code editor window displays the 'aws_ota_codesigner_certificate.h' file, which includes a PEM-encoded certificate for a code signer.

```

2      * FreeRTOS V202002.00
3
4  ifndef __AWS_CODESIGN_KEYS_H__
5  define __AWS_CODESIGN_KEYS_H__
6
7  /*
8   * PEM-encoded code signer certificate
9   *
10  * Must include the PEM header and footer:
11  * "-----BEGIN CERTIFICATE-----\n"
12  * "...base64 data...\n"
13  * "-----END CERTIFICATE-----\n";
14  */
15  static const char signingcredentialsIGNING_CERTIFICATE_PEM[] =
16  "-----BEGIN CERTIFICATE-----\n"
17  "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n"
18  "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n"
19  "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n"
20  "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n"
21  "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n"
22  "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n"
23  "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n"
24  "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n"
25  "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n"
26  "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n"
27  "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n"
28  "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n"
29  "-----END CERTIFICATE-----\n";
30
31 #endif

```

13. Task A: Install the initial version of the firmware

- Open the `vendors/renesas/boards/board/aws_demos/config_files/aws_demo_config.h` file, comment out `#define CONFIG_MQTT_DEMO_ENABLED`, and define `CONFIG_OTA_UPDATE_DEMO_ENABLED`.
- Open the `/demos/include/ aws_application_version.h` file, and set the initial version of the firmware to 0.9.2.

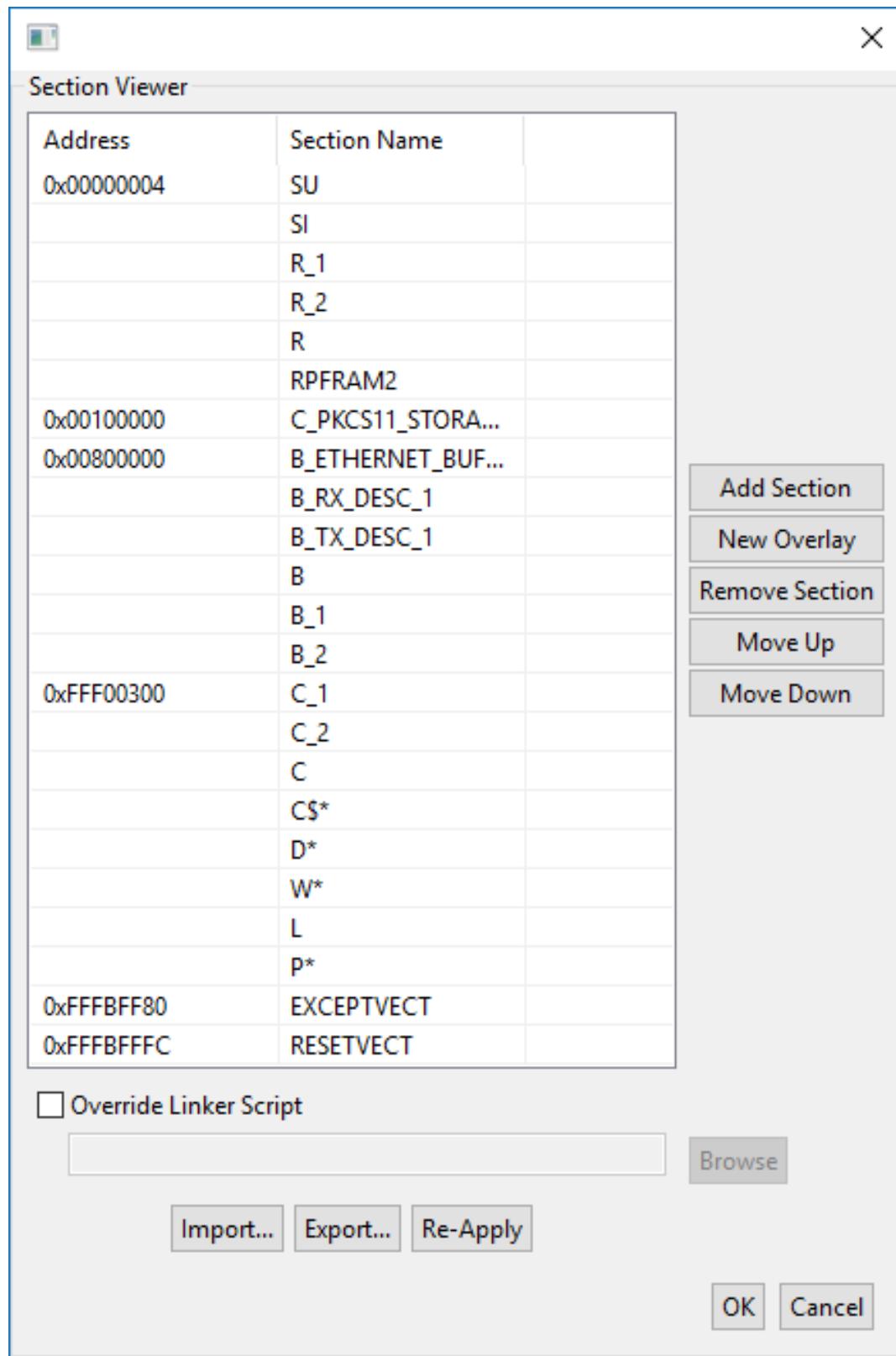
The screenshot shows the 'aws_application_version.h' file in the code editor. It defines a global constant 'xAppFirmwareVersion' of type 'AppVersion32_t' and sets its value to 0.9.2. The code also includes the 'iot_appversion32.h' header.

```

25
26  ifndef __AWS_APPLICATION_VERSION_H__
27  define __AWS_APPLICATION_VERSION_H__
28
29  include "iot_appversion32.h"
30  extern const AppVersion32_t xAppFirmwareVersion;
31
32  define APP_VERSION_MAJOR      0
33  define APP_VERSION_MINOR     9
34  define APP_VERSION_BUILD    2
35
36  endif
37

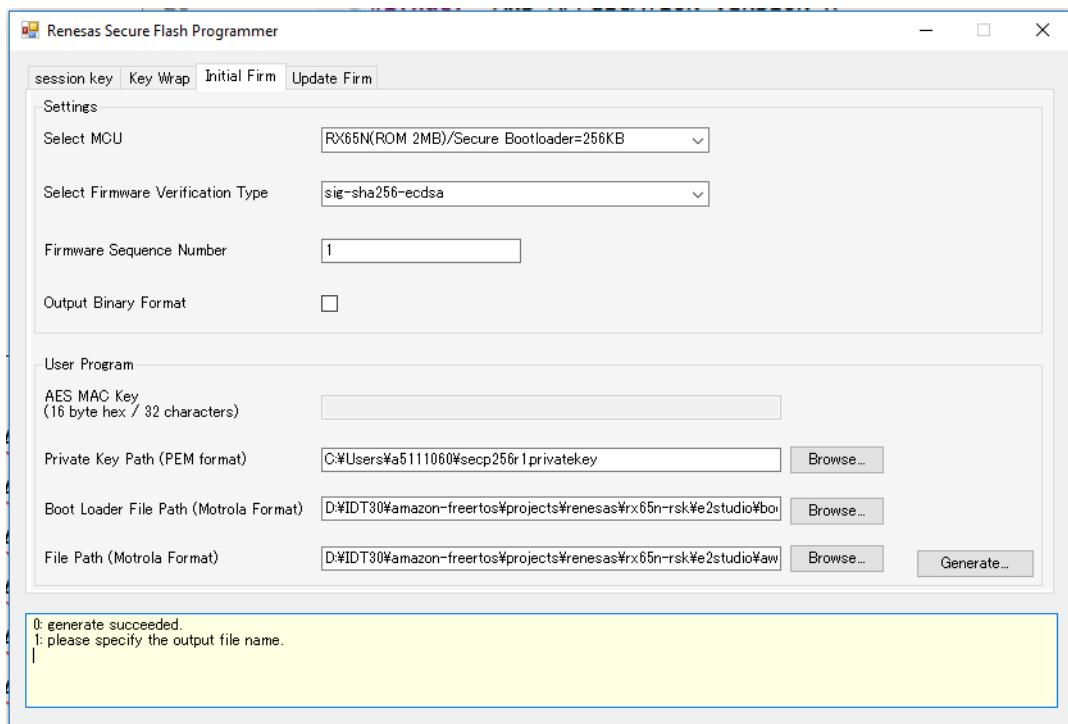
```

- Change the following settings in the Section Viewer.

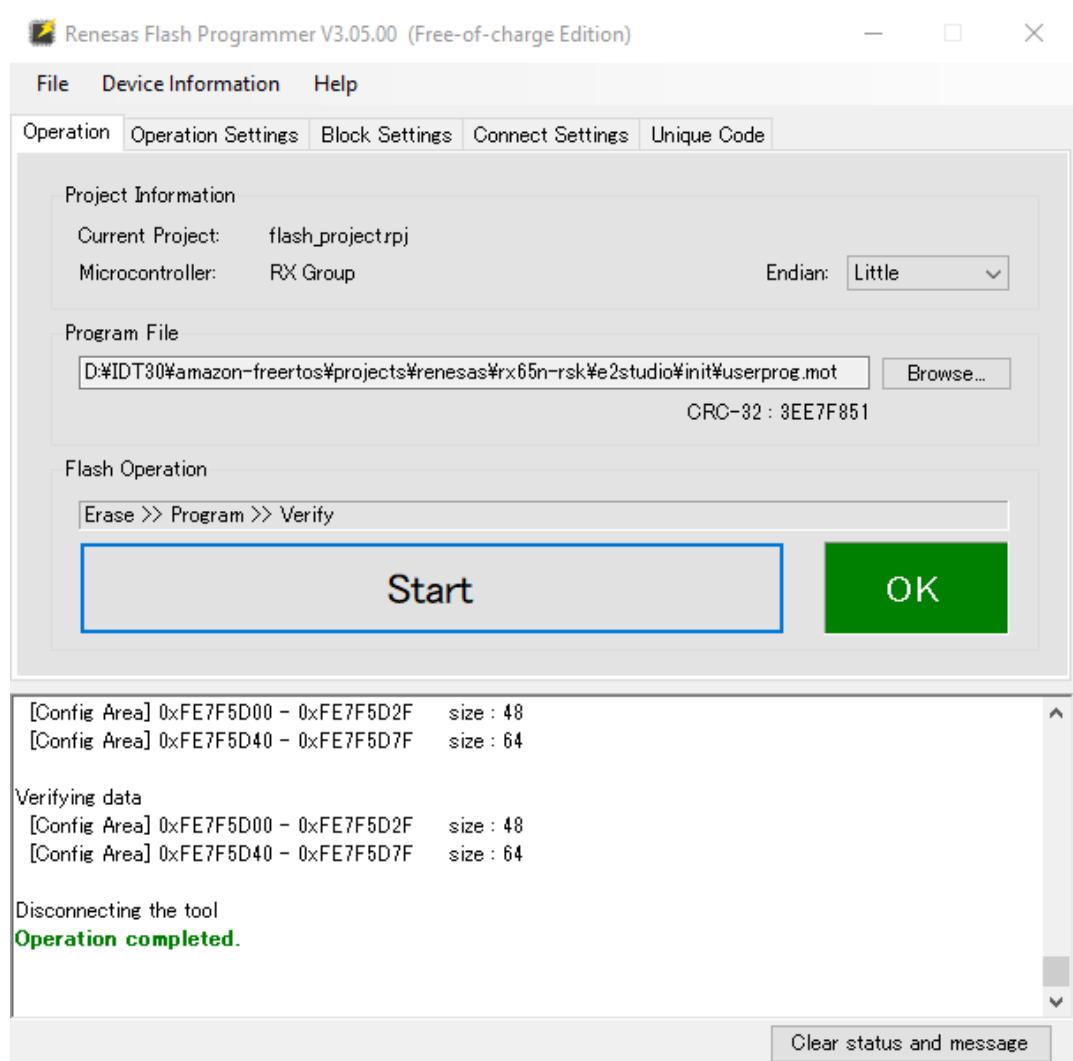


- d. Choose Build to create the aws_demos.mot file.

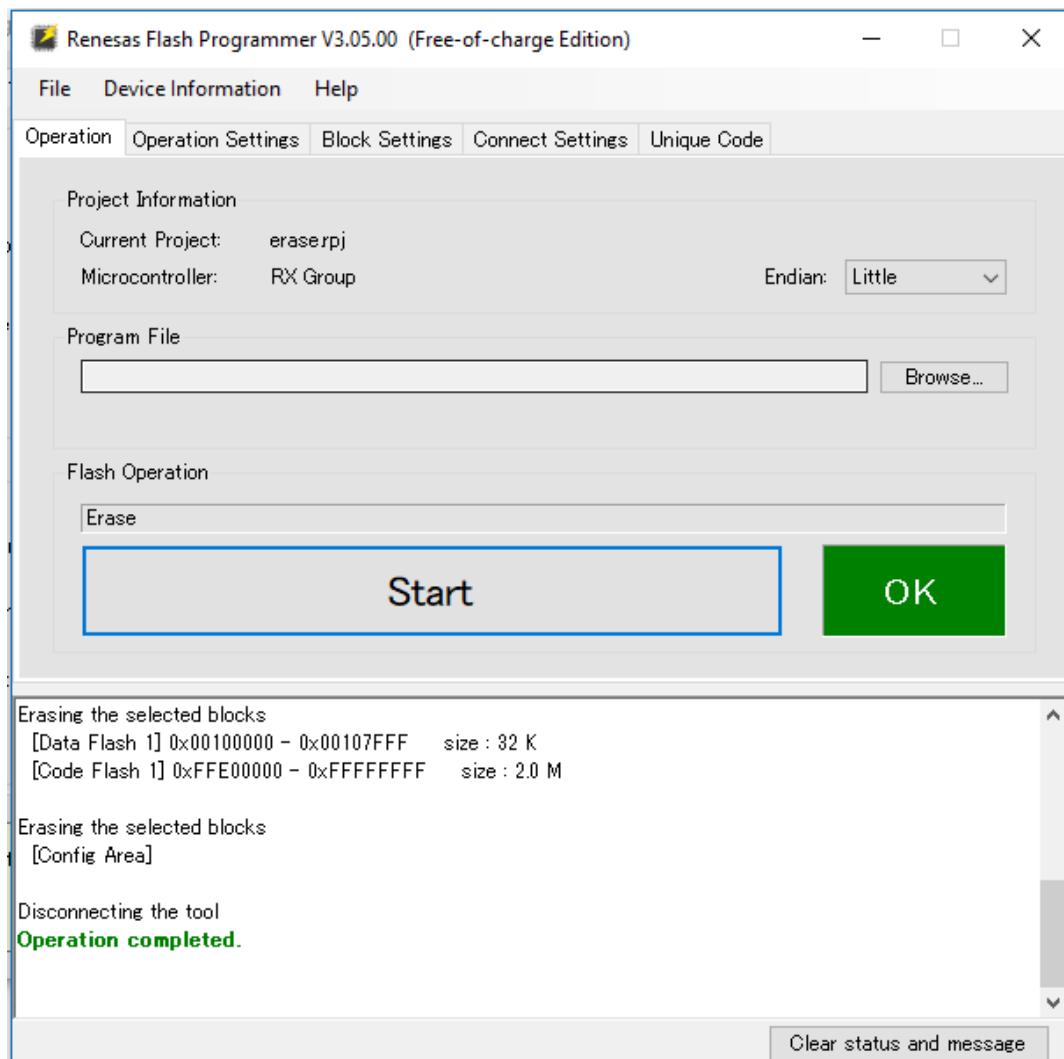
14. Create the file `userprog.mot` with the Renesas Secure Flash Programmer. `userprog.mot` is a combination of `aws_demos.mot` and `boot_loader.mot`. You can flash this file to the RX65N-RSK to install the initial firmware.
- Download <https://github.com/renesas/Amazon-FreeRTOS-Tools> and open Renesas Secure Flash Programmer.exe.
 - Choose the Initial Firm tab and then set the following parameters:
 - Private Key Path – The location of `secp256r1.privatekey`.
 - Boot Loader File Path– The location of `boot_loader.mot` (`projects\renesas\rx65n-rsk\c2studio\boot_loader\HardwareDebug`).
 - File Path – The location of the `aws_demos.mot` (`projects\renesas\rx65n-rsk\c2studio\aws_demos\HardwareDebug`).



- Create a directory named `init_firmware`, Generate `userprog.mot`, and save it to the `init_firmware` directory. Verify that the generate succeeded.
15. Flash the initial firmware on the RX65N-RSK.
- Download the latest version of the Renesas Flash Programmer (Programming GUI) from <https://www.renesas.com/tw/en/products/software-tools/tools/programmer/renesas-flash-programmer-programming-gui.html>.
 - Open the `vendors\renesas\rx_mcu_boards\boards\rx65n-rsk\aws_demos\flash_project\erase_from_bank\ erase.rpj` file to erase data on the bank.
 - Choose Start to erase the bank.



- d. To flash `userprog.mot`, choose `Browse...` and navigate to the `init_firmware` directory, select the `userprog.mot` file and choose `Start`.



16. Version 0.9.2 (initial version) of the firmware was installed to your RX65N-RSK. The RX65N-RSK board is now listening for OTA updates. If you have opened Tera Term on your PC, you see something like the following when the initial firmware runs.

```
-----
RX65N secure boot program
-----
Checking flash ROM status.
bank 0 status = 0xff [LIFECYCLE_STATE_BLANK]
bank 1 status = 0xfc [LIFECYCLE_STATE_INSTALLING]
bank info = 1. (start bank = 0)
start installing user program.
copy secure boot (part1) from bank0 to bank1...OK
copy secure boot (part2) from bank0 to bank1...OK
update LIFECYCLE_STATE from [LIFECYCLE_STATE_INSTALLING] to [LIFECYCLE_STATE_VALID]
bank1(temporary area) block0 erase (to update LIFECYCLE_STATE)...OK
bank1(temporary area) block0 write (to update LIFECYCLE_STATE)...OK
swap bank...
-----
RX65N secure boot program
-----
Checking flash ROM status.
```

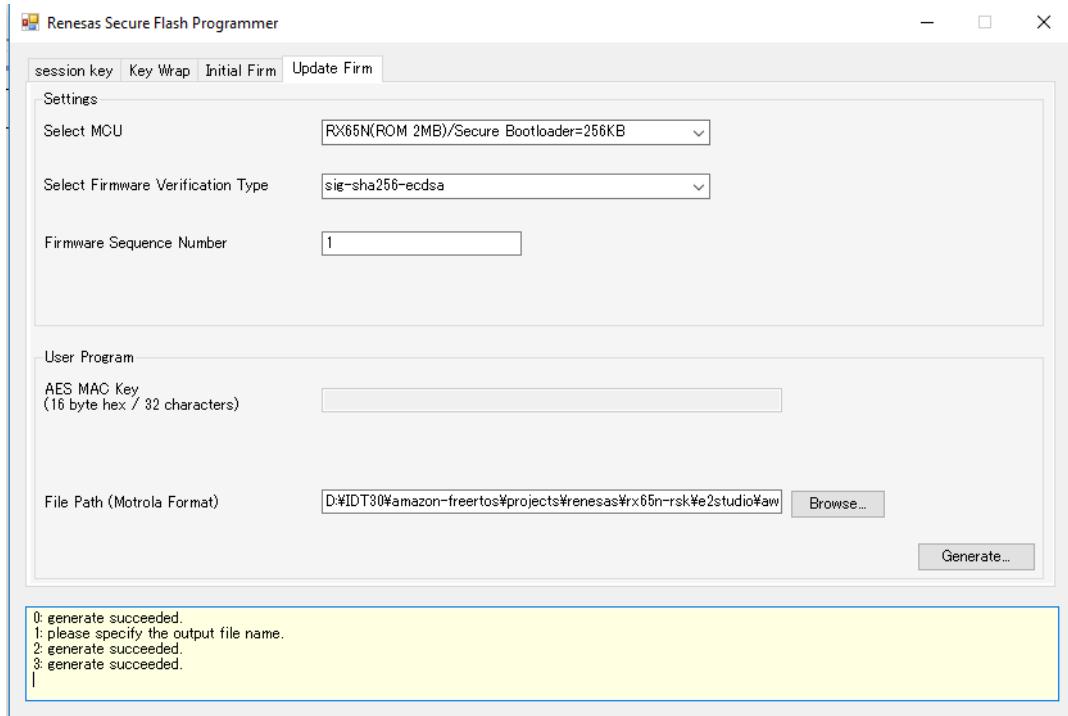
```
bank 0 status = 0xf8 [LIFECYCLE_STATE_VALID]
bank 1 status = 0xff [LIFECYCLE_STATE_BLANK]
bank info = 0. (start bank = 1)
integrity check scheme = sig-sha256-ecdsa
bank0(execute area) on code flash integrity check...OK
jump to user program
#0 1 [ETHER_RECEI] Deferred Interrupt Handler Task started
1 1 [ETHER_RECEI] Network buffers: 3 lowest 3
2 1 [ETHER_RECEI] Heap: current 234192 lowest 234192
3 1 [ETHER_RECEI] Queue space: lowest 8
4 1 [IP-task] InitializeNetwork returns OK
5 1 [IP-task] xNetworkInterfaceInitialise returns 0
6 101 [ETHER_RECEI] Heap: current 234592 lowest 233392
7 2102 [ETHER_RECEI] prvEMACHandlerTask: PHY LS now 1
8 3001 [IP-task] xNetworkInterfaceInitialise returns 1
9 3092 [ETHER_RECEI] Network buffers: 2 lowest 2
10 3092 [ETHER_RECEI] Queue space: lowest 7
11 3092 [ETHER_RECEI] Heap: current 233320 lowest 233320
12 3193 [ETHER_RECEI] Heap: current 233816 lowest 233120
13 3593 [IP-task] vDHCPPProcess: offer c0a80a09ip
14 3597 [ETHER_RECEI] Heap: current 233200 lowest 233000
15 3597 [IP-task] vDHCPPProcess: offer c0a80a09ip
16 3597 [IP-task] IP Address: 192.168.10.9
17 3597 [IP-task] Subnet Mask: 255.255.255.0
18 3597 [IP-task] Gateway Address: 192.168.10.1
19 3597 [IP-task] DNS Server Address: 192.168.10.1
20 3600 [Tmr Svc] The network is up and running
21 3622 [Tmr Svc] Write certificate...
22 3697 [ETHER_RECEI] Heap: current 232320 lowest 230904
23 4497 [ETHER_RECEI] Heap: current 226344 lowest 225944
24 5317 [iot_thread] [INFO ][DEMO][5317] -----STARTING DEMO-----
25 5317 [iot_thread] [INFO ][INIT][5317] SDK successfully initialized.
26 5317 [iot_thread] [INFO ][DEMO][5317] Successfully initialized the demo. Network
type for the demo: 4
27 5317 [iot_thread] [INFO ][MQTT][5317] MQTT library successfully initialized.
28 5317 [iot_thread] [INFO ][DEMO][5317] OTA demo version 0.9.2
29 5317 [iot_thread] [INFO ][DEMO][5317] Connecting to broker...
30 5317 [iot_thread] [INFO ][DEMO][5317] MQTT demo client identifier is rx65n-gr-rose
(length 13).
31 5325 [ETHER_RECEI] Heap: current 206944 lowest 206504
32 5325 [ETHER_RECEI] Heap: current 206440 lowest 206440
33 5325 [ETHER_RECEI] Heap: current 206240 lowest 206240
38 5334 [ETHER_RECEI] Heap: current 190288 lowest 190288
39 5334 [ETHER_RECEI] Heap: current 190088 lowest 190088
40 5361 [ETHER_RECEI] Heap: current 158512 lowest 158168
41 5363 [ETHER_RECEI] Heap: current 158032 lowest 158032
42 5364 [ETHER_RECEI] Network buffers: 1 lowest 1
43 5364 [ETHER_RECEI] Heap: current 156856 lowest 156856
44 5364 [ETHER_RECEI] Heap: current 156656 lowest 156656
46 5374 [ETHER_RECEI] Heap: current 153016 lowest 152040
47 5492 [ETHER_RECEI] Heap: current 141464 lowest 139016
48 5751 [ETHER_RECEI] Heap: current 140160 lowest 138680
49 5917 [ETHER_RECEI] Heap: current 138280 lowest 138168
59 7361 [iot_thread] [INFO ][MQTT][7361] Establishing new MQTT connection.
62 7428 [iot_thread] [INFO ][MQTT][7428] (MQTT connection 81cfc8, CONNECT operation
81d0e8) Wait complete with result SUCCESS.
63 7428 [iot_thread] [INFO ][MQTT][7428] New MQTT connection 4e8c established.
64 7430 [iot_thread] [OTA_AgentInit_internal] OTA Task is Ready.
65 7430 [OTA Agent T] [prvOTAAgentTask] Called handler. Current State [Ready] Event
[Start] New state [RequestingJob]
66 7431 [OTA Agent T] [INFO ][MQTT][7431] (MQTT connection 81cfc8) SUBSCRIBE operation
scheduled.
```

```
67 7431 [OTA Agent T] [INFO ][MQTT][7431] (MQTT connection 81cf8c, SUBSCRIBE operation 818c48) Waiting for operation completion.
68 7436 [ETHER_RECEI] Heap: current 128248 lowest 127992
69 7480 [OTA Agent T] [INFO ][MQTT][7480] (MQTT connection 81cf8c, SUBSCRIBE operation 818c48) Wait complete with result SUCCESS.
70 7480 [OTA Agent T] [prvSubscribeToJobNotificationTopics] OK: $aws/things/rx65n-gr-rose/jobs/$next/get/accepted
71 7481 [OTA Agent T] [INFO ][MQTT][7481] (MQTT connection 81cf8c) SUBSCRIBE operation scheduled.
72 7481 [OTA Agent T] [INFO ][MQTT][7481] (MQTT connection 81cf8c, SUBSCRIBE operation 818c48) Waiting for operation completion.
73 7530 [OTA Agent T] [INFO ][MQTT][7530] (MQTT connection 81cf8c, SUBSCRIBE operation 818c48) Wait complete with result SUCCESS.
74 7530 [OTA Agent T] [prvSubscribeToJobNotificationTopics] OK: $aws/things/rx65n-gr-rose/jobs/notify-next
75 7530 [OTA Agent T] [prvRequestJob_Mqtt] Request #0
76 7532 [OTA Agent T] [INFO ][MQTT][7532] (MQTT connection 81cf8c) MQTT PUBLISH operation queued.
77 7532 [OTA Agent T] [INFO ][MQTT][7532] (MQTT connection 81cf8c, PUBLISH operation 818b80) Waiting for operation completion.
78 7552 [OTA Agent T] [INFO ][MQTT][7552] (MQTT connection 81cf8c, PUBLISH operation 818b80) Wait complete with result SUCCESS.
79 7552 [OTA Agent T] [prvOTAAgentTask] Called handler. Current State [RequestingJob]
Event [RequestJobDocument] New state [WaitingForJob]
80 7552 [OTA Agent T] [prvParseJSONbyModel] Extracted parameter [ clientToken: 0:rx65n-gr-rose ]
81 7552 [OTA Agent T] [prvParseJSONbyModel] parameter not present: execution
82 7552 [OTA Agent T] [prvParseJSONbyModel] parameter not present: jobId
83 7552 [OTA Agent T] [prvParseJSONbyModel] parameter not present: jobDocument
84 7552 [OTA Agent T] [prvParseJSONbyModel] parameter not present: afr_ota
85 7552 [OTA Agent T] [prvParseJSONbyModel] parameter not present: protocols
86 7552 [OTA Agent T] [prvParseJSONbyModel] parameter not present: files
87 7552 [OTA Agent T] [prvParseJSONbyModel] parameter not present: filepath
99 7651 [ETHER_RECEI] Heap: current 129720 lowest 127304
100 8430 [iot_thread] [INFO ][DEMO][8430] State: Ready Received: 1 Queued: 0
Processed: 0 Dropped: 0
101 9430 [iot_thread] [INFO ][DEMO][9430] State: WaitingForJob Received: 1 Queued: 0
Processed: 0 Dropped: 0
102 10430 [iot_thread] [INFO ][DEMO][10430] State: WaitingForJob Received: 1 Queued: 0
Processed: 0 Dropped: 0
103 11430 [iot_thread] [INFO ][DEMO][11430] State: WaitingForJob Received: 1 Queued: 0
Processed: 0 Dropped: 0
104 12430 [iot_thread] [INFO ][DEMO][12430] State: WaitingForJob Received: 1 Queued: 0
Processed: 0 Dropped: 0
105 13430 [iot_thread] [INFO ][DEMO][13430] State: WaitingForJob Received: 1 Queued: 0
Processed: 0 Dropped: 0
106 14430 [iot_thread] [INFO ][DEMO][14430] State: WaitingForJob Received: 1 Queued: 0
Processed: 0 Dropped: 0
107 15430 [iot_thread] [INFO ][DEMO][15430] State: WaitingForJob Received: 1 Queued: 0
Processed: 0 Dropped: 0
```

17. Task B: Update the version of your firmware

- a. Open the demos/include/aws_application_version.h file and increment the APP_VERSION_BUILD token value to 0.9.3.
 - b. 重新建置專案。
18. Create the userprog.rsu file with the Renesas Secure Flash Programmer to update the version of your firmware.
- a. 開啟 Amazon-FreeRTOS-Tools\Renesis Secure Flash Programmer.exe 檔案。
 - b. Choose the Update Firm tab and set the following parameters:

- File Path – The location of the `aws_demos.mot` file (`projects\renesas\rx65n-rsk\ee2studio\aws_demos\HardwareDebug`).
- c. Create a directory named `update_firmware`. Generate `userprog.rsu` and save it to the `update_firmware` directory. Verify that the generate succeeded.



19. Upload the firmware update, `userproj.rsu`, into an Amazon S3 bucket as described in [建立 Amazon S3 儲存貯體來存放您的更新 \(p. 120\)](#).

Name	Last modified	Size
Signedimages	--	--
userprog.rsu	May 18, 2020 2:00:37 PM GMT+0900	767.5 KB

20. Create a job to update firmware on the RX65N-RSK.

AWS IoT Jobs is a service that notifies one or more connected devices of a pending [Job](#). A job can be used to manage a fleet of devices, update firmware and security certificates on devices, or perform administrative tasks such as restarting devices and performing diagnostics.

- a. 登入 [AWS IoT 主控台](#)。In the navigation pane, choose Manage, and choose Jobs.
- b. Choose Create a job, then choose Create OTA Update job. Select a thing, then choose Next.
- c. Create a FreeRTOS OTA update job as follows:

- Choose MQTT.
- Select the code signing profile you created in the previous section.
- Select the firmware image that you uploaded to an Amazon S3 bucket.
- For Pathname of firmware image on device, enter **test**.
- Choose the IAM role that you created in the previous section.

d. 選擇 Next (下一步)。

MQTT

Select and sign your firmware image

Code signing ensures that devices only run code published by trusted authors and that the code has not been altered or corrupted since it was signed. You have three options for code signing. [Learn more](#)

Sign a new firmware image for me

Select a previously signed firmware image

Use my custom signed firmware image

Code signing profile [Learn more](#)

ota_signing	SHA256	ECDSA	aaaaaaaa	Clear Change
-------------	--------	-------	----------	--

Select your firmware image in S3 or upload it

userprog.rsu	Change
--------------	------------------------

Pathname of firmware image on device [Learn more](#)

test

IAM role for OTA update job

Choose a role which grants AWS IoT access to the S3, AWS IoT jobs and AWS Code signing resources to create an OTA update job. [Learn more](#)

Role (requires S3 access)

ota_test_beginner	Select
-------------------	------------------------

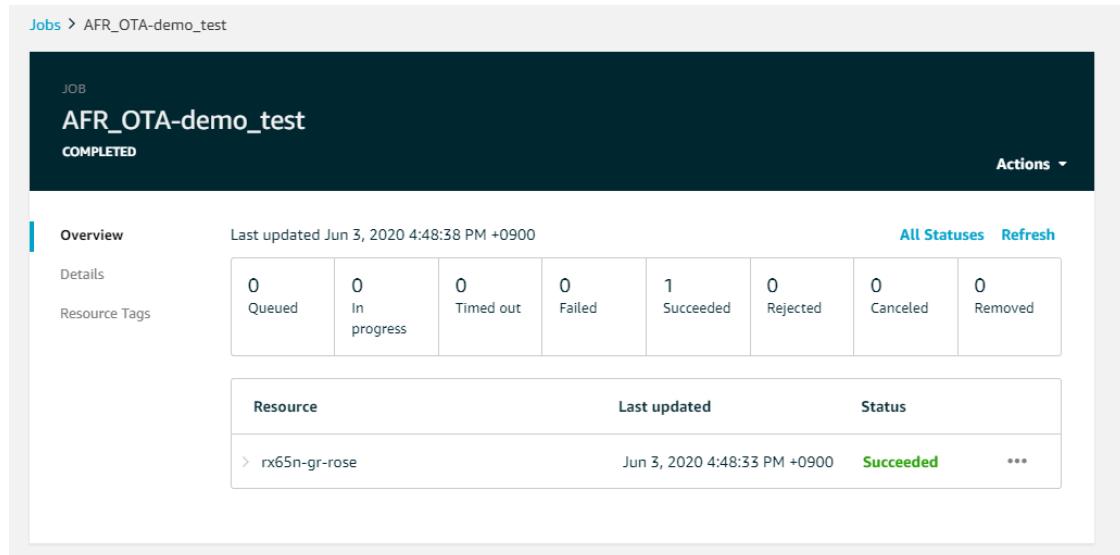
[Cancel](#) [Back](#) [Next](#)

e. Enter an ID and then choose Create.

21. Reopen Tera Term to verify that the firmware was updated successfully to OTA demo version 0.9.3.

```
11 5000 Tim over the network is up and running
22 10710 [Thr Svc] Write certificate...
23 10752 [ETHER RECEI] Heap: current 232336 lowest 232136
24 11652 [ETHER RECEI] Heap: current 226352 lowest 225952
25 12405 [iot_thread] [INFO ][DEMO][12405] -----STARTING DEMO-----
26 12405 [iot_thread] [INFO ][INIT][12405] SDK successfully initialized.
27 12405 [iot_thread] [INFO ][DEMO][12405] Successfully initialized the demo. Network type for the demo: 4
28 12405 [iot_thread] [INFO ][MQTT][12405] MQTT library successfully initialized.
29 12405 [iot_thread] [INFO ][DEMO][12405] OTA demo version 0.9.3
30 12405 [iot_thread] [INFO ][DEMO][12405] Connecting to broker...
31 12405 [iot_thread] [INFO ][DEMO][12405] MQTT demo client identifier is rx65n-gr-rose (length 13).
```

22. On the AWS IoT console, verify that the job status is Succeeded.



AWS IoT 裝置影子示範應用程式

Introduction

此示範示範如何使用 AWS IoT 裝置影子程式庫來連接到 [裝置影子服務AWS](#)。它使用 [coreMQTT 程式庫](#) (p. 195) 建立 MQTT 對 AWS IoT MQTT 中介裝置和 coreJSON 程式庫剖析器的 TLS (相互身份驗證) 連接，剖析從 AWS Shadow 服務收到的影子文件。此示範顯示基本的影子操作，例如如何更新影子文件以及如何移除影子文件。示範也示範如何使用 coreMQTT 程式庫來登記回呼函數，以處理類似影子 /update 和 /update/delta 訊息的訊息，這些訊息是由 AWS IoT Device Shadow 服務所傳送。

此示範的目的是做為學習練習，因為更新影子文件（狀態）的請求和更新回應是由相同的應用程式完成。在實際執行時，外部應用程式會請求遠端更新裝置狀態，即使裝置目前未連接也一樣。裝置會在連接時確認更新請求。

Note

若要設定和執行 FreeRTOS 示範，請遵循[FreeRTOS 入門 \(p. 14\)](#)中的步驟。

Functionality

示範會建立單一應用程式任務，以循環執行一組範例，示範陰影 /update 和 /update/delta 回呼以模擬切換遠端裝置的狀態。它會傳送新的 desired 狀態陰影更新，並等待裝置變更其 reported 狀態，以回應新的 desired 狀態。此外，陰影 /update 回呼會用於列印變更的陰影狀態。此示範也使用 AWS IoT MQTT 中介裝置的安全 MQTT 連接，並假設裝置影子中存在 powerOn 狀態。

示範會執行下列操作：

1. 使用 shadow_demo_helpers.c 中的協助程式函數來建立 MQTT 連接。
2. 使用由 AWS IoT Device Shadow 程式庫定義的巨集，為裝置陰影操作組裝 MQTT 主題字串。
3. 發行到用於將裝置影子除名給 MQTT 主題，以刪除任何現有的裝置影子。
4. 使用 /update/delta 中的協助程式函數，來訂用 /update/accepted、/update/rejected 和 shadow_demo_helpers.c 的 MQTT 主題。

5. 使用 powerOn 中的協助程式函數，發行 shadow_demo_helpers.c 所需的狀態。這會導致 /update/delta 訊息傳送到裝置。
6. 在 prvEventCallback 中處理傳入的 MQTT 訊息，並使用 AWS IoT Device Shadow 程式庫 (Shadow_MatchTopic) 定義的函數，判斷訊息是否與裝置影子相關。如果訊息是裝置影子 / update/delta 訊息，則主要示範函數會發布第二個訊息，將回報的狀態更新為 powerOn。如果收到 / update/accepted 訊息，請確認它與先前在更新訊息中發布的 clientToken 相同。這會標記示範的結尾。

```

82 9136 [ShadowDemo] [INFO] [SHADOW] [prvShadowDemoTask:641] 83 9136 [ShadowDemo] Send desired power state with 1.84 9136 [ShadowDemo]
85 9296 [ShadowDemo] [INFO] [SHADOW] [prvEventCallback:482] 86 9296 [ShadowDemo] pPublishInfo->pTopicName:$aws/things/testClient16:34:41/shadow/update/delta.87 9296 [ShadowDemo]
88 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateDeltaHandler:256] 89 9296 [ShadowDemo] /update/delta json payload:"version":1,"timestamp":1602751002,"state":{"powerOn":1},
,"metadata":{"powerOn":{"timestamp":1602751002}},"clientToken":"009136"},90 9296 [ShadowDemo]
91 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateDeltaHandler:298] 92 9296 [ShadowDemo] version: 193 9296 [ShadowDemo]
94 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateDeltaHandler:308] 95 9296 [ShadowDemo] version:1, ulCurrentVersion:0
96 9296 [ShadowDemo]
97 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateDeltaHandler:342] 98 9296 [ShadowDemo] The new power on state newState:1, ulCurrentPowerOnState:0
99 9296 [ShadowDemo]
100 9296 [ShadowDemo] [INFO] [SHADOW] [prvEventCallback:482] 101 9296 [ShadowDemo] pPublishInfo->pTopicName:$aws/things/testClient16:34:41/shadow/update/accepted.102 9296 [ShadowDemo]
103 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:376] 104 9296 [ShadowDemo] /update/accepted json payload:"state":{"desired":{"powerOn":1}}, "metadata":{"desired":{"powerOn":{"timestamp":1602751002}}}, "version":1, "timestamp":1602751002, "clientToken":"009136"},90 9296 [ShadowDemo]
106 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:424] 107 9296 [ShadowDemo] clientToken: 009136108 9296 [ShadowDemo]
109 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:429] 110 9296 [ShadowDemo] receivedToken:9136, clientToken:0
111 9296 [ShadowDemo]
112 9296 [ShadowDemo] [WARN] [SHADOW] [prvUpdateAcceptedHandler:442] 113 9296 [ShadowDemo] The received clientToken:9136 is not identical with the one@0 we sent 114 9296 [ShadowDemo]
115 9696 [ShadowDemo] [INFO] [SHADOW] [prvShadowDemoTask:670] 116 9696 [ShadowDemo] Report to the state change: 1117 9696 [ShadowDemo]
118 9856 [ShadowDemo] [INFO] [SHADOW] [prvEventCallback:482] 119 9856 [ShadowDemo] pPublishInfo->pTopicName:$aws/things/testClient16:34:41/shadow/update/accepted.120 9856 [ShadowDemo]
121 9856 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:376] 122 9856 [ShadowDemo] /update/accepted json payload:"state":{"reported":{"powerOn":1}}, "metadata":{"reported":{"powerOn":{"timestamp":1602751003}}}, "version":2, "timestamp":1602751003, "clientToken":"009696"},123 9856 [ShadowDemo]
124 9856 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:424] 125 9856 [ShadowDemo] clientToken: 009696126 9856 [ShadowDemo]
127 9856 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:429] 128 9856 [ShadowDemo] receivedToken:9696, clientToken:9696
129 9856 [ShadowDemo]
130 9856 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:437] 131 9856 [ShadowDemo] Received response from the device shadow. Previously published update with clientToken:9696 has been accepted. 132 9856 [ShadowDemo]
133 10256 [ShadowDemo] [INFO] [SHADOW] [prvShadowDemoTask:698] 134 10256 [ShadowDemo] Start to unsubscribe shadow topics and disconnect from MQTT.
135 10256 [ShadowDemo]
136 12036 [ShadowDemo] [INFO] [SHADOW] [prvShadowDemoTask:747] 137 12036 [ShadowDemo] Demo completed successfully.138 12036 [ShadowDemo]
139 12036 [ShadowDemo] [INFO] [SHADOW] [prvShadowDemoTask:755] 140 12036 [ShadowDemo] Deleting Shadow Demo Task.141 12036 [ShadowDemo]

```

以下顯示示範的結構。

```

void prvShadowDemoTask( void * pvParameters )
{
    BaseType_t demoStatus = pdPASS;

    /* A buffer containing the update document. It has static duration to
     * prevent it from being placed on the call stack. */
    static char pcUpdateDocument[ SHADOW_REPORTED_JSON_LENGTH + 1 ] = { 0 };

    demoStatus = xEstablishMqttSession( prvEventCallback );

    if( pdFAIL == demoStatus )
    {
        /* Log error to indicate connection failure. */
        LogError( ( "Failed to connect to MQTT broker." ) );
    }
    else
    {
        /* First of all, try to delete any Shadow document in the cloud. */
        demoStatus = xPublishToTopic(
                    SHADOW_TOPIC_STRING_DELETE( THING_NAME ),
                    SHADOW_TOPIC_LENGTH_DELETE( THING_NAME_LENGTH ),
                    pcUpdateDocument,
                    0U );

        /* Then try to subscribe to the shadow topics. */
        if( demoStatus == pdPASS )
        {
            demoStatus = xSubscribeToTopic(
                        SHADOW_TOPIC_STRING_UPDATE_DELTA( THING_NAME ),
                        SHADOW_TOPIC_LENGTH_UPDATE_DELTA( THING_NAME_LENGTH ) );
        }
    }
}

```

```
if( demoStatus == pdPASS )
{
    demoStatus = xSubscribeToTopic(
        SHADOW_TOPIC_STRING_UPDATE_ACCEPTED( THING_NAME ),
        SHADOW_TOPIC_LENGTH_UPDATE_ACCEPTED( THING_NAME_LENGTH ) );
}

if( demoStatus == pdPASS )
{
    demoStatus = xSubscribeToTopic(
        SHADOW_TOPIC_STRING_UPDATE_REJECTED( THING_NAME ),
        SHADOW_TOPIC_LENGTH_UPDATE_REJECTED( THING_NAME_LENGTH ) );
}

/* This demo uses a constant #THING_NAME known at compile time
 * therefore we can use macros to assemble shadow topic strings.
 * If the thing name is known at run time, then we could use the API
 * #Shadow_GetTopicString to assemble shadow topic strings, here is
 * the example for /update/delta:
 *
 * For /update/delta:
 *
 * #define SHADOW_TOPIC_MAX_LENGTH (256U)
 *
 * ShadowStatus_t shadowStatus = SHADOW_STATUS_SUCCESS;
 * char cTopicBuffer[ SHADOW_TOPIC_MAX_LENGTH ] = { 0 };
 * uint16_t usBufferSize = SHADOW_TOPIC_MAX_LENGTH;
 * uint16_t usOutLength = 0;
 * const char * pcThingName = "TestThingName";
 * uint16_t usThingNameLength = ( sizeof( pcThingName ) - 1U );
 *
 * shadowStatus = Shadow_GetTopicString(
 *     SHADOW_TOPIC_STRING_TYPE_UPDATE_DELTA,
 *     pcThingName,
 *     usThingNameLength,
 *     &( cTopicBuffer[ 0 ] ),
 *     usBufferSize,
 *     & usOutLength );
 */

/* Then we publish a desired state to the /update topic. Since we've
 * deleted the device shadow at the beginning of the demo, this will
 * cause a delta message to be published, which we have subscribed to.
 * In many real applications, the desired state is not published by
 * the device itself. But for the purpose of making this demo
 * self-contained, we publish one here so that we can receive a delta
 * message later.
 */
if( demoStatus == pdPASS )
{
    /* Desired power on state . */
    LogInfo( ( "Send desired power state with 1." ) );

    ( void ) memset( pcUpdateDocument,
                    0x00,
                    sizeof( pcUpdateDocument ) );

    snprintf( pcUpdateDocument,
              SHADOW_DESIRED_JSON_LENGTH + 1,
              SHADOW_DESIRED_JSON,
              ( int ) 1,
              ( long unsigned ) ( xTaskGetTickCount() % 1000000 ) );

    demoStatus = xPublishToTopic(
        SHADOW_TOPIC_STRING_UPDATE( THING_NAME ),
```

```
        SHADOW_TOPIC_LENGTH_UPDATE( THING_NAME_LENGTH ),
        pcUpdateDocument,
        ( SHADOW_DESIRED_JSON_LENGTH + 1 ) );
    }

    if( demoStatus == pdPASS )
    {
        /* Note that PublishToTopic already called MQTT_ProcessLoop,
         * therefore responses may have been received and the
         * prvEventCallback may have been called, which may have changed
         * the stateChanged flag. Check if the state change flag has been
         * modified or not. If it's modified, then we publish reported
         * state to update topic.
        */
        if( stateChanged == true )
        {
            /* Report the latest power state back to device shadow. */
            LogInfo( ( "Report to the state change: %d", ulCurrentPowerOnState ) );
            ( void ) memset( pcUpdateDocument,
                            0x00,
                            sizeof( pcUpdateDocument ) );

            /* Keep the client token in global variable used to compare if
             * the same token in /update/accepted. */
            ulClientToken = ( xTaskGetTickCount() % 1000000 );

            snprintf( pcUpdateDocument,
                      SHADOW_REPORTED_JSON_LENGTH + 1,
                      SHADOW_REPORTED_JSON,
                      ( int ) ulCurrentPowerOnState,
                      ( long unsigned ) ulClientToken );

            demoStatus = xPublishToTopic(
                SHADOW_TOPIC_STRING_UPDATE( THING_NAME ),
                SHADOW_TOPIC_LENGTH_UPDATE( THING_NAME_LENGTH ),
                pcUpdateDocument,
                ( SHADOW_DESIRED_JSON_LENGTH + 1 ) );
        }
        else
        {
            LogInfo( (
                "No change from /update/delta, unsubscribe all shadow topics and
disconnect from MQTT.\r\n" ) );
        }
    }

    if( demoStatus == pdPASS )
    {
        LogInfo( ( "Start to unsubscribe shadow topics and disconnect from MQTT. \r
\n" ) );

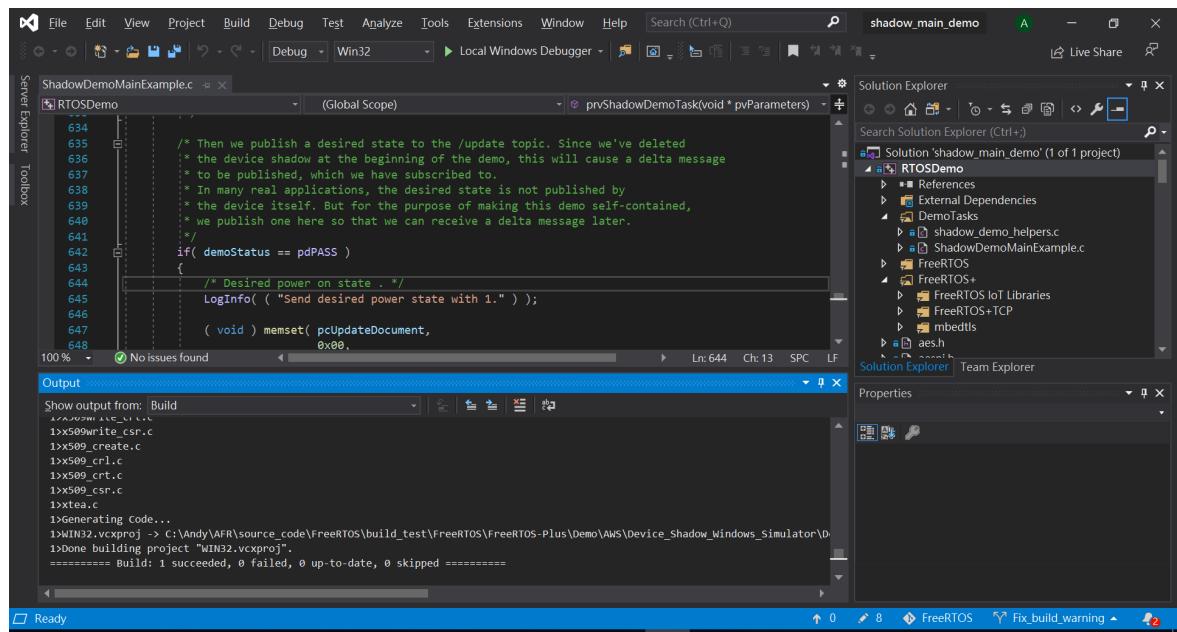
        demoStatus = xUnsubscribeFromTopic(
            SHADOW_TOPIC_STRING_UPDATE_DELTA( THING_NAME ),
            SHADOW_TOPIC_LENGTH_UPDATE_DELTA( THING_NAME_LENGTH ) );

        if( demoStatus != pdPASS )
        {
            LogError( ( "Failed to unsubscribe the topic %s",
                        SHADOW_TOPIC_STRING_UPDATE_DELTA( THING_NAME ) ) );
        }
    }

    if( demoStatus == pdPASS )
    {
        demoStatus = xUnsubscribeFromTopic(
            SHADOW_TOPIC_STRING_UPDATE_ACCEPTED( THING_NAME ),
```

```
SHADOW_TOPIC_LENGTH_UPDATE_ACCEPTED( THING_NAME_LENGTH ) );  
  
    if( demoStatus != pdPASS )  
    {  
        LogError( ( "Failed to unsubscribe the topic %s",  
                    SHADOW_TOPIC_STRING_UPDATE_ACCEPTED( THING_NAME ) ) );  
    }  
}  
  
if( demoStatus == pdPASS )  
{  
    demoStatus = xUnsubscribeFromTopic(  
        SHADOW_TOPIC_STRING_UPDATE_REJECTED( THING_NAME ),  
        SHADOW_TOPIC_LENGTH_UPDATE_REJECTED( THING_NAME_LENGTH ) );  
  
    if( demoStatus != pdPASS )  
    {  
        LogError( ( "Failed to unsubscribe the topic %s",  
                    SHADOW_TOPIC_STRING_UPDATE_REJECTED( THING_NAME ) ) );  
    }  
}  
  
/* The MQTT session is always disconnected, even there were prior  
 * failures. */  
demoStatus = xDisconnectMqttSession();  
  
/* This demo performs only Device Shadow operations. If matching the  
 * Shadow MQTT topic fails or there are failure in parsing the  
 * received JSON document, then this demo was not successful. */  
if( ( xUpdateAcceptedReturn != pdPASS ) || ( xUpdateDeltaReturn != pdPASS ) )  
{  
    LogError( ( "Callback function failed." ) );  
}  
  
if( demoStatus == pdPASS )  
{  
    LogInfo( ( "Demo completed successfully." ) );  
}  
else  
{  
    LogError( ( "Shadow Demo failed." ) );  
}  
}  
  
/* Delete this task. */  
LogInfo( ( "Deleting Shadow Demo task." ) );  
vTaskDelete( NULL );  
}
```

以下螢幕擷取畫面顯示示範成功時預期的輸出。



連接到 AWS IoT MQTT 中介裝置

若要連接到 AWS IoT MQTT 中介裝置，我們會使用與 `MQTT_Connect()` 中的 `coreMQTT` 交互身份驗證示範 (p. 241) 相同的方法。

刪除影子文件

若要刪除影子文件，請使用 `xPublishToTopic` Device Shadow 程式庫定義的巨集，以空白訊息呼叫 AWS IoT。這會使用 `MQTT_Publish` 來發布至 `/delete` 主題。下列程式碼區段顯示如何在 `prvShadowDemoTask` 函數中完成此作業。

```
/* First of all, try to delete any Shadow document in the cloud. */
returnStatus = PublishToTopic( SHADOW_TOPIC_STRING_DELETE( THING_NAME ),
                               SHADOW_TOPIC_LENGTH_DELETE( THING_NAME_LENGTH ),
                               pcUpdateDocument,
                               0U );
```

訂用影子主題

訂用 Device Shadow 主題來接收來自 AWS IoT 中介裝置有關影子變更的通知。Device Shadow 主題是由 Device Shadow 程式庫中定義的巨集整合。下列程式碼區段顯示如何在 `prvShadowDemoTask` 函數中完成此作業。

```
/* Then try to subscribe shadow topics. */
if( returnStatus == EXIT_SUCCESS )
{
    returnStatus = SubscribeToTopic(
        SHADOW_TOPIC_STRING_UPDATE_DELTA( THING_NAME ),
        SHADOW_TOPIC_LENGTH_UPDATE_DELTA( THING_NAME_LENGTH ) );
}

if( returnStatus == EXIT_SUCCESS )
{
```

```
    returnStatus = SubscribeToTopic(
        SHADOW_TOPIC_STRING_UPDATE_ACCEPTED( THING_NAME ),
        SHADOW_TOPIC_LENGTH_UPDATE_ACCEPTED( THING_NAME_LENGTH ) );
}

if( returnStatus == EXIT_SUCCESS )
{
    returnStatus = SubscribeToTopic(
        SHADOW_TOPIC_STRING_UPDATE_REJECTED( THING_NAME ),
        SHADOW_TOPIC_LENGTH_UPDATE_REJECTED( THING_NAME_LENGTH ) );
}
```

傳送影子更新

若要傳送陰影更新，示範會使用由 Device Shadow 程式庫定義的巨集，以 JSON 格式的訊息呼叫 `xPublishToTopic`。這會使用 `MQTT_Publish` 來發布至 `/delete` 主題。下列程式碼區段顯示如何在 `prvShadowDemoTask` 函數中完成此作業。

```
#define SHADOW_REPORTED_JSON \
"{" \
"\\"state\": {" \
"\\"reported\": {" \
"\\"powerOn\": "%01d" \
"}" \
"}, " \
"\\"clientToken\": \"%06lu\" " \
"}" \
snprintf( pcUpdateDocument,
    SHADOW_REPORTED_JSON_LENGTH + 1,
    SHADOW_REPORTED_JSON,
    ( int ) ulCurrentPowerOnState,
    ( long unsigned ) ulClientToken );

xPublishToTopic( SHADOW_TOPIC_STRING_UPDATE( THING_NAME ),
    SHADOW_TOPIC_LENGTH_UPDATE( THING_NAME_LENGTH ),
    pcUpdateDocument,
    ( SHADOW_DESIRED_JSON_LENGTH + 1 ) );
```

處理影子差量訊息和影子更新訊息

使用 [函數登記到 coreMQTT Client Library](#) 的使用者回呼函數將通知我們有關傳入的封包事件。`MQTT_Init` 以下是回呼函數。

```
/* This is the callback function invoked by the MQTT stack when it
 * receives incoming messages. This function demonstrates how to use the
 * Shadow_MatchTopic function to determine whether the incoming message is
 * a device shadow message or not. If it is, it handles the message
 * depending on the message type.
 */
static void prvEventCallback( MQTTContext_t * pxMqttContext,
    MQTTPacketInfo_t * pxPacketInfo,
    MQTTDeserializedInfo_t * pxDeserializedInfo )
{
    ShadowMessageType_t messageType = ShadowMessageTypeMaxNum;
    const char * pcThingName = NULL;
    uint16_t usThingNameLength = 0U;
    uint16_t usPacketIdentifier;

    ( void ) pxMqttContext;
```

```
configASSERT( pxDeserializedInfo != NULL );
configASSERT( pxMqttContext != NULL );
configASSERT( pxPacketInfo != NULL );

usPacketIdentifier = pxDeserializedInfo->packetIdentifier;

/* Handle incoming publish. The lower 4 bits of the publish packet
 * type is used for the dup, QoS, and retain flags. Hence masking
 * out the lower bits to check if the packet is publish. */
if( ( pxPacketInfo->type & 0xFOU ) == MQTT_PACKET_TYPE_PUBLISH )
{
    configASSERT( pxDeserializedInfo->pPublishInfo != NULL );
    LogInfo( ( "pPublishInfo->pTopicName:%s.",
               pxDeserializedInfo->pPublishInfo->pTopicName ) );

    /* Let the Device Shadow library tell us whether this is a device
     * shadow message. */
    if( SHADOW_SUCCESS == Shadow_MatchTopic(
            pxDeserializedInfo->pPublishInfo->pTopicName,
            pxDeserializedInfo->pPublishInfo->topicNameLength,
            &messageType,
            &pCThingName,
            &usThingNameLength ) )

    {
        /* Upon successful return, the messageType has been filled in. */
        if( messageType == ShadowMessageTypeUpdateDelta )
        {
            /* Handler function to process payload. */
            prvUpdateDeltaHandler( pxDeserializedInfo->pPublishInfo );
        }
        else if( messageType == ShadowMessageTypeUpdateAccepted )
        {
            /* Handler function to process payload. */
            prvUpdateAcceptedHandler( pxDeserializedInfo->pPublishInfo );
        }
        else if( messageType == ShadowMessageTypeUpdateDocuments )
        {
            LogInfo( ( "/update/documents json payload:%s.",
                       ( const char * ) pxDeserializedInfo->pPublishInfo->pPayload ) );
        }
        else if( messageType == ShadowMessageTypeUpdateRejected )
        {
            LogInfo( ( "/update/rejected json payload:%s.",
                       ( const char * ) pxDeserializedInfo->pPublishInfo->pPayload ) );
        }
        else
        {
            LogInfo( ( "Other message type:%d !!", messageType ) );
        }
    }
    else
    {
        LogError( ( "Shadow_MatchTopic parse failed:%s !!",
                    ( const char * ) pxDeserializedInfo->pPublishInfo->pTopicName ) );
    }
}
else
{
    vHandleOtherIncomingPacket( pxPacketInfo, usPacketIdentifier );
}
```

回呼函數會確認傳入封包的類型是 MQTT_PACKET_TYPE_PUBLISH，並使用 Device Shadow Library API Shadow_MatchTopic 來確認傳入訊息是影子訊息。

如果傳入訊息是類型為 ShadowMessageTypeUpdateDelta 的影子訊息，我們會呼叫 prvUpdateDeltaHandler 來處理此訊息。處理常式 prvUpdateDeltaHandler 使用 程式庫剖析訊息以取得 coreJSON 狀態的差量值，並將此值與本機維護的目前裝置狀態進行比較。powerOn如果這些狀態不同，本機裝置狀態會更新以反映影子文件中 powerOn 狀態的新值。

```
static void prvUpdateDeltaHandler( MQTTPublishInfo_t * pxPublishInfo )
{
    static uint32_t ulCurrentVersion = 0; /* Remember the latestVersion # we've ever
received */
    uint32_t ulVersion = 0U;
    uint32_t ulNewState = 0U;
    char * pcOutValue = NULL;
    uint32_t ulOutValueLength = 0U;
    JSONStatus_t result = JSONSuccess;

    configASSERT( pxPublishInfo != NULL );
    configASSERT( pxPublishInfo->pPayload != NULL );

    LogInfo( ( "/update/delta json payload:%s.",
                ( const char * ) pxPublishInfo->pPayload ) );

    /* The payload will look similar to this:
     * {
     *     "version": 12,
     *     "timestamp": 1595437367,
     *     "state": {
     *         "powerOn": 1
     *     },
     *     "metadata": {
     *         "powerOn": {
     *             "timestamp": 1595437367
     *         }
     *     },
     *     "clientToken": "388062"
     * }
     */

    /* Make sure the payload is a valid json document. */
    result = JSON_Validate( pxPublishInfo->pPayload,
                           pxPublishInfo->payloadLength );

    if( result == JSONSuccess )
    {
        /* Then we start to get the version value by JSON keyword "version". */
        result = JSON_Search( ( char * ) pxPublishInfo->pPayload,
                             pxPublishInfo->payloadLength,
                             "version",
                             sizeof( "version" ) - 1,
                             '.',
                             &pcOutValue,
                             ( size_t * ) &ulOutValueLength );
    }
    else
    {
        LogError( ( "The json document is invalid!!" ) );
    }

    if( result == JSONSuccess )
    {
        LogInfo( ( "version: %.*s",
                    ulOutValueLength,
                    pcOutValue ) );

        /* Convert the extracted value to an unsigned integer value. */
    }
}
```

```
        ulVersion = ( uint32_t ) strtoul( pcOutValue, NULL, 10 );
    }
else
{
    LogError( ( "No version in json document!!" ) );
}

LogInfo( ( "version:%d, ulCurrentVersion:%d \r\n",
            ulVersion, ulCurrentVersion ) );

/* When the version is much newer than the one we retained, that means
 * the powerOn state is valid for us. */
if( ulVersion > ulCurrentVersion )
{
    /* Set to received version as the current version. */
    ulCurrentVersion = ulVersion;

    /* Get powerOn state from json documents. */
    result = JSON_Search( ( char * ) pxPublishInfo->pPayload,
                          pxPublishInfo->payloadLength,
                          "state.powerOn",
                          sizeof( "state.powerOn" ) - 1,
                          '.',
                          &pcOutValue,
                          ( size_t * ) &ulOutValueLength );
}
else
{
    /* In this demo, we discard the incoming message
     * if the version number is not newer than the latest
     * that we've received before. Your application may use a
     * different approach.
    */
    LogWarn( ( "The received version is smaller than current one!!" ) );
}

if( result == JSONSuccess )
{
    /* Convert the powerOn state value to an unsigned integer value. */
    ulNewState = ( uint32_t ) strtoul( pcOutValue, NULL, 10 );

    LogInfo( ( "The new power on state newState:%d, ulCurrentPowerOnState:%d \r\n",
                ulNewState, ulCurrentPowerOnState ) );

    if( ulNewState != ulCurrentPowerOnState )
    {
        /* The received powerOn state is different from the one we
         * retained before, so we switch them and set the flag. */
        ulCurrentPowerOnState = ulNewState;

        /* State change will be handled in main(), where we will publish
         * a "reported" state to the device shadow. We do not do it here
         * because we are inside of a callback from the MQTT library, so
         * that we don't re-enter the MQTT library. */
        stateChanged = true;
    }
}
else
{
    LogError( ( "No powerOn in json document!!" ) );
    xUpdateDeltaReturn = pdFAIL;
}
}
```

如果傳入訊息是類型為 ShadowMessageTypeUpdateAccepted 的影子訊息，我們會呼叫 prvUpdateAcceptedHandler 來處理此訊息。處理常式 prvUpdateAcceptedHandler 使用 coreJSON 程式庫剖析訊息，進而從訊息取得 clientToken。此處理常式函數會檢查 JSON 訊息中的 client 字符是否符合應用程式所使用的 client 字符。如果不相符，該函數會登入警告訊息。

```
static void prvUpdateAcceptedHandler( MQTTPublishInfo_t * pxPublishInfo )
{
    char * pcOutValue = NULL;
    uint32_t ulOutValueLength = 0U;
    uint32_t ulReceivedToken = 0U;
    JSONStatus_t result = JSONSuccess;

    assert( pxPublishInfo != NULL );
    assert( pxPublishInfo->pPayload != NULL );

    LogInfo( ( "/update/accepted json payload:%s.",
               ( const char * ) pxPublishInfo->pPayload ) );

    /* Handle the reported state with state change in /update/accepted topic.
     * Thus we will retrieve the client token from the json document to see if
     * it's the same one we sent with reported state on the /update topic.
     * The payload will look similar to this:
     */
    {
        "state": {
            "reported": {
                "powerOn": 1
            }
        },
        "metadata": {
            "reported": {
                "powerOn": {
                    "timestamp": 1596573647
                }
            }
        },
        "version": 14698,
        "timestamp": 1596573647,
        "clientToken": "022485"
    }

    /* Make sure the payload is a valid json document. */
    result = JSON_Validate( pxPublishInfo->pPayload,
                           pxPublishInfo->payloadLength );

    if( result == JSONSuccess )
    {
        /* Get clientToken from json documents. */
        result = JSON_Search( ( char * ) pxPublishInfo->pPayload,
                             pxPublishInfo->payloadLength,
                             "clientToken",
                             sizeof( "clientToken" ) - 1,
                             '.',
                             &pcOutValue,
                             ( size_t * ) &ulOutValueLength );
    }
    else
    {
        LogError( ( "Invalid json documents !!" ) );
    }

    if( result == JSONSuccess )
    {
        LogInfo( ( "clientToken: %.*s", ulOutValueLength,
```

```
pcOutValue ) );  
  
/* Convert the code to an unsigned integer value. */  
ulReceivedToken = ( uint32_t ) strtoul( pcOutValue, NULL, 10 );  
  
LogInfo( ( "receivedToken:%d, clientToken:%u \r\n",
            ulReceivedToken, ulClientToken ) );  
  
/* If the clientToken in this update/accepted message matches the one
 * we published before, it means the device shadow has accepted our
 * latest reported state. We are done. */
if( ulReceivedToken == ulClientToken )
{
    LogInfo( ( "Received response from the device shadow. Previously published "
                "update with clientToken=%u has been accepted. ", ulClientToken ) );
}
else
{
    LogWarn( ( "The received clientToken=%u is not identical with the one=%u we
sent ",
                ulReceivedToken, ulClientToken ) );
}
else
{
    LogError( ( "No clientToken in json document!!" ) );
    lUpdateAcceptedReturn = EXIT_FAILURE;
}
}
```

Secure Sockets echo 用戶端示範

下列範例使用單一 RTOS 任務。您可以在 `demos/tcp/aws_tcp_echo_client_single_task.c` 找到此範例的來源程式碼。

開始之前，請確認您已將 FreeRTOS 下載到您的微型控制器，並建置和執行 FreeRTOS 示範專案。您可以從 [FreeRTOS GitHub](#) 複製或下載。請前往 [README.md](#) 檔案以獲得指示。

執行示範

Note

若要設定和執行 FreeRTOS 示範，請遵循[FreeRTOS 入門 \(p. 14\)](#)中的步驟。

目前在 Cypress CYW954907AEVAL1F 和 CYW943907AEVAL1F 開發套件上不支援 TCP 伺服器和用戶端示範。

- 按照 [移植指南](#)設定 TLS Echo ServerFreeRTOS 中的指示。

TLS echo 伺服器應該會執行並偵聽連接埠 9000。

在設定期間，您應該會產生四個檔案：

- client.pem (用戶端憑證)
- client.key (用戶端私有金鑰)
- server.pem (伺服器憑證)
- server.key (伺服器私有金鑰)

- 使用工具 `tools/certificate_configuration/CertificateConfigurator.html` 來將用戶端憑證 (`client.pem`) 和用戶端私有金鑰 (`client.key`) 複製到 `aws_clientcredential_keys.h`。
- 開啟 `FreeRTOSConfig.h` 檔案。

4. 將 configECHO_SERVER_ADDR0、configECHO_SERVER_ADDR1、configECHO_SERVER_ADDR2 和 configECHO_SERVER_ADDR3 變數設定為四個整數，組成 TLS Echo Server 執行所在的 IP 地址。
5. 將 configTCP_ECHO_CLIENT_PORT 變數設定為 9000，即 TLS Echo Server 接聽所在的連接埠。
6. 將 configTCP_ECHO_TASKS_SINGLE_TASK_TLS_ENABLED 變數設定為 1。
7. 使用工具 tools/certificate_configuration/PEMfileToCString.html 將伺服器憑證 (server.pem) 複製到檔案 aws_tcp_echo_client_single_task.c 中的 cTlsECHO_SERVER_CERTIFICATE_PEM。
8. 開啟 *freertos/vendors/vendor/boards/board/aws_demos/config_files/aws_demo_config.h*、註解 #define CONFIG_MQTT_DEMO_ENABLED，然後定義 CONFIG_TCP_ECHO_CLIENT_DEMO_ENABLED。

微型控制器和 TLS Echo Server 應該在相同網路上。示範開始時 (main.c)，您應該會看到日誌訊息，指出 Received correct string from echo server。

使用 AWS IoT Device Tester for FreeRTOS

您可以使用 AWS IoT Device Tester (IDT) for FreeRTOS 來驗證 FreeRTOS 作業系統是否可在本機裝置上運作，並與 AWS IoT 雲端通訊。值得留意的是，該工具還會確認 FreeRTOS 程式庫的移植層界面是否經過正確實作，也會使用 AWS IoT Core 執行端對端測試。舉例來說，其會驗證主機板是否能傳送和接收 MQTT 訊息，並正確處理這些項目。IDT for FreeRTOS 執行的測試會定義在 [FreeRTOS GitHub 儲存庫中](#)。

測試會以主機板內嵌應用程式的方式執行。應用程式二進位映像包含 FreeRTOS、半導體廠商的移植 FreeRTOS 界面，以及主機板裝置驅動程式。測試的目的旨在驗證移植的 FreeRTOS 界面是否能夠在裝置驅動程式上正常運作。

IDT for FreeRTOS 會產生測試報告，供您提交至 AWS IoT 以在 AWS Partner Device Catalog 中新增硬體。如需詳細資訊，請參閱 [AWS 裝置資格計劃](#)。

IDT for FreeRTOS 會在連接到要測試之電路板的主機電腦（Windows、macOS 或 Linux）上執行。此外，IDT 會執行測試案例並彙總結果，還會提供可管理測試執行作業的命令列界面。

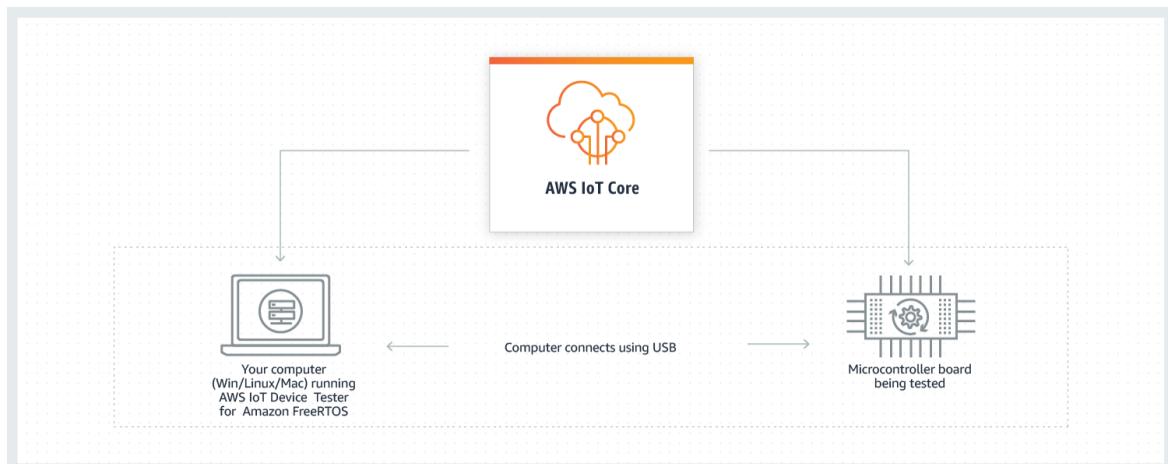
除了測試裝置，IDT for FreeRTOS 還會建立資源（例如，AWS IoT 實物、FreeRTOS 群組、Lambda 函數等），以協助資格審查程序。

為了建立這些資源，IDT for FreeRTOS 會使用 config.json 中設定的 AWS 登入資料來代替您呼叫 API。系統會在測試期間的不同時間點內佈建這些資源。

當您 在主機電腦上執行 IDT for FreeRTOS 時，該工具會執行以下步驟：

1. 載入並驗證您的裝置和登入資料組態。
2. 對所需的本機和雲端資源執行選取的測試。
3. 清除本機和雲端資源。
4. 產生測試報告以指出主機板是否通過符合資格所需的測試。

下圖顯示的是測試基礎設施設定流程。



您可使用測試資源來執行 IDT for FreeRTOS。有下列兩種類型的資源：

- 測試套件是一組測試群組，用來驗證裝置是否適用於特定版本的 FreeRTOS。
- 測試群組是一組與特定功能相關的個別測試，例如 BLE 和 MQTT 簡訊。

如需詳細資訊，請參閱 [AWS IoT Device Tester for FreeRTOS 測試套件版本 \(p. 316\)](#)。

AWS IoT Device Tester for FreeRTOS 的支援版本

本主題列出 IDT for FreeRTOS 的支援版本。建議的最佳實務就是使用支援 FreeRTOS 目標版本的最新版 IDT for FreeRTOS。每個適用於 FreeRTOS 的 IDT 版本皆對應至一或多個 FreeRTOS 版本。新版的 FreeRTOS 可能需要您下載新版本的 IDT for FreeRTOS。

下載軟體即表示您同意 IDT for FreeRTOS 授權合約。

AWS IoT Device Tester for FreeRTOS 的最新版本

您可以使用下列連結來下載適用於 FreeRTOS IDT 的最新版本。

IDT v4.0.1 和測試套件版本 1.4.1（適用於 FreeRTOS 202012.00）（使用 FreeRTOS 202012.00 LTS 程式庫）

- IDT v4.0.1 含測試套件 FRQ_1.4.1（適用於 [Linux](#)）
- IDT v4.0.1 含測試套件 FRQ_1.4.1（適用於 [macOS](#)）
- IDT v4.0.1 含測試套件 FRQ_1.4.1（適用於 [Windows](#)）

Note

不建議多位使用者從 NFS 目錄或 Windows 網路共用資料夾等共用位置執行 IDT。此實務可能會當機或資料損壞。我們建議您將 IDT 套件解壓縮到本機磁碟機，並在本機工作站上執行 IDT 二進位檔。

版本備註

- 支援使用 FreeRTOS LTS 程式庫的 202012.00FreeRTOS。如需 FreeRTOS 202012.00 版本的詳細資訊，請查看 [中的 CHANGELOG.mdGitHub 檔案](#)。
- 介紹其他 OTA（空中）E2E（端對端）測試案例。
- 支援使用 FreeRTOS LTS 程式庫執行 202012.00FreeRTOS 的開發電路板資格。
- 新增使用行動連接來支援 FreeRTOS 開發板的資格。
- 修正 echo 伺服器組態中的錯誤。
- 可讓您使用 AWS IoT Device Tester for FreeRTOS，來開發和執行自己的自訂測試套件。如需詳細資訊，請參閱 [使用 IDT 來開發和執行您自己的測試套件 \(p. 320\)](#)。
- 提供程式碼簽署的 IDT 應用程式，因此您不需要在 Windows 或 macOS 下執行它時授予許可。

測試套件版本

- FRQ_1.4.1
 - 已發行 2020.12.15。

舊版 IDT for FreeRTOS

同時支援下列舊版 IDT for FreeRTOS。

IDT v3.4.0 和測試套件版本 1.3.0 for FreeRTOS 202011.00

- IDT v3.4.0 與測試套件 FRQ_1.3.0 (適用於 [Linux](#))
- IDT v3.4.0 與測試套件 FRQ_1.3.0 (適用於 [macOS](#))
- IDT v3.4.0 與測試套件 FRQ_1.3.0 for [Windows](#)

版本備註

- 支援 FreeRTOS 202011.00。如需 FreeRTOS 202011.00 版本的詳細資訊，請查看 [中的 CHANGELOG.md](#) GitHub 檔案。
- 修正「RSA」不是有效的 PKCS11 組態選項的錯誤。
- 已修正以下錯誤：OTA 測試後，無法正確清除 Amazon S3 儲存貯體。
- 更新以支援 FullMQTT 測試群組中新的測試案例。

測試套件版本

- FRQ_1.3.0
 - 已發行 2020.11.05。

IDT v3.3.0 和 Test Suite 1.2.0 版 (適用於 FreeRTOS 202007.00)

- IDT v3.3.0 與測試套件 FRQ_1.2.0 for [Linux](#)
- IDT v3.3.0 與測試套件 FRQ_1.2.0 for [macOS](#)
- IDT v3.3.0 與測試套件 FRQ_1.2.0 for [Windows](#)

版本備註

- 支援 FreeRTOS 202007.00。如需 FreeRTOS 202007.00 版本的詳細資訊，請查看 [中的 CHANGELOG.md](#) GitHub 檔案。
- 新的端對端測試，以驗證 Over the Air (OTA) 更新暫停和繼續功能。
- 修正導致 eu-central-1 區域中的使用者無法通過 OTA 測試的組態驗證的錯誤。
- 新增 --update-idt 參數至 run-suite 命令。您可以使用此選項來設定 IDT 更新提示的回應。
- 新增 --update-managed-policy 參數至 run-suite 命令。您可以使用此選項來設定受管政策更新提示的回應。
- 外部改進與錯誤修正，包括：
 - 如需自動測試套件更新，請改善組態檔案升級。

測試套件版本

- FRQ_1.2.0
 - 已發行 2020.09.17。
 - 外部改進與錯誤修正，包括：
 - 對於 OTA，現在正確執行中斷及繼續測試，會在 MQTT 設定時使用 MQTT 資料平面。
 - 對於 BLE，您現在可以變更 Raspberry Pi 映像的密碼。

如需詳細資訊，請參閱 AWS IoT Device Tester for FreeRTOS 的支援政策 (p. 379)。

適用於 的不支援的 IDT 版本 FreeRTOS

本節列出不支援的 IDT for FreeRTOS 版本。不支援的版本不會收到錯誤修正或更新。如需詳細資訊，請參閱 AWS IoT Device Tester for FreeRTOS 的支援政策 (p. 379)。

不再支援下列版本的 IDT-FreeRTOS。

適用於 FreeRTOS 202002.00 的 IDT v3.0.2

- IDT for FreeRTOS : Linux
- IDT for FreeRTOS : macOS
- IDT for FreeRTOS : Windows

版本備註

- 支援 FreeRTOS 202002.00。如需 FreeRTOS 202002.00 版本的詳細資訊，請前往 [中的 CHANGELOG.mdGitHub 檔案](#)。
- 新增 IDT 內的測試套件自動更新。IDT 現可下載可供 FreeRTOS 版本使用的最新測試套件。透過此功能，您可以：
 - 使用 `upgrade-test-suite` 命令下載最新測試套件。
 - 在啟動 IDT 時設定旗標以下載最新測試套件。

使用 `-u flag` 選項，其中 `flag` 可以是 'y' 以一律下載或 'n' 來使用現有版本。

當有多個可用的測試套件版本時，除非您在啟動 IDT 時指定測試套件 ID，否則就會使用最新版。

- 使用新 `list-supported-versions` 選項列出 FreeRTOS，以及安裝的 IDT 版本所支援的測試套件版本。
- 列出群組中的測試案例，然後執行個別測試。

測試套件會使用從 1.0.0 開始的 `major.minor.patch` 格式進行版本化。

- 新增 `list-supported-products` 命令 – 列出安裝的 IDT 版本所支援的 FreeRTOS 和測試套件版本。
- 新增 `list-test-cases` 命令 – 列出測試群組中的可用測試案例。
- 新增 `run-suite` 命令的 `test-id` 選項 – 使用此選項來執行測試群組中的個別測試案例。

測試套件版本

- FRQ_1.0.1 (FRQ_1.0.1)

適用於 FreeRTOS 202002.00 的 IDT v1.7.1

- IDT for FreeRTOS : Linux
- IDT for FreeRTOS : macOS
- IDT for FreeRTOS : Windows

版本備註

- 支援 FreeRTOS 202002.00。如需 FreeRTOS 202002.00 版本的詳細資訊，請查看 [中的 CHANGELOG.mdGitHub 檔案](#)。

- 支援透過空中 (OTA) 端對端測試案例的自訂程式碼簽章方法，讓您可以使用自己的程式碼簽章命令和指令碼來簽署 OTA 承載。
- 在測試開始前新增序列埠的預先檢查。如果 `device.json` 檔案中的序列埠配置錯誤，測試將會迅速失敗，並提供需要改進的錯誤訊息。
- 新增具有執行 [所需許可的 AWS](#) 受管政策 `AWSIoTDeviceTesterForFreeRTOSFullAccess`。AWS IoT Device Tester 如果新版本需要其他許可，我們會將其新增至此受管政策，因此您不必手動更新 IAM 許可。
- 結果目錄命名為 `AFO_Report.xml` 的檔案現在是 `FRQ_Report.xml`。

適用於 FreeRTOS 201912.00 的 IDT v1.6.2

- IDT for FreeRTOS : [Linux](#)
- IDT for FreeRTOS : [macOS](#)
- IDT for FreeRTOS : [Windows](#)

版本備註

- 支援 FreeRTOS 201912.00。
- 支援透過 HTTPS 進行 OTA 的選用測試，以符合 FreeRTOS 的開發主機板的資格。
- 在測試中支援 AWS IoT ATS 端點。
- 支援在啟動測試套件之前通知使用者有最新 IDT 版本的功能。

IDT v1.5.2 for FreeRTOS 201910.00

- IDT for Amazon FreeRTOS : [Linux](#)
- IDT for Amazon FreeRTOS : [macOS](#)
- IDT for Amazon FreeRTOS : [Windows](#)

版本備註

- 支援使用安全元素 (內建金鑰) 的 FreeRTOS 裝置資格。
- 支援 SecureSocket 和 Wifi 測試群組的可設定 echo 伺服器連接埠。
- 支援逾時乘數標記，以便在您故障診斷逾時相關錯誤時得以方便增加逾時。
- 新增用於日誌剖析的錯誤修正。
- 在測試中支援 IoT ATS 端點。

IDT v1.4.1 for FreeRTOS 201908.00

- IDT for Amazon FreeRTOS : [Linux](#)
- IDT for Amazon FreeRTOS : [macOS](#)
- IDT for Amazon FreeRTOS : [Windows](#)

版本備註

- 新增對新 PKCS11 程式庫和測試案例更新的支援。
- 引進可動作的錯誤代碼。如需詳細資訊，請參閱「[IDT 錯誤代碼 \(p. 374\)](#)」
- 更新用於執行 IDT 的 IAM 政策。

IDT v1.3.2 for FreeRTOS 201906.00 主要版本

- IDT for FreeRTOS : [Linux](#)
- IDT for FreeRTOS : [macOS](#)
- IDT for FreeRTOS : [Windows](#)

版本備註

- 新增對測試低功耗藍牙 (BLE) 的支援。
- 改善 IDT 命令列介面 (CLI) 命令的使用者體驗。
- 更新用於執行 IDT 的 IAM 政策。

IDT-FreeRTOS 1.2 版

- FreeRTOS 版本：
 - FreeRTOS 1.4.9 版
 - FreeRTOS 1.4.8 版
- 版本備註：
 - 新增對 FreeRTOS v1.4.8 和 v1.4.9 的支援。
 - 新增使用 CMAKE 建置系統測試 FreeRTOS 裝置的支援。

IDT-FreeRTOS 1.1 版

- FreeRTOS 版本：
 - FreeRTOS 1.4.7 版

IDT-FreeRTOS 1.0 版

- FreeRTOS 版本：
 - FreeRTOS 1.4.6 版
 - FreeRTOS 1.4.5 版
 - FreeRTOS 1.4.4 版
 - FreeRTOS 1.4.3 版
 - FreeRTOS 1.4.2 版

Prerequisites

本節說明使用 AWS IoT Device Tester 測試微控制器的先決條件。

下載 FreeRTOS

您可以使用下列命令從 [FreeRTOS GitHub 下載](#) 的版本：

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

其中 <FREERTOS_RELEASE_VERSION> 是 FreeRTOS 版本 (例如 202007.00) , 對應到 [AWS IoT Device Tester for FreeRTOS 的支援版本 \(p. 291\)](#) 中列出的 IDT 版本。這可確保您擁有完整的原始程式碼，包括子模組，以及使用 IDT for 您的 FreeRTOS 版本的正確版本，反之亦然。

Windows 的路徑長度限制為 260 個字元。FreeRTOS 的路徑結構有許多層次，因此若您使用的是 Windows，請將檔案路徑長度維持在 260 個字元以內。例如，建議您將 FreeRTOS 檔案複製至 C:\FreeRTOS，而不是 C:\Users\username\programs\projects\myproj\FreeRTOS\。

LTS 資格 (使用 LTS 程式庫的 FreeRTOS 資格)

- 為了在 AWS Partner Device Catalog 中將您的微型控制器指定為支援長期支援 (LTS) 版本 FreeRTOS，您必須提供資訊清單檔案。如需詳細資訊，請查看 [資格指南FreeRTOS中的 資格檢查清單](#) FreeRTOS。
- 為了驗證您的微型控制器是否支援 FreeRTOS 的 LTS 版本，並使其符合提交至 AWS Partner Device Catalog 的資格，您必須針對 AWS IoT Device Tester v4.0.1 使用 FreeRTOS (IDT)。
- 目前，FreeRTOS 以 LTS 為基礎的 202012.00 的 FreeRTOS 版本。

下載 IDT for FreeRTOS

執行資格測試時，每個 FreeRTOS 版本都有相對應的 IDT for FreeRTOS 版本。您可以從 [AWS IoT Device Tester for FreeRTOS 的支援版本 \(p. 291\)](#) 下載適當的 IDT for FreeRTOS 版本。

請將 IDT for FreeRTOS 解壓縮到檔案系統的某個位置上，您在該位置中需具有讀取和寫入許可。因為 Microsoft Windows 有路徑長度的字元限制，所以請將 IDT for FreeRTOS 解壓縮到根目錄，例如 C:\ 或 D:\。

Note

不建議多位使用者從 NFS 目錄或 Windows 網路共用資料夾等共用位置執行 IDT。這麼做可能會導致當機或資料損毀。建議您將 IDT 套件解壓縮至本機磁碟機。

建立並設定 AWS 帳戶

請遵循下列步驟來建立和設定 AWS 帳戶、IAM 使用者，以及可授權 IDT for FreeRTOS 在執行測試期間代表您存取資源的 IAM 政策。

- 若您已經擁有 AWS 帳戶，請跳至下一項任務，建立 [AWS 帳戶](#)。
- 建立授與 IDT for FreeRTOS 權限的 IAM 原則，以建立具有特定 IAM 權限的服務角色。
 - 登入 [IAM 主控台](#)。
 - 在導覽窗格上選擇 Policies (政策)。
 - 在內容窗格中，選擇 Create policy (建立政策)。
 - 選擇 JSON 索引標籤，並將以下權限複製到 JSON 文字方塊中。

Important

下列政策範本會授予 IDT 許可，允許使用者建立角色、建立政策，以及將政策附加至角色。IDT for FreeRTOS 會使用這些許可處理會建立角色的測試。雖然這些政策範本不會向使用者提供管理員權限，但這些許可仍可能用於取得管理員對您 AWS 帳戶的存取權。

Most Regions

{

```
"Version": "2012-10-17",
"Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "iam:CreatePolicy",
            "iam:DetachRolePolicy",
            "iam:DeleteRolePolicy",
            "iam:DeletePolicy",
            "iam:CreateRole",
            "iam:DeleteRole",
            "iam:AttachRolePolicy"
        ],
        "Resource": [
            "arn:aws:iam::*:policy/idt*",
            "arn:aws:iam::*:role/idt*"
        ]
    }
]
```

北京與寧夏區域

您可以在 北京與寧夏區域 中使用下列原則範本。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreatePolicy",
                "iam:DetachRolePolicy",
                "iam:DeleteRolePolicy",
                "iam:DeletePolicy",
                "iam:CreateRole",
                "iam:DeleteRole",
                "iam:AttachRolePolicy"
            ],
            "Resource": [
                "arn:aws-cn:iam::*:policy/idt*",
                "arn:aws-cn:iam::*:role/idt*"
            ]
        }
    ]
}
```

- e. 完成時，請選擇 Review policy (檢閱政策)。
 - f. 在 Review (檢閱) 頁面上，針對政策名稱輸入 IDTFreeRTOSIAMPermissions。檢閱政策摘要，確認您的政策所授予的許可。
 - g. 選擇 Create policy (建立政策)。
3. 建立具有執行 AWS IoT Device Tester 必要許可的 IAM 使用者。
- a. 請遵循[建立 IAM 使用者 \(主控台\)](#)中的步驟 1 至步驟 5。
 - b. 若要將必要的許可與您的 IAM 使用者連接：
 - i. 在 Set permissions (設定許可) 頁面上，選擇 Attach existing policies directly (直接連接現有的政策)。
 - ii. 搜尋您在步驟 2 中建立的 IDTFreeRTOSIAMPermissions 政策。選取核取方塊。
 - iii. 搜尋 AWSIoTDeviceTesterForFreeRTOSFullAccess 政策。選取核取方塊。
 - c. 選擇 Next : (下一步 :)。標籤。

- d. 選擇 Next : (下一步 :)。檢視以檢視選項的摘要。
- e. 選擇 Create user (建立使用者)。
- f. 若要檢視使用者的存取金鑰 (存取金鑰 IDs 和私密存取金鑰)，請選擇每個密碼和存取金鑰旁的 Show (顯示)，然後選擇 Download.csv。將此檔案儲存至安全位置。

AWS IoT Device Tester 受管政策

AWSIoTDeviceTesterForFreeRTOSFullAccess 受管政策包含下列許可，可讓 Device Tester 執行和收集指標：

- `iot-device-tester:SupportedVersion`
授予取得 IDT 支援之 FreeRTOS 版本和測試套件版本清單的許可，這樣就能從 AWS CLI 中取得。
- `iot-device-tester:LatestIdt`
授予取得可供下載之最新 AWS IoT Device Tester 版本的許可。
- `iot-device-tester:CheckVersion`
授予檢查產品、測試套件和 AWS IoT Device Tester 版本之組合是否相容的許可。
- `iot-device-tester:DownloadTestSuite`
授予 AWS IoT Device Tester 下載測試套件的許可。
- `iot-device-tester:SendMetrics`
授予發佈 AWS IoT Device Tester 用量指標資料的許可。

(選用) 安裝 AWS Command Line Interface

您可能偏好使用 AWS CLI 來執行部分操作。如果您未安裝 AWS CLI，請遵循[安裝 AWS CLI](#) 中的指示操作。

請從命令列執行 `aws configure`，以針對您要使用的 AWS 區域設定 CLI。如需支援 IDT for FreeRTOS 的 AWS 區域相關資訊，請參閱[AWS 區域與端點](#)。如需 `aws configure` 的詳細資訊，請前往[使用 aws configure 的快速組態](#)。

首次準備測試微型控制器主機板

在移植 FreeRTOS 界面時，您可以使用 IDT for FreeRTOS 來進行測試。為主機板的裝置驅動程式移植 FreeRTOS 界面後，您即可使用 AWS IoT Device Tester 在微型控制器主機板上執行資格測試。

新增程式庫移植層

若要將 FreeRTOS 移植到您的裝置，請按照[FreeRTOS 移植指南](#)中的指示操作。

設定 AWS 登入資料

您需要為 Device Tester 設定 AWS 登入資料，才能與 AWS 雲端通訊。如需詳細資訊，請參閱[設定開發用的 AWS 登入資料和區域](#)。請務必在 `devicetester_extract_location/` `devicetester_afreertos_[win/mac/linux]/configs/config.json` 組態檔案中指定有效的 AWS 登入資料。

在 IDT for FreeRTOS 中建立裝置集區

系統會將要測試的裝置整理為裝置集區，每個裝置集區都包含一個或多個相同的裝置。您可以設定 IDT for FreeRTOS 來測試集區中的單一裝置，或是測試集區中的多個裝置。IDT for FreeRTOS 能夠同時測試相同規格的裝置，藉此加速資格審查程序。該工具會採用循環配置資源方法，在裝置集區的每個裝置上執行不同的測試群組。

您可以在 `configs` 資料夾中編輯 `device.json` 範本的 `devices` 區段，進而新增一或多個裝置至裝置集區。

Note

同一個集區中的所有裝置皆需採用相同技術規格和 SKU。

IDT for FreeRTOS 會將原始程式碼複製到 IDT for FreeRTOS 解壓縮資料夾內的結果資料夾，以便針對不同測試群組啟用原始程式碼的平行建置功能。您必須使用 `testdata.sourcePath` 或 `sdkPath` 變數來參考建置或刷新命令中的來源碼路徑。IDT for FreeRTOS 會將此變數替換成所複製原始程式碼的暫存路徑。如需詳細資訊，請參閱「[IDT for FreeRTOS 變數 \(p. 309\)](#)」。

下方範例 `device.json` 檔案可用來建立具有多個裝置的裝置集區。

```
[  
  {  
    "id": "pool-id",  
    "sku": "sku",  
    "features": [  
      {  
        "name": "WIFI",  
        "value": "Yes | No"  
      },  
      {  
        "name": "Cellular",  
        "value": "Yes | No"  
      },  
      {  
        "name": "OTA",  
        "value": "Yes | No",  
        "configs": [  
          {  
            "name": "OTADataPlaneProtocol",  
            "value": "HTTP | MQTT | Both"  
          }  
        ]  
      },  
      {  
        "name": "BLE",  
        "value": "Yes | No"  
      },  
      {  
        "name": "TCP/IP",  
        "value": "On-chip | Offloaded | No"  
      },  
      {  
        "name": "TLS",  
        "value": "Yes | No"  
      },  
      {  
        "name": "PKCS11",  
        "value": "RSA | ECC | Both | No"  
      },  
      {  
        "name": "KeyProvisioning",  
        "value": "Import | Onboard | No"  
      }  
    ]  
  }]
```

```
        },
    ],
    "devices": [
        {
            "id": "device-id",
            "connectivity": {
                "protocol": "uart",
                "serialPort": "/dev/tty*"
            },
            *****Remove the section below if the device does not support onboard key
generation*****
            "secureElementConfig" : {
                "publicKeyAsciiHexFilePath": "absolute-path-to/public-key-txt-file: contains-
the-hex-bytes-public-key-extracted-from-onboard-private-key",
                "secureElementSerialNumber": "secure-element-serialNo-value"
            },
            ****
            "identifiers": [
                {
                    "name": "serialNo",
                    "value": "serialNo-value"
                }
            ]
        }
    ]
}
```

以下是 device.json 檔案中使用的屬性：

id

使用者定義的英數字 ID，可唯一識別裝置集區。屬於集區的裝置必須為相同類型。執行測試套件時，集區中的裝置將用來將工作負載平行化。

sku

可唯一識別您要測試之主機板的英數字元值。SKU 用來追蹤合格的主機板。

Note

如果您想要在 AWS 合作夥伴裝置目錄中列出主機板，在此處指定的 SKU 必須符合您在清單程序中使用的 SKU。

features

包含裝置支援功能的陣列。Device Tester 使用此資訊來選取要執行的資格測試。

支援的值如下：

TCP/IP

指出您的面板是否支援 TCP/IP 堆疊，以及支援晶載 (MCU) 或轉移到另一個模組。你需要 TCP/IP 才能符合資格。

WIFI

指出您的面板是否具有 Wi-Fi 功能。如果 No 設為 Cellular，則必須設定為 Yes。

Cellular

指出您的主機板是否具有行動功能。如果 No 設為 WIFI，則必須設定為 Yes。此功能設為 Yes 時，FullSecureSockets 測試將使用 AWS t2.micro EC2 執行個體執行，且可能需要支付額外的費用。如需詳細資訊，請查看 [Amazon EC2 定價](#)。

TLS

指出您的面板是否支援 TLS。您需要 TLS 才能獲得資格。

PKCS11

指出面板支援的公有金鑰密碼編譯演算法。您需要 PKCS11 才能獲得資格。支援的值為 ECC、RSA、Both 和 No。Both 表示主機板同時支援 ECC 和 RSA 演算法。

KeyProvisioning

指出將受信任的 X.509 用戶端憑證寫入面板的方法。有效值為 Import、Onboard 和 No。您需要索引鍵配置才符合資格。

- 如果主機板可允許匯入私有金鑰，請使用 Import。IDT 會建立私有金鑰並將此建置至 FreeRTOS 原始程式碼。
- 如果主機板可支援產生內建私有金鑰（例如，如果裝置有安全元素，或您偏好產生自己的裝置金鑰對和憑證），請使用 Onboard。請務必在每個裝置區段中新增一個 secureElementConfig 元素，並在 publicKeyAsciiHexFilePath 欄位中放置公有金鑰檔案的絕對路徑。
- 如果主機板不支援金鑰佈建，請使用 No。

OTA

指出您的面板是否支援無線（OTA）更新功能。otaDataPlaneProtocol 屬性指出裝置支援哪個 OTA 資料平面通訊協定。如果裝置不支援 OTA 功能，則會忽略此屬性。選取 "Both" 時，由於同時執行 MQTT、HTTP 和混合測試，OTA 測試執行時間會延長。

BLE

指出您的面板是否支援低功耗藍牙（BLE）。

devices.id

使用者定義的唯一識別符，用於識別要測試的裝置。

devices.connectivity.protocol

用來與此裝置通訊的通訊協定。支援的值為：uart -。

devices.connectivity.serialPort

要測試用於連接到裝置之主機電腦的序列埠。

devices.secureElementConfig.PublicKeyAsciiHexFilePath

檔案的絕對路徑，此檔案包含從內建私有金鑰擷取的十六進位位元組公有金鑰。

格式範例：

```
3059 3013 0607 2a86 48ce 3d02 0106 082a
8648 ce3d 0301 0703 4200 04cd 6569 ceb8
1bb9 1e72 339f e8cf 60ef 0f9f b473 33ac
6f19 1813 6999 3fa0 c293 5fae 08f1 1ad0
41b7 345c e746 1046 228e 5a5f d787 d571
dcbb 4e8d 75b3 2586 e2cc 0c
```

如果您的公有金鑰為 .der 格式，您可以直接以十六進位編碼處理公有金鑰，以製作 hex 檔案。

.der 公開金鑰取得 hex 檔案的範例命令：

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

如果您的公有金鑰為 .pem 格式，您可以擷取 base64 編碼部分，將其解碼為二進位格式，然後使用十六進位編碼來生成 hex 檔案。

例如，使用這些命令可建立 .pem 公有金鑰的 hex 檔案：

1. 漢取金鑰的 base64 編碼部分 (Strip the header and footer)，並將其儲存在檔案中（例如命名 base64key），執行此命令將其轉換為 .der 格式：

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

2. 執行 xxd 命令，將其轉換為 hex 格式。

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

`devices.secureElementConfig.SecureElementSerialNumber`

(選用) 安全元素的序號。執行 FreeRTOS 示範/測試專案時，若序號要隨著裝置公開金鑰一起列印出來，請提供此欄位。

`identifiers`

(選用) 任意的名稱/值組的陣列。您可以在下一節所述的建置和刷新命令中使用這些值。

設定建置、刷新和測試設定

如果要讓 IDT for FreeRTOS 自動在主機板上建置和刷新測試案例，則需設定 IDT 以執行硬體的建置與刷新命令。您可以在位於 config 資料夾的 `userdata.json` 範本檔案中配置建置和刷新命令設定。

配置設定以測試裝置

您可以在 `configs/userdata.json` 檔案中進行建置、刷新和測試設定。我們支援 Echo Server 組態，方法是在 `customPath` 中載入使用者端與伺服器憑證及金鑰。如需詳細資訊，請前往 [移植指南](#) 中的設定 echo 伺服器FreeRTOS。下列 JSON 範例顯示如何設定 IDT for FreeRTOS 測試多個裝置：

```
{
    "sourcePath": "/absolute-path-to/freertos",
    "vendorPath": "{{ testData.sourcePath }} /vendors /vendor-name /boards /board-name",
    // *****The sdkConfiguration block below is needed if you are not using the
    default, unmodified FreeRTOS repo.
    // In other words, if you are using the default, unmodified FreeRTOS repo then remove
    this block*****
    "sdkConfiguration": {
        "name": "sdk-name",
        "version": "sdk-version",
        "path": "/absolute-path-to/sdk"
    },
    "buildTool": {
        "name": "your-build-tool-name",
        "version": "your-build-tool-version",
        "command": [
            "/absolute-path-to/build-parallel.sh {{ testData.sourcePath }} {{ enableTests }}"
        ]
    },
    "flashTool": {
        "name": "your-flash-tool-name",
        "version": "your-flash-tool-version",
        "command": [
            "/absolute-path-to/flash-parallel.sh {{ testData.sourcePath }}"
            {{ device.connectivity.serialPort }} {{ buildImageName }}"
        ],
        "buildImageInfo" : {
            "testsImageName": "tests-image-name",
            "imageSize": 1000000000
        }
    }
}
```

```
        "demosImageName": "demos-image-name"
    },
    "clientWifiConfig": {
        "wifiSSID": "ssid",
        "wifiPassword": "password",
        "wifiSecurityType": "eWiFiSecurityOpen | eWiFiSecurityWEP | eWiFiSecurityWPA | eWiFiSecurityWPA2 | eWiFiSecurityWPA3"
    },
    "testWifiConfig": {
        "wifiSSID": "ssid",
        "wifiPassword": "password",
        "wifiSecurityType": "eWiFiSecurityOpen | eWiFiSecurityWEP | eWiFiSecurityWPA | eWiFiSecurityWPA2 | eWiFiSecurityWPA3"
    },
    //*****
    //This section is used to start echo server based on server certificate generation method,
    //When certificateGenerationMethod is set as Automatic specify the eccCurveFormat to generate certifcate and key based on curve format,
    //When certificateGenerationMethod is set as Custom specify the certificatePath and PrivateKeyPath to be used to start echo server
    //*****
    "echoServerCertificateConfiguration": {
        "certificateGenerationMethod": "Automatic | Custom",
        "customPath": {
            "clientCertificatePath": "/path/to/clientCertificate",
            "clientPrivateKeyPath": "/path/to/clientPrivateKey",
            "serverCertificatePath": "/path/to/serverCertificate",
            "serverPrivateKeyPath": "/path/to/serverPrivateKey"
        },
        "eccCurveFormat": "P224 | P256 | P384 | P521"
    },
    "echoServerConfiguration": {
        "securePortForSecureSocket": 33333, // Secure tcp port used by SecureSocket test. Default value is 33333. Ensure that the port configured isn't blocked by the firewall or your corporate network
        "insecurePortForSecureSocket": 33334, // Insecure tcp port used by SecureSocket test. Default value is 33334. Ensure that the port configured isn't blocked by the firewall or your corporate network
        "insecurePortForWiFi": 33335 // Insecure tcp port used by Wi-Fi test. Default value is 33335. Ensure that the port configured isn't blocked by the firewall or your corporate network
    },
    "otaConfiguration": {
        "otaFirmwareFilePath": "{{ testData.sourcePath }}/{relative-path-to/ota-image-generated-in-build-process}",
        "deviceFirmwareFileName": "ota-image-name-on-device",
        "otaDemoConfigFilePath": "{{ testData.sourcePath }}/{relative-path-to/ota-demo-config-header-file}",
        "codeSigningConfiguration": {
            "signingMethod": "AWS | Custom",
            "signerHashingAlgorithm": "SHA1 | SHA256",
            "signerSigningAlgorithm": "RSA | ECDSA",
            "signerCertificate": "arn:partition:service:region:account-id:resource:qualifier | /absolute-path-to/signer-certificate-file",
            "signerCertificateFileName": "signerCertificate-file-name",
            "compileSignerCertificate": boolean,
            // *****Use signerPlatform if you choose aws for signingMethod*****
            "signerPlatform": "AmazonFreeRTOS-Default | AmazonFreeRTOS-TI-CC3220SF",
            "untrustedSignerCertificate": "arn:partition:service:region:account-id:resource:type:resource:qualifier",
            // *****Use signCommand if you choose custom for signingMethod*****
            "signCommand": [

```

```
        "/absolute-path-to/sign.sh {{inputImagePath}}
{{outputSignatureFilePath}}"
    ]
}
},
// *****Remove the section below if you're not configuring CMake*****
"cmakeConfiguration": {
    "boardName": "board-name",
    "vendorName": "vendor-name",
    "compilerName": "compiler-name",
    "frToolchainPath": "/path/to/freertos/toolchain",
    "cmakeToolchainPath": "/path/to/cmake/toolchain"
},
"freertosFileConfiguration": {
    "required": [
        {
            "configName": "pkcs11Config",
            "filePath": "{{testData.sourcePath}}/vendors/vendor-name/boards/board-path/
aws_tests/config_files/core_pkcs11_config.h"
        },
        {
            "configName": "pkcs11TestConfig",
            "filePath": "{{testData.sourcePath}}/vendors/vendor-name/boards/board-path/
aws_tests/config_files/iot_test_pkcs11_config.h"
        }
    ],
    "optional": [
        {
            "configName": "otaAgentTestsConfig",
            "filePath": "{{testData.sourcePath}}/vendors/vendor-name/boards/board-path/
aws_tests/config_files/aws_ota_agent_config.h"
        },
        {
            "configName": "otaAgentDemosConfig",
            "filePath": "{{testData.sourcePath}}/vendors/vendor-name/boards/board-path/
aws_demos/config_files/aws_ota_agent_config.h"
        }
    ]
}
}
```

以下列出 `userdata.json` 中使用的屬性：

sourcePath

移植的 FreeRTOS 原始程式碼的根目錄路徑。對於使用開發套件進行平行測試，可使用 `sourcePath` 預留位置來設定 `{{userData.sdkConfiguration.path}}`。例如：

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

vendorPath

開發廠商特定 FreeRTOS 程式碼的路徑。針對序列測試，`vendorPath` 可設定為絕對路徑。例如：

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

針對平行測試，可使用 `{{testData.sourcePath}}` 預留位置設定 `vendorPath`。例如：

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

只有在不使用軟體開發套件執行時，才會需要 `vendorPath` 變數，否則可以將其移除。

Note

平行執行測試而不使用開發套件時，必須在

`{{{testData.sourcePath}}}, vendorPath, buildTool` 欄位中使用 `flashTool` 預留位置。使用單一裝置執行測試時，必須在 `vendorPath, buildTool, flashTool` 欄位中使用絕對路徑。以 開發套件執行時，必須在 `{{{sdkPath}}}, sourcePath` 和 `buildTool` 命令中使用 `flashTool` 預留位置。

`sdkConfiguration`

如果您符合 FreeRTOS 的資格，而且對檔案和資料夾結構的任何修改，超過移植所需的修改，則您在本區塊中需要設定您的軟體開發套件資訊。如果您在 開發套件中使用移植的 FreeRTOS 不符合 的資格，您應該完全省略此區塊。

`sdkConfiguration.name`

您搭配 FreeRTOS 使用的開發套件名稱。如果您不是使用開發套件，則應該省略整個 `sdkConfiguration` 區塊。

`sdkConfiguration.version`

您搭配 FreeRTOS 使用的開發套件版本。如果您不是使用開發套件，則應該省略整個 `sdkConfiguration` 區塊。

`sdkConfiguration.path`

指向您開發套件資料夾（包含 FreeRTOS 程式碼）的真實路徑。如果您不是使用開發套件，則應該省略整個 `sdkConfiguration` 區塊。

`buildTool`

建置指令碼的完整路徑 (.bat 或 .sh)，其中包含用來建立來源碼的命令。建置命令中來源碼路徑的所有參考，必須由 AWS IoT Device Tester 變數 `{{{testdata.sourcePath}}}` 取代，軟體開發套件路徑的參考應該由 `{{{sdkPath}}}` 取代。

`buildImageInfo`

`testsImageName`

從 `freertos-source/tests` 資料夾建置測試時，由建置命令所建立的檔案名稱。

`demosImageName`

從 `freertos-source/demos` 資料夾建置測試時，由建置命令所建立的檔案名稱。

`flashTool`

包含裝置刷新命令的刷新指令碼 (.sh 或 .bat) 完整路徑。刷入命令中來源碼路徑的所有參考，必須由 IDT for FreeRTOS 變數 `{{{testdata.sourcePath}}}` 取代，並且 IDT for FreeRTOS 變數 `{{{sdkPath}}}` 必須取代對軟體開發套件路徑的所有參考。

`clientWifiConfig`

用戶端 Wi-Fi 組態。Wi-Fi 程式庫測試需要 MCU 主機板，以連接到兩個存取點。（兩個存取點可以相同。）此屬性會設定第一個存取點的 Wi-Fi 設定。某些 Wi-Fi 測試案例希望存取點有一些安全保障，因此並未開放使用。請確認兩個存取點與執行 IDT 的主機電腦位於相同的子網路。

`wifi_ssid`

Wi-Fi SSID。

`wifi_password`

Wi-Fi 密碼。

`wifiSecurityType`

使用的 Wi-Fi 安全性類型。其中一個值：

- `eWiFiSecurityOpen`

- eWiFiSecurityWEP
- eWiFiSecurityWPA
- eWiFiSecurityWPA2
- eWiFiSecurityWPA3

Note

即使主機板不支援 Wi-Fi，您仍需在 device.json 檔案中加入 clientWifiConfig 區段，但可以省略這些屬性的值。

testWifiConfig

測試 Wi-Fi 組態。Wi-Fi 程式庫測試需要 MCU 主機板，以連接到兩個存取點。（兩個存取點可以相同。）此屬性會設定第二個存取點的 Wi-Fi 設定。某些 Wi-Fi 測試案例希望存取點有一些安全保障，因此並未開放使用。請確認兩個存取點與執行 IDT 的主機電腦位於相同的子網路。

wifiSSID

Wi-Fi SSID。

wifiPassword

Wi-Fi 密碼。

wifiSecurityType

使用的 Wi-Fi 安全性類型。其中一個值：

- eWiFiSecurityOpen
- eWiFiSecurityWEP
- eWiFiSecurityWPA
- eWiFiSecurityWPA2
- eWiFiSecurityWPA3

Note

即使主機板不支援 Wi-Fi，您仍需在 device.json 檔案中加入 testWifiConfig 區段，但可以省略這些屬性的值。

echoServerCertificateConfiguration

用於安全通訊端測試的可設定 echo 伺服器憑證衍生預留位置。此欄位為必填。

certificateGenerationMethod

指定伺服器憑證是自動生成還是手動提供。

customPath

如果 certificateGenerationMethod 為「自訂」，則 certificatePath 和 privateKeyPath 是必要的。

certificatePath

指定伺服器憑證的檔案路徑。

privateKeyPath

指定私有金鑰的檔案路徑。

eccCurveFormat

指定電路板支援的彎道格式。在 PKCS11 中將 device.json 設為 "ecc" 時，此為必要項目。有效值為 "P224"、"P256"、"P384" 或 "P521"。

echoServerConfiguration

和安全通訊端測試的可設定 echo 伺服器連接埠。WiFi此欄位為選用的。

`securePortForSecureSocket`

用於設定具有 TLS 的 echo 伺服器連接埠，以供安全通訊端測試之用。預設值為 33333。請確定防火牆或您公司的網路未封鎖設定的連接埠。

`insecurePortForSecureSocket`

用於設定不具有 TLS 的 echo 伺服器連接埠，以供安全通訊端測試之用。測試中使用的預設值為 33334。請確定防火牆或您公司的網路未封鎖設定的連接埠。

`insecurePortForWiFi`

用於設定無 TLS 的 echo 伺服器連接埠，以供 WiFi 測試之用。測試中使用的預設值為 33335。請確定防火牆或您公司的網路未封鎖設定的連接埠。

`otaConfiguration`

OTA 組態。[選用]

`otaFirmwareFilePath`

建置之後建立的 OTA 映像的完整路徑。例如，`\{{testData.sourcePath}\}/relative-path/to/ota/image/from/source/root`。

`deviceFirmwareFileName`

OTA 韌體所在之 MCU 裝置上的完整檔案路徑。有些裝置不會使用此欄位，但您仍須提供一值。

`otaDemoConfigFilePath`

的完整路徑，可在 `aws_demo_config.h` 找到。`afr-source/vendors/vendor/boards/board/aws_demos/config_files/` 這些檔案包括在 FreeRTOS 提供的移植程式碼範本中。

`codeSigningConfiguration`

程式碼簽章組態。

`signingMethod`

程式碼簽章方法。可能的值為 AWS 或 Custom。

`Note`

對於北京與寧夏區域，請使用 Custom。這些區域不支援 AWS 程式碼簽章。

`signerHashingAlgorithm`

裝置上支援的雜湊演算法。可能的值為 SHA1 或 SHA256。

`signerSigningAlgorithm`

裝置上支援的簽署演算法。可能的值為 RSA 或 ECDSA。

`signerCertificate`

用於 OTA 的信任憑證。

對於 AWS 程式碼簽章方法，請對上傳到 AWS Certificate Manager 的信任憑證使用 Amazon Resource Name (ARN)。

對於自訂程式碼簽署方法，請使用簽署者憑證檔案的絕對路徑。

如需建立信任憑證的詳細資訊，請參閱 [建立程式碼簽署憑證 \(p. 124\)](#)。

`signerCertificateFileName`

在裝置上的程式碼簽署憑證位置。

`compileSignerCertificate`

如果未佈建或刷新程式碼簽署者的簽章驗證，則設為 `true`，這樣就必須將其編譯到專案中。AWS IoT Device Tester 會獲取可信憑證並將其編譯至 `aws_codesigner_certificate.h` 中。

`untrustedSignerCertificateArn`

上傳至 ACM 之程式碼簽署憑證的 ARN。

`signerPlatform`

AWS 程式碼簽署者在建立 OTA 更新任務時使用的簽署和雜湊演算法。目前，此欄位的可能值為 AmazonFreeRTOS-TI-CC3220SF 和 AmazonFreeRTOS-Default。

- 如果是 AmazonFreeRTOS-TI-CC3220SF 和 SHA1，請選擇 RSA。
- 如果是 AmazonFreeRTOS-Default 和 SHA256，請選擇 ECDSA。

如果您的組態需要 SHA256 | RSA 或 SHA1 | ECDSA，請聯絡我們以取得進一步支援。

如果您針對 `signCommand` 選擇 Custom，請設定 `signingMethod`。

`signCommand`

用於執行自訂程式碼簽署的命令。您可以在 `/configs/script_templates` 目錄中找到範本。

命令中需要兩個預留位置 `{{inputImagePath}}` 和 `{{outputSignatureFilePath}}`。`{{inputImagePath}}` 是由 IDT 建置以進行簽署的影像檔案路徑。`{{outputSignatureFilePath}}` 是將由指令碼生成的簽章檔案路徑。

`otaDemoConfigFilePath`

在 `aws_demo_config.h` 中找到的 *afr-source*/vendors/vendor/boards/board/`aws_demos/config_files/` 完整路徑。這些檔案包括在 FreeRTOS 提供的移植程式碼範本中。

`cmakeConfiguration`

CMake 組態 [選用]

Note

若要執行 CMake 測試案例，您必須提供電路板名稱、廠商名稱，以及 `frToolchainPath` 或 `compilerName`。如果您有 `cmakeToolchainPath` 工具鏈的自訂路徑，也可以提供 CMake。

`boardName`

待測主機板的名稱。主機板名稱應與 `path/to/afr/source/code/vendors/vendor/boards/board` 下的資料夾名稱相同。

`vendorName`

待測主機板的廠商名稱。廠商應與 `path/to/afr/source/code/vendors/vendor` 下的資料夾名稱相同。

`compilerName`

編譯器的名稱。

`frToolchainPath`

編譯器工具鏈的完整路徑。

`cmakeToolchainPath`

工具鏈的完整路徑。CMake 此為選用欄位。

`freertosFileConfiguration`

IDT 在執行測試前修改的 FreeRTOS 檔案組態。

`required`

本節指定必要測試，其您已移動組態檔，例如 PKCS11、TLS 等。

`configName`

正在設定的測試名稱。

filePath

組態檔案在 *freertos* 儲存庫中的正確路徑。使用 {{testData.sourcePath}} 變數來定義路徑。

optional

本節指定選用測試，其組態檔案已移動，例如 OTA、WiFi 等。

configName

正在設定的測試名稱。

filePath

組態檔案在 *freertos* 儲存庫中的正確路徑。使用 {{testData.sourcePath}} 變數來定義路徑。

Note

若要執行 CMake 測試案例，您必須提供電路板名稱、廠商名稱，以及 afrToolchainPath 或 compilerName。如果您有 cmakeToolchainPath 工具鏈的自訂路徑，也可以提供 CMake。

IDT for FreeRTOS 變數

建置程式碼的命令和刷新裝置可能需要連接或有關裝置的其他資訊才能成功執行。AWS IoTDevice Tester 可讓您使用 [JsonPath 參考](#) 刷新和建置命令中的裝置資訊。使用簡易 JsonPath 表達式，即可擷取 device.json 檔案中指定的必要資訊。

路徑變數

IDT for FreeRTOS 可定義下列路徑變數，以便在命令列和組態檔案中使用：

 {{testData.sourcePath}}

 展開至原始程式碼路徑。如果您使用此變數，則必須同時在快閃和建置命令中使用。

 {{sdkPath}}

 用於建置和刷新命令時，展開 userData.sdkConfiguration.path 中的數值。

 {{device.connectivity.serialPort}}

 展開至序列埠。

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

 展開至裝置的序號。

 {{enableTests}}

 指出建置是否用於測試 (值 1) 或示範 (值 0) 的整數值。

 {{buildImageName}}

 檔案名稱使用建置命令產生的映像建置。

 {{otaCodeSignerPemFile}}

 OTA 程式碼簽署者的 PEM 檔案。

執行低功耗藍牙測試

本節將說明如何使用適用於 FreeRTOS 的 AWS IoT 裝置測試器來設定和執行藍牙測試。核心資格不需要進行藍牙測試。如果您不想透過 FreeRTOS 藍牙支援測試您的裝置，您可略過此設定，請務必將 device.json 中的 BLE 功能設為 No。

Prerequisites

- 請遵循首次準備測試微型控制器主機板 (p. 298) 中的說明進行。
- Raspberry Pi 3B+。(需要執行 Raspberry Pi BLE 配套應用程式)
- Raspberry Pi 軟體使用的 MicroSD 卡和 SD 卡的轉接卡。

Raspberry Pi 設定

若要測試待測裝置 (DUT) 的 BLE 功能，您必須擁有一個 Raspberry Pi Model 3B+。

設定您的 Raspberry Pi 以執行 BLE 測試

- 下載包含執行測試所需軟體的自訂 Yocto 映像。
- 將 yocto 映像刷到 Raspberry Pi 的 SD 卡上。
 - 使用 SD 卡片撰寫工具 (例如 Etcher) 將下載的 *image-name.rpi-sdimg* 檔案刷到 SD 卡片上。由於作業系統映像較大，此步驟需要時間才能完成。然後，退出您的 SD 卡，並將 microSD 卡插入 Raspberry Pi。
- 設定您的 Raspberry Pi。
 - 第一次啟動時，建議您將 Raspberry Pi 連接到螢幕、鍵盤和滑鼠。
 - 將您的 Raspberry Pi 連接到 micro USB 電源。
 - 使用預設的登入資料進行登入。針對使用者 ID，輸入 **root**。密碼請輸入 **idtafr**。
 - 使用乙太網路或 Wi-Fi 連線將 Raspberry Pi 連接到您的網路。
 - 若要透過 Wi-Fi 連接 Raspberry Pi，請開啟 Raspberry Pi 的 */etc/wpa_supplicant.conf*，然後將您的 Wi-Fi 登入資料新增到 Network 組態。

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

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

- 執行 `ifup wlan0` 以啟動 Wi-Fi 連線。連接到 Wi-Fi 網路可能需時一分鐘。
- 如需乙太網路連接，請執行 `ifconfig eth0`。如果是 Wi-Fi 連接，請執行 `ifconfig wlan0`。記下在命令輸出中顯示為 `inet addr` 的 IP 地址。在此程序稍後您將需要該 IP 地址。
- (選用) 該測試會使用 yocto 映像預設的登入資料，透過 SSH 在 Pi Raspberry 上執行命令。如需增加安全性，我們建議您為 SSH 設定公開金鑰身分驗證並停用以密碼為基礎的 SSH。
 - 使用 OpenSSL `ssh-keygen` 命令建立 SSH 金鑰。如果您的主機電腦中已有 SSK 金鑰對，最佳實務是建立新的金鑰對，讓 FreeRTOS 的 AWS IoT Device Tester 登入您的 Raspberry Pi。

Note

Windows 不附帶已安裝的 SSH 用戶端。有關如何在 Windows 安裝 SSH 用戶端的詳細資訊，請參閱[下載 SSH 軟體](#)。

- ii. `ssh-keygen` 命令會提示您提供金鑰對的存放名稱和路徑。根據預設，該金鑰對檔案會命名為 `id_rsa`(私有金鑰) 和 `id_rsa.pub`(公有金鑰)。在 macOS 和 Linux 上，這些檔案的預設位置是 `~/.ssh/`。在 Windows 上，預設位置為 `C:\Users\user-name`。
- iii. 提示您輸入金鑰字詞時，只要按 Enter 鍵即可。
- iv. 若要在 Raspberry Pi 上新增您的 SSH 金鑰，以使 FreeRTOS 的 AWS IoT Device Tester 可登入到您的裝置，請從您的主機電腦使用 `ssh-copy-id` 命令。此命令會將您的公有金鑰新增至 Raspberry Pi 上的 `~/.ssh/authorized_keys` 檔案。

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

- v. 當系統提示您輸入密碼時，請輸入 `idtafr`。這是 yocto 映像的預設密碼。

Note

命令會假設公有金鑰名稱為 `ssh-copy-id`。`id_rsa.pub` 在 macOS 和 Linux 上，預設位置為 `~/.ssh/`。在 Windows 上，預設位置為 `C:\Users\user-name\.ssh`。如果您給公有金鑰不同的名稱，或將其存放在不同的位置中，則必須使用 `-i` 選項向 `ssh-copy-id` 指定 SSH 公有金鑰的完整路徑（例如，`ssh-copy-id -i ~/my/path/myKey.pub`）。如需有關建立 SSH 金鑰和複製公有金鑰的詳細資訊，請參閱 [SSH-COPY-ID](#)。

- vi. 若要測試公有金鑰身分驗證運作正常，請執行 `ssh -i /my/path/myKey root@raspberry-pi-device-ip`。

如果您未被提示輸入密碼，則您的公開金鑰身分驗證運作正常。

- vii. 請確認您是用公開金鑰登入您的 Raspberry Pi，然後停用密碼為基礎的 SSH。

- A. 在 Raspberry Pi 上編輯 `/etc/ssh/sshd_config` 檔案。
- B. 將 `PasswordAuthentication` 屬性設為 `no`。
- C. 儲存並關閉 `sshd_config` 檔案。
- D. 執行 `/etc/init.d/sshd reload` 以重新載入 SSH 伺服器。

- g. 建立 `resource.json` 檔案。

- i. 在您擷取 AWS IoT 裝置測試器的目錄中建立名為 `resource.json` 的檔案。
- ii. 將 Raspberry Pi 的以下資訊新增至檔案，並取代 `rasp-pi-ip-address` 取代為 Raspberry Pi 的 IP 地址。

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

- iii. 如果您未選擇使用 SSH 的公開金鑰身分驗證，請將下列項目新增至 `connectivity` 檔案的 `resource.json` 區段。

```
"connectivity": {
```

```
"protocol": "ssh",
"ip": "rasp-pi-ip-address",
"auth": {
    "method": "password",
    "credentials": {
        "user": "root",
        "password": "idtafr"
    }
}
```

- iv. (選用) 如果您選擇使用 SSH 的公開金鑰身分驗證，請將下列項目新增至 `resource.json` 檔案的 `connectivity` 部分。

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

FreeRTOS 裝置設定

在您的 `device.json` 檔案中，將 BLE 功能設定為 Yes。如果您在藍牙測試可用之前開始使用 `device.json` 檔案，則需要將 BLE 功能新增至 `features` 陣列：

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

執行 BLE 測試

在您啟用 `device.json` 中的 BLE 功能後，BLE 測試會在您執行 `devicetester_[linux | mac | win_x86-64] run-suite` 時執行，且不會指定群組 ID。

如果您想要單獨執行 BLE 測試，您可以指定 BLE 的群組 ID：`devicetester_[linux | mac | win_x86-64] run-suite --userdata path-to-userdata userdata.json --group-id FullBLE -`。

為獲得最可靠效能，請將您的 Raspberry Pi 靠近待測裝置 (DUT)。

故障診斷 BLE 測試

確定您已遵循 [首次準備測試微型控制器主機板 \(p. 298\)](#) 中的步驟。如果 BLE 以外的測試失敗，則該問題可能並非因為 藍牙組態所導致。

執行 FreeRTOS 資格套件

您可以使用 AWS IoT Device Tester for FreeRTOS 可執行檔來與 IDT for FreeRTOS 互動。以下命令列範例會說明如何執行裝置集區 (一組相同的裝置) 的資格測試。

IDT v3.0.0 and later

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

在裝置集區上執行測試套件。userdata.json 檔案必須位於 `devicetester_extract_location/devicetester_afreertos_[win/mac/linux]/configs/` 目錄。

Note

如果您是在 Windows 上執行 IDT for FreeRTOS，請使用正斜線 (/) 來指定 userdata.json 檔案的路徑。

使用下列命令來執行特定的測試群組：

```
devicetester_[linux | mac | win] run-suite  \
--suite-id FRO_1.0.1 \
--group-id group-id \
--pool-id pool-id \
--userdata userdata.json
```

如果您是在單一裝置集區上執行單一測試套件 (也就是說，您在 device.json 檔案中僅定義了一個裝置集區)，suite-id 和 pool-id 參數則為選用。

使用下列命令，在測試群組中執行特定的測試案例：

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

您可以使用 list-test-cases 命令列出測試群組中的測試案例。

IDT for FreeRTOS 命令列選項

group-id

(選用) 要執行的測試群組，以逗號分隔的清單。如果未指定，IDT 會執行測試套件中的所有測試群組。

pool-id

(選用) 要測試的裝置集區。如果您在 device.json 中定義多個裝置集區，這則為必要。如果您只有一個裝置集區，則可以省略此選項。

suite-id

(選用) 要執行的測試套件版本。如果未指定，IDT 則會使用系統的測試目錄中的最新版本。

Note

從 IDT v3.0.0 開始，IDT 會在線上檢查是否有更新的測試套件。如需詳細資訊，請參閱 [測試套件版本 \(p. 316\)](#)。

test-id

(選用) 要執行的測試，以逗號分隔的清單。若已指定，group-id 必須指定單一群組。

Example

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

update-idt (update-idt)

(選用) 如果未設定此參數且有較新的 IDT 版本可用，系統會提示您更新 IDT。如果此參數設為 Y，IDT 會在偵測到較新版本可用時停止測試執行。如果此參數設為 N，IDT 會繼續測試執行。

更新受管政策

(選用) 如果未使用此參數，IDT 會偵測到您的受管政策不是最新的，系統會提示您更新受管政策。如果此參數設為 Y，IDT 會在偵測到您的受管政策不是最新狀態時停止測試執行。如果此參數設為 N，IDT 會繼續測試執行。

upgrade-test-suite

(選用) 若未使用，且有可用的更新測試套件版本，則會提示您進行下載。若要隱藏提示，請指定 y 以一律下載最新測試套件，或指定 n 以使用指定的測試套件或系統上的最新版本。

Example

若要一律下載並使用最新測試套件，請使用下列命令。

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

若要在系統上使用最新測試套件，請使用下列命令。

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

h

使用說明選項以進一步了解 run-suite 選項。

Example

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

IDT v1.7.0 and earlier

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

userdata.json 檔案應位於 [devicetester_extract_location/](#)
[devicetester_afreertos_\[win/mac/linux\]/configs/](#) 目錄中。

Note

如果您是在 Windows 上執行 IDT for FreeRTOS，請使用正斜線 (/) 來指定 `userdata.json` 檔案的路徑。

使用下列命令來執行特定的測試群組。

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

如果您是在單一裝置集區上執行單一測試套件 (也就是說，您在 `device.json` 檔案中僅定義了一個裝置集區)，則 `suite-id` 和 `pool-id` 為選用參數。

IDT for FreeRTOS 命令列選項

`group-id`

(選用) 指定測試群組。

`pool-id`

指定要測試的裝置集區。如果您只有一個裝置集區，就可以省略此選項。

`suite-id`

(選用) 指定要執行的測試套件。

IDT for FreeRTOS 命令

IDT for FreeRTOS 命令支援下列操作：

IDT v3.0.0 and later

`help`

列出所指定命令的相關資訊。

`list-groups`

列出指定套件中的群組。

`list-suites`

列出可用套件。

`list-supported-products`

列出支援的產品和測試套件版本。

`list-supported-versions`

列出目前 IDT 版本所支援的 FreeRTOS 和測試套件版本。

`list-test-cases`

列出指定群組中的測試案例。

`run-suite`

在裝置集區上執行測試套件。

使用 `--suite-id` 選項以指定測試套件版本，或省略它以使用系統上的最新版本。

使用 --test-id 執行個別測試案例。

Example

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

如需選項的完整清單，請參閱[執行 FreeRTOS 資格套件 \(p. 313\)](#)。

Note

從 IDT v3.0.0 開始，IDT 會在線上檢查是否有更新的測試套件。如需詳細資訊，請參閱[測試套件版本 \(p. 316\)](#)。

IDT v1.7.0 and earlier

`help`

列出所指定命令的相關資訊。

`list-groups`

列出指定套件中的群組。

`list-suites`

列出可用套件。

`run-suite`

在裝置集區上執行測試套件。

重新取得資格的測試

當新版本的 IDT for FreeRTOS 資格測試推出，或在您更新主機板特定的套件或裝置驅動程式時，都可使用 IDT for FreeRTOS 來測試微型控制器主機板。如果要進行後續資格審查，請確保您有最新版本的 FreeRTOS 和 IDT for FreeRTOS，並再次執行資格審查程序。

AWS IoT Device Tester for FreeRTOS 測試套件版本

IDT for FreeRTOS 會將測試整理為測試套件和測試群組。

- 測試套件是一組測試群組，用來驗證裝置是否適用於特定版本的 FreeRTOS。
- 測試群組是一組與特定功能相關的單一測試，例如 PKCS、MQTT 或 OTA。

從 IDT v3.0.0 開始，測試套件使用從 1.0.0 開始的 `major.minor.patch` 格式進行版本化。當您下載 IDT 時，套件會包含最新的測試套件版本。

當您在命令列界面中啟動 IDT 時，IDT 會檢查是否有可用的更新測試套件版本。如果有，則會提示您更新至新版。您可以選擇進行更新或繼續使用目前的測試。

Note

IDT 支援三種最新的測試套件版本，以符合資格。如需詳細資訊，請參閱[AWS IoT Device Tester for FreeRTOS 的支援政策 \(p. 379\)](#)。

您可以使用 `upgrade-test-suite` 命令下載測試套件。或者，當您開始 IDT 位置時，您可以使用選用參數 `-upgrade-test-suite flag`。`flag` 可以是 'y' 以一律下載最新版本，或 'n' 來使用現有版本。

您也可以執行 `list-supported-versions` 命令，列出 IDT 目前版本所支援的 FreeRTOS 和測試套件版本。

新測試可能會引入新的 IDT 組態設定。如果這些是選用設定，IDT 會通知您並繼續執行測試。如果這些是必要設定，IDT 會通知您並停止執行。完成設定後，您可以繼續執行測試。

了解結果和日誌

本節說明如何檢視和解譯 IDT 結果報告與日誌。

檢視結果

執行期間，IDT 會將錯誤寫入主控台、日誌檔和測試報告。IDT 完成資格測試套件後，即會將測試執行摘要寫入主控台，並產生兩份測試報告。您可以在 `devicetester-extract-location/results/execution-id/` 中找到這些報告。兩種報告都會從資格測試套件執行擷取結果。

`awsiotdevicetester_report.xml` 是您提交給 AWS 的資格測試報告，以在 AWS 合作夥伴裝置目錄中列出您的裝置。該報告包含下列元素：

- IDT for FreeRTOS 版本。
- 經過測試的 FreeRTOS 版本。
- 裝置根據已通過測試所支援的 FreeRTOS 功能。
- `device.json` 檔案中指定的 SKU 和裝置名稱。
- `device.json` 檔案中所指定裝置的功能。
- 測試案例結果的彙總摘要。
- 根據裝置功能（例如，FullWiFi、FullMQTT 等）進行測試的程式庫測試案例結果明細。
- 此資格是否屬於使用 LTS 程式庫的 FreeRTOS 版。202012.00

是標準 `FRQ_Report.xml` XML 格式的報告。[JUnit](#)您可以將它整合到 CI/CD 平台，例如 [Jenkins](#)、[Bamboo](#) 等等。該報告包含下列元素：

- 測試案例結果的彙總摘要。
- 根據裝置功能進行測試的程式庫測試案例結果明細。

解譯 IDT for FreeRTOS 結果

`awsiotdevicetester_report.xml` 或 `FRQ_Report.xml` 中的報告部分會列出所執行測試的結果。

第一個 XML 標籤 `<testsuites>` 包含測試執行的整體摘要。例如：

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

`<testsuites>` 標籤中使用的屬性

`name`

測試套件的名稱。

time

執行資格套件所花費的時間 (以秒為單位)。

tests

所執行的測試案例數目。

failures

已執行但未通過的測試案例數目。

errors

IDT for FreeRTOS 無法執行的測試案例數量。

disabled

此屬性未使用，可忽略。

如果沒有測試案例故障或錯誤，您的裝置即符合執行 FreeRTOS 的技術需求，可與 AWS IoT 服務相互運作。如果您選擇在 AWS Partner Device Catalog 中列出裝置，就能將此報告做為資格審查的證據。

在測試案例失敗或發生錯誤的情況下，您可以透過檢閱 `<testsuites>` XML 標籤來識別失敗的測試案例。`<testsuites>` 標籤內的 `<testsuite>` XML 標籤會顯示測試群組的測試案例結果摘要。

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

該格式與 `<testsuites>` 標籤相似，但具有不使用且可忽略的 `skipped` 屬性。在每個 `<testsuite>` XML 標籤內，系統為測試群組執行的每個測試案例都有 `<testcase>` 標籤。例如：

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

<awsproduct> 標籤中使用的屬性

name

受測產品名稱。

version

受測產品版本。

sdk

如果您使用 開發套件執行 IDT，則此區塊會包含您開發套件的名稱和版本。如果您未使用 開發套件執行 IDT，則此區塊包含：

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

features

驗證的功能。標記為 `required` 的功能為提交主機板獲得資格時所需。以下程式碼片段示範如何將此顯示在 `awsiotdevicetester_report.xml` 檔案中。

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

標記為 optional 的功能不需要進行資格測試。以下程式碼片段顯示選用功能。

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

如果所需功能沒有測試失敗或錯誤，您的裝置即符合執行 FreeRTOS 的技術要求，可與 AWS IoT 服務相互運作。若您希望在 [AWS Partner Device Catalog](#) 中列出您的裝置，您可以使用此報告做為資格審查的證據。

如果測試發生失敗或錯誤，您可以檢閱 `<testsuites>` XML 標籤來識別失敗的測試。`<testsuites>` 標籤內的 `<testsuite>` XML 標籤會顯示測試群組的測試結果摘要。例如：

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

其格式類似於 `<testsuites>` 標籤，但具有未使用且可忽略的 `skipped` 屬性。在每個 `<testsuite>` XML 標籤內，測試群組每個執行的測試都有 `<testcase>` 標籤。例如：

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

lts

如果您符合使用 LTS 程式庫的 FreeRTOS 版本資格，否則為 true。

`<testcase>` 標籤中使用的屬性

`name`

測試案例的名稱。

`attempts`

IDT for FreeRTOS 執行測試案例的次數。

當測試案例失敗或發生錯誤時，系統就會將 `<failure>` 或 `<error>` 標籤新增至 `<testcase>` 標籤，其中附有相關資訊以利故障診斷。例如：

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

如需詳細資訊，請參閱 [Troubleshooting \(p. 373\)](#)。

檢視日誌

您可以在 FreeRTOS 中找到 IDT for `devicetester-extract-location/results/execution-id/logs` 從測試執行而建立的對數。系統會建立兩組物件：

`test_manager.log`

包含產生自 IDT for FreeRTOS 的日誌 (例如，與組態和報告產生相關的日誌皆在此)。

`test_group_id__test_case_id.log` (例如，`FullMQTT__Full_MQTT.log`)

測試案例的日誌檔案，包括來自測試中裝置的輸出。日誌檔案會根據執行的測試群組和測試案例命名。

使用 IDT 來開發和執行您自己的測試套件

從 IDT v4.0.1 開始，IDT for FreeRTOS 結合了標準化的組態設定和結果格式與測試套件環境，可讓您為裝置和裝置軟體開發自訂測試套件。您可以為自己的驗證新增自訂測試，或將其提供給使用者以進行裝置驗證。

使用 IDT 開發和執行自訂測試套件，如下所示：

開發自訂測試套件

- 搭配您想要測試的裝置的自訂測試邏輯建立測試套件。
- 將自訂測試套件提供給 IDT 以測試執行器。包含測試套件的特定設定組態資訊。

執行自訂測試套件

- 設定您要測試的裝置。
- 依您想要使用的測試套件要求來實作設定組態。
- 使用 IDT 來執行您的自訂測試套件。
- 檢視 IDT 所執行測試的測試結果和執行數。

下載最新版本的 AWS IoT Device Tester for FreeRTOS

下載 IDT 的最新版本 ([p. 291](#))，並將軟體解壓縮到檔案系統上，您在該系統中具有讀取和寫入許可。

Note

IDT 不支援由多位使用者從共用位置執行，例如 NFS 目錄或 Windows 網路共用資料夾。我們建議您將 IDT 套件解壓縮到本機磁碟機，並在本機工作站上執行 IDT 二進位檔。

Windows 的路徑長度限制為 260 個字元。如果您使用的是 Windows，請將 IDT 解壓縮到根目錄，例如 C:\ 或 D:\，使路徑保持在 260 個字元的限制以下。

測試套件建立工作流程

測試套件包含三種類型的檔案：

- JSON 組態檔案，提供 IDT 如何執行測試套件的相關資訊。
- 測試 IDT 用來執行測試案例的可執行檔。
- 執行測試所需的其他檔案。

完成下列基本步驟來建立自訂 IDT 測試：

1. 為您的測試套件建立 JSON 組態檔案 ([p. 330](#))。
2. 建立測試案例可執行檔 ([p. 350](#))，其中包含您的測試套件的測試邏輯。
3. 驗證並記載執行測試套件所需的測試執行器組態資訊 ([p. 357](#))。
4. 確認 IDT 可以執行您的測試套件並如預期地製作測試結果 ([p. 364](#))。

若要快速建置並執行範例自訂套件，請遵循教學課程：建置並執行範例 IDT 測試套件 ([p. 320](#)) 中的指示。

若要開始在 Python 中建立自訂測試套件，請前往 教學課程：開發簡單的 IDT 測試套件 ([p. 324](#))。

教學課程：建置並執行範例 IDT 測試套件

下載包含範例測試套件的來源碼。AWS IoT Device Tester 您可以完成此教學來建置和執行範例測試套件，以了解您可以如何使用 AWS IoT Device Tester for FreeRTOS 來執行自訂測試套件。

在此教學課程中，您將完成下列步驟：

1. 建置範例測試套件 (p. 323)
2. 使用 IDT 來執行範例測試套件 (p. 323)

Prerequisites

為了完成本教學，您需要以下項目：

- 主機電腦要求
 - AWS IoT Device Tester 的最新版本
 - Python 3.7 或更新版本

若要檢查電腦上安裝的 Python 版本，請執行下列命令：

```
python3 --version
```

在 Windows 上，如果使用此命令，會傳回錯誤，請改為使用 `python --version`。如果傳回的版本編號為 3.7 或更新版本，請在 Powershell 終端機中執行以下命令，以將 `python3` 設定為 `python` 命令的 alias。

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

如果未傳回版本資訊或版本號碼低於 3.7，請按照[下載 Python](#) 中的指示安裝 Python 3.7+。如需詳細資訊，請查看[Python 文件](#)。

- `urllib3` (`urllib3`)

若要確認 `urllib3` 已正確安裝，請執行下列命令：

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

如果未安裝 `urllib3`，請執行下列命令來安裝它：

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

- 裝置要求

- 具備 Linux 作業系統及網路連接至相同網路的裝置會與您的主機電腦連接。

我們建議您使用 [Raspberry Pi](#) 搭配 Raspberry Pi OS。請確定您已在 Raspberry Pi 上設定 [SSH](#) 遠端連接。

設定 IDT 的裝置資訊

為 IDT 設定裝置資訊以執行測試。您必須以下列資訊來更新位於 `device.json` 資料夾中的 `<device-tester-extract-location>/configs` 範本。

```
[  
 {  
   "id": "pool",  
   "sku": "N/A",  
   "devices": [  
     {  
       "id": "<device-id>",  
     }  
   ]  
 }]
```

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

在 devices 物件中，提供下列資訊：

`id`

使用者定義的裝置唯一標識符。

`connectivity.ip`

裝置的 IP 地址。

`connectivity.port`

選用。用於裝置 SSH 連接的連接埠號碼。

`connectivity.auth`

連線的驗證資訊。

只有當 `connectivity.protocol` 設為 `ssh` 時，才會套用此屬性。

`connectivity.auth.method`

用來透過指定的連線通訊協定存取裝置的驗證方法。

支援的值如下：

- `pki`
- `password`

`connectivity.auth.credentials`

用於驗證的登入資料。

`connectivity.auth.credentials.user`

用來登入您裝置的使用者名稱。

`connectivity.auth.credentials.privKeyPath`

用來登入至您的裝置之私有金鑰的完整路徑。

只有當 `connectivity.auth.method` 設為 `pki` 時，才會套用此值。

`devices.connectivity.auth.credentials.password`

用於登入裝置的密碼。

只有當 `connectivity.auth.method` 設為 `password` 時，才會套用此值。

Note

如果 method 是設定為 pki , 則指定 privKeyPath
如果 method 是設定為 password , 則指定 password

建置範例測試套件

資料夾包含範例組態檔案、原始碼，以及您可以使用提供的建置指令碼結合到測試套件的 IDT Client 開發套件。`<device-tester-extract-location>/samples/python`下列樹狀資料可顯示這些範例檔案的位置：

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

若要建置測試套件，請在您的主機電腦上執行下列命令：

Windows

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

Linux, macOS, or UNIX

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

這會在 `IDTSampleSuitePython_1.0.0` 資料夾的 `<device-tester-extract-location>/tests` 資料夾中建立範例測試套件。檢視 `IDTSampleSuitePython_1.0.0` 資料夾中的檔案，以了解範例測試套件的結構方式，並查看測試案例可執行檔的各種範例和測試組態 JSON 檔案。

後續步驟：使用 IDT 來執行您建立的範例測試套件 (p. 323)。

使用 IDT 來執行範例測試套件

若要執行範例測試套件，請在您的主機電腦上執行下列命令：

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

IDT 執行範例測試套件並將結果串流至主控台。測試執行完成時，您會看到以下資訊：

```
===== Test Summary =====
```

```
Execution Time:      5s
Tests Completed:    4
Tests Passed:        4
Tests Failed:       0
Tests Skipped:      0
-----
Test Groups:
  sample_group:      PASSED
-----
Path to AWS IoT Device Tester Report: /path/to/devicetester/
results/87e673c6-1226-11eb-9269-8c8590419f30/awsiotdevicetester_report.xml
Path to Test Execution Logs: /path/to/devicetester/
results/87e673c6-1226-11eb-9269-8c8590419f30/logs
Path to Aggregated JUnit Report: /path/to/devicetester/
results/87e673c6-1226-11eb-9269-8c8590419f30/IDTSampleSuitePython_Report.xml
```

Troubleshooting

使用下列資訊，協助解析任何完成教學課程的問題。

測試案例未成功執行

如果測試未成功執行，IDT 會將錯誤串流到主控台，協助您對測試執行進行故障診斷。請確定您符合本教學的所有事前準備 (p. 321)。

無法連接到測試的裝置

請確認下列內容：

- 您的 device.json 檔案包含正確的 IP 地址、連接埠和身份驗證資訊。
- 您可以從主機電腦透過 SSH 連接到您的裝置。

教學課程：開發簡單的 IDT 測試套件

測試套件結合了下列各項：

- 包含測試邏輯的測試可執行檔
- 描述測試套件的 JSON 組態檔案

此教學課程示範如何使用 IDT for FreeRTOS 開發包含單一測試案例的 Python 測試套件。在此教學課程中，您將完成下列步驟：

1. 建立測試套件資料夾 (p. 325)
2. 建立 JSON 組態檔案 (p. 325)
3. 建立測試案例可執行檔 (p. 327)
4. 執行測試套件 (p. 329)

Prerequisites

為了完成本教學，您需要以下項目：

- 主機電腦要求
 - AWS IoT Device Tester 的最新版本

- Python 3.7 或更新版本

若要檢查電腦上安裝的 Python 版本，請執行下列命令：

```
python3 --version
```

在 Windows 上，如果使用此命令，會傳回錯誤，請改為使用 `python --version`。如果傳回的版本編號為 3.7 或更新版本，請在 Powershell 終端機中執行以下命令，以將 `python3` 設定為 `python` 命令的 alias。

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

如果未傳回版本資訊或版本號碼低於 3.7，請按照[下載 Python](#) 中的指示安裝 Python 3.7+。如需詳細資訊，請查看[Python 文件](#)。

- `urllib3` (`urllib3`)

若要確認 `urllib3` 已正確安裝，請執行下列命令：

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

如果未安裝 `urllib3`，請執行下列命令來安裝它：

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

- **裝置要求**

- 具備 Linux 作業系統及網路連接至相同網路的裝置會與您的主機電腦連接。

我們建議您使用 [Raspberry Pi](#) 搭配 Raspberry Pi OS。請確定您已在 Raspberry Pi 上設定 [SSH](#) 遠端連接。

建立測試套件資料夾

IDT 會邏輯地將測試案例分成每個測試套件中的測試群組。每個測試案例必須位於測試群組中。在本教學課程中，建立名為 `MyTestSuite_1.0.0` 的資料夾，並在此資料夾中建立下列資料夾：

```
MyTestSuite_1.0.0
### suite
### myTestGroup
### myTestCase
```

建立 JSON 組態檔案

您的測試套件必須包含下列必要的 [JSON 組態檔案](#) (p. 330)：

必要的 JSON 檔案

`suite.json`

包含測試套件的資訊。請參閱[設定 suite.json](#) (p. 331)。

`group.json`

包含測試群組的資訊。您必須為測試套件中的每個測試群組建立一個 `group.json` 檔案。請參閱[設定 group.json](#) (p. 332)。

test.json

包含測試案例的資訊。您必須為測試套件中的每個測試案例建立一個 test.json 檔案。請參閱[設定 test.json \(p. 332\)](#)。

1. 在 MyTestSuite_1.0.0/suite 資料夾中，建立具有以下結構的 suite.json 檔案：

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

2. 在 MyTestSuite_1.0.0/myTestGroup 資料夾中，建立具有以下結構的 group.json 檔案：

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

3. 在 MyTestSuite_1.0.0/myTestGroup/myTestCase 資料夾中，建立具有以下結構的 test.json 檔案：

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

您 MyTestSuite_1.0.0 資料夾的加密樹現在看起來應該與下列類似：

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

```
### myTestCase
### test.json
```

取得 IDT 使用者開發套件

您可以使用 [IDT client SDK \(p. 350\)](#) 與測試裝置互動，並報告測試結果。在本教學課程中，您將使用 Python 版本的開發套件。

從 `<device-tester-extract-location>/sdks/python/` 資料夾將 `idt_client` 資料夾複製到您的 `MyTestSuite_1.0.0/suite/myTestGroup/myTestCase` 資料夾。

若要驗證是否已成功複製軟體開發套件，請執行下列命令。

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

建立測試案例可執行檔

測試案例可執行檔包含您要執行的測試邏輯。測試套件可包含多個測試案例可執行檔。在此教學課程中，您將只會建立一個測試案例可執行檔。

1. 建立測試套件檔案。

在 `MyTestSuite_1.0.0/suite/myTestGroup/myTestCase` 資料夾中，建立包含下列參數的 `myTestCase.py` 檔案：

```
from idt_client import *

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

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

2. 使用 `client` 開發套件函數，將下列測試邏輯新增至 `myTestCase.py` 檔案：

- a. 在待測裝置上執行 SSH 命令。

```
from idt_client import *

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

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

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

    # Print the standard output
    print(exec_resp.stdout)

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

- b. 將測試結果傳送到 IDT。

```
from idt_client import *
```

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

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

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

    # Print the standard output
    print(exec_resp.stdout)

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

    # Send the result
    client.send_result(sr_req)

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

設定 IDT 的裝置資訊

為 IDT 設定裝置資訊以執行測試。您必須以下列資訊來更新位於 device.json 資料夾中的 `<device-tester-extract-location>/configs` 範本。

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

在 devices 物件中，提供下列資訊：

id

使用者定義的裝置唯一標識符。

connectivity.ip

裝置的 IP 地址。

connectivity.port

選用。用於裝置 SSH 連接的連接埠號碼。

connectivity.auth

連線的驗證資訊。

只有當 connectivity.protocol 設為 ssh 時，才會套用此屬性。

connectivity.auth.method

用來透過指定的連線通訊協定存取裝置的驗證方法。

支援的值如下：

- pki
- password

connectivity.auth.credentials

用於驗證的登入資料。

connectivity.auth.credentials.user

用來登入您裝置的使用者名稱。

connectivity.auth.credentials.privKeyPath

用來登入至您的裝置之私有金鑰的完整路徑。

只有當 connectivity.auth.method 設為 pki 時，才會套用此值。

devices.connectivity.auth.credentials.password

用於登入裝置的密碼。

只有當 connectivity.auth.method 設為 password 時，才會套用此值。

Note

如果 method 是設定為 pki，則指定 privKeyPath

如果 method 是設定為 password，則指定 password

執行測試套件

建立測試套件之後，您想要確定其可如預期運作。完成以下步驟，以您現有的裝置集區來執行測試套件。

1. 將您的 MyTestSuite_1.0.0 資料夾複製到 <device-tester-extract-location>/tests。
2. 執行下列命令：

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

IDT 會執行您的測試套件，並將結果串流至主控台。測試執行完成時，您會看到以下資訊：

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

```
time="2020-10-19T09:24:47-07:00" level=info msg=All tests finished.  
executionId=9a52f362-1227-11eb-86c9-8c8590419f30  
time="2020-10-19T09:24:48-07:00" level=info msg=  
  
===== Test Summary =====  
Execution Time: 1s  
Tests Completed: 1  
Tests Passed: 1  
Tests Failed: 0  
Tests Skipped: 0  
  
-----  
Test Groups:  
myTestGroup: PASSED  
-----  
Path to AWS IoT Device Tester Report: /path/to/devicetester/  
results/9a52f362-1227-11eb-86c9-8c8590419f30/awsiotdevicetester_report.xml  
Path to Test Execution Logs: /path/to/devicetester/  
results/9a52f362-1227-11eb-86c9-8c8590419f30/logs  
Path to Aggregated JUnit Report: /path/to/devicetester/  
results/9a52f362-1227-11eb-86c9-8c8590419f30/MyTestSuite_Report.xml
```

Troubleshooting

使用下列資訊，協助解析任何完成教學課程的問題。

測試案例未成功執行

如果測試未成功執行，IDT 會將錯誤串流到主控台，協助您對測試執行進行故障診斷。在檢查錯誤碼之前，請檢查下列各項：

- IDT 使用者開發套件位於正確的資料夾，如[此步驟 \(p. 327\)](#)所述。
- 您符合此教學課程的所有[事前準備 \(p. 324\)](#)。

無法連接到測試的裝置

請確認下列內容：

- 您的 device.json 檔案包含正確的 IP 地址、連接埠和身份驗證資訊。
- 您可以從主機電腦透過 SSH 連接到您的裝置。

建立 IDT 測試套件組態檔案

本節描述編寫自訂測試套件時，所包含之 JSON 組態檔案的撰寫格式。

必要的 JSON 檔案

`suite.json`

包含測試套件的資訊。請參閱[設定 suite.json \(p. 331\)](#)。

`group.json`

包含測試群組的資訊。您必須為測試套件中的每個測試群組建立一個 group.json 檔案。請參閱[設定 group.json \(p. 332\)](#)。

`test.json`

包含測試案例的資訊。您必須為測試套件中的每個測試案例建立一個 test.json 檔案。請參閱[設定 test.json \(p. 332\)](#)。

選用的 JSON 檔案

state_machine.json

定義 IDT 在執行測試套件時，執行測試的方式。請參閱[設定 state_machine.json \(p. 334\)](#)。

userdata_schema.json

定義測試執行器可以包含在其設定組態中的 [userdata.json 檔案 \(p. 360\)](#)結構描述。檔案用於執行測試所需的任何其他組態資訊，但 userdata.json 檔案中不存在。device.json請參閱[設定 userdata_schema.json \(p. 335\)](#)。

JSON 組態檔案會放置在您的 `<custom-test-suite-folder>` 中，如下所示。

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

設定 suite.json

檔案會設定環境變數，並判斷是否需要使用者資料來執行測試套件。suite.json使用以下範本來設定您的 `<custom-test-suite-folder>/suite/suite.json` 檔案：

```
{
  "id": "<suite-name>_<suite-version>",
  "title": "<suite-title>",
  "details": "<suite-details>",
  "userDataRequired": true | false,
  "environmentVariables": [
    {
      "key": "<name>",
      "value": "<value>",
    },
    ...
    {
      "key": "<name>",
      "value": "<value>",
    }
  ]
}
```

如下所述，包含值的所有欄位皆為必要：

id

測試套件的唯一使用者定義 ID。的值必須符合 id 檔案所在的測試套件資料夾名稱。suite.json套件名稱和套件版本也必須符合下列要求：

- `<suite-name>` 不能包含分數不足。
- `<suite-version>` 表示 `x.x.x`，其中 x 是數字。

ID 會顯示在 IDT 發出的測試報告中。

title

此測試套件測試的 product 或功能的使用者定義名稱。該名稱會顯示在 IDT CLI for 測試執行器中。

details

測試套件用途的簡短描述。

userDataRequired

定義測試執行器是否需要在 `userdata.json` 檔案中包含自訂資訊。若您將此值設為 `true`，您也必須在測試套件資料夾中包含 [檔案 `userdata_schema.json` \(p. 335\)](#)

environmentVariables

選用。要為此測試套件設定的環境變數陣列。

environmentVariables.key

環境變數的名稱。

environmentVariables.value

環境變數的值。

設定 `group.json`

檔案定義測試群組是必要項目或選用項目。`group.json` 使用以下範本來設定您的 [`<custom-test-suite-folder>/suite/<test-group>/group.json` 檔案](#)：

```
{  
    "id": "<group-id>",  
    "title": "<group-title>",  
    "details": "<group-details>",  
    "optional": true | false,  
}
```

如下所述，包含值的所有欄位皆為必要：

id

測試群組的唯一使用者定義 ID。的值必須符合 `id` 檔案所在的測試群組資料夾名稱。`group.json` 該 ID 會用於 IDT 發出的測試報告。

title

測試群組的描述性名稱。該名稱會顯示在 IDT CLI for 測試執行器中。

details

測試群組目的的簡短描述。

optional

選用。IDT 完成執行必要測試之後，設定為 `true` 可將此測試群組顯示為選用群組。預設值為 `false`。

設定 `test.json`

檔案會判斷測試案例可執行檔，以及測試案例所使用的環境變數。`test.json` 如需建立測試案例可執行檔的詳細資訊，請前往 [建立 IDT 測試案例可執行檔 \(p. 350\)](#)。

使用以下範本來設定您的 [`<custom-test-suite-folder>/suite/<test-group>/<test-case>/test.json` 檔案](#)：

```
{  
    "id": "<test-id>",  
    "title": "<test-title>",
```

```
"details": "<test-details>",
"requireDUT": true | false,
"requiredResources": [
    {
        "name": "<resource-name>",
        "features": [
            {
                "name": "<feature-name>",
                "version": "<feature-version>",
                "jobSlots": <job-slots>
            }
        ]
    }
],
"execution": {
    "timeout": <timeout>,
    "mac": {
        "cmd": "/path/to/executable",
        "args": [
            "<argument>"
        ],
        ...
    },
    "linux": {
        "cmd": "/path/to/executable",
        "args": [
            "<argument>"
        ],
        ...
    },
    "win": {
        "cmd": "/path/to/executable",
        "args": [
            "<argument>"
        ]
    }
},
"environmentVariables": [
    {
        "key": "<name>",
        "value": "<value>",
        ...
    }
]
}
```

如下所述，包含值的所有欄位皆為必要：

id

測試案例的唯一使用者定義 ID。的值必須符合 id 檔案所在的測試案例資料夾名稱。test.json該 ID 會用於 IDT 發出的測試報告。

title

測試案例的描述性名稱。該名稱會顯示在 IDT CLI for 測試執行器中。

details

測試案例用途的簡短描述。

requireDUT

選用。如果需要執行此測試，請設定為 true，否則設定為 false。預設值為 true。測試執行器會設定它們在 device.json 檔案中用來執行測試的裝置。

requiredResources

選用。陣列，提供執行此測試所需資源裝置的相關資訊。

`requiredResources.name`

此測試執行時提供給資源裝置的唯一名稱。

`requiredResources.features`

使用者定義的資源裝置功能的陣列。

`requiredResources.features.name`

功能的名稱。您要使用此裝置的裝置功能。此名稱會與 `resource.json` 檔案中測試執行器所提供的功能名稱進行比對。

`requiredResources.features.version`

選用。功能的版本。這個值會與 `resource.json` 檔案中測試執行器所提供的功能版本相符。

如果未提供版本，則不會檢查功能。如果功能不需要版本編號，請將此欄位保留空白。

`requiredResources.features.jobSlots`

選用。此功能可支援的同步測試次數。預設值為 1。如果您希望 IDT 針對各個功能使用不同的裝置，建議您將此值設定為 1。

`execution.timeout`

IDT 等待測試完成執行的時間量（毫秒）。如需設定此值的詳細資訊，請前往 [建立 IDT 測試案例可執行檔 \(p. 350\)](#)。

`execution.os`

根據執行 IDT 之主機電腦作業系統執行的測試案例可執行檔。支援的值為 linux、mac 和 win。

`execution.os.cmd`

您想要針對指定作業系統執行之測試案例可執行檔的路徑。此位置必須位於系統路徑。

`execution.os.args`

選用。提供以執行測試案例可執行檔的引數。

`environmentVariables`

選用。為此測試案例設定的環境變數陣列。

`environmentVariables.key`

環境變數的名稱。

`environmentVariables.value`

環境變數的值。

`Note`

如果您在 `test.json` 檔案中和 `suite.json` 檔案中指定相同的環境變數，`test.json` 檔案中的值優先。

設定 `state_machine.json`

狀態機器是一種建構，可控制測試套件執行流程。它會判斷測試套件的開始狀態、根據使用者定義的規則管理狀態轉換，然後繼續透過這些狀態轉換，直到它達到結束狀態。

如果您的測試套件不包含使用者定義的狀態機器，IDT 會為您建立狀態機器。預設狀態機器會執行下列函數：

- 提供測試執行器選擇和執行特定測試群組（而非整個測試套件）的能力。

- 如果未選取特定測試群組，會以隨機順序執行測試套件中的每個測試群組。
- 製作報告並列印主控台摘要，以顯示每個測試群組和測試案例的測試結果。

如需 IDT 狀態機器函數的詳細資訊，請前往 [設定 IDT 狀態機器 \(p. 335\)](#)。

設定 userdata_schema.json

檔案會判斷測試執行器提供使用者資料的結構描述。userdata_schema.json如果您的測試套件需要的資訊不存在於 device.json 檔案中，則需要使用者資料。例如，您的測試可能需要 Wi-Fi 網路登入資料、特定的開放連接埠或使用者必須提供的憑證。您可以將此資訊提供給 IDT 做為輸入參數（稱為 userdata），該參數是使用者在其 userdata.json 資料夾中建立的 *<device-tester-extract-location>/config* 檔案值。檔案的格式是根據您在測試套件中包含的 userdata.json 檔案而定。userdata_schema.json

指出測試執行器必須提供 userdata.json 檔案：

1. 在 suite.json 檔案中，將 userDataRequired 設定為 true。
2. 在您的 *<custom-test-suite-folder>* 中建立 userdata_schema.json 檔案。
3. 編輯 userdata_schema.json 檔案以建立有效的 IETF Draft v4 JSON 結構描述。

IDT 執行您的測試套件時，會自動讀取結構描述，並使用它來驗證測試執行器提供的 userdata.json 檔案。如果有效，userdata.json 檔案的 [檔案分在 \(p. 354\)](#)IDT context (IDT 細節) 和 [\(p. 343\)](#)state machine context (狀態機器語境) 中都有提供。

設定 IDT 狀態機器

狀態機器是一種建構，可控制測試套件執行流程。它會判斷測試套件的開始狀態、根據使用者定義的規則管理狀態轉換，然後繼續透過這些狀態轉換，直到它達到結束狀態。

如果您的測試套件不包含使用者定義的狀態機器，IDT 會為您建立狀態機器。預設狀態機器會執行下列函數：

- 提供測試執行器選擇和執行特定測試群組（而非整個測試套件）的能力。
- 如果未選取特定測試群組，會以隨機順序執行測試套件中的每個測試群組。
- 製作報告並列印主控台摘要，以顯示每個測試群組和測試案例的測試結果。

IDT 測試套件的狀態機器必須符合下列條件：

- 每個狀態對應至供 IDT 執行的動作，例如執行測試群組或將報告檔案積存。
- 轉換為 狀態會執行與狀態相關聯的動作。
- 每個狀態都會定義下一個狀態的轉換規則。
- 結束狀態必須是 Succeed 或 Fail。

狀態機器格式

您可以使用下列範本來設定自己的 *<custom-test-suite-folder>/suite/state_machine.json* 檔案：

```
{  
  "Comment": "<description>",  
  "StartAt": "<state-name>,"
```

```
"States": {  
    "<state-name>": {  
        "Type": "<state-type>",  
        // Additional state configuration  
    }  
  
    // Required states  
    "Succeed": {  
        "Type": "Succeed"  
    },  
    "Fail": {  
        "Type": "Fail"  
    }  
}
```

如下所述，包含值的所有欄位皆為必要：

Comment

狀態機器的描述。

StartAt

IDT 開始執行測試套件的狀態名稱。的值必須設定為 StartAt 物件中所列的其中一個狀態。States States

將使用者定義狀態名稱映射至有效 IDT 狀態的物件。每個狀態。*state-name* 物件包含映射到 的有效狀態定義 *state-name*。

物件必須包含 States 和 Succeed 狀態。Fail如需有效狀態的資訊，請前往 [有效狀態和狀態定義 \(p. 336\)](#)。

有效狀態和狀態定義

本節描述可用於 IDT 狀態機器的所有有效狀態的狀態定義。下列某些狀態支援測試案例層級的組態。不過，我們建議您在測試群組層級設定狀態轉換規則，而不是在測試案例層級設定，除非為必要。

狀態定義

- [RunTask \(p. 336\)](#)
- [Choice \(p. 337\)](#)
- [Parallel \(p. 338\)](#)
- [AddProductFeatures \(p. 339\)](#)
- [Report \(p. 341\)](#)
- [LogMessage \(p. 341\)](#)
- [SelectGroup \(p. 342\)](#)
- [Fail \(p. 342\)](#)
- [Succeed \(p. 342\)](#)

RunTask

狀態會從測試套件中定義的測試群組執行測試案例。RunTask

```
{  
    "Type": "RunTask",
```

```
"Next": "<state-name>",
"TestGroup": "<group-id>",
"TestCases": [
    "<test-id>"
],
"ResultVar": "<result-name>"}
```

如下所述，包含值的所有欄位皆為必要：

Next

在目前狀態中執行動作後要轉換為的狀態名稱。

TestGroup

選用。要執行的測試群組 ID。如果未指定此值，則 IDT 會執行測試執行器選取的測試群組。

TestCases

選用。中指定群組中 IDs 的測試案例陣列。TestGroup IDT 會根據 TestGroup 和 TestCases 的值，判斷測試執行行為，如下所示：

- 指定 TestGroup 和 TestCases 時，IDT 會從測試群組執行指定的測試案例。
- 指定 TestCases 但未指定 TestGroup 時，IDT 會執行指定的測試案例。
- 指定 TestGroup 時，但未指定 TestCases，IDT 會在指定的測試群組中執行所有測試案例。
- 未指定 TestGroup 或 TestCases 時，IDT 會從測試執行器從 IDT CLI 所選的測試群組執行所有測試案例。若要讓群組選擇測試執行器，您必須在 RunTask 檔案中同時包含 Choice 和 statemachine.json 狀態。如需此運作方式的範例，請查看範例狀態機器：執行使用者選取的測試群組 (p. 346)。

如需讓測試執行器執行 IDT CLI 命令的詳細資訊，請前往[the section called “Enable IDT CLI 命令” \(p. 352\)](#)。

ResultVar

要用來設定測試執行結果的環境變數名稱。如果您未指定 TestGroup 的值，請勿指定此值。IDT 會根據下列項目，將您在 ResultVar 中定義的變數值設為 true 或 false：

- 如果變數名稱的格式為 `text_text_passed`，則值設為第一個測試群組中的所有測試是否傳遞或略過。
- 在所有其他案例中，會將此值設為所有測試群組中的所有測試傳遞或略過。

一般而言，您將使用 RunTask 狀態來指定測試群組 ID，而未指定單一測試案例 IDs，以便 IDT 執行指定測試群組中的所有測試案例。此狀態執行的所有測試案例都會以隨機順序平行執行。不過，如果所有測試案例都需要執行裝置，而且只有單一裝置可用，則測試案例會依序執行。

錯誤處理

如果任何指定的測試群組或測試案例 IDs 無效，則此狀態會發出 RunTaskError 執行錯誤。如果狀態遇到執行錯誤，它也會在狀態機器上下文中將 hasExecutionError 變數設為 true。

Choice

狀態可讓您根據使用者定義的條件，動態設定下一個狀態轉換為。Choice

```
{  
    "Type": "Choice",  
    "Default": "<state-name>",  
    "FallthroughOnError": true | false,
```

```
"Choices": [
    {
        "Expression": "<expression>",
        "Next": "<state-name>"
    }
]
```

如下所述，包含值的所有欄位皆為必要：

Default

如果 Choices 中定義的表達式都無法評估為 true，要轉換為此預設狀態。

FallthroughOnError

選用。指定當狀態在評估表達式時發生錯誤時的行為。如果您想要在評估造成錯誤時跳過運算式，請設為 true。如果表達式不相符，則狀態機器會轉換到 Default 狀態。如果未指定 FallthroughOnError 值，則會預設為 false。

Choices

表達式和狀態的陣列，以判斷在目前狀態中執行動作後要轉換為哪個狀態。

Choices.Expression

評估為布林值的表達式字串。如果表達式評估結果為 true，則狀態機器會轉換到 Choices.Next 中定義的狀態。表達式字串會從狀態機器環境擷取值，然後對其執行操作，以使其達到布林值。如需取得狀態機器語境的資訊，請前往 [狀態機器語境 \(p. 343\)](#)。

Choices.Next

在 Choices.Expression 中定義的表達式評估為 true 時，要轉換為此狀態的名稱。

錯誤處理

狀態可能需要在下列案例中處理錯誤：Choice

- 選擇表達式中的某些變數不存在於狀態機器上下文中。
- 運算式的結果不是布林值。
- JSON 查詢的結果不是字串、數字或布林值。

您無法使用 Catch 區塊來處理此狀態的錯誤。如果您希望在遇到錯誤時停止執行狀態機器，您必須將 FallthroughOnError 設定為 false。不過，我們建議您將 FallthroughOnError 設定為 true，並視您的使用案例而定，請執行以下其中一項：

- 如果您要存取的變數在某些案例中預計不存在，請使用 Default 值和其他 Choices 區塊來指定下一個狀態。
- 如果您要存取的變數應始終存在，請將 Default 狀態設定為 Fail。

Parallel

狀態可讓您同時定義和執行新的狀態機器。Parallel

```
{
    "Type": "Parallel",
    "Next": "<state-name>",
    "Branches": [

```

```
<state-machine-definition>
}
]
```

如下所述，包含值的所有欄位皆為必要：

Next

在目前狀態中執行動作後要轉換為的狀態名稱。

Branches

要執行的狀態機器定義陣列。每個狀態機器定義都必須包含自己的 StartAt、Succeed 和 Fail 狀態。此陣列中的狀態機器定義不能參考自己的定義外部的狀態。

Note

因為每個分支狀態機器都會共享相同的狀態機器環境，所以在某個分支中設定變數，然後從另一個分支讀取這些變數可能會導致未預期的行為。

狀態只會在執行所有分支狀態機器之後移至下一個狀態。Parallel每個需要裝置等待執行的狀態，直到裝置可用為止。如果有多個裝置可用，此狀態會從多個群組平行執行測試案例。如果裝置無法使用就不足，測試案例就會循序執行。因為測試案例平行執行時是以隨機順序執行，所以可能會使用不同的裝置從相同的測試群組執行測試。

錯誤處理

確定分支狀態機器和父狀態機器都會轉換到 Fail 狀態，以處理執行錯誤。

由於分支狀態機器不會將執行錯誤傳輸至父狀態機器，因此您無法使用 Catch 區塊來處理分支狀態機器中的執行錯誤。反之，請在共用狀態機器環境中使用 hasExecutionErrors 值。如需其運作方式的範例，請查看 [範例狀態機器：平行執行兩個測試群組 \(p. 348\)](#)。

AddProductFeatures

狀態可讓您新增 product 功能到 IDT 生成的 AddProductFeatures 檔案中。`awsiotdevicetester_report.xml`

product 功能是裝置可能滿足的特定條件的使用者定義資訊。例如，MQTT product 功能可以指定裝置正確發布 MQTT 訊息。在報告中，根據指定的測試是否通過，將 product 功能設定為 supported、not-supported 或自訂值。

Note

狀態無法自行建立報告。AddProductFeatures此狀態必須轉換到 [Report 狀態 \(p. 341\)](#) 才能取得報告。

```
{
    "Type": "Parallel",
    "Next": "<state-name>",
    "Features": [
        {
            "Feature": "<feature-name>",
            "Groups": [
                "<group-id>"
            ],
            "OneOfGroups": [
                "<group-id>"
            ],
        }
    ]
}
```

```
"TestCases": [  
    "<test-id>"  
],  
"IsRequired": true | false,  
"ExecutionMethods": [  
    "<execution-method>"  
]  
}  
]  
}
```

如下所述，包含值的所有欄位皆為必要：

Next

在目前狀態中執行動作後要轉換為的狀態名稱。

Features

要在 `awsiotdevicetester_report.xml` 檔案中顯示的 product 功能陣列。

Feature

功能的名稱

FeatureValue

選用。要在報告中使用的自訂值，而不是 `supported`。如果未指定此值，則根據測試結果，特徵值設定為 `supported` 或 `not-supported`。

如果您對 `FeatureValue` 使用自訂值，您可以使用不同的條件測試相同的功能，IDT 串連支援條件的功能值。例如，以下摘要顯示了具有兩個不同功能值的 `MyFeature` 功能：

```
...  
{  
    "Feature": "MyFeature",  
    "FeatureValue": "first-feature-supported",  
    "Groups": ["first-feature-group"]  
},  
{  
    "Feature": "MyFeature",  
    "FeatureValue": "second-feature-supported",  
    "Groups": ["second-feature-group"]  
},  
...
```

如果兩個測試群組都通過，則功能值會設為 `first-feature-supported`, `second-feature-supported`。

Groups

選用。測試群組 IDs 的陣列。每個指定測試群組中的所有測試都必須通過，功能才能受到支援。

OneOfGroups

選用。測試群組陣列。IDs 至少要通過其中一個指定測試群組中的所有測試，才能支援此功能。

TestCases

選用。測試案例的陣列。IDs 如果您指定此值，則適用下列：

- 所有指定的測試案例都必須通過，功能才能受到支援。
- `Groups` 只能包含一個測試群組 ID。
- 不得指定 `OneOfGroups`。

IsRequired

選用。設定為 `false` 可在報告中將此功能標記為選用功能。預設值為 `true`。

ExecutionMethods

選用。符合 `protocol` 檔案中所指定 `device.json` 值的執行方法陣列。如果指定此值，則測試執行器必須指定符合此陣列中其中一個值的 `protocol` 值，才能將功能包含在報告中。如果未指定此值，則該功能將一律包含在報告中。

若要使用 `AddProductFeatures` 狀態，您必須將處於 `ResultVar` 狀態的 `RunTask` 的值設為下列其中一個值：

- 如果您指定了一個測試案例 IDs，則請將 `ResultVar` 設定為 `group-id_test-id_passed`。
- 如果您未指定單一測試案例 IDs，請將 `ResultVar` 設定為 `group-id_passed`。

狀態會以下列方式檢查測試結果：`AddProductFeatures`

- 如果您未指定任何測試案例 IDs，則會從狀態機器環境中的 `group-id_passed` 變數值判斷每個測試群組的結果。
- 如果您已指定測試案例 IDs，則每個測試的結果都會從狀態機器環境中的 `group-id_test-id_passed` 變數值判斷。

錯誤處理

如果此狀態提供的群組 ID 不是有效的群組 ID，則此狀態會導致 `AddProductFeaturesError` 執行錯誤。如果狀態遇到執行錯誤，它也會在狀態機器上下文中將 `hasExecutionErrors` 變數設定為 `true`。

Report

狀態會建立 `Report` 和 `suite-name_Report.xml` 檔案。`awsiotdevicetester_report.xml` 此狀態也會將報告串流至主控台。

```
{  
    "Type": "Report",  
    "Next": "<state-name>"  
}
```

如下所述，包含值的所有欄位皆為必要：

Next

在目前狀態中執行動作後要轉換為的狀態名稱。

您應該一律轉換至 `Report` 狀態到測試執行流程結束，以便測試執行器可以檢視測試結果。通常，此狀態後的下一個狀態會是 `Succeed`。

錯誤處理

如果此狀態在製作報告時發生問題，則它會發出 `ReportError` 執行錯誤。

LogMessage

狀態會建立 `LogMessage` 檔案，並將 log 訊息串流至主控台。`test_manager.log`

```
{
```

```
{  
    "Type": "LogMessage",  
    "Next": "<state-name>"  
    "Level": "info | warn | error"  
    "Message": "<message>"  
}
```

如下所述，包含值的所有欄位皆為必要：

Next

在目前狀態中執行動作後要轉換為的狀態名稱。

Level

建立 log 訊息的錯誤層級。如果您指定的層級無效，此狀態會發出錯誤訊息並捨棄。

Message

登入訊息。

SelectGroup

狀態會更新狀態機器上下文，以指出已選取哪些群組。SelectGroup此狀態設定的值會由任何後續的Choice 狀態使用。

```
{  
    "Type": "SelectGroup",  
    "Next": "<state-name>"  
    "TestGroups": [  
        "<group-id>"  
    ]  
}
```

如下所述，包含值的所有欄位皆為必要：

Next

在目前狀態中執行動作後要轉換為的狀態名稱。

TestGroups

將標示為選取的測試群組陣列。對於此陣列中的每個測試群組 ID，*group-id_selected* 變數在上下文中設定為 true。請確定您提供有效的測試群組 IDs，因為 IDT 不會驗證指定的群組是否存在。

Fail

狀態表示狀態機器未正確執行。Fail這是狀態機器的結束狀態，而且每個狀態機器定義都必須包含此狀態。

```
{  
    "Type": "Fail"  
}
```

Succeed

狀態表示狀態機器正確執行。Succeed這是狀態機器的結束狀態，而且每個狀態機器定義都必須包含此狀態。

```
{
```

```
    "Type": "Succeed"  
}
```

狀態機器語境

狀態機器語境是唯讀 JSON 文件，其中包含狀態機器在執行期間可用的資料。狀態機器環境只能從狀態機器存取，並包含判斷測試流程的資訊。例如，您可以使用 `userdata.json` 檔案中的測試執行器設定的資訊，判斷是否需要執行特定的測試。

狀態機器環境使用以下格式：

```
{  
    "pool": {  
        <device-json-pool-element>  
    },  
    "userData": {  
        <userdata-json-content>  
    },  
    "config": {  
        <config-json-content>  
    },  
    "suiteFailed": true | false,  
    "specificTestGroups": [  
        "<group-id>"  
    ],  
    "specificTestCases": [  
        "<test-id>"  
    ],  
    "hasExecutionErrors": true  
}
```

pool

針對測試執行所選取的裝置集區相關資訊。對於選取的裝置集區，此資訊是從 `device.json` 檔案中定義的對應最上層裝置集區陣列元素擷取。

userData

檔案中的資訊。`userdata.json`

config

資訊會鎖定 `config.json` 檔案。

suiteFailed

當狀態機器開始時，此值會設為 `false`。如果測試群組失敗處於 RunTask 狀態，則此值在狀態機器執行的剩餘期間設定為 `true`。

specificTestGroups

如果測試執行器選擇要執行的特定測試群組（而非整個測試套件），則會建立此金鑰並包含特定測試群組 IDs 的清單。

specificTestCases

如果測試執行器選取要執行的特定測試案例，而非整個測試套件，則會建立此金鑰並包含特定測試案例 IDs 的清單。

hasExecutionErrors

狀態機器開始時，不退出。如果任何狀態發生執行錯誤，此變數會在狀態機器執行的剩餘期間建立和設定為 `true`。

您可以使用 JSONPath 表示法來查詢上下文。狀態定義中 JSONPath 查詢的語法為 `{$.query}`。您可以使用 JSONPath 查詢做為某些狀態中的預留位置字串。IDT 會使用來自上下文之已評估 JSONPath 查詢的值，取代預留位置字串。您可以使用預留位置做為下列值：

- 狀態下的 TestCases 值。RunTask
- 值 Expression 狀態。Choice

當您從狀態機器上下文存取資料時，請確定符合下列條件：

- JSON 路徑必須以 `$.` 開頭
- 每個值都必須評估為字串、數字或布林值。

如需使用 JSONPath 表示法從上下文存取資料的詳細資訊，請前往 [使用 IDT 環境 \(p. 354\)](#)。

執行錯誤

執行錯誤是狀態機器定義中的錯誤，狀態機器在執行狀態時遇到這些錯誤。IDT 會針對 `test_manager.log` 檔案中每個錯誤來登入，並將 log 訊息串流至主控台。

您可以使用下列方法來處理執行錯誤：

- 在狀態定義中新增 [Catch 區塊 \(p. 344\)](#)。
- 檢查狀態機器環境中 `hasExecutionErrors` 值 ([p. 344](#)) 的值。

Catch

若要使用 Catch，請將下列項目新增至狀態定義中：

```
"Catch": [
  {
    "ErrorEquals": [
      "<error-type>"
    ]
    "Next": "<state-name>"
  }
]
```

如下所述，包含值的所有欄位皆為必要：

`Catch.ErrorEquals`

要擷取的錯誤類型陣列。如果執行錯誤符合其中一個指定的值，則狀態機器會轉換到 `Catch.Next` 中指定的狀態。請查看每個狀態定義，以了解其所發生錯誤類型的相關資訊。

`Catch.Next`

如果目前的狀態遇到與 `Catch.ErrorEquals` 中指定值之一相符的執行錯誤，下一個狀態會轉換成。

快取區塊會循序處理，直到一個配對。如果無錯誤符合 Catch 區塊中列出的錯誤，則狀態機器會繼續執行。因為執行錯誤是因為不正確的狀態定義所造成，建議您在狀態發生執行錯誤時轉換成 Fail 狀態。

hasExecutionError

當某些狀態遇到執行錯誤時，除了發出錯誤，也會在狀態機器環境中將 `hasExecutionError` 值設為 `true`。您可以使用此值來偵測何時發生錯誤，然後使用 Choice 狀態將狀態機器轉換到 Fail 狀態。

這個方法具有下列特性：

- 狀態機器不會以指派給 hasExecutionError 的任何值開始，且除非特定狀態設定該值，否則此值將無法使用。這表示您必須明確將 FallthroughOnError 狀態的 false 設定為 Choice，而 狀態會存取此值，以防止狀態機器在無執行錯誤時停止。
- 一旦設定為 true，hasExecutionError 便永遠不會設定為 false，或從上下文中移除。這表示此值只有在第一次設定為 true 時才有用，並且對於所有後續狀態，不會提供有意義的值。
- 值會與處於 hasExecutionError 狀態的所有分支狀態機器共用，根據其存取順序可能導致未預期的結果。Parallel

由於這些特性，如果可以改用 Catch 區塊，我們不建議您使用此方法。

範例狀態機器

本節提供一些範例狀態機器組態。

範例

- 範例狀態機器：執行單一測試群組 (p. 345)
- 範例狀態機器：執行使用者選取的測試群組 (p. 346)
- 範例狀態機器：執行具備 product 功能的單一測試群組 (p. 347)
- 範例狀態機器：平行執行兩個測試群組 (p. 348)

範例狀態機器：執行單一測試群組

此狀態機器：

- 執行 ID 為 GroupA 的測試群組，該群組必須存在於 group.json 檔案的套件中。
- 檢查執行錯誤，若找到任何 Fail 則轉換為。
- 製作報告，若無錯誤則轉換為 Succeed，否則轉換為 Fail。

```
{  
    "Comment": "Runs a single group and then generates a report.",  
    "StartAt": "RunGroupA",  
    "States": {  
        "RunGroupA": {  
            "Type": "RunTask",  
            "Next": "Report",  
            "TestGroup": "GroupA",  
            "Catch": [  
                {  
                    "ErrorEquals": [  
                        "RunTaskError"  
                    ],  
                    "Next": "Fail"  
                }  
            ]  
        },  
        "Report": {  
            "Type": "Report",  
            "Next": "Succeed",  
            "Catch": [  
                {  
                    "ErrorEquals": [  
                        "ReportError"  
                    ],  
                    "Next": "Fail"  
                }  
            ]  
        }  
    }  
}
```

```
        "Next": "Fail"
    }
]
},
"Succeed": {
    "Type": "Succeed"
},
"Fail": {
    "Type": "Fail"
}
}
```

範例狀態機器：執行使用者選取的測試群組

此狀態機器：

- 檢查測試執行器是否已選取特定的測試群組。狀態機器不會檢查特定測試案例，因為測試執行器無法選取測試案例，而不需要選取測試群組。
- 如果選取測試群組：
 - 執行所選取測試群組中的測試案例。若要這樣做，狀態機器不會明確地指定 RunTask 狀態中的任何測試群組或測試案例。
 - 在執行所有測試和結束之後建立報告。
- 如果未選取測試群組：
 - 執行測試群組中的測試 GroupA。
 - 生成報告和退出。

```
{
    "Comment": "Runs specific groups if the test runner chose to do that, otherwise runs GroupA.",
    "StartAt": "SpecificGroupsCheck",
    "States": {
        "SpecificGroupsCheck": {
            "Type": "Choice",
            "Default": "RunGroupA",
            "FallthroughOnError": true,
            "Choices": [
                {
                    "Expression": "{$.specificTestGroups[0]} != ''",
                    "Next": "RunSpecificGroups"
                }
            ]
        },
        "RunSpecificGroups": {
            "Type": "RunTask",
            "Next": "Report",
            "Catch": [
                {
                    "ErrorEquals": [
                        "RunTaskError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "RunGroupA": {
            "Type": "RunTask",
            "Next": "Report",
            "TestGroup": "GroupA",
            "Catch": [

```

```
{  
    "ErrorEquals": [  
        "RunTaskError"  
    ],  
    "Next": "Fail"  
}  
]  
},  
"Report": {  
    "Type": "Report",  
    "Next": "Succeed",  
    "Catch": [  
        {  
            "ErrorEquals": [  
                "ReportError"  
            ],  
            "Next": "Fail"  
        }  
    ]  
},  
"Succeed": {  
    "Type": "Succeed"  
},  
"Fail": {  
    "Type": "Fail"  
}  
}  
}  
}
```

範例狀態機器：執行具備 product 功能的單一測試群組

此狀態機器：

- 執行測試群組 GroupA。
- 檢查執行錯誤，若找到任何 Fail 則轉換為。
- 將 FeatureThatDependsOnGroupA 功能新增至 awsiotdevicetester_report.xml 檔案：
 - 如果 GroupA 傳遞，功能會設定為 supported。
 - 報告中未將功能標示為選用。
- 製作報告，若無錯誤則轉換為 Succeed，否則轉換為Fail

```
{  
    "Comment": "Runs GroupA and adds product features based on GroupA",  
    "StartAt": "RunGroupA",  
    "States": {  
        "RunGroupA": {  
            "Type": "RunTask",  
            "Next": "AddProductFeatures",  
            "TestGroup": "GroupA",  
            "ResultVar": "GroupA_passed",  
            "Catch": [  
                {  
                    "ErrorEquals": [  
                        "RunTaskError"  
                    ],  
                    "Next": "Fail"  
                }  
            ]  
        },  
        "AddProductFeatures": {  
            "Type": "AddProductFeatures",  
            "Next": "Report",  
            "Catch": [  
                {  
                    "ErrorEquals": [  
                        "RunTaskError"  
                    ],  
                    "Next": "Fail"  
                }  
            ]  
        }  
    }  
}
```

```
"Features": [
    {
        "Feature": "FeatureThatDependsOnGroupA",
        "Groups": [
            "GroupA"
        ],
        "IsRequired": true
    }
],
"Report": {
    "Type": "Report",
    "Next": "Succeed",
    "Catch": [
        {
            "ErrorEquals": [
                "ReportError"
            ],
            "Next": "Fail"
        }
    ]
},
"Succeed": {
    "Type": "Succeed"
},
"Fail": {
    "Type": "Fail"
}
}
```

範例狀態機器：平行執行兩個測試群組

此狀態機器：

- 平行執行 GroupA 和 GroupB 測試群組。由 ResultVar 在分支狀態機器中 RunTask 狀態在上下文中存放的 AddProductFeatures 變數可供 狀態使用。
- 檢查執行錯誤，若找到任何 Fail 則轉換為。此狀態機器不會使用 Catch 區塊，因為該方法不會偵測分支狀態機器中的執行錯誤。
- 根據傳遞的群組，將功能新增至 awsiotdevicetester_report.xml 檔案
 - 如果 GroupA 傳遞，功能會設定為 supported。
 - 報告中未將功能標示為選用。
- 製作報告，若無錯誤則轉換為 Succeed，否則轉換為Fail

如果在裝置集區中設定兩個裝置，則 GroupA 和 GroupB 可同時執行。不過，如果 GroupA 或 GroupB 中有多個測試，則這兩個裝置可能會被配置到那些測試。如果只設定一個裝置，測試群組會循序執行。

```
{
    "Comment": "Runs GroupA and GroupB in parallel",
    "StartAt": "RunGroupAAndB",
    "States": {
        "RunGroupAAndB": {
            "Type": "Parallel",
            "Next": "CheckForErrors",
            "Branches": [
                {
                    "Comment": "Run GroupA state machine",
                    "StartAt": "RunGroupA",
                    "States": {
                        "RunGroupA": {
                            "Type": "Machine"
                        }
                    }
                }
            ]
        }
    }
}
```

```
"Type": "RunTask",
"Next": "Succeed",
"TestGroup": "GroupA",
"ResultVar": "GroupA_passed",
"Catch": [
    {
        "ErrorEquals": [
            "RunTaskError"
        ],
        "Next": "Fail"
    }
],
"Succeed": {
    "Type": "Succeed"
},
"Fail": {
    "Type": "Fail"
}
},
{
    "Comment": "Run GroupB state machine",
    "StartAt": "RunGroupB",
    "States": {
        "RunGroupA": {
            "Type": "RunTask",
            "Next": "Succeed",
            "TestGroup": "GroupB",
            "ResultVar": "GroupB_passed",
            "Catch": [
                {
                    "ErrorEquals": [
                        "RunTaskError"
                    ],
                    "Next": "Fail"
                }
            ],
            "Succeed": {
                "Type": "Succeed"
            },
            "Fail": {
                "Type": "Fail"
            }
        }
    }
},
"CheckForErrors": {
    "Type": "Choice",
    "Default": "AddProductFeatures",
    "FallthroughOnError": true,
    "Choices": [
        {
            "Expression": "{$.hasExecutionErrors} == true",
            "Next": "Fail"
        }
    ]
},
"AddProductFeatures": {
    "Type": "AddProductFeatures",
    "Next": "Report",
    "Features": [
        {
            "Feature": "FeatureThatDependsOnGroupA",

```

```
        "Groups": [
            "GroupA"
        ],
        "IsRequired": true
    },
    {
        "Feature": "FeatureThatDependsOnGroupB",
        "Groups": [
            "GroupB"
        ],
        "IsRequired": true
    }
]
},
"Report": {
    "Type": "Report",
    "Next": "Succeed",
    "Catch": [
        {
            "ErrorEquals": [
                "ReportError"
            ],
            "Next": "Fail"
        }
    ]
},
"Succeed": {
    "Type": "Succeed"
},
"Fail": {
    "Type": "Fail"
}
}
}
```

建立 IDT 測試案例可執行檔

您可以透過下列方式，在測試套件資料夾中建立並放置測試案例可執行檔：

- 針對使用來自 `test.json` 檔案之引數或環境變數來判斷要執行的測試套件，您可以為整個測試套件建立單一測試案例可執行檔，或為測試套件中的每個測試群組建立測試可執行檔。
- 針對您想要根據指定命令執行特定測試的測試套件，您可以為測試套件中的每個測試案例各建立一個測試案例可執行檔。

身為測試撰寫者，您可以判斷哪種方法適合您的使用案例，並相應地建構測試案例可執行檔。請確認 在每個 `test.json` 檔案中提供正確的測試案例可執行檔路徑，並且指定的可執行檔正確執行。

當所有裝置都準備好執行測試案例時，IDT 會讀取以下檔案：

- 所選測試案例的 `test.json` 會判斷要開始的程序和要設定的環境變數。
- 測試套件的 `suite.json` 會判斷要設定的環境變數。

IDT 會根據 `test.json` 檔案中指定的命令和引數，開始所需的測試可取代程序，並將必要的環境變數傳遞到程序。

使用 IDT Client SDK

IDT Client SDKs 可讓您使用 API 命令，簡化在測試可執行檔中撰寫測試邏輯的方式，您可以使用這些命令與 IDT 及待測裝置互動。IDT 目前提供下列SDKs：

- IDT Client SDK for Python
- 適用於 Go 的 IDT Client SDK

這些 SDKs 位於 `<device-tester-extract-location>/sdks` 資料夾。當您建立新的測試案例可執行檔時，您必須將想要使用的開發套件複製到包含測試案例可執行檔的資料夾，並在程式碼中參考軟體開發套件。本節提供可用於測試案例可執行檔之可用 API 命令的簡短描述。

本節內容

- [裝置互動 \(p. 351\)](#)
- [IDT 互動 \(p. 351\)](#)
- [主機互動 \(p. 351\)](#)

裝置互動

以下命令可讓您與待測裝置通訊，而無需實作任何其他裝置互動和連接管理功能。

ExecuteOnDevice

允許測試套件在支援 SSH 或 Docker shell 連接的裝置上執行 shell 命令。

CopyToDevice

允許測試套件從執行 IDT 的主機機器，將本機檔案複製到支援 SSH 或 Docker shell 連接的裝置上的指定位置。

ReadFromDevice

允許測試套件從支援 UART 連接的裝置序列埠讀取。

Note

因為 IDT 不會管理使用裝置存取資訊透過環境對裝置的直接連接，建議您在測試案例可執行檔中使用這些裝置互動 API 命令。不過，如果這些命令不符合您的測試案例需求，您可以從 IDT 環境擷取裝置存取資訊，並用它來從測試套件直接連接到裝置。

若要直接連接，請為待測裝置和資源裝置擷取 `device.connectivity` 和 `resource.devices.connectivity` 欄位中的資訊。如需使用 IDT 環境的詳細資訊，請前往 [使用 IDT 環境 \(p. 354\)](#)。

IDT 互動

以下命令可讓您的測試套件與 IDT 通訊。

PollForNotifications

允許測試套件檢查 IDT 的通知。

GetContextValue 以及 GetContextString

允許測試套件從 IDT 環境擷取值。如需詳細資訊，請參閱 [使用 IDT 環境 \(p. 354\)](#)。

SendResult

允許測試套件向 IDT 報告測試案例結果。此命令必須在測試套件的每個測試案例結尾呼叫。

主機互動

以下命令可讓您的測試套件與主機機器通訊。

PollForNotifications

允許測試套件檢查 IDT 的通知。

GetContextValue 以及 GetContextString

允許測試套件從 IDT 環境擷取值。如需詳細資訊，請參閱 [使用 IDT 環境 \(p. 354\)](#)。

ExecuteOnHost

允許測試套件在本機電腦上執行命令，並讓 IDT 管理測試案例可執行的生命周期。

Enable IDT CLI 命令

命令 IDT CLI 提供數個選項，可讓測試執行器自訂測試執行。run-suite 若要允許測試執行器使用這些選項來執行自訂測試套件，您可以實作 IDT CLI 的支援。如果您不實作支援，測試執行器仍然可執行測試，但某些 CLI 選項無法正確運作。為了提供理想的消費者體驗，建議您在 IDT CLI 中針對 run-suite 命令實作下列引數的支援：

timeout-multiplier

指定大於 1.0 的值，該值會在執行測試時套用至所有逾時。

測試執行器可以使用此引數來增加他們想要執行之測試案例的逾時。當測試執行器在 run-suite 命令中指定此引數時，IDT 會使用此引數來計算 IDT_TEST_TIMEOUT 環境變數的值，並在 IDT 環境中設定 config.timeoutMultiplier 欄位值。若要支援此引數，您必須執行下列作業：

- 讀取 IDT_TEST_TIMEOUT 環境變數，以取得正確的計算逾時值，而不是直接使用 test.json 檔案中的逾時值。
- 從 IDT 環境擷取 config.timeoutMultiplier 值，並將其套用至長時間執行的逾時。

有關由於逾時事件而提早退出的詳細資訊，請前往 [指定離開行為 \(p. 353\)](#)。

stop-on-first-failure

指定 IDT 應在發生故障時停止執行所有測試。

當測試執行器在其 run-suite 命令中指定此引數時，IDT 會在發生失敗時立即停止執行測試。不過，如果測試案例平行執行，則可能會出現未預期的結果。若要實作支援，請確保如果 IDT 遇到此事件，您的測試邏輯會指示所有執行中的測試案例停止、清除暫存資源，並將測試結果回報給 IDT。如需有關提早退出失敗的詳細資訊，請查看 [指定離開行為 \(p. 353\)](#)。

group-id 以及 test-id

指定 IDT 應該只執行選取的測試群組或測試案例。

測試執行器可以使用這些引數搭配其 run-suite 命令，來指定下列測試執行行為：

- 在指定的測試群組中執行所有測試。
- 從指定的測試群組中執行選取測試。

為了支援這些引數，您測試套件的狀態機器必須在狀態機器中包含一組特定的 RunTask 和 Choice 狀態。如果您不使用自訂狀態機器，則預設 IDT 狀態機器會包含必要的狀態，而且您不需要執行額外動作。不過，如果您使用自訂狀態機器，請使用 [範例狀態機器：執行使用者選取的測試群組 \(p. 346\)](#) 做為範例，以新增狀態機器中所需的狀態。

如需 IDT CLI 命令的詳細資訊，請前往 [偵錯和執行自訂測試套件 \(p. 364\)](#)。

寫入事件 log

在測試執行期間，將資料傳送到 stdout 和 stderr 以將事件登入和錯誤訊息寫入主控台。如需主控台訊息格式的資訊，請查看 [主控台訊息格式 \(p. 366\)](#)。

IDT 完成執行測試套件之後，此資訊也可以在位於 `test_manager.log` 資料夾的 `<devicetester-extract-location>/results/<execution-id>/logs` 檔案中取得。

您可以設定每個測試案例來從其測試執行中寫入資料，包括從待測裝置將資料寫入位於 `<group-id>_<test-id>` 資料夾的 `<device-tester-extract-location>/results/<execution-id>/logs` 檔案。若要這樣做，請從 IDT context 使用 IDT 查詢擷取 log 檔案的路徑、在該路徑建立檔案，並寫入您想要的動態資料。`testData.logFilePath`IDT 會根據執行中的測試案例自動更新路徑。如果您選擇不為測試案例建立 log 檔案，則不會為該測試案例建立任何檔案。

您也可以設定文字可執行檔，視需要在 `<device-tester-extract-location>/logs` 資料夾中建立額外的 log 檔案。我們建議您為 log 檔案名稱指定唯一字首，以免覆寫您的檔案。

向 IDT 報告結果

IDT 會將測試結果寫入 `awsiotdevicetester_report.xml` 和 `suite-name_report.xml` 檔案。這些報告檔案位於 `<device-tester-extract-location>/results/<execution-id>/` 中。兩個報告都會從測試套件執行擷取結果。如需 IDT 用於這些報告的結構描述詳細資訊，請前往 [檢視 IDT 測試結果和 Logs \(p. 366\)](#)。

若要填入 `suite-name_report.xml` 檔案，您必須在測試執行完成之前，使用 `SendResult` 命令向 IDT 報告測試結果。如果 IDT 找不到測試的結果，就會發出測試案例的錯誤。以下 Python 範例顯示將測試結果傳送到 IDT 的命令：

```
request-variable = SendResultRequest(TestResult(result))
client.send_result(request-variable)
```

如果您未透過 API 報告結果，IDT 會在測試成品資料夾中尋找測試結果。此資料夾的路徑會存放在 `testData.testArtifactsPath` 中，IDT 環境會存檔。在此資料夾中，IDT 會使用第一個依字母排序的 XML 檔案來找出測試結果。

如果您的測試邏輯得到 JUnit XML 結果，您可以將測試結果寫入成品資料夾中的 XML 檔案，直接將結果提供給 IDT，而不是剖析結果，然後使用 API 將它們提交給 IDT。

如果您使用此方法，請確定您的測試邏輯可準確描述測試結果，並將結果檔案的格式與 `suite-name_report.xml` 檔案的格式相同。IDT 不會針對您提供的資料執行任何驗證，但下列例外：

- IDT 會忽略 `testsuites` 標籤的所有屬性。相反地，它會從其他報告的測試群組結果計算標籤屬性。
- 必須在 `testsuite` 中存在至少一個 `testsuites` 標籤。

由於 IDT 針對所有測試案例使用相同的成品資料夾，而且要在測試執行之間偵測結果檔案，因此如果 IDT 讀取不正確的檔案，此方法也可能會導致錯誤報告。我們建議您在所有測試案例中，對生成的 XML 結果檔案使用相同的名稱，以覆寫每個測試案例的結果，並確保 IDT 可以使用正確的結果。雖然您可以使用混合方式報告測試套件，亦即，針對某些測試案例使用 XML 結果檔案，並透過 API 來提交結果給其他案例，我們不建議此方法。

指定離開行為

將您的文字可執行檔設定為一律以結束代碼 0 結束，即使測試案例報告失敗或錯誤結果。僅使用非零結束代碼，表示測試案例未執行，或測試案例可執行檔無法將任何結果傳達給 IDT。當 IDT 收到非零結束代碼時，會標記測試案例發生錯誤而使測試無法執行。

IDT 可能會在下列事件中，請求或預期測試案例結束執行之前停止執行。使用此資訊來設定您的測試案例可執行檔，以偵測測試案例的每一個事件：

逾時

當測試案例執行的時間超過 `test.json` 檔案中所指定的逾時值時發生。如果測試執行器使用 `timeout-multiplier` 引數來指定逾時乘數，IDT 會使用乘數來計算逾時值。

若要偵測此事件，請使用 IDT_TEST_TIMEOUT 環境變數。當測試執行器推出測試時，IDT 會將 IDT_TEST_TIMEOUT 環境變數的值設為計算逾時值（以秒為單位），並將變數傳遞至測試案例可執行檔。您可以讀取變數值來設定適當的計時器。

中斷

當測試執行器中斷 IDT 時發生。例如，按下 Ctrl+C.

由於終端機會將訊號傳播到所有子處理序，您只需在測試案例中設定訊號處理常式，即可偵測中斷訊號。

或者，您可以定期輪詢 API，來檢查 CancellationRequested API 回應中的 PollForNotifications 布林值。IDT 收到中斷訊號時，會將 CancellationRequested 布林值設為 true。

第一次失敗時停止

當與目前測試案例平行執行的測試案例失敗，且測試執行器使用 stop-on-first-failure 引數來指定 IDT 遇到任何失敗時應停止，便會發生此錯誤。

若要偵測此事件，您可以定期輪詢 API，檢查 CancellationRequested API 回應中 PollForNotifications 布林值。當 IDT 發生故障，且設定為在第一次故障時停止，則將 CancellationRequested 布林值設為 true。

當其中任何事件發生時，IDT 會等待 5 分鐘，讓任何目前執行中測試案例完成執行。如果所有執行中的測試案例不會在 5 分鐘後結束，IDT 會動力每個程序停止。如果在程序結束之前 IDT 尚未收到測試結果，則會將測試案例標記為已逾時。根據最佳實務，您應該確保您的測試案例在遇到其中一個事件時執行下列動作：

1. 停止執行正常的測試邏輯。
2. 清除任何臨時資源，例如待測裝置上的測試成品。
3. 將測試結果報告給 IDT，例如測試失敗或錯誤。
4. 結束。

使用 IDT 環境

IDT 執行測試套件時，測試套件可以存取一組資料，用來判斷每個測試執行的方式。此資料稱為 IDT 環境。例如，userdata.json 檔案中之測試執行器所提供的使用者資料組態，可供 IDT 環境中的測試套件使用。

IDT 上下文可視為唯讀 JSON 文件。測試套件可以使用標準 JSON 資料類型（例如物件、陣列、數字等），從中擷取資料，並將資料寫入上下文。

環境結構描述

IDT 上下文使用下列格式：

```
{  
    "config": {  
        <config-json-content>  
        "timeoutMultiplier": timeout-multiplier  
    },  
    "device": {  
        <device-json-device-element>  
    },  
    "devicePool": {  
        <device-json-pool-element>  
    },  
    "resource": {  
        "devices": [  
            {  
                "name": "Device A",  
                "type": "Sensor",  
                "status": "Normal"  
            },  
            {  
                "name": "Device B",  
                "type": "Actuator",  
                "status": "Standby"  
            }  
        ]  
    }  
}
```

```
        <resource-json-device-element>
        "name": "<resource-name>"
    }
]
},
"testData": {
    "awsCredentials": {
        "awsAccessKeyId": "<access-key-id>",
        "awsSecretAccessKey": "<secret-access-key>",
        "awsSessionToken": "<session-token>"
    },
    "logFilePath": "/path/to/log/file"
},
"userData": {
    <userdata-json-content>
}
}
```

config

來自 [config.json 檔案的資訊。 \(p. 362\)](#) 欄位也包含下列其他欄位： config
config.timeoutMultiplier

測試套件所使用之任何逾時值的乘數。這個值是由 IDT CLI 中的測試執行器指定。預設值為 1。

device

針對測試執行而選取之裝置的相關資訊。此資訊等同於所選裝置在 devices 檔案中的 [device.json 陣列元素。 \(p. 357\)](#)

devicePool

針對測試執行所選取的裝置集區相關資訊。此資訊等同於所選裝置集區的 device.json 檔案中所定義的最上層裝置集區陣列元素。

resource

檔案中資源裝置的相關資訊。 resource.json

resource.devices

此資訊等同於 devices 檔案中定義的 resource.json 陣列。每個 devices 元素都包含下列額外的欄位：

resource.device.name

資源裝置的名稱。此值設定為 requiredResource.name 檔案中的 test.json 值。

testData.awsCredentials

測試用來連接到 AWS 雲端的 AWS 登入資料。此資訊取自 config.json 檔案。

testData.logFilePath

測試案例寫入的 log 檔案路徑。測試套件會建立此檔案（如果該檔案不存在）。

userData

由測試執行器在 [檔案userdata.json中提供的資訊。 \(p. 360\)](#)

存取環境中的資料

您可以從 JSON 檔案和文字可執行檔使用 JSONPath 和 GetContextValue GetContextString 來查詢上下文。APIs 存取 IDT 上下文的字串語法有所不同，如下所示：JSONPath

- 在 suite.json 和 test.json 中，您可以使用 {{**query**}}。亦即，請勿使用根元素 \$. 來開始您的表達式。

- 在 `statemachine.json` 中，您可以使用 `{$.query}`。
- 在 API 命令中，根據命令而定，您可以使用 `query` 或 `{$.query}`。如需詳細資訊，請參閱 SDKs 中的 inline 文件。

下表描述典型 JSONPath 表達式中的運算子：

Operator	Description
<code>\$</code>	The root element. Because the top-level context value for IDT is an object, you will typically use <code>\$.</code> to start your queries.
<code>.childName</code>	Accesses the child element with name <code>childName</code> from an object. If applied to an array, yields a new array with this operator applied to each element. The element name is case sensitive. For example, the query to access the <code>awsRegion</code> value in the <code>##</code> object is <code>\$.config.awsRegion</code> .
<code>[start##end] #####</code>	Filters elements from an array, retrieving items beginning from the <code>start</code> index and going up to the <code>##</code> index, both inclusive.
<code>[index1#index2#... #indexN]</code>	Filters elements from an array, retrieving items from only the specified indices.
<code>[?#expr#]</code>	Filters elements from an array using the <code>expr</code> expression. This expression must evaluate to a boolean value.

若要建立篩選條件表達式，請使用下列語法：

```
<jsonpath> | <value> operator <jsonpath> | <value>
```

在此語法中：

- `jsonpath` 是使用標準 JSON 語法的 JSONPath。
- `value` 是使用標準 JSON 語法的任何自訂值。
- `operator` 是下列其中一個運算子：
 - `<` (小於)
 - `<=` (小於或等於)
 - `==` (等於)

如果您表達式中的 JSONPath 或值為陣列、布林值或物件值，則這是您可以使用的唯一受支援的二進位運算子。

- `>=` (大於或等於)
- `>` (大於)
- `=~` (正規表達式比對)。若要在篩選條件表達式中使用此運算子，您表達式左側的 JSONPath 或值必須評估為字串，而且右側必須是接在 [RE2 語法](#)後面的模式值。

您可以使用 JSONPath 查詢的形式為 `{$.query}` 做為 `args` 檔案之 `environmentVariables` 和 `test.json` 欄位中的預留位置字串，以及 `environmentVariables` 檔案中 `suite.json` 欄位。IDT 會執行上下文查詢，並以查詢的評估值填入欄位。例如，在 `suite.json` 檔案中，您可以使用預留位置字

串來指定每個測試案例變更的環境變數值，IDT 會將每個測試案例的正確值填入環境變數。不過，當您在 test.json 和 suite.json 檔案中使用預留位置字串時，查詢適用下列考量：

- 您必須使用所有小寫，在查詢中每個出現的 devicePool 金鑰。亦即，請改用 devicepool。
- 對於陣列，您只能使用字串陣列。此外，陣列會使用非標準 item1, item2, ..., itemN 格式。如果陣列只包含一個元素，則會序列化為 item，使其與字串欄位無關。
- 您不能使用預留位置從上下文擷取物件。

由於這些考量，建議您盡可能使用 API 來存取測試邏輯中的細節，而非在 test.json 和 suite.json 檔案中的預留位置字串。不過，在某些案例中，使用 JSONPath 預留位置來擷取單一字串設定為環境變數可能會更方便。

設定測試執行器的設定

若要執行自訂測試套件，測試執行器必須依據他們要執行的測試套件來設定其設定。設定是根據 JSON 組態檔案範本（位於 <device-tester-extract-location>/configs/ 資料夾）來指定。如有必要，測試執行器還必須設定 AWS 登入資料，IDT 將用於連接到 AWS 雲端。

做為測試撰寫者，您需要設定這些檔案來[偵錯您的測試套件 \(p. 364\)](#)。您必須提供測試執行器的指示，以便他們可以視需要設定以下設定以執行您的測試套件。

設定 device.json

檔案包含測試執行所在裝置的相關資訊（例如 IP 地址、登入資訊、作業系統和 CPU 架構）。device.json

測試執行器可以使用下列位於 device.json 資料夾中的範本 <device-tester-extract-location>/configs/ 檔案，來提供這項資訊。

```
[  
  {  
    "id": "<pool-id>",  
    "sku": "<pool-sku>",  
    "features": [  
      {  
        "name": "<feature-name>",  
        "value": "<feature-value>",  
        "configs": [  
          {  
            "name": "<config-name>",  
            "value": "<config-value>"  
          }  
        ],  
      }  
    ],  
    "devices": [  
      {  
        "id": "<device-id>",  
        "connectivity": {  
          "protocol": "ssh | uart | docker",  
          // ssh  
          "ip": "<ip-address>",  
          "port": <port-number>,  
          "auth": {  
            "method": "pki | password",  
            "credentials": {  
              "user": "<user-name>",  
              // pki  
              "privKeyPath": "/path/to/private/key",  
            }  
          }  
        }  
      }  
    ]  
  }  
]
```

```
// password
"password": "<password>",

},
),

// uart
"serialPort": "<serial-port>",

// docker
"containerId": "<container-id>",
"containerUser": "<container-user-name>",

}
]
]
```

如下所述，包含值的所有欄位皆為必要：

id

使用者定義的英數字元 ID，可唯一識別裝置的集合，稱為「裝置集區」。屬於同一個集區的裝置必須有相同的硬體。當您執行測試套件，集區中的裝置會用來將工作負載平行化。多個裝置用來執行不同的測試。

sku

可唯一識別測試裝置的英數字元值。SKU 用於追蹤合格的裝置。

Note

如果您想要在 AWS 合作夥伴裝置目錄中列出主機板，在此處指定的 SKU 必須符合您在清單程序中使用的 SKU。

features

選用。包含裝置支援功能的陣列。裝置功能是您在測試套件中設定的使用者定義值。您必須提供測試執行器關於要在 device.json 檔案中包含的功能名稱和值的資訊。例如，如果您想要測試做為 MQTT 伺服器用於其他裝置的裝置，則您可以設定測試邏輯，以驗證名為 MQTT_QOS 功能的特定支援層級。測試執行器提供此功能名稱，並將功能值設定為其裝置支援的 QOS 層級。您可以使用 [查詢來從 \(p. 354\)](#)IDT contextdevicePool.features 摘取提供的資訊，或是從[狀態機器 context \(p. 343\)](#) 摘取 pool.features 查詢。

features.name

功能的名稱。

features.value

支援的功能值。

features.configs

功能的組態設定（若需要）。

features.config.name

組態設定的名稱。

features.config.value

支援的設定值。

devices

要測試之集區中的裝置陣列。至少需要一個裝置。

`devices.id`

使用者定義的唯一識別符，用於識別要測試的裝置。

`connectivity.protocol`

用來與此裝置通訊的通訊協定。集區中的每個裝置都必須使用相同的通訊協定。

目前，唯一支援的值是實體裝置的 `ssh` 和 `uart`，以及 Docker 容器的 `docker`。

`connectivity.ip`

要測試之裝置的 IP 位址。

只有當 `connectivity.protocol` 設為 `ssh` 時，才會套用此屬性。

`connectivity.port`

選用。用於 SSH 連接的連接埠號碼。

預設值為 22。

只有當 `connectivity.protocol` 設為 `ssh` 時，才會套用此屬性。

`connectivity.auth`

連線的驗證資訊。

只有當 `connectivity.protocol` 設為 `ssh` 時，才會套用此屬性。

`connectivity.auth.method`

用來透過指定的連線通訊協定存取裝置的驗證方法。

支援的值如下：

- `pki`
- `password`

`connectivity.auth.credentials`

用於驗證的登入資料。

`connectivity.auth.credentials.password`

用於登入要測試裝置的密碼。

只有當 `connectivity.auth.method` 設為 `password` 時，才會套用此值。

`connectivity.auth.credentials.privKeyPath`

用來登入待測裝置之私有金鑰的完整路徑。

只有當 `connectivity.auth.method` 設為 `pki` 時，才會套用此值。

`connectivity.auth.credentials.user`

登入要測試之裝置的使用者名稱。

`connectivity.serialPort`

選用。裝置連接的序列埠。

只有當 `connectivity.protocol` 設為 `uart` 時，才會套用此屬性。

`connectivity.containerId`

要測試之 Docker 容器的容器 ID 或名稱。

只有當 `connectivity.protocol` 設為 `docker` 時，才會套用此屬性。

connectivity.containerUser

選用。容器中使用者的使用者名稱。預設值是 Dockerfile 中提供的使用者。

預設值為 22。

只有當 connectivity.protocol 設為 docker 時，才會套用此屬性。

Note

若要檢查測試執行器是否設定不正確的測試裝置連接，您可以從狀態機器語境擷取 pool.Devices[0].Connectivity.Protocol，並將其與 Choice 狀態中的預期值進行比較。如果使用不正確的協定，則使用 LogMessage 狀態列印訊息並轉換到 Fail 狀態。或者，您可以使用錯誤處理代碼來報告不正確裝置類型的測試失敗。

(選用) 設定 userdata.json

檔案包含測試套件需要但 userdata.json 檔案中未指定的任何其他資訊。device.json 此檔案的格式視測試套件中定義的 [檔案 userdata_scheme.json 而定。\(p. 335\)](#) 如果您是測試撰寫者，請務必將此資訊提供給將執行您所撰寫測試套件的使用者。

(選用) 設定 resource.json

檔案包含做為資源裝置使用之任何裝置的相關資訊。resource.json 資源裝置是測試待測裝置特定功能所需的裝置。例如，若要測試裝置的藍牙功能，您可以使用資源裝置來測試您的裝置是否可成功連接裝置。資源裝置是選用的，您可以視需要使用任意數量的資源裝置。做為測試撰寫者，您可以使用 [test.json 檔案 \(p. 332\)](#) 來定義測試所需的資源裝置功能。然後測試執行器會使用 resource.json 檔案來提供具有所需功能的資源裝置的集區。請務必將此資訊提供給將執行您編寫之測試套件的使用者。

測試執行器可以使用位於 resource.json 資料夾中的下列範本 <device-tester-extract-location>/configs/ 檔案，來提供這項資訊。

```
[  
  {  
    "id": "<pool-id>",  
    "features": [  
      {  
        "name": "<feature-name>",  
        "version": "<feature-value>",  
        "jobSlots": <job-slots>  
      }  
    ],  
    "devices": [  
      {  
        "id": "<device-id>",  
        "connectivity": {  
          "protocol": "ssh | uart | docker",  
          // ssh  
          "ip": "<ip-address>",  
          "port": <port-number>,  
          "auth": {  
            "method": "pki | password",  
            "credentials": {  
              "user": "<user-name>",  
              // pki  
              "privKeyPath": "/path/to/private/key",  
              // password  
              "password": "<password>",  
            }  
          },  
        }  
      },  
    ]  
  }]
```

```
// uart
"serialPort": "<serial-port>",

// docker
"containerId": "<container-id>",
"containerUser": "<container-user-name>",
}

]

}

]

]
```

如下所述，包含值的所有欄位皆為必要：

id

使用者定義的英數字元 ID，可唯一識別裝置的集合，稱為「裝置集區」。屬於同一個集區的裝置必須有相同的硬體。當您執行測試套件，集區中的裝置會用來將工作負載平行化。多個裝置用來執行不同的測試。

features

選用。包含裝置支援功能的陣列。此欄位所需的資訊是在測試套件的 [test.json 檔案 \(p. 332\)](#) 中定義，並判斷要執行哪些測試以及執行這些測試的方式。如果測試套件不需要任何功能，則此欄位非必要。

features.name

功能的名稱。

features.version

功能版本。

features.jobSlots

設定以指出可以同時使用裝置進行多少次測試。預設值為 1。

devices

要測試之集區中的裝置陣列。至少需要一個裝置。

devices.id

使用者定義的唯一識別符，用於識別要測試的裝置。

connectivity.protocol

用來與此裝置通訊的通訊協定。集區中的每個裝置都必須使用相同的通訊協定。

目前，唯一支援的值是實體裝置的 ssh 和 uart，以及 Docker 容器的 docker。

connectivity.ip

要測試之裝置的 IP 位址。

只有當 connectivity.protocol 設為 ssh 時，才會套用此屬性。

connectivity.port

選用。用於 SSH 連接的連接埠號碼。

預設值為 22。

只有當 connectivity.protocol 設為 ssh 時，才會套用此屬性。

connectivity.auth

連線的驗證資訊。

只有當 connectivity.protocol 設為 ssh 時，才會套用此屬性。

connectivity.auth.method

用來透過指定的連線通訊協定存取裝置的驗證方法。

支援的值如下：

- pki

- password

connectivity.auth.credentials

用於驗證的登入資料。

connectivity.auth.credentials.password

用於登入要測試裝置的密碼。

只有當 connectivity.auth.method 設為 password 時，才會套用此值。

connectivity.auth.credentials.privKeyPath

用來登入待測裝置之私有金鑰的完整路徑。

只有當 connectivity.auth.method 設為 pki 時，才會套用此值。

connectivity.auth.credentials.user

登入要測試之裝置的使用者名稱。

connectivity.serialPort

選用。裝置連接的序列埠。

只有當 connectivity.protocol 設為 uart 時，才會套用此屬性。

connectivity.containerId

要測試之 Docker 容器的容器 ID 或名稱。

只有當 connectivity.protocol 設為 docker 時，才會套用此屬性。

connectivity.containerUser

選用。容器中使用者的使用者名稱。預設值是 Dockerfile 中提供的使用者。

預設值為 22。

只有當 connectivity.protocol 設為 docker 時，才會套用此屬性。

(選用) 設定 config.json

檔案包含 IDT 的組態資訊。config.json一般而言，測試執行器不需要修改此檔案，除非為 IDT 提供其 AWS 使用者登入資料，以及選擇性地提供 AWS 區域。如果提供具有必要許可的 AWS 登入資料，則 AWS IoT Device Tester 會收集並將用量指標提交到 AWS。這是選擇加入功能，可用於改善 IDT 功能。如需詳細資訊，請參閱 [IDT 用量指標 \(p. 370\)](#)。

測試執行器可以透過下列其中一種方式設定其 AWS 登入資料：

- 登入資料檔案

IDT 會使用與 AWS CLI 相同的登入資料檔案。如需詳細資訊，請參閱[組態與登入資料檔案](#)。

登入資料檔案的位置會有所不同，取決於您使用的作業系統：

- macOS , Linux : `~/.aws/credentials`

- Windows : C:\Users\UserName\.aws\credentials
- 環境變數

環境變數是由作業系統維護且由系統命令使用的變數。在 SSH 工作階段期間定義的變數，在工作階段關閉之後將無法使用。IDT 可以使用 AWS_ACCESS_KEY_ID 和 AWS_SECRET_ACCESS_KEY 環境變數來存放 AWS 登入資料

若要在 Linux、macOS 或 Unix 上設定這些變數，請使用 export：

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

若要在 Windows 上設定這些變數，請使用 set：

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

若要設定 IDT 的 AWS 登入資料，測試執行器會編輯位於 auth 資料夾之 config.json 檔案中的 <device-tester-extract-location>/configs/ 區段。

```
{
    "log": {
        "location": "logs"
    },
    "configFiles": {
        "root": "configs",
        "device": "configs/device.json"
    },
    "testPath": "tests",
    "reportPath": "results",
    "awsRegion": "<region>",
    "auth": {
        "method": "file | environment",
        "credentials": {
            "profile": "<profile-name>"
        }
    }
}
```

如下所述，包含值的所有欄位皆為必要：

Note

這個檔案的所有路徑都由相對於 <device-tester-extract-location>.

log.location

中 logs 資料夾的路徑 <device-tester-extract-location>.

configFiles.root

包含組態檔案的資料夾路徑。

configFiles.device

檔案的路徑。device.json

testPath

包含測試套件的資料夾路徑。

reportPath

IDT 執行測試套件之後，資料夾的路徑將包含測試結果。

awsRegion

選用。測試套件將使用的 AWS 區域。如果未設定，則測試套件會使用每個測試套件中指定的預設區域。

auth.method

IDT 用來擷取 AWS 登入資料的方法。支援值為 `file`，可從登入資料檔案擷取登入資料，以及使用環境變數擷取登入資料的 `environment`。

auth.credentials.profile

要從登入資料檔案使用的登入資料描述檔。只有當 `auth.method` 設為 `file` 時，才會套用此屬性。

偵錯和執行自訂測試套件

在設定 [Required configuration \(必要組態\) \(p. 357\)](#) 之後，IDT 即可執行您的測試套件。完整測試套件的執行時間視硬體和測試套件的組成而定。做為參考，在 Raspberry Pi 3B 上完成完整的 FreeRTOS 資格測試套件大約需要 30 分鐘。

當您撰寫測試套件時，可以使用 IDT 在偵錯模式中執行測試套件來檢查您的程式碼，之後再執行，或用來測試執行器。

在偵錯模式中執行 IDT

由於測試套件依存於 IDT 來與裝置互動、提供環境並接收結果，因此您無法直接在 IDE 中偵錯測試套件，無需任何 IDT 互動。若要這樣做，IDT CLI 提供 `debug-test-suite` 命令，可讓您在偵錯模式中執行 IDT。執行以下命令來檢視 `debug-test-suite` 可用的選項：

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

在偵錯模式中執行 IDT 時，IDT 不會實際執行測試套件或執行狀態機器；而是會與您的 IDE 互動，以回應從 IDE 中執行之測試套件所做的請求，並將 Logs 列印至主控台。IDT 不會在逾時，等待手動中斷之後才能結束。在偵錯模式中，IDT 也不會執行狀態機器，也不會建立任何報告檔案。若要為測試套件除錯，您必須使用 IDE 來提供 IDT 通常從組態 JSON 檔案取得的一些資訊。請務必提供下列資訊：

- 每個測試的環境變數和引數。IDT 不會從 `test.json` 或 `suite.json` 讀取此資訊。
- 選取資源裝置的引數。IDT 不會從 `test.json` 讀取此資訊。

若要偵錯您的測試套件，請完成以下步驟：

- 建立執行測試套件所需的設定組態檔案。例如，如果您的測試套件需要 `device.json`、`resource.json` 和 `user data.json`，請確定您已視需要設定所有套件。
- 執行以下命令，將 IDT 置於偵錯模式，並選取執行測試所需的任何裝置。

```
devicetester_[linux | mac | win_x86-64] debug-test-suite [options]
```

執行此命令後，IDT 會等待來自測試套件的請求，然後回應它們。IDT 也會建立 IDT Client SDK 之案例程序所需的環境變數。

- 在您的 IDE 中，使用 `run` 或 `debug` 組態執行下列動作：

- 設定 IDT 發出的環境變數的值。

- b. 設定您在 `test.json` 和 `suite.json` 檔案中指定的任何環境變數或引數的值。
- c. 視需要設定中斷點。
4. 在您的 IDE 中執行測試套件。
您可以視需要偵錯和重新執行測試套件，次數不限。IDT 不會在除錯模式中逾時。
5. 完成偵錯之後，請中斷 IDT 以結束偵錯模式。

執行測試的 IDT CLI 命令

下列各節描述 IDT CLI 命令：

IDT v4.0.1

`help`

列出所指定命令的相關資訊。

`list-groups`

列出指定測試套件中的群組。

`list-suites`

列出可用的測試套件。

`list-supported-products`

列出 IDT 版本支援的版本，在此案例中為 FreeRTOS 版本，而 FreeRTOS 資格測試套件版本適用於目前的 IDT 版本。

`list-test-cases`

列出特定測試群組中的測試案例。支援下列選項：

- `group-id-`。要搜尋的測試群組。此選項為必要選項，且必須指定單一群組。

`run-suite`

在裝置集區上執行測試套件。以下是一些常用選項：

- `suite-id-`。要執行的測試套件版本。如果未指定，IDT 會使用 `tests` 資料夾中的最新版本。
- `group-id-`。要執行的測試群組，以逗號分隔的清單。如果未指定，IDT 會執行測試套件中的所有測試群組。
- `test-id-`。要執行的測試案例，以逗號分隔的清單。指定時，`group-id` 必須指定單一群組。
- `pool-id-`。要測試的裝置集區。測試執行器如果在您的 `device.json` 檔案中定義了多個裝置集區，則必須指定集區。
- `timeout-multiplier-`。設定 IDT 以使用者定義的乘數來修改測試的 `test.json` 檔案中指定的測試執行逾時。
- `stop-on-first-failure-`。將 IDT 設定為在第一次失敗時停止執行。此選項應與 `group-id` 搭配使用以偵錯指定的測試群組。
- `userdata-`。設定檔案，其中包含執行測試套件所需的使用者資料資訊。只有在測試套件的 `userdataRequired` 檔案中將 `suite.json` 設為 `true` 時，才需要此項目。

如需 `run-suite` 選項的詳細資訊，請使用下列 `help` 選項：

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

`debug-test-suite`

在偵錯模式中執行測試套件。如需詳細資訊，請參閱 [在偵錯模式中執行 IDT \(p. 364\)](#)。

檢視 IDT 測試結果和Logs

本節描述 IDT 生成主控台登入和測試報告的格式。

主控台訊息格式

當主控台開始測試套件時，AWS IoT Device Tester 會使用標準格式以列印訊息。以下摘要顯示 IDT 所建立主控台訊息的範例。

```
time="2000-01-02T03:04:05-07:00" level=info msg=Using suite: MyTestSuite_1.0.0
executionId=9a52f362-1227-11eb-86c9-8c8590419f30
```

大多數主控台訊息由下列欄位組成：

time

登入事件的完整 ISO 8601 時間戳記。

level

所登入事件的訊息層級。通常，登入的訊息層級為 info、warn 或 error 其中之一。如果 IDT 遇到可提早結束的預期事件，則會發出 fatal 或 panic 訊息。

msg

登入的訊息。

executionId

目前 IDT 程序的唯一 ID 字串。此 ID 用於區分各個 IDT 執行。

從測試套件衍生的主控台訊息，提供待測裝置以及 IDT 執行之測試套件、測試群組和測試案例的其他資訊。以下摘要顯示從測試套件所衍生的主控台訊息範例。

```
time="2000-01-02T03:04:05-07:00" level=info msg=Hello world! suiteId=MyTestSuite
groupId=myTestGroup testCaseId=myTestCase deviceId=my-device
executionId=9a52f362-1227-11eb-86c9-8c8590419f30
```

主控台訊息的特定測試套件部分包含下列欄位：

suiteId

目前正在執行之測試套件的名稱。

groupId

目前執行中測試群組的 ID。

testCaseId

目前執行中之測試案例的 ID。

deviceId

測試目前測試案例正在使用的裝置 ID。

若要在 IDT 完成執行測試時，將測試摘要列印至主控台，您必須在狀態機器中包含 [狀態Report](#)。[\(p. 341\)](#)測試摘要包含測試套件的相關資訊、執行每個群組的測試結果，以及生成的登入和報告檔案的位置。以下範例顯示測試摘要訊息。

```
===== Test Summary =====
```

```
Execution Time:      5m00s
Tests Completed:    4
Tests Passed:       3
Tests Failed:       1
Tests Skipped:      0
-----
Test Groups:
  GroupA:          PASSED
  GroupB:          FAILED
-----
Failed Tests:
  Group Name: GroupB
    Test Name: TestB1
      Reason: Something bad happened
-----
Path to AWS IoT Device Tester Report: /path/to/awsiotdevicetester_report.xml
Path to Test Execution Logs: /path/to/logs
Path to Aggregated JUnit Report: /path/to/MyTestSuite_Report.xml
```

AWS IoT Device Tester 報告結構描述

`awsiotdevicetester_report.xml` 是一個已簽署的報告，其中包含以下資訊：

- IDT 版本。
- 測試套件版本。
- 用來簽署報告的報告簽章和索引鍵。
- 檔案中指定的裝置 SKU 和裝置集區名稱。`device.json`
- 已測試的 product 版本和裝置功能。
- 測試結果的彙總摘要。此資訊與 `suite-name_report.xml` 檔案中所包含的資訊相同。

```
<apnreport>
  <awsiotdevicetesterversion>idt-version</awsiotdevicetesterversion>
  <testsuiteversion>test-suite-version</testsuiteversion>
  <signature>signature</signature>
  <keyname>keyname</keyname>
  <session>
    <testsession>execution-id</testsession>
    <starttime>start-time</starttime>
    <endtime>end-time</endtime>
  </session>
  <awsproduct>
    <name>product-name</name>
    <version>product-version</version>
    <features>
      <feature name="feature-name" value="supported | not-supported | feature-value" type="optional | required"/>
    </features>
  </awsproduct>
  <device>
    <sku>device-sku</sku>
    <name>device-name</name>
    <features>
      <feature name="feature-name" value="feature-value" />
    </features>
    <executionMethod>ssh | uart | docker</executionMethod>
  </device>
  <devenvironment>
    <os name="os-name" />
  </devenvironment>
</report>
```

```
<suite-name-report-contents>
</report>
</apnreport>
```

awsiotdevicetester_report.xml 檔案包含 **<awsproduct>** 標籤，其中包含關於受測產品和經過一系列測試驗證後之產品功能的資訊。

<awsproduct> 標籤中使用的屬性

name

受測產品名稱。

version

受測產品版本。

features

驗證的功能。標記為 **required** 的功能為測試套件驗證裝置所需要的項目。以下片段顯示此資訊如何出現在 awsiotdevicetester_report.xml 檔案中。

```
<feature name="ssh" value="supported" type="required"></feature>
```

標記為 **optional** 的功能不需要進行驗證。以下程式碼片段顯示選用功能。

```
<feature name="hs1" value="supported" type="optional"></feature>
<feature name="mqtt" value="not-supported" type="optional"></feature>
```

測試套件報告結構描述

報告為 **suite-name_Result.xml** XML 格式。[JUnit](#)您可以將它整合到持續整合和部署平台，例如 [Jenkins](#)、[Bamboo](#) 等。此報告包含測試結果的彙總摘要。

```
<testsuites name="<suite-name>" results="<run-duration>" tests="<number-of-test>" failures="<number-of-tests>" skipped="<number-of-tests>" errors="<number-of-tests>" disabled="0">
    <testsuite name="<test-group-id>" package="" tests="<number-of-tests>" failures="<number-of-tests>" skipped="<number-of-tests>" errors="<number-of-tests>" disabled="0">
        <!--success-->
        < testcase classname="<classname>" name="<name>" time="<run-duration>" />
        <!--failure-->
        < testcase classname="<classname>" name="<name>" time="<run-duration>" >
            < failure type="<failure-type>" >
                reason
            </ failure >
        </ testcase >
        <!--skipped-->
        < testcase classname="<classname>" name="<name>" time="<run-duration>" >
            < skipped >
                reason
            </ skipped >
        </ testcase >
        <!--error-->
        < testcase classname="<classname>" name="<name>" time="<run-duration>" >
            < error >
                reason
            </ error >
        </ testcase >
```

```
</testsuite>  
</testsuites>
```

或 `awsiotdevicetester_report.xml` 中的報告區段會列出已執行的測試和結果。`suite-name_report.xml`

第一個 XML 標籤 `<testsuites>` 包含測試執行的摘要。例如：

```
<testsuites name="MyTestSuite results" time="2299" tests="28" failures="0" errors="0"  
disabled="0">
```

<testsuites> 標籤中使用的屬性

name

測試套件的名稱。

time

執行測試套件所花費的時間（以秒為單位）。

tests

執行的測試次數。

failures

已執行但未通過的測試次數。

errors

IDT 無法執行的測試次數。

disabled

此屬性未使用，可忽略。

如果測試發生失敗或錯誤，您可以檢閱 `<testsuites>` XML 標籤來識別失敗的測試。`<testsuites>` 標籤內的 `<testsuite>` XML 標籤會顯示測試群組的測試結果摘要。例如：

```
<testsuite name="combination" package="" tests="1" failures="0" time="161" disabled="0"  
errors="0" skipped="0">
```

其格式類似於 `<testsuites>` 標籤，但有不使用且可忽略的 `skipped` 屬性。在每個 `<testsuite>` XML 標籤內，測試群組每個執行的測試都有 `<testcase>` 標籤。例如：

```
<testcase classname="Security Test" name="IP Change Tests" attempts="1"></testcase>
```

<testcase> 標籤中使用的屬性

name

測試的名稱。

attempts

IDT 執行測試案例的次數。

當測試案例失敗或發生錯誤時，系統就會將 `<failure>` 或 `<error>` 標籤新增至 `<testcase>` 標籤，其中附有相關資訊以利故障診斷。例如：

```
<testcase classname="mcu.Full_MQTT" name="MQTT_TestCase" attempts="1">
<failure type="Failure">Reason for the test failure</failure>
<error>Reason for the test execution error</error>
</testcase>
```

IDT 用量指標

如果您提供具有必要許可的 AWS 登入資料，則 AWS IoT Device Tester 會收集用量指標並將其提交至 AWS。這是選擇加入功能，可用於改善 IDT 功能。IDT 收集的資訊如下：

- 用來執行 IDT 的 AWS 帳號 ID
- 用於執行測試的 IDT CLI 命令
- 執行的測試套件
- 中的測試套件 *<device-tester-extract-location>* folder
- 在裝置集區中設定的裝置數量
- 測試案例名稱和執行時間
- 測試結果資訊，例如是否通過、失敗、遇到錯誤或略過
- 測試的 Product 特徵
- IDT 結束行為，例如未預期或提早退出

IDT 傳送的所有資訊也會登入 metrics.log 資料夾中的 *<device-tester-extract-location>/results/<execution-id>/* 檔案。您可以檢視 log 檔案，以查看在測試執行期間收集的資訊。這個檔案只有在您選擇收集用量指標時才會建立。

若要停用指標集合，您不需要執行其他動作。只要不存放您的 AWS 登入資料，而且如果您有存放的 AWS 登入資料，請不要設定 config.json 檔案來存取登入資料。

設定您的 AWS 登入資料

如果您尚未擁有 AWS 帳號，您必須[建立一個 \(p. 370\)](#)。如果您已經有 AWS 帳號，您只需要為允許 IDT 代您傳送用量指標至 [的帳號](#)，[\(p. 370\)](#)設定必要許可AWS。

步驟 1：建立 AWS 帳戶

在此步驟中，建立和設定 AWS 帳戶。如果您已擁有 AWS 帳戶，請跳到 [the section called “步驟 2：設定 IDT 的許可” \(p. 370\)](#)。

1. 開啟 [AWS 首頁](#)，接著選擇 Create an AWS Account (建立 AWS 帳戶)。

Note

如果您最近有登入 AWS，您看到的可能會是 Sign In to the Console (登入主控台)。

2. 請遵循線上指示進行。註冊程序的一部分包括註冊信用卡、接收文字簡訊或電話，以及輸入 PIN。

如需詳細資訊，請參閱[如何建立和啟用新的 Amazon Web Services 帳戶](#)？

步驟 2：設定 IDT 的許可

在此步驟中，設定 IDT 用來執行測試和收集 IDT 用量資料的許可。您可以使用 AWS 管理主控台 或 AWS Command Line Interface (AWS CLI) 建立 IAM 政策和 IDT 使用者，然後將政策連接到使用者。

- [設定 IDT \(主控台\) 的許可 \(p. 371\)](#)

- [步驟 2：設定 IDT \(AWS CLI\) 的許可 \(p. 371\)](#)

設定 IDT (主控台) 的許可

請依照下列步驟使用主控台來設定 IDT for FreeRTOS 的許可。

1. 登入 [IAM 主控台](#)。
2. 建立客戶受管政策，該政策授與建立具有特定許可之角色的許可。
 - a. 在導覽窗格中，選擇 Policies (政策)，然後選擇 Create policy (建立政策)。
 - b. 在 JSON 標籤上，用以下政策取代預留位置內容。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "iot-device-tester:SendMetrics"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```
 - c. 選擇 Review policy (檢閱政策)。
 - d. 針對 Name (名稱)，輸入 **IDTUsageMetricsIAMPermissions**。在 Summary (摘要) 下，檢視政策授予的許可。
 - e. 選擇 Create policy (建立政策)。
3. 建立 IAM 使用者並將許可連接到使用者。
 - a. 建立 IAM 使用者。請遵循[建立 IAM 使用者（主控台）](#)中的步驟 1 到 5IAM 使用者指南。如果您已經建立 IAM 使用者，請跳至下一個步驟。
 - b. 將許可連接到您的 IAM 使用者：
 - i. 在 Set permissions (設定許可) 頁面上，選擇 Attach existing policies directly (直接連接現有的政策)。
 - ii. 搜尋您在上一個步驟中建立的 **IDTUsageMetricsIAMPermissions** 政策。選取核取方塊。
 - c. 選擇 Next : (下一步 :)。標籤。
 - d. 選擇 Next : (下一步 :)。檢視以檢視選項的摘要。
 - e. 選擇 Create user (建立使用者)。
 - f. 若要檢視使用者的存取金鑰 (存取金鑰 IDs 和私密存取金鑰)，請選擇密碼和存取金鑰旁的 Show (顯示)。若要儲存存取金鑰，請選擇 Download.csv，並將檔案儲存到安全的位置。您稍後可以使用此資訊來設定您的 AWS 登入資料檔案。

步驟 2：設定 IDT (AWS CLI) 的許可

請依照下列步驟使用 AWS CLI 來設定 IDT for FreeRTOS 的許可。

1. 如果尚未安裝 AWS CLI，請在電腦上安裝並設定。依照 [AWS CLI 中安裝](#) 的步驟執行。AWS Command Line Interface 使用者指南

Note

AWS CLI 是開放原始碼工具，您可從命令列殼層中用它來與 AWS 服務互動。

2. 建立下列 customer 受管政策，其會授予管理 IDT 和 FreeRTOS 角色的許可。

Linux, macOS, or Unix

```
aws iam create-policy --policy-name IDTUsageMetricsIAMPermissions --policy-document
'{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot-device-tester:SendMetrics"
      ],
      "Resource": "*"
    }
  ]
}'
```

Windows command prompt

```
aws iam create-policy --policy-name IDTUsageMetricsIAMPermissions --policy-document
'{"Version": "2012-10-17", "Statement": [{"Effect": "Allow", "Action": ["iot-device-tester:SendMetrics"], "Resource": "*"}]}'
```

Note

此步驟包含一個 Windows 命令提示字元範例，因為它使用的 JSON 語法與 Linux、macOS 或 Unix 終端機命令不同。

3. 建立 IAM 使用者，並連接 IDT for FreeRTOS 所需的許可。

- a. 建立 IAM 使用者。

```
aws iam create-user --user-name user-name
```

- b. 將您建立的 IDTUsageMetricsIAMPermissions 政策連接到您的 IAM 使用者。Replace *user-name* 取代為您的 IAM 使用者名稱，以及 <*account-id*>，位於具有 AWS 帳號 ID 的命令中。

```
aws iam attach-user-policy --user-name user-name --policy-arn
arn:aws:iam::<account-id>:policy/IDTFreeRTOSIAMPermissions
```

4. 為使用者建立私密存取金鑰。

```
aws iam create-access-key --user-name user-name
```

將輸出儲存在安全的位置。您稍後可以使用此資訊來設定 AWS 登入資料檔案。

將 AWS 登入資料提供給 IDT

若要允許 IDT 存取您的 AWS 登入資料並將指標提交給 AWS，請執行下列動作：

1. 存放 AWS 使用者的 IAM 登入資料做為環境變數或登入資料檔案：

- a. 若要使用環境變數，請執行下列命令：

```
AWS_ACCESS_KEY_ID=access-key
AWS_SECRET_ACCESS_KEY=secret-access-key
```

- b. 若要使用登入資料檔案，請將以下資訊新增至 .aws/credentials file::。

```
[profile-name]
aws_access_key_id=access-key
aws_secret_access_key=secret-access-key
```

2. 設定 auth 檔案的 config.json 區段。如需詳細資訊，請參閱（選用）設定 config.json (p. 362)。

Troubleshooting

每個測試套件執行都有唯一的執行 ID，用於 results 目錄中建立名為 results/*execution-id* 的資料夾。個別測試群組日誌位於 results/*execution-id*/logs 目錄下。使用 IDT for FreeRTOS 主控台輸出來尋找測試案例的執行 ID、測試案例 ID 和測試群組 ID 失敗，然後針對名為 results/*execution-id*/logs/*test_group_id*_*test_case_id*.log 的測試案例打開 log 檔案。此檔案中的資訊包括：

- 完整的建置和刷入命令輸出。
- 測試執行輸出。
- 更多 IDT for FreeRTOS 主控台的詳細輸出。

我們建議您採用以下工作流程來進行故障診斷：

1. 如果您看到錯誤「*user/role* 未獲授權存取此資源，請確定您設定 建立並設定 AWS 帳戶 (p. 296) 中指定的許可。
2. 讀取主控台輸出以找出相關資訊，如執行 UUID 和目前執行中的任務。
3. 在 FRO_Report.xml 檔案中查看每個測試的錯誤陳述式。此目錄會包含每個測試群組的執行日誌。
4. 查看 /results/*execution-id*/logs 下方的 log 檔案。
5. 調查下列其中一個問題領域：
 - 裝置組態，例如 /configs/ 資料夾中的 JSON 組態檔案。
 - 裝置界面。您可以查看日誌以判斷故障的界面。
 - 裝置工具。請確認用來建置和刷新裝置的工具鏈皆已正確安裝與設定。
 - 請確定您具有清晰、複製的 FreeRTOS 原始程式碼版本。FreeRTOS 版本已根據 FreeRTOS 版本標記。若要複製特定版本的程式碼，請使用下列命令：

```
git clone --branch version-number https://github.com/aws/amazon-freertos.git
cd amazon-freertos
git submodule update --checkout --init --recursive
```

故障診斷裝置組態

使用 IDT for FreeRTOS 時，您必須備妥正確的組態檔案，才能執行二進位檔。如果您遇到剖析和組態錯誤，第一步應該是找到適合環境的組態範本並加以運用。這些範本皆位於 *IDT_ROOT*/configs 目錄中。

如果您仍然有問題，請參閱下列除錯程序。

查詢位置

首先，請讀取主控台輸出以找出相關資訊，例如本文件中做為 execution-id 參考的執行 UUID。

接著，在 `/results/execution-id` 目錄中查看 `FRQ_Report.xml` 檔案。此檔案包含已執行的所有測試案例，以及每次失敗的錯誤片段。若要取得所有的執行日誌，請尋找每個測試案例的 `/results/execution-id/logs/test_group_id_test_case_id.log` 檔案。

IDT 錯誤代碼

下表說明由 IDT for FreeRTOS 產生的錯誤代碼：

錯誤代碼	錯誤代碼名稱	可能的根本原因	故障診斷
201	InvalidInputError	device.json、config.json 或 userdata.json 中的欄位遺失或格式不正確。	請確定列出的檔案中未遺漏必要欄位且為必要格式。如需詳細資訊，請參閱 首次準備測試微型控制器主機板 (p. 298) 。
202	ValidationError	device.json、config.json 或 userdata.json 中的欄位包含無效的值。	請查看報告中錯誤代碼右方的錯誤訊息： <ul style="list-style-type: none"> 無效的 AWS 區域 - 請在 config.json 檔案中指定有效的 AWS 區域。如需 AWS 區域的詳細資訊，請參閱 區域和端點。 無效的 AWS 登入資料 - 在您的測試機器設定有效的 AWS 登入資料 (透過環境變數或登入資料檔案)。確認身份驗證欄位設定正確。如需詳細資訊，請參閱 建立並設定 AWS 帳戶 (p. 296)。
203	CopySourceCodeError	無法將 FreeRTOS 來源碼複製到指定的目錄。	請確認下列項目： <ul style="list-style-type: none"> 檢查有效的 <code>sourcePath</code> 在您指定的 <code>userdata.json</code> 檔案中。 刪除 FreeRTOS 原始碼目錄下的 <code>build</code> 資料夾 (如果存在)。如需詳細資訊，請參閱 設定建置、刷新和測試設定 (p. 302)。
204	BuildSourceError	無法編譯 FreeRTOS 來源碼。	請確認下列項目： <ul style="list-style-type: none"> 檢查您 <code>userdata.json</code> 檔案底下的 <code>buildTool</code> 資訊是否正確。

錯誤代碼	錯誤代碼名稱	可能的根本原因	故障診斷
			<ul style="list-style-type: none"> 如果您使用 <code>cmake</code> 做為建置工具，請確定 <code>enableTests</code> 已指定在 <code>buildTool</code> 命令中。如需詳細資訊，請參閱 設定建置、刷新和測試設定 (p. 302)。 如果您已將 IDT for FreeRTOS 擷取到包含空格的系統上的檔案路徑（例如 <code>c:\Users\My Name\Desktop\</code>），則您可能需要建置命令中額外的引號，以確保路徑已正確剖析。插入命令也可能需要相同的操作。
205	FlashOrRunTestError	IDT FreeRTOS 無法在 DUT 上刷新或執行 FreeRTOS。	確認您 <code>userdata.json</code> 檔案底下的 <code>flashTool</code> 資訊是否正確。如需詳細資訊，請參閱 設定建置、刷新和測試設定 (p. 302) 。
206	StartEchoServerError	IDT FreeRTOS 無法開始 WiFi 或安全通訊端測試的 echo 伺服器。	請驗證防火牆或網路設定未使用或封鎖設定在 <code>userdata.json</code> 檔案 <code>echoServerConfiguration</code> 項下的連接埠。

偵錯剖析錯誤

JSON 組態中的錯字有時會導致剖析錯誤。在大部分的情況下，問題是出在 JSON 檔案中省略了括弧、逗號或引號。IDT for FreeRTOS 會執行 JSON 驗證作業並列印除錯資訊。該工具會印出發生錯誤的行、行號和語法錯誤的欄號。此資訊應足以協助您修正錯誤，但如果尚未找到錯誤，則可以在 IDE 中手動執行驗證、文字編輯器（例如 Atom 或 Sublime），或透過 JSONLint 等網路工具執行驗證。

偵錯完整性檢查失敗

當您執行 FreeRTOSIntegrity 測試群組且遇到失敗時，請先確定您尚未修改任何 `freertos` 目標檔案。如果尚未看到問題，但仍然遇到問題，請確保您使用的是正確的分支。如果您執行 IDT 的 `list-supported-products` 命令，您可以找到您應該使用之 `freertos` 庫的哪些標記分支。

如果您複製了 `freertos` 庫的正確標記分支，但仍發生問題，請確保您也已執行 `submodule update` 命令。儲存庫的複製工作流程如下。`freertos`

```
git clone --branch version-number https://github.com/aws/amazon-freertos.git
cd amazon-freertos
git submodule update --checkout -init -recursive
```

完整性檢查程式尋找的檔案清單位於 `checksums.json` 資料夾中的 `freertos` 檔案。若要限定 FreeRTOS 連接埠，而無須修改檔案及資料夾結構，請確定 `exhaustive` 檔案的 'minimal' 和 'checksums.json' 部分中列出的檔案均已修改。若要以 SDK 設定執行，請確認「minimal」區段下的檔案均已修改。

如果您以 開發套件執行 IDT 且已在 `freertos` 中修改一些檔案，則請確定您在 `userdata` 檔案中正確設定軟體開發套件。否則，完整性檢查程式會驗證 `freertos` 中的所有檔案。

偵錯 FullWiFi 測試群組失敗

如果您在 FullWiFi 測試群組中看到故障，且「`AFQP_WiFiConnectMultipleAP`」測試失敗，則此原因可能是這兩個存取點不在執行 IDT 的主機電腦之相同子網路中。確定兩個存取點位於與執行 IDT 的主機電腦相同的子網路中。

偵錯必要參數遺漏錯誤

IDT for FreeRTOS 中加入了新功能，因此您可能需要變更組態檔案，使用舊的組態檔案可能會破壞組態。如果發生這種情況，`results/execution-id/logs` 目錄下的 `test_group_id_test_case_id.log` 檔案會明確列出所有遺漏的參數。IDT for FreeRTOS 會驗證 JSON 組態檔案結構描述，藉此確保您使用的是最新支援的版本。

偵錯「無法開始測試」錯誤

可能會出現錯誤，指向測試開始期間發生的失敗。由於這類錯誤有多種可能原因，請務必檢查下列領域的正確性：

- 確定您在執行命令中加入的集區名稱實際存在。系統會直接參考您的 `device.json` 檔案。
- 確保集區中的一或多個裝置都有正確的組態參數。

偵錯「未授權存取資源」錯誤

您可能會看到錯誤「`user/role` 未獲授權存取此資源」，其位於終端機輸出或 `test_manager.log` 下的 `/results/execution-id/logs` 檔案中。若要解析此問題，請將 `AWSIoTDeviceTesterForFreeRTOSFullAccess` 受管政策連接到測試使用者。如需詳細資訊，請參閱 [建立並設定 AWS 帳戶 \(p. 296\)](#)。

偵錯網路測試錯誤

進行以網路為基礎的測試時，IDT 會在主機機器上啟動連結至非預留連接埠的 echo 伺服器。如果您因為 WiFi 或安全通訊端測試中的逾時或無法使用連接而發生錯誤，請確定網路已設為允許流量傳到 1024 - 49151 範圍的連接埠。

根據預設，安全通訊端測試會使用連接埠 33333 和 33334。根據預設，WiFi 測試會使用連接埠 33335。如果這三個連接埠都為防火牆或網路使用或封鎖，您可以在 `userdata.json` 中選擇不同的連接埠以供測試。如需詳細資訊，請參閱 [設定建置、刷新和測試設定 \(p. 302\)](#)。您可以使用下列命令檢查某個特定的連接埠是否正在使用中：

- Windows : `netsh advfirewall firewall show rule name=all | grep port`
- Linux : `sudo netstat -pan | grep port`
- macOS: `netstat -nat | grep port`

由於相同版本有效承載，OTA 更新失敗

如果 OTA 測試案例在執行 OTA 之後，由於裝置上有相同版本而失敗，則原因可能是您的建置系統（例如 `cmake`）沒有注意到 IDT 對 FreeRTOS 原始碼的變更，且未構置已更新的二進位檔。這會導致 OTA 與目前

位於裝置上的相同二進位檔搭配執行，因而測試失敗。若要疑難排解 OTA 更新失敗，請先確定您使用的是建置系統的最新支援版本。

測試案例上的 OTA PresignedUrlExpired 測試失敗

此測試的其中一個先決條件是 OTA 更新時間應該超過 60 秒，否則測試將會失敗。如果發生此問題，可在 log 中找到以下錯誤訊息：「測試時間少於 60 秒（URL 過期時間）才能完成。請聯絡我們。」

偵錯裝置界面和連接埠錯誤

本節包含 IDT 用來連接至裝置的裝置界面相關資訊。

支援的平台

IDT 支援 Linux、macOS 和 Windows。這三個平台對與其連接的序列裝置有不同的命名機制：

- Linux : /dev/tty*
- macOS : /dev/tty.* 或 /dev/cu.*
- Windows : COM* (COM)

檢查裝置 ID：

- 若為 Linux/macOS，請開啟終端機並執行 ls /dev/tty*。
- 若為 macOS，請打開終端機並執行 ls /dev/tty.* 或 ls /dev/cu.*。
- 若為 Windows，請開啟裝置管理員並展開序列裝置群組。

驗證連接至連接埠的裝置：

- 對於 Linux，請確定已安裝 udev 套件，然後執行 udevadm info -name=PORT。此公用程式會印出裝置驅動程式資訊，可協助您驗證使用的連接埠是否正確。
- 若為 macOS，請打開 Launchpad 並搜尋 **System Information**。
- 若為 Windows，請開啟裝置管理員並展開序列裝置群組。

裝置界面

每個內嵌裝置均不同，這表示它們可能有一或多個序列埠。裝置與機器連結時有兩個連接埠是很常見的情況，

- 其中一個是用來刷新裝置的資料連接埠，
- 另一個則是可讀取輸出的讀取連接埠。

請務必在 device.json 檔案中設定正確的讀取連接埠。否則，系統可能無法讀取來自裝置的輸出。

在有多個連接埠的情況下，您應在 device.json 檔案中使用裝置的讀取連接埠。例如，如果您插入 Espressif WROVER 裝置，且指派給它的兩個連接埠為 /dev/ttUSB0 和 /dev/ttUSB1，請在 /dev/ttUSB1 檔案中使用 device.json。

使用 Windows 時，也是遵循相同的邏輯操作。

讀取裝置資料

IDT for FreeRTOS 會使用個別裝置建置和刷新工具來指定連接埠組態。如果您正在測試裝置但沒有取得輸出，可以嘗試使用下列預設設定：

- 傳輸速率：115200
- 資料位元：8
- 同位：無
- 停止位元：1
- 流量控制：無

IDT for FreeRTOS 會負責處理這些設定，因此您不需要自行設定。不過，您可以使用相同方法來手動讀取裝置輸出。在 Linux 或 macOS 上，您可以使用 screen 命令來執行此操作。在 Windows 上，您可以使用 TeraTerm 之類的程式。

```
Screen: screen /dev/cu.usbserial 115200
```

```
TeraTerm: Use the above-provided settings to set the fields explicitly in the GUI.
```

開發工具鏈問題

本節討論您的工具鏈可能發生的問題。

Ubuntu 上的 Code Composer Studio

Ubuntu 較新版本 (17.10 和 18.04) 具備的 glibc 套件版本與 Code Composer Studio 7.x 版並不相容，建議您安裝 Code Composer Studio 8.2 版或更新版本。

不相容的症狀可能包括：

- FreeRTOS 無法對您的裝置建置或刷入。
- Code Composer Studio 安裝程式可能會凍結。
- 執行建置或刷新程序期間，主控台中不會顯示任何日誌輸出。
- 即使是在無界面模式中叫用建置命令，該命令仍會嘗試以 GUI 模式啟動。

Logging

系統會將 IDT for FreeRTOS 日誌放在單一位置。從根 IDT 目錄中，這些檔案可在 results/*execution-id*/ 下取得：

- FRO_Report.xml
- awsiotdevicetester_report.xml
- logs/*test_group_id*_*test_case_id*.log

FRO_Report.xml 和 logs/*test_group_id*_*test_case_id*.log 是要檢查的最重要日誌。FRO_Report.xml 包含有關失敗測試案例並顯示特定錯誤訊息的資訊。然後，您可以使用 logs/*test_group_id*_*test_case_id*.log 進一步挖掘問題，深入了解來龍去脈。

主控台錯誤

當 AWS IoT Device Tester 執行時，失敗會以簡短訊息回報到主控台。您可以在 results/*execution-id*/logs/*test_group_id*_*test_case_id*.log 中查看該內容，以進一步了解錯誤。

日誌錯誤

每個測試套件執行都有一個唯一的執行 ID，用來建立名為 results/*execution-id* 的資料夾。但每個測試案例的字號都會放在 results/*execution-id*/logs 下方。使用 IDT for FreeRTOS 主控台的輸出來查

找失敗測試案例的執行 ID、測試案例 ID 和測試群組 ID。接著，使用這項資訊來尋找並開啟該測試案例名為 `results/execution-id/logs/test_group_id_test_case_id.log` 的日誌檔案。這個檔案中的資訊包括完整的建置和刷入命令輸出、測試執行輸出，以及更詳細的 AWS IoT Device Tester 主控台輸出。

S3 儲存貯體問題

如果按下 CTRL+C 在執行 IDT 時，IDT 將開始清理程序。清除作業的一部分是移除 IDT 測試過程中建立的 Amazon S3 資源。如果清除無法完成，您可能會發生問題，而您有太多 Amazon S3 儲存貯體已建立。這表示下次執行 IDT 時測試將開始失敗。

如果按下 CTRL+C 若要停止 IDT，您必須讓 IDT 完成清理程序，以避免發生此問題。您也可以從手動建立的 Amazon S3 儲存貯體中移除 儲存貯體。

故障診斷逾時錯誤

如果在執行測試套件時出現逾時錯誤，請指定逾時乘數係數增加逾時。此係數會套用至預設的逾時值。此旗標設定的任何值必須大於或等於 1.0。若要使用逾時乘數，請在執行測試套件時使用標記 `--timeout-multiplier`。

Example

IDT v3.0.0 and later

```
./devicetester_linux run-suite --suite-id FRO_1.0.1 --pool-id DevicePool1 --timeout-multiplier 2.5
```

IDT v1.7.0 and earlier

```
./devicetester_linux run-suite --suite-id FRO_1 --pool-id DevicePool1 --timeout-multiplier 2.5
```

行動功能和 AWS 費用

當 Cellular 功能在您的 Yes 檔案中設定為 `device.JSON` 時，FullSecureSockets 將使用 t.micro EC2 執行個體來執行測試，這可能會導致您的 AWS 帳號增加成本。如需詳細資訊，請查看 [Amazon EC2 定價](#)。

AWS IoT Device Tester for FreeRTOS 的支援政策

適用於 AWS IoT 的 FreeRTOS Device Tester 是一種測試自動化工具，可驗證您的 [裝置是否有資格](#) FreeRTOS 納入 [AWS Partner Device Catalog](#) 中。建議您使用最新版本的 FreeRTOS 和 AWS IoT Device Tester，來測試您的裝置是否有資格。我們支援 AWS IoT Device Tester 可用於最新版本的 FreeRTOS，以及前六個月內發行的 FreeRTOS 版本。支援以 LTS 為基礎之 AWS IoT Device Tester 的 FreeRTOS 版本支援兩年。目前，僅支援 AWS IoT Device Tester FreeRTOS 202012.00 的版本，支援兩年。最新版本的 FreeRTOS 可在 [GitHub 上取得](#)。Device Tester 的支援版本，請前往 AWS IoT。[AWS IoT Device Tester for FreeRTOS 的支援版本 \(p. 291\)](#)

對於每個版本的 IDT 架構，將支援三個版本的測試套件，以符合裝置資格。

您也可以使用任何支援的 AWS IoT Device Tester 版本，搭配對應的 FreeRTOS 版本來測試您的裝置是否有資格。雖然您可以繼續使用 [適用於的不支援的 IDT 版本 FreeRTOS \(p. 293\)](#)，但這些不會收到錯誤修正或更新。

如果您對支援政策有任何疑問，請聯絡 [AWS 客戶支援](#)。

AWS 中的安全

雲端安全是 AWS 最重視的一環。身為 AWS 客戶的您，將能從資料中心和網路架構的建置中獲益，以滿足組織最為敏感的安全要求。

安全是 AWS 與您共同肩負的責任。[共同的責任模型](#) 將此描述為雲端本身的安全和雲端內部的安全：

- 雲端本身的安全 – AWS 負責保護執行 AWS 雲端內 AWS 服務的基礎設施。AWS 也會提供可讓您安全使用的服務。第三方稽核人員定期檢測和驗證安全效率也是 [AWS 合規計劃](#) 的一部分。若要了解適用於 AWS 服務的合規計劃，請參閱 [AWS 合規計劃範圍內的服務](#)。
- 雲端內部的安全 – 您的責任取決於所使用的 AWS 服務。您也必須對資料敏感度、組織要求，以及適用法律和法規等其他因素負責。

本文件會協助您了解使用 AWS 時共同責任模型的適用情形。下列主題將示範如何設定 AWS 以達到您的安全和合規目標。您也將了解如何使用其他 AWS 服務，以幫助您監控並保護 AWS 資源。

如需 AWS IoT 安全性深入的詳細資訊，請參閱 [AWS IoT 的安全性與身分](#)。

主題

- [適用於 AWS 資源的 Identity and Access Management \(p. 380\)](#)
- [合規驗證 \(p. 389\)](#)
- [AWS 中的復原功能 \(p. 389\)](#)
- [FreeRTOS 中的基礎設施安全 \(p. 390\)](#)

適用於 AWS 資源的 Identity and Access Management

AWS Identity and Access Management (IAM) 是一種 AWS 服務，可協助管理員安全地控制 AWS 資源的存取。IAM 管理員可以控制能進行身份驗證（登入）及獲得授權（具備許可）以使用 AWS 資源的人員。IAM 是一種 AWS 服務，無須支付任何額外的費用即可使用。

主題

- [Audience \(p. 380\)](#)
- [使用身分來驗證 \(p. 381\)](#)
- [使用政策管理存取權 \(p. 382\)](#)
- [進一步了解 \(p. 383\)](#)
- [AWS 服務如何使用 IAM \(p. 383\)](#)
- [身分型政策範例 \(p. 385\)](#)
- [對身分與存取進行疑難排解 \(p. 387\)](#)

Audience

您使用 AWS Identity and Access Management (IAM) 的方式會因您在 AWS 中進行的工作而有所差異。

服務使用者 – 若您使用 AWS 來執行任務，您的管理員可以提供您需要的登入資料和許可。隨著您為了執行作業而使用的 功能數量變多，您可能會需要額外的許可。了解存取控制的管理方式可協助您向管理員請求正確的許可。如果您無法存取 AWS 服務中的功能，請參閱 [對身分與存取進行疑難排解 \(p. 387\)](#)。

服務管理員 – 若您在公司負責管理 AWS 資源，您應該具備服務使用的完整存取權限。您的任務是要判斷員工應存取哪些 功能和資源。您接著必須將請求提交給您的 IAM 管理員，來變更您服務使用者的許可。檢閱此頁面上的資訊，了解 IAM 的基本概念。若要進一步了解您公司可搭配 AWS 使用 IAM 的方式，請參閱 [AWS 服務如何使用 IAM \(p. 383\)](#)。

IAM 管理員 – 如果您是 IAM 管理員，請了解如何撰寫政策來管理 AWS 存取的詳細資訊。如需可在 AWS 中使用的以 IAM 身分為基礎的政策詳細資訊，請參閱 [AWS Identity and Access Management 使用者指南中的政策與許可](#)。

使用身分來驗證

身份驗證是使用身分登入資料登入 AWS 的方式。如需使用 AWS 管理主控台 登入的詳細資訊，請前往 [中的 IAM 主控台和登入頁面](#)。IAM 使用者指南

您必須以 AWS 帳戶根使用者、IAM 使用者，或取得 IAM 角色身分的方式進行身份驗證 (登入 AWS)。您也可以使用貴公司的單一登入身分驗證，甚至使用 Google 或 Facebook 進行登入。在上述案例中，您的管理員會使用 IAM 角色預先設定聯合身分。當您使用來自其他公司的身份驗證來存取 AWS 時，您是間接地擔任角色。

若要直接登入 [AWS 管理主控台](#)，請使用您的密碼及您的 根使用者 電子郵件或您的 IAM 使用者名稱。您可以使用您的 根使用者 或 IAM 使用者存取金鑰，透過編寫程式的方式存取 AWS。AWS 提供軟體開發套件和命令列工具，以加密的方式使用您的登入資料簽署您的請求。如果您不使用 AWS 工具，您必須自行簽署請求。請使用 Signature 第 4 版來執行此作業，它是針對傳入 API 請求進行身份驗證的通訊協定。如需驗證請求的詳細資訊，請參閱 [AWS General Reference 中的 Signature 第 4 版簽署程序](#)。

無論您使用何種身份驗證方法，您可能還需要提供額外的安全資訊。例如，AWS 建議您使用多重驗證 (MFA) 來提高帳戶的安全。若要進一步了解，請前往 [AWS 中的 使用多重因素認證 \(MFA\)](#)。IAM 使用者指南

AWS 帳戶根使用者

當您首次建立 AWS 帳戶時，您會先有單一的登入身分，可以完整存取帳戶中所有 AWS 服務與資源。此身分稱為 AWS 帳戶 根使用者，是藉由您用來建立帳戶的電子郵件地址和密碼以登入並存取。強烈建議您不要以 根使用者 處理日常作業，即使是管理作業。反之，請遵循 [僅以 根使用者 建立您第一個 IAM 使用者的最佳實務](#)。接著請妥善鎖定 根使用者 登入資料，只用來執行少數的帳戶與服務管理作業。

IAM 使用者和群組

[IAM 使用者](#)是您 AWS 帳戶中的一種實體，具備單一人員或應用程式的特定許可。IAM 使用者可有長期登入資料 (例如，使用者名稱和密碼或一組存取金鑰)。若要了解如何建立存取金鑰，請查看 [IAM 中的 管理 使用者的存取金鑰](#)。IAM 使用者指南。當您產生 IAM 使用者的存取金鑰時，請確認您已檢視且安全地儲存金鑰對。您在這之後便無法復原私密存取金鑰。屆時您必須改為產生新的存取金鑰對。

[IAM 群組](#)是一種指定 IAM 使用者集合的實體。您無法以群組身分登入。您可以使用群組來一次為多名使用者指定許可。群組可讓管理大量使用者許可的過程變得更為容易。例如，您可以擁有一個名為 IAMAdmin 的群組，並給予該群組管理 IAM 資源的許可。

使用者與角色不同。使用者只會與單一人員或應用程式建立關聯，但角色的目的是在由任何需要它的人員取得。使用者擁有永久的長期登入資料，但角色僅提供暫時登入資料。若要進一步了解，請查看 [中的 IAM 建立使用者的時間 \(而不是角色\)](#)。IAM 使用者指南

IAM 角色

[IAM 角色](#)是您 AWS 帳戶中的一種實體，具備特定許可。它與 IAM 使用者相似，但是不會與特定人員建立關聯。您可以在 AWS 管理主控台 中透過 [切換角色](#)來暫時取得 IAM 角色。您可以透過呼叫 AWS CLI 或 AWS API 操作，或是使用自訂 URL 來假定角色。如需使用角色之方法的詳細資訊，請前往 [中的 IAM 使用 角色](#)。IAM 使用者指南

使用臨時登入資料的 IAM 角色在下列情況中非常有用：

- 暫時 IAM 使用者許可 – IAM 使用者可以取得 IAM 角色來暫時針對特定任務具備不同的許可。
- 聯合身分使用者存取 – 您可以使用 AWS Directory Service、您企業使用者目錄或 Web 身分供應商現有的使用者身分，而不需要建立 IAM 使用者。這些稱為聯合身分使用者。透過**身分供應商**來請求存取時，AWS 會指派角色給聯合身分使用者。如需聯合身份使用者的詳細資訊，請參閱 IAM 使用者指南中的**聯合身份使用者與角色**。
- 跨帳戶存取 – 您可以使用 IAM 角色，允許不同帳戶中的某人 (信任的委託人) 存取您帳戶的資源。角色是授予跨帳戶存取的主要方式。但是，針對某些 AWS 服務，您可以將政策直接連接到資源 (而非使用角色做為代理)。若要了解跨帳號存取的角色和以資源為基礎的政策之間的不同，請查看 [IAM中的與以資源為基礎的政策不同](#)。IAM 使用者指南
- AWS 服務存取 – 服務角色是 AWS 服務擔任的 [IAM角色](#)，以代表您執行動作。服務角色提供的存取權僅限在您的帳戶內，不能用來授予存取其他帳戶中的服務。IAM 管理員可以從 IAM 內建立、修改和刪除服務角色。如需詳細資訊，請參閱 IAM 使用者指南中的[建立角色以將許可委派給 AWS 服務](#)。
- 在 Amazon EC2 上執行的應用程式 – 針對在 EC2 執行個體上執行並提出 AWS CLI 或 AWS API 請求的應用程式，您可以使用 IAM 角色來管理臨時登入資料。這是在 EC2 執行個體內存放存取金鑰的較好方式。若要指派 AWS 角色給 EC2 執行個體並提供其所有應用程式使用，您可以建立連接到執行個體的執行個體描述檔。執行個體描述檔包含該角色，並且可讓 EC2 執行個體上執行的程式取得臨時登入資料。如需詳細資訊，請參閱 IAM 使用者指南中的[使用 IAM 角色授予許可給 Amazon EC2 執行個體上執行的應用程式](#)。

若要了解是否使用 IAM 角色，請查看 [中的IAM何時建立 角色 \(而不是使用者\)](#)。IAM 使用者指南

使用政策管理存取權

您可以透過建立政策並將其連接到 IAM 身分或 AWS 資源，在 AWS 中控制存取。政策是 AWS 中的一個物件，當其和實體或資源建立關聯時，便可定義其許可。AWS 會在實體 (根使用者、IAM 使用者或 IAM 角色) 發出請求時評估這些政策。政策中的許可，決定是否允許或拒絕請求。大部分政策以 JSON 文件形式存放在 AWS 中。如需 JSON 政策文件的結構和資訊，請查看 https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies.html#access_policies-json 中的 JSON 政策概觀 IAM 使用者指南。

IAM 管理員可以使用政策指定能存取 AWS 資源的人員，以及他們能在這些資源上執行的動作。每個 IAM 實體 (使用者或角色) 在開始時都沒有許可。換句話說，根據預設，使用者無法執行任何作業，甚至也無法變更他們自己的密碼。若要授予使用者執行動作的許可，管理員必須將許可政策連接到使用者。或者，管理員可以將使用者新增到具備預定許可的群組。管理員將許可給予群組時，該群組中的所有使用者都會獲得那些許可。

IAM 政策會定義該動作的許可，無論您使用何種方法來執行操作。例如，假設您有一個允許 `iam:GetRole` 動作的政策。具備該政策的使用者便可以從 AWS 管理主控台、AWS CLI 或 AWS API 取得使用者資訊。

以身分為基礎的政策

身分類型政策是您可以連接到身分 (例如 IAM 使用者、角色或群組) 的 JSON 許可政策文件。這些政策可控制身分在何種條件下能對哪些資源執行哪些動作。若要了解如何建立以身分為基礎的政策，請查看 [IAM中的建立 政策](#) IAM 使用者指南。

身分類型政策可進一步分類成內嵌政策或受管政策。內嵌政策會直接內嵌到單一使用者、群組或角色。受管政策則是獨立的政策，您可以將這些政策連接到 AWS 帳戶中的多個使用者、群組和角色。受管政策包含 AWS 受管政策和客戶受管政策。若要了解如何選擇受管政策或輸入政策，請查看 https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies_managed-vs-inline.html#choosing-managed-or-inline 中的受管政策和輸入政策之間的選擇 IAM 使用者指南。

以資源為基礎的政策

資源型政策是附加到資源 (如 Amazon S3 儲存貯體) 的 JSON 政策文件。服務管理員可使用這些政策來定義指定委託人 (帳戶成員、使用者或角色) 可以在什麼情況下對該資源執行什麼動作。以資源為基礎的政策是內嵌政策。不存在受管的以資源為基礎的政策。

存取控制清單 (ACL)

存取控制政策 (ACL) 可控制哪些委託人 (帳戶成員、使用者或角色) 擁有存取某資源的許可。ACLs 類似以資源為基礎的政策，雖然是不使用 JSON 政策文件格式的唯一政策類型。Amazon S3、AWS WAF 和 Amazon VPC 是支援 ACLs 的服務範例。若要進一步了解 ACLs，請前往 [開發人員指南](#) 中的存取控制清單 (ACL) 概觀 Amazon Simple Storage Service。

其他政策類型

AWS 支援其他較少見的政策類型。這些政策類型可設定較常見政策類型授予您的最大許可。

- 許可界限 – 許可界限是一種進階功能，可供您設定身分類型政策能授予 IAM 實體 (IAM 使用者或角色) 的最大許可。您可以為實體設定許可界限。所產生的許可會是實體的身分類型政策和其許可界限的交集。會在 Principal 欄位中指定使用者或角色的資源類型政策則不會受到許可界限限制。所有這類政策中的明確拒絕都會覆寫該允許。如需許可界限的詳細資訊，請查看 [IAM 中 實體的許可界限](#)。IAM 使用者指南
- 服務控制政策 (SCP) – SCPs 是 JSON 政策，可指定 AWS Organizations 中組織或組織單位 (OU) 的最大許可。AWS Organizations 是用於分組和集中管理您商業所擁有多個 AWS 帳號的服務。若您啟用組織中的所有功能，您可以將服務控制政策 (SCP) 套用到任何或所有帳戶。這個 SCP 會限制成員帳戶中實體的許可，包括每個 AWS 帳戶根使用者。如需組織和 SCPs 的詳細資訊，請前往 [中的 SCPs How](#) How work (運作方式) AWS Organizations 使用者指南。
- 工作階段政策 – 工作階段政策是一種進階政策，您可以在透過編寫程式的方式建立角色或聯合身分使用者的暫時工作階段時，做為參數傳遞。所產生工作階段的許可會是使用者或角色的身分類型政策和工作階段政策的交集。許可也可能來自資源類型政策。所有這類政策中的明確拒絕都會覆寫該允許。如需詳細資訊，請前往 https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies.html#policies_session 中的工作階段政策 IAM 使用者指南。

多種政策類型

將多種政策類型套用到請求時，其結果形成的許可會更為複雜、更加難以理解。若要了解 AWS 如何判斷是否在涉及多個政策類型時允許請求，請查看 [中的 Policy Evaluation Logic](#) (政策評估邏輯)。IAM 使用者指南

進一步了解

如需針對 AWS 資源的身分和存取權管理的詳細資訊，請繼續參閱以下頁面：

- [AWS 服務如何使用 IAM \(p. 383\)](#)
- [身分型政策範例 \(p. 385\)](#)
- [對身分與存取進行疑難排解 \(p. 387\)](#)

AWS 服務如何使用 IAM

在您使用 IAM 管理 AWS 服務的存取權前，建議您先了解可使用的 IAM 功能有哪些。若要取得 AWS 服務如何搭配 IAM 運作的高階檢視，請查看 [中的 AWS 使用 IAM 的 服務](#)。IAM 使用者指南

主題

- [以身分為基礎的政策 \(p. 384\)](#)
- [AWS 資源型政策 \(p. 385\)](#)
- [根據標籤的授權 \(p. 385\)](#)
- [IAM 角色 \(p. 385\)](#)

以身分為基礎的政策

使用 IAM 身分型政策，您可以指定允許或拒絕的動作和資源，以及在何種條件下允許或拒絕動作。若要了解您在 JSON 政策中使用的所有元素，請參考 [IAM 中的 JSON 政策元素參考](#)。IAM 使用者指南

Actions

IAM 身分類型政策的 Action 元素會描述政策將允許或拒絕的特定動作。政策動作的名稱通常會和相關聯的 AWS API 操作相同。政策會使用動作來授予執行相關聯操作的許可。

政策動作會在動作前使用前綴。政策陳述式必須包含 Action 或 NotAction 元素。每個服務都有自己的一組定義動作來描述您可以使用該服務執行的任務。

若要在單一陳述式中指定多個動作，請用逗號分隔，如下所示：

```
"Action": [  
    "service-prefix:action1",  
    "service-prefix:action2"]
```

您也可以使用萬用字元 (*) 來指定多個動作。例如，若要指定開頭是 Describe 文字的所有動作，請包含以下動作：

```
"Action": "service-prefix:Describe*"
```

若要查看 AWS 動作的清單，請參閱 IAM 使用者指南中的 [Actions, Resources, and Condition Keys for AWS Services](#)。

Resources

Resource 元素可指定動作套用的物件。陳述式必須包含 Resource 或 NotResource 元素。您可以使用 ARN 來指定資源，或是使用萬用字元 (*) 來指定陳述式套用到所有資源。

如需 ARNs 格式的詳細資訊，請前往 [Amazon Resource Names \(ARN\) 和 AWS Service Namespaces](#)。

若要指定屬於特定帳戶的所有執行個體，請使用萬用字元 (*)：

```
"Resource": "arn:aws:service-prefix:us-east-1:123456789012:resource-type/*"
```

有些 動作（例如用來建立資源的動作）無法在特定資源上執行。在那些情況下，您必須使用萬用字元 (*)。

```
"Resource": "*"
```

有些 API 動作涉及多個資源，因此 IAM 使用者必須具有使用所有資源的許可。若要在單一陳述式中指定多項資源，請用逗號分隔 ARNs。

```
"Resource": [  
    "resource1",  
    "resource2"]
```

您若要了解您可以使用哪些動作指定每項資源的 ARN，請參閱 [Actions, Resources, and Condition Keys for AWS Services](#)。

條件鍵

Condition 元素（或 Condition「區塊」）可讓您指定使陳述式生效的條件。Condition 元素是選用的。您可以建置使用 [條件運算子](#) 的條件表達式（例如等於或小於），來比對政策中的條件和請求中的值。

若您在陳述式中指定多個 Condition 元素，或是在單一 Condition 元素中指定多個金鑰，AWS 會使用邏輯 AND 操作評估他們。若您為單一條件金鑰指定多個值，AWS 會使用邏輯 OR 操作評估條件。必須符合所有條件，才會授予陳述式的許可。

您也可以在指定條件時使用預留位置變數。例如，您可以只在使用者使用其 IAM 使用者名稱標記時，將存取資源的許可授予該 IAM 使用者。如需詳細資訊，請查看 [政策元素: IAM](#)。《IAM 使用者指南》中的變數和標籤。

若要查看所有 AWS 全域條件金鑰，請查看 [AWSGlobal Condition Context Keys](#)。IAM 使用者指南

AWS 資源型政策

以資源為基礎的政策是 JSON 政策文件，這些文件會指定指定的委託人可對資源以及在怎樣的條件下執行哪些動作。以資源為基礎的政策可讓您依資源將使用許可授予至其他帳戶。

若要啟用跨帳戶存取，您可以指定在其他帳戶內的所有帳戶或 IAM 實體，做為 [資源型政策的委託人](#)。新增跨帳戶委託人至資源型政策，只是建立信任關係的一半。當委託人和資源分屬不同的 AWS 帳戶時，您也必須授與委託人實體資源的存取權。透過將身分型政策連接到實體來授予許可。不過，如果資源型政策會為相同帳戶中的委託人授與存取，這時就不需要額外的身分型政策。如需詳細資訊，請查看 [中 IAM 與資源型政策不同的角色 IAM 使用者指南](#)。

若要檢視詳細資源類型政策頁面的範例，請參閱 <https://docs.aws.amazon.com/lambda/latest/dg/access-control-resource-based.html>。

根據標籤的授權

您可以將標籤連接到資源，或是在請求中傳遞標籤。若要根據標籤控制存取，請使用 `prefix:ResourceTag/key-name`、`aws:RequestTag/key-name` 或 `aws:TagKeys` 條件金鑰，在政策的 [條件元素](#) 中提供標籤資訊。

若要檢視身分型政策範例以根據該資源上的標籤來限制存取資源，請參閱 [根據標籤檢視 資源 \(p. 387\)](#)。

IAM 角色

[IAM 角色](#)是您 AWS 帳戶中的一種實體，具備特定許可。

使用暫時登入資料

您可以使用暫時登入資料登入聯合、取得 IAM 角色，或是取得跨帳戶角色。您取得暫時安全登入資料的方式是透過呼叫 AWS Security Token Service (AWS STS) API 操作 (例如, `AssumeRole` 或 `GetFederationToken`)。

服務連結角色

[服務連結角色](#)可讓 AWS 服務存取其他服務中的資源，以代您完成動作。服務連結角色會顯示在您的 IAM 帳戶中，並由該服務所擁有。IAM 管理員可以檢視，但不能編輯服務連結角色的許可。

服務角色

此功能可讓服務代表您擔任 [服務角色](#)。此角色可讓服務存取其他服務中的資源，以代表您完成動作。服務角色會出現在您的 IAM 帳戶，且由該帳戶所擁有。這表示 IAM 管理員可以變更此角色的許可。不過，這樣可能會破壞此服務的功能。

身分型政策範例

根據預設，IAM 使用者和角色不具備建立或修改 AWS 資源的許可。他們也無法使用 AWS 管理主控台、AWS CLI 或 AWS API 執行任務。IAM 管理員必須建立 IAM 政策，授予使用者和角色在他們所需指定資源上執行特定 API 操作的許可。管理員接著必須將這些政策連接至需要這些許可的 IAM 使用者或群組。

若要了解如何使用這些範例 JSON 政策文件建立以 IAM 身分為基礎的政策,請查看 https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies_create.html#access_policies_create-json-editor 中的 JSON 標籤上的建立政策IAM 使用者指南。

主題

- [政策最佳實務 \(p. 386\)](#)
- [使用 AWS 主控台 \(p. 386\)](#)
- [允許使用者檢視他們自己的許可 \(p. 386\)](#)
- [根據標籤檢視 資源 \(p. 387\)](#)

政策最佳實務

身分類型政策相當強大。他們可以判斷您帳戶中的某個人員是否可以建立、存取或刪除資源。這些動作可能會讓您的 AWS 帳戶產生成本。當您建立或編輯身分類型政策時，請遵循下列準則及建議事項：

- 開始使用 AWS 受管政策 – 若要快速開始使用 AWS，請使用 AWS 受管政策來給予員工所需許可。這些政策已在您的帳戶中提供，並由 AWS 所維護和更新。如需詳細資訊,請前往 [中的AWS使用](#) 受管政策入門。IAM 使用者指南
- 授予最低權限 – 當您建立自訂政策時，請只授予執行任務所需要的許可。以最小一組許可開始，然後依需要授予額外的許可。如此做比一開始使用太寬鬆的許可更為安全，然後稍後再嘗試將它們限縮。如需詳細資訊,請前往 [中的](#) 授予最低權限。IAM 使用者指南
- 為敏感操作啟用 MFA – 為了增加安全，請要求 IAM 使用者使用多重驗證 (MFA) 存取敏感資源或 API 操作。如需詳細資訊，請參閱 [AWS](#) 中的在 IAM 使用者指南 中使用 Multi-Factor Authentication (MFA) 相關文章。
- 使用政策條件以增加安全 – 在切實可行的範圍中，請定義您身分類型政策允許存取資源的條件。例如，您可以撰寫條件，來指定必須發出各種請求的可允許 IP 地址範圍。您也可以撰寫條件，只在指定的日期或時間範圍內允許請求，或是要求使用 SSL 或 MFA。如需詳細資訊,請前往 [IAM JSON 政策元素: 中的 Condition \(條件\)](#)。IAM 使用者指南

使用 AWS 主控台

若要存取 AWS 服務的主控台，您必須擁有最低的一組許可。這些許可必須允許您列出和檢視您 AWS 帳戶中資源的詳細資訊。如果您建立比最基本必要許可更嚴格的身分型政策，則對於具有該政策的實體 (IAM 使用者或角色) 而言，主控台就無法如預期運作。

為確保那些實體仍可使用主控台，請也將 AWS 受管政策連接到實體。如需詳細資訊,請前往 [中的](#) 將許可新增至使用者。IAM 使用者指南

您不需要針對只呼叫 AWS CLI 或 AWS API 的使用者允許最基本的主控台許可。反之，只需允許存取符合您嘗試執行之 API 操作的動作就可以了。

允許使用者檢視他們自己的許可

此範例會示範如何建立政策，允許 IAM 使用者檢視連接到他們使用者身分的內嵌及受管政策。此政策包含在主控台上，或是使用 AWS CLI 或 AWS API 透過編寫程式的方式完成此動作的許可。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "ViewOwnUserInfo",  
            "Effect": "Allow",  
            "Action": [  
                "iam:GetUserPolicy",  
                "iam>ListGroupsForUser",  
            ]  
        }  
    ]  
}
```

```
        "iam>ListAttachedUserPolicies",
        "iam>ListUserPolicies",
        "iam GetUser"
    ],
    "Resource": [
        "arn:aws:iam::*:user/${aws:username}"
    ]
},
{
    "Sid": "NavigateInConsole",
    "Effect": "Allow",
    "Action": [
        "iam:GetGroupPolicy",
        "iam:GetPolicyVersion",
        "iam:GetPolicy",
        "iam>ListAttachedGroupPolicies",
        "iam>ListGroupPolicies",
        "iam>ListPolicyVersions",
        "iam>ListPolicies",
        "iam>ListUsers"
    ],
    "Resource": "*"
}
]
```

根據標籤檢視 資源

您可以在身分型政策中使用條件，來根據標籤控制存取 資源。此範例會示範如何建立允許檢視資源的政策。但是，只有在資源標籤 Owner 的值是該使用者的使用者名稱時，才會授予該許可。此政策也會授予在主控台完成此動作的必要許可。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ListInputsInConsole",
            "Effect": "Allow",
            "Action": "prefix>ListInputs",
            "Resource": "*"
        },
        {
            "Sid": "ViewResourceIfOwner",
            "Effect": "Allow",
            "Action": "prefix>ListInputs",
            "Resource": "arn:aws:prefix::resource-name/*",
            "Condition": {
                "StringEquals": {"prefix:ResourceTag/Owner": "${aws:username}"}
            }
        }
    ]
}
```

您可以將此政策連接至您帳戶中的 IAM 使用者。如果名為 richard-roe 的使用者嘗試檢視 *resource-name*,*resource-name* 必須標記 Owner=richard-roe 或 owner=richard-roe。否則，他存取遭拒。條件標籤鍵 Owner 符合 Owner 和 owner，因為條件金鑰名稱不區分大小寫。如需詳細資訊，請前往 [IAM JSON 政策元素：中的 Condition \(條件\)](#)。IAM 使用者指南

對 身分與存取進行疑難排解

使用下列資訊，協助您針對在使用 IAM 時可能遇到的常見問題，進行診斷與排除。

主題

- 我未獲得執行動作的授權 (p. 388)
- 我未獲得執行 iam:PassRole 的授權 (p. 388)
- 我想要檢視我的存取金鑰 (p. 388)
- 我是管理員，想要允許其他人存取 AWS 資源 (p. 389)
- 我想要允許 AWS 帳戶以外的人員存取我的資源 (p. 389)

我未獲得執行動作的授權

若 AWS 管理主控台 告知您並未獲得執行動作的授權，您必須聯絡您的管理員以取得協助。您的管理員是提供您使用者名稱和密碼的人員。

以下範例錯誤會在 mateojackson IAM 使用者嘗試使用主控台來檢視 的詳細資訊時發生 *my-example-resource* 但不具備 *prefix:Action* 許可。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to
perform: prefix:Action on resource: my-example-resource
```

在此情況下，Mateo 會請求管理員更新他的政策，允許他使用 *prefix:Action* 存取 *my-example-resource*。

我未獲得執行 iam:PassRole 的授權

若您收到錯誤，告知您並未獲得執行 iam:PassRole 動作的授權，您必須聯絡您的管理員以取得協助。您的管理員是提供您使用者名稱和密碼的人員。要求該人員更新您的政策，允許您將角色傳遞給服務。

有些 AWS 服務允許您傳遞現有的角色至該服務，而無須建立新的服務角色或服務連結角色。若要執行此作業，您必須擁有將角色傳遞至該服務的許可。

以下範例錯誤會在名為 marymajor 的 IAM 使用者嘗試使用主控台在服務中執行動作時發生。但是，動作要求服務具備服務角色授予的許可。Mary 沒有將角色傳遞至該服務的許可。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform: iam:PassRole
```

在這種情況下，Mary 會請求管理員更新她的政策，允許她執行 iam:PassRole 動作。

我想要檢視我的存取金鑰

在您建立 IAM 使用者存取金鑰後，您可以隨時檢視您的存取金鑰 ID。但是，您無法再次檢視您的私密存取金鑰。若您遺失了秘密金鑰，您必須建立新的存取金鑰對。

存取金鑰包含兩個部分：存取金鑰 ID (例如 AKIAIOSFODNN7EXAMPLE) 和私密存取金鑰 (例如 wJalrXUtnFEMI/K7MDENG/bPxRfCYEXAMPLEKEY)。如同使用者名稱和密碼，您必須一起使用存取金鑰 ID 和私密存取金鑰來驗證您的請求。就如對您的使用者名稱和密碼一樣，安全地管理您的存取金鑰。

Important

請不要將您的存取金鑰提供給第三方，甚至是協助尋找您的標準使用者 ID。執行此作業，可能會讓他人能夠永久存取您的帳戶。

建立存取金鑰對時，您會收到提示，要求您將存取金鑰 ID 和私密存取金鑰儲存在安全位置。只有在您建立的時候才可使用私密存取金鑰。若您遺失了私密存取金鑰，您必須將新的存取金鑰新增到您的 IAM 使用者。您最多可有兩種存取金鑰。若您已有兩個存取金鑰，您必須先刪除其中一個金鑰對，才能建立新的金鑰對。若要檢視指示，請查看 中的管理存取金鑰。IAM 使用者指南

我是管理員，想要允許其他人存取 AWS 資源

若要允許其他人存取服務，您必須針對需要存取的人員或應用程式建立 IAM 實體 (使用者或角色)。他們將使用該實體的登入資料來存取 AWS。您接著必須將政策連接到實體，在 AWS 中授予正確的許可。

若要立即開始，請查看 [IAM 中的建立您的第一個 委派使用者和群組 IAM 使用者指南](#)。

我想要允許 AWS 帳戶以外的人員存取我的資源

您可以建立一個角色，讓其他帳戶中的使用者或您組織外部的人員存取您的資源。您可以指定要允許哪些信任對象取得該角色。針對支援資源類型政策或存取控制清單 (ACL) 的服務，您可以使用那些政策來授予人員存取您資源的權限。

若要進一步了解，請參閱以下內容：

- 若要了解服務是否支援這些功能，請參閱 [AWS 服務如何使用 IAM \(p. 383\)](#)。
- 若要了解如何跨您擁有的 AWS 項目提供資源存取，請查看 [《IAM》AWS 中的授與存取權給另一個由您擁有 IAM 使用者指南的使用者](#)。
- 若要了解如何將資源的存取權提供給第三方 AWS 帳號，請參考 [中的提供存取給第三方擁有的 AWS Accounts IAM 使用者指南](#)
- 若要了解如何透過聯合身分提供存取，請前往 https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_common-scenarios_federated-users.html 中的 IAM 使用者指南提供存取給外部驗證使用者 (聯合身分)。
- 若要了解使用角色和以資源為基礎的政策取得跨帳號存取的不同，請查看 [中的 IAM 與資源型政策不同的角色 IAM 使用者指南](#)。

合規驗證

FreeRTOS 不在任何 AWS 合規計劃範圍內。如需特定合規計劃範圍內的 AWS 服務清單，請參閱 [合規計劃內的 AWS 服務](#)。如需一般資訊，請參閱 [AWS 合規計劃](#)。

您可使用 AWS Artifact 下載第三方稽核報告。如需詳細資訊，請參閱 [在 AWS Artifact 中下載報告](#)。

您使用 FreeRTOS 時的合規責任取決於資料的敏感度、您公司的合規目標，以及適用的法律和法規。AWS 提供以下資源協助您處理合規事宜：

- [安全與合規快速入門指南](#) – 這些部署指南討論在 AWS 上部署以安全及合規為重心之基準環境的架構考量和步驟。
- [HIPAA 安全與合規架構白皮書](#) – 本白皮書說明公司可如何運用 AWS 來建立 HIPAA 合規的應用程式。
- [AWS 合規資源](#) – 這組手冊和指南可能適用於您的產業和位置。
- [AWS Config](#) – 此 AWS 服務可評定資源組態與內部實務、業界準則和法規的合規狀態。
- [AWS Security Hub](#) – 此 AWS 服務可供您檢視 AWS 中的安全狀態，可助您檢查是否符合安全產業標準和最佳實務。

AWS 中的復原功能

AWS 全球基礎設施是以 AWS 區域與可用區域為中心建置的。AWS 區域提供多個分開且隔離的實際可用區域，並以低延遲、高輸送量和高度備援網路連線相互連結。透過可用區域，您所設計與操作的應用程式和資料庫，就能夠在可用區域之間自動容錯移轉，而不會發生中斷。可用區域的可用性、容錯能力和擴充能力，均較單一或多個資料中心的傳統基礎設施還高。

如需 AWS 區域與可用區域的詳細資訊，請參閱 [AWS 全球基礎設施](#)。

FreeRTOS 中的基礎設施安全

AWS 受管服務受到 [Amazon Web Services : 安全程序概觀](#)白皮書所述的 AWS 全球網路安全程序所保護。

您可使用 AWS 發佈的 API 呼叫，透過網路存取 AWS 服務。用戶端必須支援 Transport Layer Security (TLS) 1.0 或更新版本。建議使用 TLS 1.2 或更新版本。用戶端也必須支援具備完美轉送私密 (PFS) 的密碼套件，例如臨時 Diffie-Hellman (DHE) 或橢圓曲線臨時 Diffie-Hellman (ECDHE)。現代系統 (如 Java 7 和更新版本) 大多會支援這些模式。

此外，請求必須使用存取金鑰 ID 和與 IAM 委託人相關聯的私密存取金鑰來簽署。或者，您可以使用 [AWS Security Token Service](#) (AWS STS) 來產生暫時安全登入資料來簽署請求。

本文為英文版的機器翻譯版本，如內容有任何歧義或不一致之處，概以英文版為準。