aws

使用者指南

FreeRTOS

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

FreeRTOS: 使用者指南

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

Amazon 的商標和商業外觀不得用於任何非 Amazon 的產品或服務，也不能以任何可能造成客戶混淆、任何貶低或使 Amazon 名譽受損的方式使用 Amazon 的商標和商業外觀。所有其他非 Amazon 擁有的商標均為其各自擁有者的財產，這些擁有者可能附屬於 Amazon，或與 Amazon 有合作關係，亦或受到 Amazon 贊助。

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 2
FreeRTOS 合格硬體平台 2
開發工作流程 3
其他資源 3
FreeRTOS 核心基礎 5
FreeRTOS 核心排程器 5
記憶體管理 6
核心記憶體配置 6
應用程式記憶體管理 6
任務間協調 7
佇列 7
旗號與 Mutex 7
直達任務通知 8
串流緩衝區 8
訊息緩衝區 9
軟體計時器 11
低電力支援 11
FreeRTOSConfig.h 11
AWS IoT 適用於 Embedded C 的裝置 SDK 12
FreeRTOS 入門 13
FreeRTOS 示範應用程式 13
首要步驟 14
設定 AWS 您的帳戶和許可 14
向 註冊您的 MCU 電路板 AWS IoT 16
下載 FreeRTOS 19
設定 FreeRTOS 示範 19
故障診斷 22
一般入門故障診斷提示 22
安裝終端機模擬器 22
搭配 FreeRTOS 使用 CMake 24
先決條件 25

使用第三方程式碼編輯器和偵錯工具開發 FreeRTOS 應用程式 26
使用 CMake 建置 FreeRTOS 26
開發人員模式金鑰佈建 32
簡介 33
選項 #1：從 匯入私有金鑰 AWS IoT 33
選項 #2：產生內建私有金鑰 33
主機板特定的入門指南 35
Cypress CYW943907AEVAL1F 開發套件 36
Cypress CYW954907AEVAL1F 開發套件 41
Cypress CY8CKIT-064S0S2-4343W 45
Microchip ATECC608A 安全元素與 Windows 模擬器 51
Espressif ESP32-DevKitC 和 ESP-WROVER-KIT 58
Espressif ESP32-WROOM-32SE 98
Infineon XMC4800 IoT 連線套件 112
Infineon OPTIGA Trust X 和 XMC4800 IoT Connectivity Kit 118
Marvell MW32x AWS IoT 入門套件 124
MediaTek MT7697Hx 開發套件 146
Microchip Curiosity PIC32MZ EF 150
Nordic nRF52840-DK 156
Nuvoton NuMaker-IoT-M487 161
NXP LPC54018 IoT 模組 168
Renesas Starter Kit+ for RX65N-2MB 172
IoT 節點用的 STMicroelectronics STM32L4 探索套件 177
Texas Instruments CC3220SF-LAUNCHXL 180
Windows 裝置模擬器 185
Xilinx Avnet MicroZed Industrial IoT 套件 189
無線更新 198
標記 OTA 資源 198
OTA 更新先決條件 199
建立 Amazon S3 儲存貯體以存放您的更新 199
建立 OTA 更新服務角色 200
建立 OTA 使用者政策 203
建立程式碼簽署憑證 204
授予 程式碼簽署的存取權 AWS IoT 213
使用 OTA 程式庫下載 FreeRTOS 214
使用 MQTT 進行 OTA 更新的先決條件 214

使用 HTTP 進行 OTA 更新的先決條件 218
OTA 教學 222
安裝初始韌體 223
更新您的韌體版本 236
建立 OTA 更新AWS IoT （主控台） 237
使用 建立 OTA 更新 AWS CLI 241
OTA Update Manager 服務 264
將 OTA 代理程式整合到您的應用程式 265
連線管理 266
簡單 OTA 示範 266
針對 OTA 完成事件使用自訂回撥 269
OTA 安全性 270
的程式碼簽署 AWS IoT 271
OTA 故障診斷 271
設定 CloudWatch Logs 以進行 OTA 更新 272
使用 AWS IoT 記錄 OTA API 呼叫 AWS CloudTrail 277
使用 取得 CreateOTAUpdate 失敗詳細資訊 AWS CLI 280
使用 取得 OTA 失敗代碼 AWS CLI 282
故障診斷多個裝置的 OTA 更新 283
故障診斷 Texas Instruments CC3220SF Launchpad 的 OTA 更新 284
FreeRTOS 程式庫 285
FreeRTOS 移植程式庫 285
FreeRTOS 應用程式程式庫 287
FreeRTOS 通用程式庫 288
設定 FreeRTOS 程式庫 289
常見的程式庫 289
不可部分完成操作 290
Linear Containers 291
日誌 291
Static Memory 292
Task Pool 292
backoffAlgorithm 296
簡介 296
低功耗藍牙 297
概觀 297
架構 298

相依性和要求 300
程式庫組態檔案 300
最佳化 301
使用限制 301
初始化 302
API 參考 304
範例使用方式 304
移植 307
適用於 FreeRTOS 藍牙裝置的 Mobile SDKs 310
通用 I/O 311
AWS IoT Device Defender 312
簡介 312
AWS IoT Greengrass 313
概觀 313
相依性和要求 313
API 參考 314
範例使用方式 315
coreHTTP 316
簡介 316
coreJSON 317
簡介 317
記憶體使用 317
coreMQTT 318
簡介 318
OTA 代理程式 319
概觀 319
功能 320
API 參考 320
範例使用方式 321
移植 321
記憶體使用 321
corePKCS11 322
概觀 322
功能 323
非對稱加密系統支援 324
移植 326

記憶體使用 326
Secure Sockets 327
概觀 327
相依性和要求 327
功能 328
故障診斷 328
開發人員支援 328
使用限制 329
初始化 329
API 參考 329
範例使用方式 329
移植 332
AWS IoT 裝置影子 332
簡介 332
AWS IoT 任務 333
簡介 333
Transport Layer Security 334
Wi-Fi 334
概觀 334
相依性和要求 334
功能 335
組態 336
初始化 337
API 參考 338
範例使用方式 338
移植 340
FreeRTOS 示範 341
執行 FreeRTOS 示範 341
設定示範 342
低功耗藍牙 342
概觀 342
先決條件 343
常見元件 346
透過低功耗藍牙執行的 MQTT 353
Wi-Fi 佈建 354
一般屬性伺服器 360

Microchip Curiosity PIC32MZEF 的開機載入器 362
開機載入器狀態 363
快閃裝置 364
應用程式映像結構 365
映像標頭 365
映像描述項 366
映像尾部 367
開機載入器組態 368
建置開機載入器 369
AWS IoT Device Defender 369
簡介 369
功能 370
AWS IoT Greengrass 373
Amazon EC2 376
coreHTTP 377
coreHTTP 相互身分驗證 378
coreHTTP Amazon S3 上傳 380
coreHTTP S3 下載 382
coreHTTP 多執行緒 385
AWS IoT 任務 389
簡介 389
原始碼組織 389
設定 AWS IoT MQTT 代理程式連線 390
功能 390
coreMQTT 391
coreMQTT 相互身分驗證 392
coreMQTT 連線共用 396
無線更新 401
Over-the-air示範組態 406
Texas Instruments CC3220SF-LAUNCHXL 407
Microchip Curiosity PIC32MZEF 409
Espressif ESP32 414
Renesas RX65N 415
AWS IoT 裝置影子 439
簡介 439
功能 440

連線至 AWS IoT MQTT 代理程式 442
刪除影子文件 442
訂閱陰影主題 442
傳送陰影更新 443
處理陰影差異訊息和陰影更新訊息 443
Secure Sockets 444
使用 AWS IoT Device Tester for FreeRTOS 446
IDT for FreeRTOS 支援的版本 447
最新版本的 IDT for FreeRTOS 447
舊版 IDT 448
不支援的 IDT 版本 451
先決條件 454
下載 FreeRTOS 454
LTS 資格（使用 LTS 程式庫的 FreeRTOS 資格） 454
下載 IDT for FreeRTOS 455
..... 455
AWS IoT Device Tester 受管政策 457
（選用）安裝 AWS Command Line Interface 457
首次準備測試微型控制器主機板 458
新增程式庫移植層 458
設定您的 AWS 登入資料 458
在 IDT for FreeRTOS 中建立裝置集區 458
設定建置、刷新和測試設定 463
執行低功耗藍牙測試 475
先決條件 475
Raspberry Pi 設定 476
FreeRTOS 裝置設定 479
執行 BLE 測試 480
故障診斷 BLE 測試 480
執行 FreeRTOS 資格套件 480
IDT for FreeRTOS 命令 484
重新取得資格的測試 485
測試套件版本 486
了解結果和日誌 487
檢視結果 487
使用 IDT 來開發和執行您自己的測試套件 491

下載最新版本的 IDT for FreeRTOS 491
測試套件建立工作流程 492
教學課程：建置並執行範例 IDT 測試套件 492
教學課程：開發簡單的 IDT 測試套件 498
建立 IDT 測試套件組態檔案 507
設定 IDT 狀態機器 514
建立 IDT 測試案例可執行檔 536
使用 IDT 內容 542
設定測試執行器的設定 546
偵錯並執行自訂測試套件 557
檢閱 IDT 測試結果和日誌 559
IDT 用量指標 565
故障診斷 571
故障診斷裝置組態 572
故障診斷逾時錯誤 581
行動功能和 AWS 費用 582
資格報告產生政策 582
支援政策 582
中的安全性 AWS 583
身分和存取權管理 583
目標對象 584
使用身分驗證 584
使用政策管理存取權 587
FreeRTOS 如何與 IAM 搭配使用 589
身分型政策範例 594
故障診斷 597
法規遵循驗證 598
恢復能力 599
基礎架構安全 600
..... dci

什麼是 FreeRTOS？

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

FreeRTOS 與全球領先的晶片公司在 15 年期間共同開發，現在每 175 秒下載一次，是微型控制器和小型微處理器的市場領導即時作業系統 (RTOS)。FreeRTOS 在 MIT 開放原始碼授權下自由分佈，包括核心和一組不斷增長的程式庫，適用於所有產業領域。FreeRTOS 的建置重點是可靠性和易用性。

FreeRTOS 包含連線、安全性和over-the-air(OTA) 更新的程式庫。FreeRTOS 也包含在合格主機板上顯示 FreeRTOS 功能的示範應用程式。

FreeRTOS 是開放原始碼專案。您可以在 GitHub 網站 https://github.com/aws/amazon-freertos 下載原始程式碼、提供變更或增強功能，或回報問題。我們根據 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它。

我們也歡迎對 FreeRTOS 文件的貢獻 (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 GitHub 複製或下載 FreeRTOS。如需說明，請參閱 README.md 檔案。

FreeRTOS 版本控制

FreeRTOS 核心和元件會個別發行，並使用語意版本控制。整合式 FreeRTOS 版本會定期進行。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 7 月的第二個版本為 202107.01。

先前，FreeRTOS 版本針對主要版本使用語意版本控制。雖然它已移至以日期為基礎的版本控制 (FreeRTOS 1.4.8 更新至 FreeRTOS AWS 參考整合 201906.00)，但 FreeRTOS 核心和每個個別 FreeRTOS 程式庫仍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) 表示該版本是為主要、次要或小版本更新。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 通常會以單一編譯映像的方式刷入裝置，其中包含裝置應用程式所需的所有元件。此映像會合併由內嵌開發人員撰寫的應用程式功能、Amazon 提供的軟體程式庫、FreeRTOS 核心，以及硬體平台的驅動程式及電路板支援套件 (BSP)。嵌入式應用程式開發人員可以預期與 FreeRTOS 核心和所有 FreeRTOS 軟體程式庫相同的標準化界面，而不受使用個別微控制器的影響。

FreeRTOS 合格硬體平台

下列硬體平台符合 FreeRTOS 的資格：

- 適用於 的 ATECC608A 零接觸佈建套件 AWS IoT
- Cypress CYW943907AEVAL1F 開發套件
- Cypress CYW954907AEVAL1F 開發套件
- Espressif ESP32-DevKitC
- Espressif ESP-WROVER-KIT
- Infineon XMC4800 IoT 連線套件
- Marvell MW320 AWS IoT Starter 套件
- Marvell MW322 AWS IoT Starter 套件
- MediaTek MT7697Hx 開發套件
- Microchip Curiosity PIC32MZEF 套件
- Nordic nRF52840-DK
- NuMaker-IoT-M487
- NXP LPC54018 IoT 模組
- OPTIGA Trust X 安全解決方案

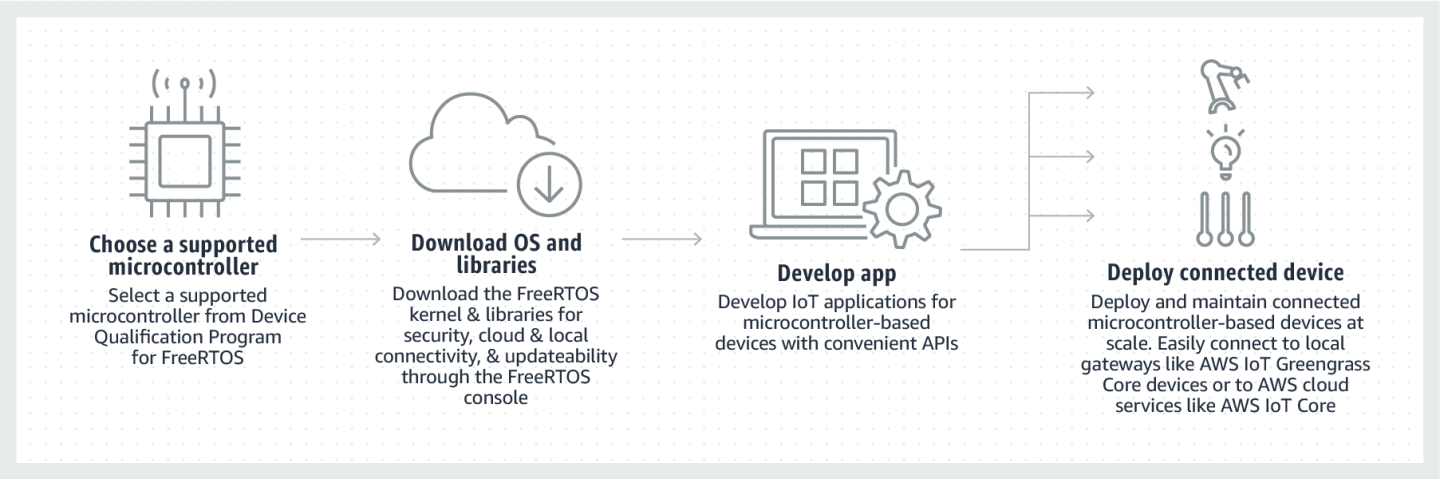
- Renesas RX65N RSK IoT 模組
- STMicroelectronicsSTM32L4 Discovery Kit IoT Node
- Texas Instruments CC3220SF-LAUNCHXL
- Microsoft Windows 7 或更新版本，含至少一個雙核心和有線乙太網路連線
- Xilinx Avnet MicroZed Industrial IoT 套件

合格的裝置也列在 AWS Partner Device Catalog 中。

如需符合新裝置資格的詳細資訊，請參閱 FreeRTOS 資格指南。

開發工作流程

您可以透過下載 FreeRTOS 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。部署的裝置可以連線至 AWS IoT 服務或 AWS IoT Greengrass 做為完整 IoT 解決方案的一部分。



其他資源

這些資源可能對您有所幫助。

- 其他 FreeRTOS 文件可在 https：//freertos.org 上取得，包括 FreeRTOS v10.0.0 參考手冊。
- 如需有關 FreeRTOS 工程團隊 FreeRTOS 的問題，您可以在 FreeRTOS GitHub 頁面上開啟問題。
- 如需有關 FreeRTOS 的技術問題，請造訪 FreeRTOS 社群論壇。
- 如需將裝置連線至 的詳細資訊 AWS IoT，請參閱《AWS IoT Core 開發人員指南》和該指南》中的裝置佈建章節。

- 如需 的技術支援 AWS，請造訪 AWS 支援中心。
- 如需有關 AWS 帳單、帳戶服務、事件、濫用或其他問題的問題 AWS，請造訪聯絡我們頁面。

FreeRTOS 核心基礎

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

FreeRTOS 核心是一種即時作業系統，可支援許多架構。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。

FreeRTOS 核心絕不會執行不具確定性的操作 (例如查核連結清單、處於關鍵區段內部，或是插斷)。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。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。

FreeRTOS 核心小型、簡易且易於使用。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up-to-date文件，請參閱 FreeRTOS.org。FreeRTOS.org 提供了一系列有關使用 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 也會建立閒置任務，只在其他任務尚未準備好執行的期間執行。

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 保存在其可攜式 layer 中。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 堆積進行配置。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。

FreeRTOS 核心包含五種堆積實作：

`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 FreeRTOS 基本功能的相關資訊。

佇列

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。) 訊息會以複製的方式透過佇列傳送，即表示資料 (可為指向大型緩衝區的指標) 本身會複製到佇列中，而非僅只是將參考存放到資料內。

佇列 API 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

FreeRTOS 核心可提供二進位旗號、計數旗號，以及適用於互斥及同步處理的 mutex。

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。將較於可用於執行相同功能的 FreeRTOS 功能，任務通知在速度及 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()` 用於在任務中將資料傳送到串流緩衝區。`xStreamBufferSendFromISR()` 則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()` 為巨集，會在資料寫入串流緩衝區時呼叫 (由 FreeRTOS API 從內部呼叫)。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 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() 用於在任務中從串流緩衝區讀取資料。xStreamBufferReceiveFromISR() 則用於在插斷服務常式 (ISR) 中從串流緩衝區中讀取資料。

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() 為巨集，會在從串流緩衝區讀取資料時呼叫 (由 FreeRTOS API 從內部呼叫)。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 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() 用於從任務中將資料傳送到訊息緩衝區。xMessageBufferSendFromISR() 則用於從插斷服務常式 (ISR) 中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() 為巨集，會在資料寫入串流緩衝區時呼叫 (由 FreeRTOS API 從內部呼叫)。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 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() 用於在任務中從訊息緩衝區讀取資料。xMessageBufferReceiveFromISR() 則用於在插斷服務常式 (ISR) 中從訊息緩衝區讀取資料。xMessageBufferReceive() 則允許指定封鎖時間。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() 為巨集，會在從串流緩衝區讀取資料時呼叫 (由 FreeRTOS API 從內部呼叫)。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 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。FreeRTOS 核心提供高效率的軟體計時器實作，因為：

- 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

與大多數的內嵌作業系統相似，FreeRTOS 核心會使用硬體計時器產生定期刻度插斷，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。

為了解決這項限制，FreeRTOS 包含了一個適用於低電力應用程式的無刻度計時器模式。FreeRTOS 無刻度閒置模式會在閒置期間停止定期刻度插斷 (即沒有可執行應用程式任務的期間)，然後在重新啟動刻度插斷時對 RTOS 刻度計數值進行修正調整。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 `FreeRTOSConfig.h` 標頭檔案設定特定主機板和應用程式的 FreeRTOS 核心。每個建置在核心上的應用程式都必須在其前置處理器包含路徑中具有 `FreeRTOSConfig.h` 標頭檔案。`FreeRTOSConfig.h` 是應用程式特定項目，應該放在應用程式目錄底下，而不是在任一個 FreeRTOS 核心原始程式碼目錄底下。

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

如需可在 `FreeRTOSConfig.h` 中指定的可用配置參數清單，請參閱 FreeRTOS.org。

AWS IoT 適用於 Embedded C 的裝置 SDK

Important

這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

Note

此 SDK 適合經驗豐富的嵌入式軟體開發人員使用。

適用於 Embedded C 的 AWS IoT Device SDK (C-SDK) 是 MIT 開放原始碼授權下的 C 來源檔案集合，可用於內嵌應用程式，以安全地將 IoT 裝置連接到其中 AWS IoT Core。它包含 MQTT 用戶端、HTTP 用戶端、JSON Parser 和 AWS IoT Device Shadow、 AWS IoT Jobs 和 AWS IoT Device Defender 程式庫。此 SDK 以來源形式分配，並且可與應用程式碼、其他程式庫及您選擇的作業系統 (OS) 作業系統一起內建於客戶韌體中。

適用於 Embedded C 的 AWS IoT Device SDK (C-SDK) 是 MIT 開放原始碼授權下的 C 來源檔案集合，可用於內嵌應用程式，以安全地將 IoT 裝置連接到其中 AWS IoT Core。它包含 MQTT 用戶端、HTTP 用戶端、JSON Parser 和 AWS IoT Device Shadow、 AWS IoT Jobs、 AWS IoT Fleet Provisioning 和 AWS IoT Device Defender 程式庫。此 SDK 以來源形式分配，並且可與應用程式碼、其他程式庫及您選擇的作業系統 (OS) 作業系統一起內建於客戶韌體中。

通常以需要最佳化 C 語言執行時間的資源限制裝置 適用於 Embedded C 的 AWS IoT Device SDK 為目標。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 (例如 MCU 和 MPU) 上。不過，如果您的裝置有足夠的記憶體和處理資源，我們建議您使用其中一個順序較高的AWS IoT 裝置 SDKs。

如需詳細資訊，請參閱下列內容：

- AWS IoT GitHub 上適用於內嵌 C 的裝置 SDK
- AWS IoT 適用於嵌入式 C 的 裝置 SDK 讀我檔案
- AWS IoT 適用於內嵌 C 的 裝置 SDK 範例

FreeRTOS 入門

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

本 FreeRTOS 入門教學課程說明如何在主機電腦上下載和設定 FreeRTOS，然後在合格的微控制器主機板上編譯和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
- 首要步驟
- 故障診斷入門
- 搭配 FreeRTOS 使用 CMake
- 開發人員模式金鑰佈建
- 主機板特定的入門指南

FreeRTOS 示範應用程式

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

本教學課程中的示範應用程式是 *freertos*/demos/coreMQTT/mqtt_demo_mutual_auth.c 檔案中定義的 coreMQTT 相互身分驗證示範。它會使用 coreMQTT 程式庫連線至 AWS 雲端，然後定期將訊息發佈至 MQTT AWS IoT 代理程式託管的 MQTT 主題。

一次只能執行單一 FreeRTOS 示範應用程式。當您建置 FreeRTOS 示範專案時，*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。coreMQTT 相互身分驗證示範預設為啟用。

如需 FreeRTOS 隨附的示範應用程式的詳細資訊，請參閱 FreeRTOS 示範。

首要步驟

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

若要開始使用 FreeRTOS AWS IoT，您必須擁有 AWS 帳戶、具有存取 AWS IoT 和 FreeRTOS 雲端服務許可的 IAM 使用者。您還必須下載 FreeRTOS 並設定主機板的 FreeRTOS 示範專案來使用 AWS IoT。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。
> - 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。

1. 設定 AWS 您的帳戶和許可
2. 向 註冊您的 MCU 電路板 AWS IoT
3. 下載 FreeRTOS
4. 設定 FreeRTOS 示範

設定 AWS 您的帳戶和許可

註冊 AWS 帳戶

如果您沒有 AWS 帳戶，請完成下列步驟來建立一個。

註冊 AWS 帳戶

1. 開啟 https://portal.aws.amazon.com/billing/signup。
2. 請遵循線上指示進行。

   註冊程序的一部分包括接聽電話或文字訊息，以及在電話鍵盤上輸入驗證碼。

   當您註冊 時 AWS 帳戶，AWS 帳戶根使用者會建立 。根使用者有權存取該帳戶中的所有 AWS 服務 和資源。作為安全最佳實務，請將管理存取權指派給使用者，並且僅使用根使用者來執行需要根使用者存取權的任務。

AWS 會在註冊程序完成後傳送確認電子郵件給您。您可以隨時登錄 https://aws.amazon.com/ 並選擇我的帳戶，以檢視您目前的帳戶活動並管理帳戶。

建立具有管理存取權的使用者

註冊 後 AWS 帳戶，請保護 AWS 帳戶根使用者、啟用 AWS IAM Identity Center和建立管理使用者，以免將根使用者用於日常任務。

保護您的 AWS 帳戶根使用者

1. 選擇根使用者並輸入 AWS 帳戶 您的電子郵件地址，以帳戶擁有者AWS Management Console身分登入 。在下一頁中，輸入您的密碼。

   如需使用根使用者登入的說明，請參閱 AWS 登入 使用者指南中的以根使用者身分登入。
2. 若要在您的根使用者帳戶上啟用多重要素驗證 (MFA)。

   如需說明，請參閱《IAM 使用者指南》中的為您的 AWS 帳戶 根使用者 （主控台） 啟用虛擬 MFA 裝置。

建立具有管理存取權的使用者

1. 啟用 IAM Identity Center。

   如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的啟用 AWS IAM Identity Center。
2. 在 IAM Identity Center 中，將管理存取權授予使用者。

   如需使用 IAM Identity Center 目錄 做為身分來源的教學課程，請參閱AWS IAM Identity Center《 使用者指南》中的使用預設值設定使用者存取 IAM Identity Center 目錄。

以具有管理存取權的使用者身分登入

- 若要使用您的 IAM Identity Center 使用者簽署，請使用建立 IAM Identity Center 使用者時傳送至您電子郵件地址的簽署 URL。

  如需使用 IAM Identity Center 使用者登入的說明，請參閱AWS 登入 《 使用者指南》中的登入 AWS 存取入口網站。

指派存取權給其他使用者

1. 在 IAM Identity Center 中，建立一個許可集來遵循套用最低權限的最佳實務。

   如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的建立許可集。
2. 將使用者指派至群組，然後對該群組指派單一登入存取權。

   如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的新增群組。

若要提供存取權，請新增權限至您的使用者、群組或角色：

- 中的使用者和群組 AWS IAM Identity Center：

  建立權限合集。請按照 AWS IAM Identity Center 使用者指南 中的 建立權限合集 說明進行操作。
- 透過身分提供者在 IAM 中管理的使用者：

  建立聯合身分的角色。遵循「IAM 使用者指南」的為第三方身分提供者 (聯合) 建立角色中的指示。
- IAM 使用者：
  - 建立您的使用者可擔任的角色。請按照「IAM 使用者指南」的為 IAM 使用者建立角色中的指示。
  - (不建議) 將政策直接附加至使用者，或將使用者新增至使用者群組。請遵循 IAM 使用者指南的新增許可到使用者 (主控台) 中的指示。

設定 AWS 帳戶和許可後，請繼續 向 註冊您的 MCU 電路板 AWS IoT。

向 註冊您的 MCU 電路板 AWS IoT

您的主機板必須向 註冊 AWS IoT ，才能與 AWS 雲端通訊。若要向 註冊您的電路板 AWS IoT，您必須擁有：

AWS IoT 政策

AWS IoT 政策會授予您的裝置存取 AWS IoT 資源的許可。它存放在 AWS 雲端。

AWS IoT 實物

物件 AWS IoT 可讓您在 中管理裝置 AWS IoT。它存放在 AWS 雲端。

私密金鑰和 X.509 憑證

私有金鑰和憑證可讓您的裝置進行身分驗證 AWS IoT。

若要註冊您的電路板，請遵循下列程序。

建立 AWS IoT 政策

1. 若要建立 IAM 政策，您必須知道您 AWS 的區域和 AWS 帳戶號碼。

   若要尋找 AWS 您的帳戶號碼，請開啟 AWS 管理主控台，在右上角找到並展開帳戶名稱下方的選單，然後選擇我的帳戶。您的帳戶 ID 會顯示在 Account Settings (帳戶設定) 下方。

   若要尋找 AWS 帳戶的 AWS 區域，請使用 AWS Command Line Interface。若要安裝 AWS CLI，請遵循 AWS Command Line Interface 使用者指南中的指示。安裝 之後 AWS CLI，請開啟命令提示視窗，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. 在導覽窗格中，選擇安全，選擇政策，然後選擇建立。
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。將 *`aws-region`* 和 *`aws-account`* 取代為您的 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`

授予您的裝置許可，以使用任何用戶端 ID 連線至 AWS IoT 訊息中介裝置。

`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. 選擇建立。

為裝置建立 IoT 實物、私有金鑰和憑證

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. 如果您的帳戶中尚未註冊任何 IoT 實物，則會顯示 You don't have any things yet (尚無任何實物) 頁面。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. 在建立 AWS IoT 物件頁面上，選擇建立單一物件。
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. 選擇連接政策，將政策連接至您的憑證，以授予裝置存取 AWS IoT 操作的權限。
10. 選擇您剛建立的政策，然後選擇 Register thing (註冊實物)。

向 註冊電路板後 AWS IoT，您可以繼續 下載 FreeRTOS。

下載 FreeRTOS

您可以從 FreeRTOS GitHub 儲存庫下載 FreeRTOS。

> **Note**
>
> 如果您要開始使用 Cypress CYW954907AEVAL1F 或 CYW943907AEVAL1F 開發套件，您必須從 GitHub 下載 FreeRTOS。如需說明，請參閱 README.md 檔案。這些主機板的 FreeRTOS 組態目前無法從 FreeRTOS 主控台取得。

設定 FreeRTOS 示範

您必須先編輯 FreeRTOS 目錄中的一些組態檔案，才能在電路板上編譯和執行任何示範。

設定您的 AWS IoT 端點

您必須向 FreeRTOS 提供 AWS IoT 端點，以便在電路板上執行的應用程式才能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，選擇設定。

   您的 AWS IoT 端點會顯示在端點中。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)。

   您的裝置應該有 AWS IoT 物件名稱。記下此名稱。
4. 打開 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

如果您的主機板透過 Wi-Fi 連線連線至網際網路，您必須向 FreeRTOS 提供 Wi-Fi 登入資料，才能連線至網路。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`*Wi-Fi #######*

     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 必須擁有與您的已註冊物件相關聯的 AWS IoT 憑證和私有金鑰及其許可政策，才能 AWS IoT 代表您的裝置成功與 通訊。

> **Note**
>
> 若要設定登入 AWS IoT 資料，您必須擁有註冊裝置時從 AWS IoT 主控台下載的私有金鑰和憑證。將裝置註冊為 AWS IoT 物件後，您可以從 AWS IoT 主控台擷取裝置憑證，但無法擷取私有金鑰。

FreeRTOS 是 C 語言專案，憑證和私有金鑰必須經過特殊格式，才能新增至專案。

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 (產生並儲存 aws_clientcredential_keys.h)，並將檔案儲存到 `demos/include` 中。這會覆寫目錄中現有的檔案。

   > **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。

設定 FreeRTOS 之後，您可以繼續參閱主機板入門指南，以設定平台的硬體及其軟體開發環境，然後在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。入門教學中使用的示範應用程式是 coreMQTT 相互身分驗證示範，位於 `demos/coreMQTT/mqtt_demo_mutual_auth.c`。

故障診斷入門

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

下列主題可協助您針對開始使用 FreeRTOS 時遇到的問題進行疑難排解：

主題

- 一般入門故障診斷提示
- 安裝終端機模擬器

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 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：

1. 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。
2. 確認您的網路登入資料有效。

執行示範時，我的終端機中顯示的日誌會被截斷。如何增加長度？

針對您正在執行的示範，將 `FreeRTOSconfig.h` 檔案中的值`configLOGGING_MAX_MESSAGE_LENGTH`增加為 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	Value
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.*
```

搭配 FreeRTOS 使用 CMake

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

您可以使用 CMake 從 FreeRTOS 應用程式原始碼產生專案建置檔案，以及建置和執行原始碼。

您也可以使用 IDE 在符合 FreeRTOS 資格的裝置上編輯、偵錯、編譯、刷新和執行程式碼。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。如果您偏好在沒有 IDE 的情況下作業，則可使用第三方的程式碼編輯與除錯工具來開發和除錯程式碼，再接著使用 CMake 建置並執行應用程式。

以下主機板支援使用 CMake：

- Espressif ESP32-DevKitC
- Espressif ESP-WROVER-KIT
- Infineon XMC4800 IoT 連線套件
- Marvell MW320 AWS IoT Starter 套件
- Marvell MW322 AWS IoT Starter 套件
- Microchip Curiosity PIC32MZEF 套件
- Nordic nRF52840 DK Development kit
- STMicroelectronicsSTM32L4 Discovery Kit IoT Node
- Texas Instruments CC3220SF-LAUNCHXL
- Microsoft Windows 模擬器

如需搭配 FreeRTOS 使用 CMake 的詳細資訊，請參閱下列主題。

主題

- 先決條件
- 使用第三方程式碼編輯器和偵錯工具開發 FreeRTOS 應用程式
- 使用 CMake 建置 FreeRTOS

先決條件

請確認您的主機機器符合下列先決條件，再繼續操作：

- 您的裝置的編譯工具鏈必須支援該機器的作業系統。且 CMake 需支援 Windows、macOS 和 Linux 的所有版本。

  不支援「適用於 Linux 的 Windows 子系統 (WSL)」。在 Windows 機器上使用原生 CMake。
- 您必須安裝 CMake 3.13 版或更新版本。

  您可以從 CMake.org 下載 CMake 的二進位分發。

  > **Note**
  >
  > 如果您已下載 CMake 的二進位分發版本，請務必先新增 CMake 可執行檔至 PATH 環境變數，再從命令列使用 CMake。

  您也可以使用套件管理員來下載並安裝 CMake，例如 macOS 上的 homebrew，以及 Windows 上的 scoop 或 chocolatey。

  > **Note**
  >
  > 在許多 Linux 分發版本的套件管理員中，提供的 CMake 套件版本都已過時。如果分發版本的套件管理員未提供 CMake 最新版本，您可以嘗試換成其他套件管理員，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 都可透過套件管理員來安裝在 Linux、macOS 和 Windows 上。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**
  >
  > MinGW 中的 Make 可執行檔稱為 `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。

使用第三方程式碼編輯器和偵錯工具開發 FreeRTOS 應用程式

您可以使用程式碼編輯器和除錯延伸模組或第三方除錯工具來開發 FreeRTOS 的應用程式。

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。應用程式開發完畢後，即可呼叫 CMake 命令列工具從 VS 程式碼內建置專案。如需使用 CMake 建置 FreeRTOS 應用程式的詳細資訊，請參閱使用 CMake 建置 FreeRTOS 。

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/stm32l475_discovery/aws_demos.elf",
        "request": "launch",
        "type": "cortex-debug",
        "servertype": "stutil"
    }
]
```

使用 CMake 建置 FreeRTOS

根據預設，CMake 以您的主機作業系統作為目標系統。若要將它用於跨編譯，CMake 需要指定您想使用之編譯器的工具鏈檔案。在 FreeRTOS 中，我們在 中提供預設的工具鏈檔案*freertos*/tools/cmake/toolchains。提供此檔案到 CMake 的方法取決於您是否使用 CMake 命令列界面或 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) 指示。如需有關在 CMake 中跨編譯的詳細資訊，請參閱官方 CMake wiki 上的跨編譯。

建置以 CMake 為基礎的專案

1. 執行 CMake 來產生原生建置系統的建置檔案，例如 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。

   如需產生 FreeRTOS 組建檔案的資訊，請參閱 產生建置檔案 (CMake 命令行工具)和 產生建置檔案 (CMake GUI)。
2. 叫用原生建置系統將專案製成可執行檔。

   如需製作 FreeRTOS 組建檔案的相關資訊，請參閱 從產生的建置檔案建置 FreeRTOS 。

產生建置檔案 (CMake 命令行工具)

您可以使用 CMake 命令列工具 (cmake) 來產生 FreeRTOS 的建置檔案。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**
>
> 如果您使用 Windows，則必須指定原生建置系統，因為 CMake 預設會使用 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
```

如果您想要使用的工具鏈檔案不是位於預設目錄 cmake/toolchains，您可以使用 CMAKE_TOOLCHAIN_FILE 選項。例如：

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

如果工具鏈檔案不使用絕對路徑來尋找您的編譯器，而且您未將您的編譯器新增到 PATH 環境變數，則 CMake 可能找不到它。為了確保 CMake 可找到您的工具鏈檔案，您可以使用 AFR_TOOLCHAIN_PATH 選項。此選項會搜尋指定的工具鏈目錄路徑和 bin 下的工具鏈子資料夾。例如：

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

若要啟用除錯，請將 CMAKE_BUILD_TYPE 設為 debug。啟用此選項後，CMake 會將偵錯旗標新增至編譯選項，並使用偵錯符號建置 FreeRTOS。

```
# 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。

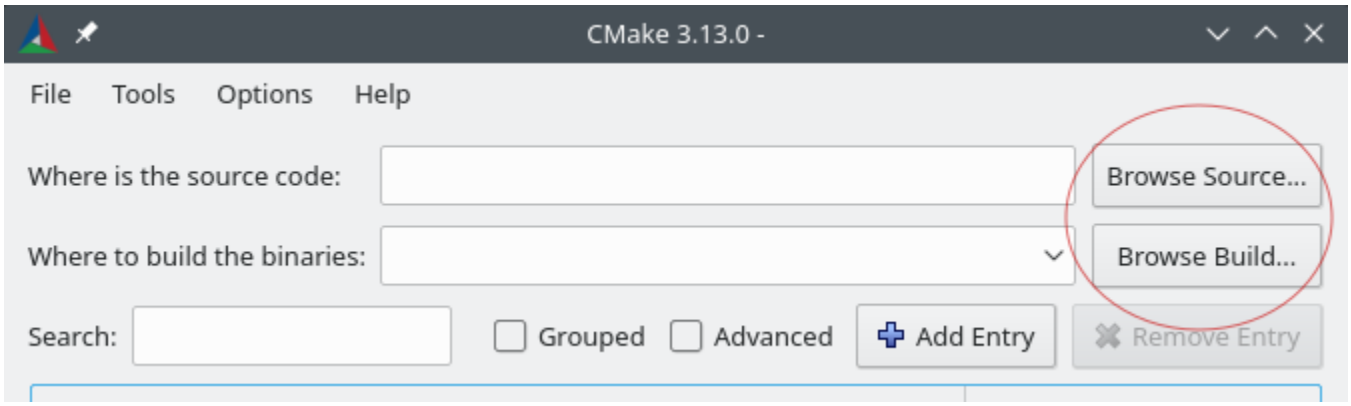
產生建置檔案 (CMake GUI)

您可以使用 CMake GUI 來產生 FreeRTOS 組建檔案。

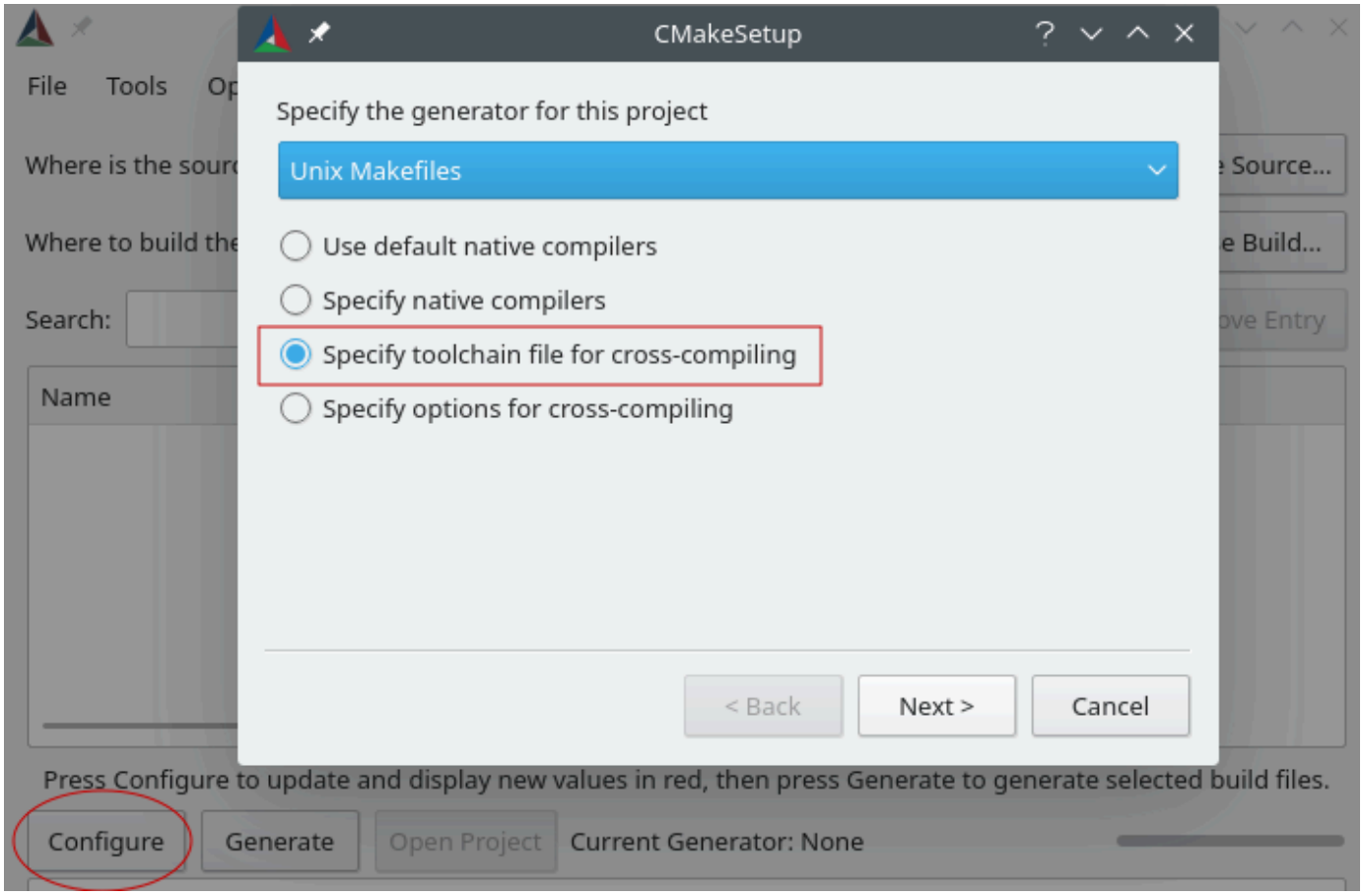
使用 CMake GUI 產生建置檔案

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，尋找和選擇您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。在此情況下，請從檔案功能表中選擇刪除快取，以刪除 CMake 快取。



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)。

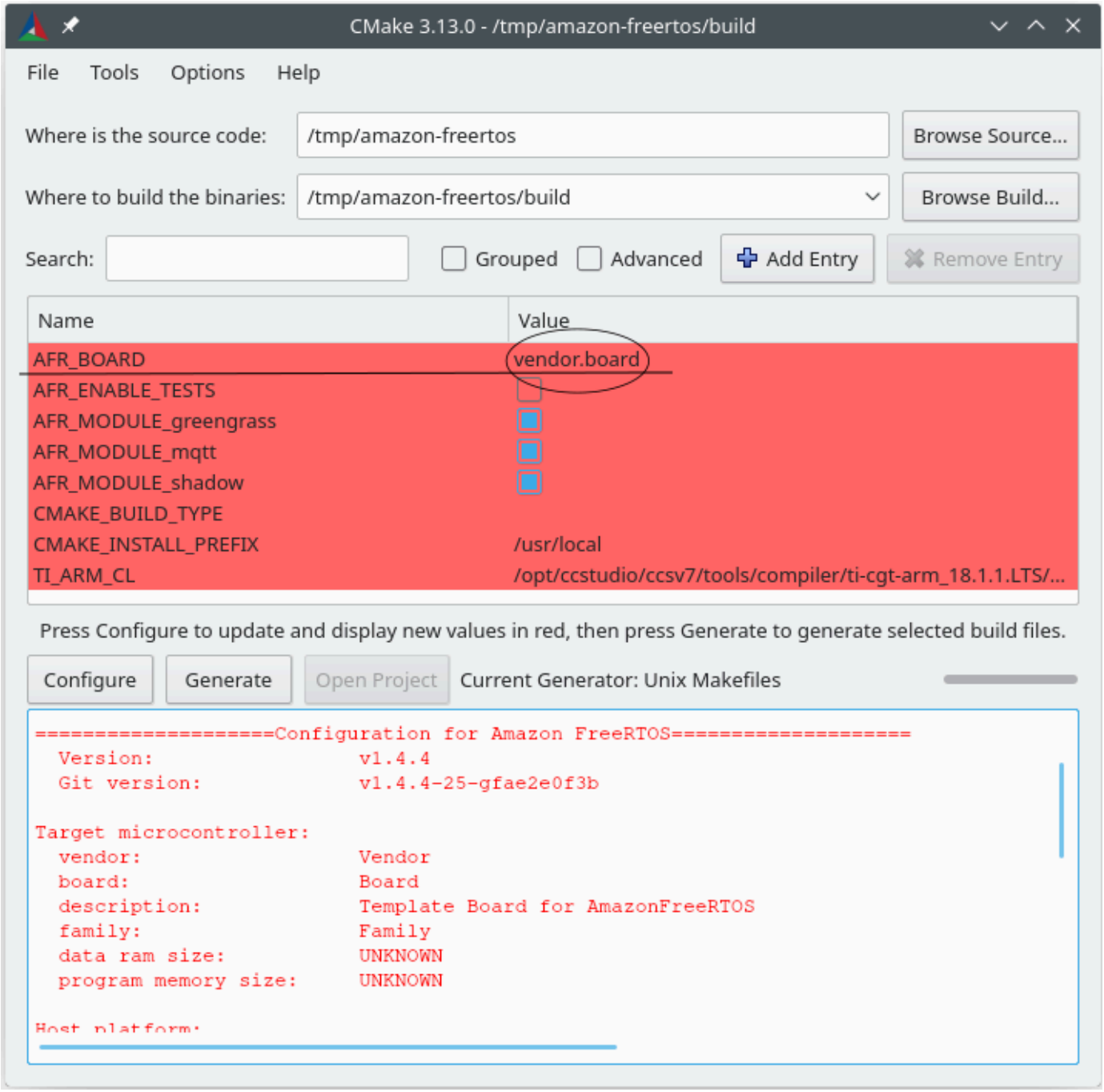
5. 選擇工具鏈檔案 (例如，*freertos*/tools/cmake/toolchains/arm-ti.cmake)，然後選擇 Finish (完成)。

   FreeRTOS 的預設組態是範本板，它不提供任何可攜式 layer 目標。因此，將出現含有訊息 Error in configuration process 的視窗。

   > **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。您可以在 GUI 中設定 AFR_TOOLCHAIN_PATH 變數，以告訴 CMake 您安裝編譯器的位置。如果您沒有看到 AFR_TOOLCHAIN_PATH 變數，請選擇 Add Entry (新增項目)。在彈出視窗中的 Name (名稱) 下輸入 **AFR_TOOLCHAIN_PATH**。在 Compiler Path (編譯器路徑) 輸入編譯器路徑。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 Generate (產生)。CMake 會產生建置系統檔案 (例如，makefiles 或 ninja 檔案)，這些檔案會出現在您在第一個步驟中指定的建置目錄中。請按照下一節的指示來產生二進位影像。

從產生的建置檔案建置 FreeRTOS

使用原生建置系統進行建置

您可以透過從輸出二進位目錄呼叫建置系統命令，使用原生建置系統建置 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
```

使用 CMake 建置

您也可以使用 CMake 命令列工具來建置 FreeRTOS。CMake 提供抽象層來呼叫原生建置系統。例如：

```
cmake --build build_dir
```

以下是 CMake 命令列工具建置模式的一些其他常見用法：

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

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

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

如需 CMake 建置模式的詳細資訊，請參閱 CMake 文件。

開發人員模式金鑰佈建

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

簡介

本節會討論兩個選項，將受信任的 X.509 用戶端憑證放置到 IoT 裝置，以供實驗室測試。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 匯入私有金鑰 AWS IoT

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 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，在 中執行步驟設定 FreeRTOS 示範，但略過最後一個步驟 （ 也就是說，不要將 AWS IoT 登入資料格式化)。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

開啟 coreMQTT 相互身分驗證示範，如 中的電路板指南中所述主機板特定的入門指南。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用戶端憑證佈建，示範將無法向 進行身分驗證 AWS IoT。不過，coreMQTT 相互身分驗證示範會從執行開發人員模式金鑰佈建開始，如果私有金鑰尚未存在，則會導致建立私有金鑰。您應該會在序列主控台輸出的開頭附近看到類似以下內容。

```
7 910 [IP-task] Device public key, 91 bytes:
3059 3013 0607 2a86 48ce 3d02 0106 082a
8648 ce3d 0301 0703 4200 04cd 6569 ceb8
```

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

將六行金鑰位元組複製到名為 DevicePublicKeyAsciiHex.txt 的檔案中。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 憑證建立裝置憑證」一節中所述順序執行作業。

在此情況下，裝置不會簽署憑證請求 （ 也就是憑證服務請求或 CSR)，因為建立和簽署 CSR 所需的 X.509 編碼邏輯已從 FreeRTOS 示範專案中排除，以減少 ROM 大小。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
```

憑證授權機構建立並註冊後 AWS IoT，請使用下列命令，根據上一個步驟中簽署的裝置 CSR 發出用戶端憑證：

```
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。您也需要匯入您的憑證授權機構 (CA) 憑證，因為這是第一次驗證 AWS IoT 在使用 JITP 時成功的必要條件。在您專案的 `aws_clientcredential_keys.h` 檔案中，將 `keyCLIENT_CERTIFICATE_PEM` 巨集設為 deviceCert.pem 的內容，並將 `keyJITR_DEVICE_CERTIFICATE_AUTHORITY_PEM` 巨集設為 `rootCA.pem` 的內容。

裝置授權

`deviceCert.pem` 匯入至 AWS IoT 登錄檔，如使用您自己的憑證中所述。您必須建立新的 AWS IoT 物件，將待定憑證和政策連接至您的物件，然後將憑證標記為 ACTIVE。所有這些步驟都可以在 AWS IoT 主控台中手動執行。

一旦新的用戶端憑證為 ACTIVE 並與物件和政策相關聯，請再次執行 coreMQTT 相互身分驗證示範。這次，與 AWS IoT MQTT 代理程式的連線將會成功。

主機板特定的入門指南

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

完成 後首要步驟，請參閱您的主機板指南，以取得開始使用 FreeRTOS 的主機板特定指示：

- Cypress CYW943907AEVAL1F 開發套件入門
- Cypress CYW954907AEVAL1F 開發套件入門
- Cypress CY8CKIT-064S0S2-4343W 套件入門
- Infineon XMC4800 IoT Connectivity Kit 入門
- MW32x AWS IoT 入門套件入門
- MediaTek MT7697Hx 開發套件入門
- Microchip Curiosity PIC32MZ EF 入門

- Nuvoton NuMaker-IoT-M487 入門
- NXP LPC54018 IoT Module 入門
- Renesas Starter Kit+ for RX65N-2MB 入門
- IoT 節點用的 STMicroelectronics STM32L4 探索套件入門
- Texas Instruments CC3220SF-LAUNCHXL 入門
- Windows 裝置模擬器入門
- Xilinx Avnet MicroZed Industrial IoT 入門套件

> **Note**
>
> 您不需要完成下列獨立式 FreeRTOS 入門指南首要步驟的 ：
>
> - 開始使用 Microchip ATECC608A 安全元素與 Windows 模擬器
> - Espressif ESP32-DevKitC 和 ESP-WROVER-KIT 入門
> - Espressif ESP32-WROOM-32SE 入門
> - 開始使用 Infineon OPTIGA Trust X 和 XMC4800 IoT Connectivity Kit
> - Nordic nRF52840-DK 入門

Cypress CYW943907AEVAL1F 開發套件入門

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

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**
>
> 本教學課程會逐步引導您設定和執行 coreMQTT 相互身分驗證示範。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，您必須設定 AWS IoT 和 FreeRTOS 下載，將您的裝置連線至 AWS 雲端。如需說明，請參閱 首要步驟。

Important

- 在本主題中，FreeRTOS 下載目錄的路徑稱為 *freertos*。
- *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。長 FreeRTOS 下載目錄路徑可能會導致建置失敗。
- 如 所述下載 FreeRTOS，Cypress 的 FreeRTOS 連接埠目前只能在 GitHub 上使用。

概觀

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. 跨編譯 FreeRTOS 示範應用程式至二進位映像。
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，您可以使用 Cypress WICED Studio 開發套件，透過 FreeRTOS 示範來編寫電路板的程式。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，系統依預設會在 `Documents` 資料夾中為軟體開發套件建立 `WICED-Studio-m.n` 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. 在 Variable name (變數名稱) 中，輸入 **`WICED_STUDIO_SDK_PATH`**。在 Variable value (變數值) 中，輸入 WICED Studio 軟體開發套件安裝目錄。

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。

建置並執行 FreeRTOS 示範專案

設定電路板的序列連線後，您可以建置 FreeRTOS 示範專案、將示範刷新到電路板，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 Cloud 的訊息。

使用 MQTT 用戶端訂閱 AWS IoT 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 ***your-thing-name*/example/topic**，然後選擇訂閱主題。

故障診斷

- 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。

Cypress CYW954907AEVAL1F 開發套件入門

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

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**
>
> 本教學課程會逐步引導您設定和執行 coreMQTT 相互身分驗證示範。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，您必須設定 AWS IoT 和 FreeRTOS 下載，將您的裝置連線至 AWS 雲端。如需說明，請參閱 首要步驟。在本教學課程中，FreeRTOS 下載目錄的路徑稱為 *`freertos`*。

> **Important**
>
> - 在本主題中，FreeRTOS 下載目錄的路徑稱為 *`freertos`*。
> - *`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。長 FreeRTOS 下載目錄路徑可能會導致建置失敗。
> - 如 所述下載 FreeRTOS，Cypress 的 FreeRTOS 連接埠目前只能在 GitHub 上使用。

概觀

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. 跨編譯 FreeRTOS 示範應用程式至二進位映像。
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，您可以使用 Cypress WICED Studio 開發套件，透過 FreeRTOS 示範來編寫電路板的程式。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，系統依預設會在 `Documents` 資料夾中為軟體開發套件建立 `WICED-Studio-m.n` 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. 在 Variable name (變數名稱) 中，輸入 **`WICED_STUDIO_SDK_PATH`**。在 Variable value (變數值) 中，輸入 WICED Studio 軟體開發套件安裝目錄。

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。

建置並執行 FreeRTOS 示範專案

設定電路板的序列連線後，您可以建置 FreeRTOS 示範專案、將示範刷新到電路板，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 Cloud 的訊息。

使用 MQTT 用戶端訂閱 AWS IoT 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 ***`your-thing-name`*`/example/topic`**，然後選擇訂閱主題。

故障診斷

- 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。

Cypress CY8CKIT-064S0S2-4343W 套件入門

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

本教學課程提供開始使用 CY8CKIT-064S0S2-4343W 套件的說明。如果您還沒有，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。

開始使用

開始之前，您必須設定 AWS IoT 和 FreeRTOS 將您的裝置連線至 AWS 雲端。如需說明，請參閱 首要步驟。完成先決條件後，您將擁有具有 AWS IoT Core 登入資料的 FreeRTOS 套件。

> **Note**
>
> 在本教學課程中，在「第一步驟」區段中建立的 FreeRTOS 下載目錄路徑稱為 *freertos*。

設定開發環境

FreeRTOS 適用於 CMake 或建立建置流程。您可以使用 ModusToolbox 進行建立流程。您可以使用與 ModusToolbox 一起交付的 Eclipse IDE 或合作夥伴 IDE，例如 IAR EW-Arm、Arm MDK 或 Microsoft Visual Studio Code。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 ModusToolbox 軟體。如需詳細資訊，請參閱 ModusToolbox 安裝指南。

更新 ModusToolbox 2.1 或更新版本的工具

如果您使用 ModusToolbox 2.1 Eclipse IDE 來編寫此套件的程式，則需要更新 OpenOCD 和韌體載入器工具。

在下列步驟中，依預設， 的*ModusToolbox*路徑為：

- Windows 是 C:\Users*user_name*\ModusToolbox。
- Linux 是您選擇擷取封存檔案*user_home*/ModusToolbox的位置。
- MacOS 位於您在精靈中選取的磁碟區中的應用程式資料夾下。

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 版本頁面。
2. 下載作業系統的封存檔案 (Windows/Mac/Linux)。
3. 刪除 中的現有檔案*ModusToolbox*/tools_2.x/openocd。
4. 將 中的檔案取代為您*ModusToolbox*/tools_2.x/openocd在上一個步驟中下載的封存擷取內容。

更新 Firmware-loader

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 版本頁面。

2. 下載作業系統的封存檔案 (Windows/Mac/Linux)。
3. 刪除 中的現有檔案*ModusToolbox*/tools_2.x/fw-loader。
4. 將 中的檔案取代為您*ModusToolbox*/tools_2.x/fw-loader在上一個步驟中下載的封存擷取內容。

或者，您可以使用 CMake 從 FreeRTOS 應用程式原始程式碼產生專案建置檔案、使用您偏好的建置工具建置專案，然後使用 OpenOCD 編寫套件的程式。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 Solutions 網頁下載並安裝 Cypress Programmer。如需詳細資訊，請參閱搭配 FreeRTOS 使用 CMake 。

設定您的硬體

請依照下列步驟來設定套件的硬體。

1. 佈建您的套件

   請遵循 CY8CKIT-064S0S2-4343W 套件的佈建指南說明，安全地佈建您的套件 AWS IoT。
2. 設定序列連線

   a. 將套件連接至您的主機電腦。

   b. 主機電腦上會自動列舉套件的 USB 序列連接埠。識別連接埠號碼。在 Windows 中，您可以使用連接埠 (COM & LPT) 下的 Device Manager 來識別它。

   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，您將建置並執行示範。

1. 請務必遵循 CY8CKIT-064S0S2-4343W 套件佈建指南中的步驟。
2. 建置 FreeRTOS 示範。

a. 開啟適用於 ModusToolbox 的 Eclipse IDE，然後選擇或建立工作區。

b. 從 File (檔案) 功能表中，選擇 Import (匯入)。

展開一般，選擇現有專案到工作區，然後選擇下一步。

c. 在根目錄中，輸入 *freertos*/projects/cypress/CY8CKIT-064S0S2-4343W/mtb/aws_demos ，然後選取專案名稱 aws_demos。預設應選取它。

d. 選擇完成，將專案匯入您的工作區。

e. 透過執行下列其中一項來建置應用程式：

- 從快速面板中，選取建置 aws_demos 應用程式。
- 選擇專案，然後選擇全部建置。

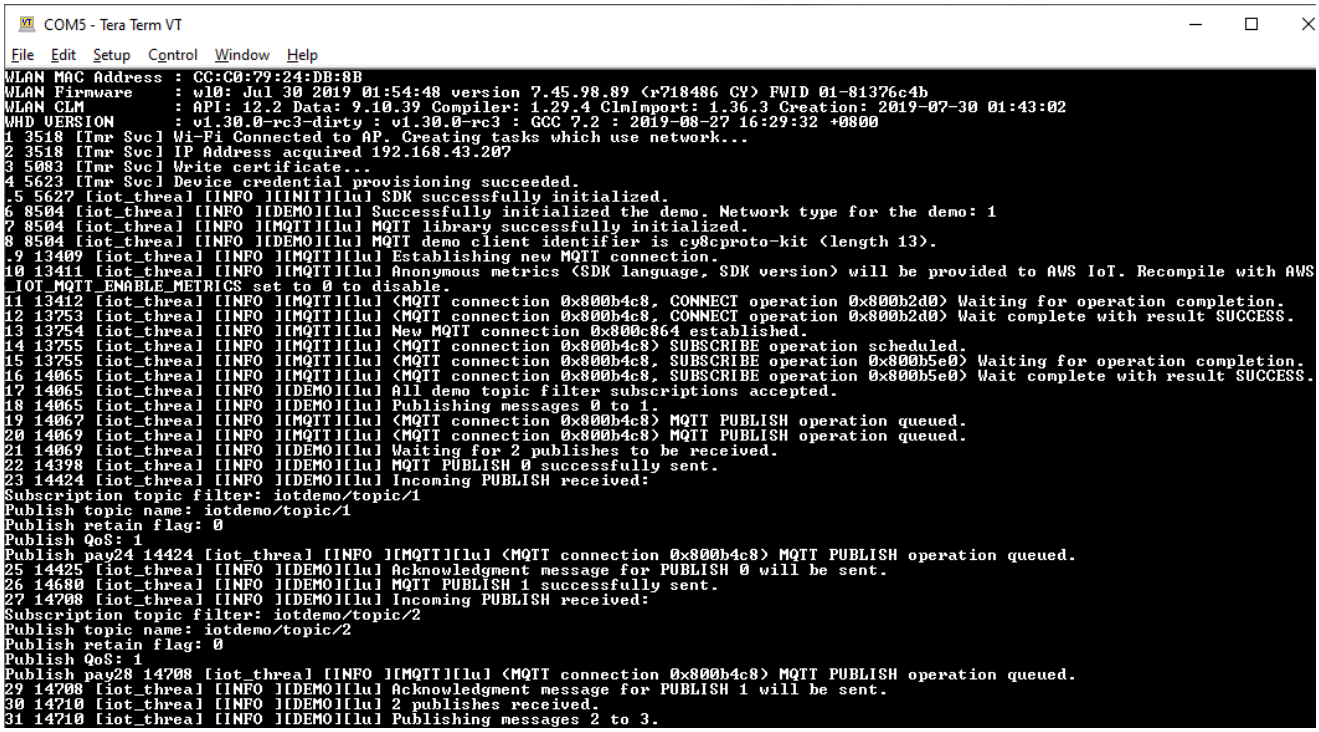
確保專案編譯時沒有錯誤。

3. 執行 FreeRTOS 示範專案

a. 在工作區aws_demos中選取專案。

b. 從快速面板中，選取 aws_demos Program (KitProg3)。這會在程式設計完成後，開始為電路板和示範應用程式進行程式設計。

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。



MQTT 示範會在四個不同的主題 (iotdemo/topic/*n*，其中 n=1 到 4) 上發佈訊息，並訂閱所有這些主題以接收相同的訊息。收到訊息時，示範會在主題 上發佈確認訊息iotdemo/acknowledgements。下列清單說明終端輸出中顯示的偵錯訊息，並參考訊息的序號。在輸出中，會先列印 WICED 主機驅動程式 (WHD) 驅動程式詳細資訊，而不需序列編號。

1. 1 到 4 – 裝置連線至設定的存取點 (AP)，並使用設定的端點和憑證連線至 AWS 伺服器來佈建。
2. 5 到 13 – coreMQTT 程式庫已初始化，裝置會建立 MQTT 連線。
3. 14 到 17 – 裝置訂閱所有主題，以接收已發佈的訊息。
4. 18 到 30 – 裝置發佈兩個訊息，並等待接收回訊息。收到每個訊息時，裝置會傳送確認訊息。

發佈、接收和確認的相同週期會持續到所有訊息發佈為止。每個週期會發佈兩則訊息，直到設定的週期數完成為止。

4. 搭配 FreeRTOS 使用 CMake

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，請參閱 先決條件。

a. 使用下列命令來產生建置檔案。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
```

如果您使用的是 Windows，則必須使用 -G選項指定原生建置系統，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
```

即使 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，使用 CMake 建置專案。

```
cmake --build build_dir
```

c. 最後，使用 Cypress Programmer *build_dir*編寫 cm0.hex和 下產生的cm4.hex檔案。

5. 監控雲端的 MQTT 訊息

您可以使用 AWS IoT 主控台中的 MQTT 用戶端來監控裝置傳送至 AWS Cloud 的訊息。若要使用 MQTT 用戶端訂閱 AWS IoT MQTT 主題，請遵循下列步驟。

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. 針對訂閱主題，輸入 ***your-thing-name*/example/topic**，然後選擇訂閱主題。

d. 重設套件以強制其發佈 MQTT 訊息，然後可在主控台測試用戶端上查看。

執行其他示範

下列示範應用程式已經過測試和驗證，可與目前的版本搭配使用。您可以在 *freertos*/demos目錄下找到這些示範。如需如何執行這些示範的詳細資訊，請參閱 FreeRTOS 示範。

- 低功耗藍牙示範
- Over-the-Air更新示範
- Secure Sockets Echo 用戶端示範
- AWS IoT 裝置影子示範

除錯

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，請在工作區中選取 aws_demos 專案，然後從 Quick Panel 中選取 aws_demos Debug (KitProg3)。

OTA 更新

PSoC 64 MCUs 已通過所有必要的 FreeRTOS 資格測試。不過，在 PSoC 64 標準安全 AWS 韌體程式庫中實作的選用over-the-air(OTA) 功能仍在等待評估。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。

當成功驗證的 OTA 映像使用 PSoC64 標準安全 – AWS MCU 套用至裝置，且裝置無法與之通訊時 AWS IoT Core，裝置就無法自動復原至原始已知的良好映像。這可能會導致無法從 連線裝置 AWS IoT Core 以進行進一步更新。Cypress 團隊仍在開發此功能。

如需詳細資訊，請參閱 的 OTA 更新 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

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

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，您必須設定 AWS IoT 和 FreeRTOS 下載，將您的裝置連線至 AWS 雲端。在本教學課程中，FreeRTOS 下載目錄的路徑稱為 *freertos*。

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 示範應用程式交叉編譯至二進位映像。
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。

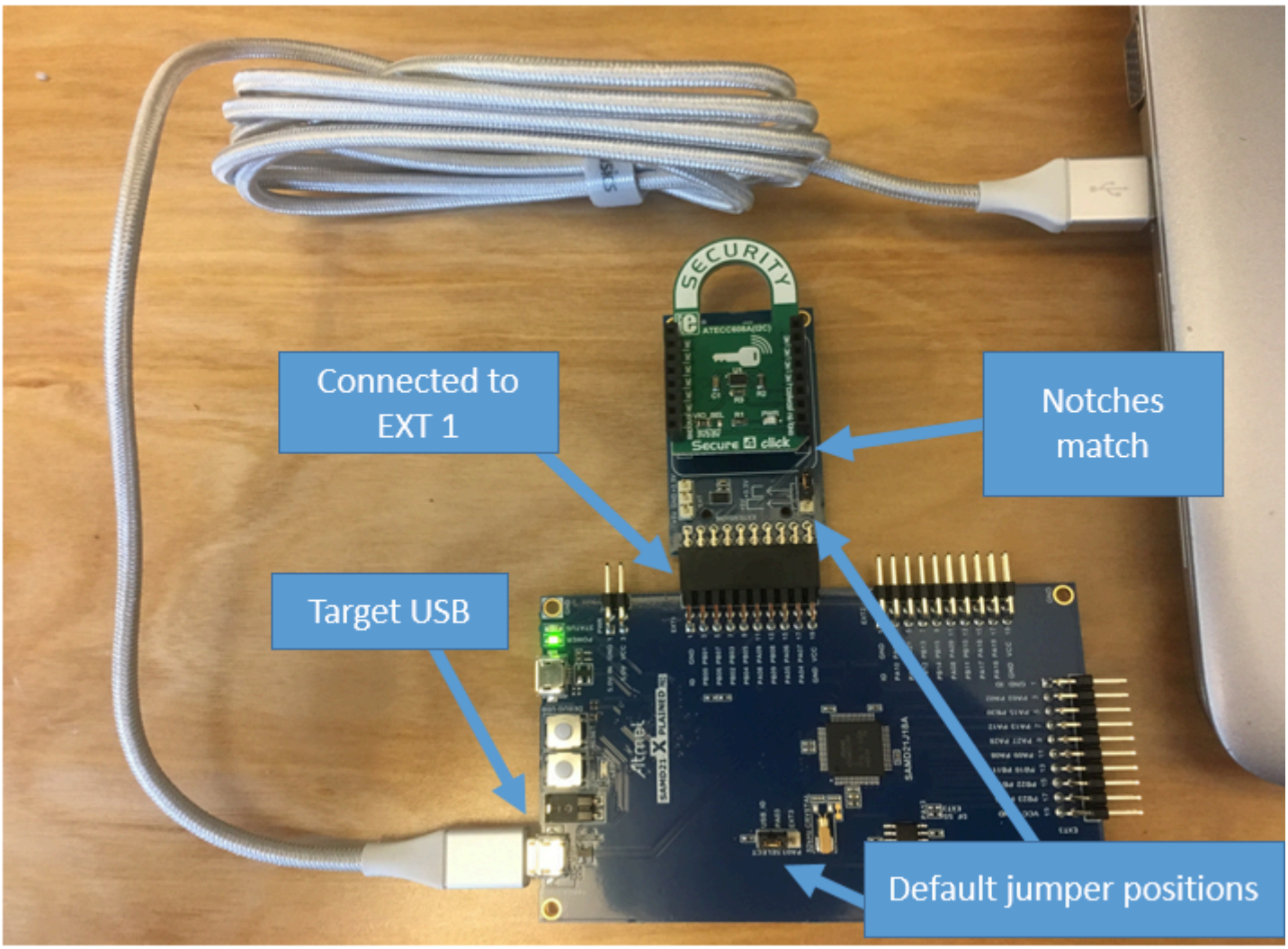
設定 SAMD21 XPlained Pro 面板

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

註冊 AWS 帳戶

如果您沒有 AWS 帳戶，請完成下列步驟來建立一個。

註冊 AWS 帳戶

1. 開啟 https://portal.aws.amazon.com/billing/signup。
2. 請遵循線上指示進行。

   註冊程序的一部分包括接聽電話或文字訊息，並在電話鍵盤上輸入驗證碼。

當您註冊 時 AWS 帳戶，AWS 帳戶根使用者會建立 。根使用者有權存取該帳戶中的所有 AWS 服務 和資源。作為安全最佳實務，請將管理存取權指派給使用者，並且僅使用根使用者來執行需要根使用者存取權的任務。

AWS 會在註冊程序完成後傳送確認電子郵件給您。您可以隨時登錄 https://aws.amazon.com/ 並選擇我的帳戶，以檢視您目前的帳戶活動並管理帳戶。

建立具有管理存取權的使用者

註冊 後 AWS 帳戶，請保護 AWS 帳戶根使用者、啟用 AWS IAM Identity Center和建立 管理使用者，以免將根使用者用於日常任務。

保護您的 AWS 帳戶根使用者

1. 選擇根使用者並輸入 AWS 帳戶 您的電子郵件地址，以帳戶擁有者AWS Management Console身分登入 。在下一頁中，輸入您的密碼。

   如需使用根使用者登入的說明，請參閱 AWS 登入 使用者指南中的以根使用者身分登入。
2. 若要在您的根使用者帳戶上啟用多重要素驗證 (MFA)。

   如需說明，請參閱《IAM 使用者指南》中的為您的 AWS 帳戶 根使用者 （主控台） 啟用虛擬 MFA 裝置。

建立具有管理存取權的使用者

1. 啟用 IAM Identity Center。

   如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的啟用 AWS IAM Identity Center。
2. 在 IAM Identity Center 中，將管理存取權授予使用者。

   如需使用 IAM Identity Center 目錄 做為身分來源的教學課程，請參閱AWS IAM Identity Center《 使用者指南》中的使用預設值設定使用者存取 IAM Identity Center 目錄。

以具有管理存取權的使用者身分登入

- 若要使用您的 IAM Identity Center 使用者簽署，請使用建立 IAM Identity Center 使用者時傳送至您電子郵件地址的簽署 URL。

如需使用 IAM Identity Center 使用者登入的說明，請參閱AWS 登入 《 使用者指南》中的登入 AWS 存取入口網站。

指派存取權給其他使用者

1. 在 IAM Identity Center 中，建立一個許可集來遵循套用最低權限的最佳實務。

   如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的建立許可集。
2. 將使用者指派至群組，然後對該群組指派單一登入存取權。

   如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的新增群組。

若要提供存取權，請新增權限至您的使用者、群組或角色：

- 中的使用者和群組 AWS IAM Identity Center：

  建立權限合集。請按照 AWS IAM Identity Center 使用者指南 中的 建立權限合集 說明進行操作。
- 透過身分提供者在 IAM 中管理的使用者：

  建立聯合身分的角色。遵循「IAM 使用者指南」的為第三方身分提供者 (聯合) 建立角色中的指示。
- IAM 使用者：
  - 建立您的使用者可擔任的角色。請按照「IAM 使用者指南」的為 IAM 使用者建立角色中的指示。
  - (不建議) 將政策直接附加至使用者，或將使用者新增至使用者群組。請遵循 IAM 使用者指南的新增許可到使用者 (主控台) 中的指示。

設定

1. 若要從 GitHub 下載 FreeRTOS：

   1. 瀏覽至 FreeRTOS GitHub 儲存庫。
   2. 選擇程式碼，然後選擇複製或下載 ZIP。
   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 下載目錄的路徑稱為 *freertos*。
> - *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。長 FreeRTOS 下載目錄路徑可能會導致建置失敗。

4. 從 *freertos* 目錄中，查看要使用的分支。

2. 設定開發環境。

   a. 安裝最新版本的 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 建置和執行 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。

   從建置功能表中，選擇建置解決方案，並確保解決方案建置無誤。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。

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。

Espressif ESP32-DevKitC 和 ESP-WROVER-KIT 入門

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

遵循本教學課程，開始使用配備 ESP32-WROOM-32, ESP32-SOLO-132-WROOM-32、ESP32-SOLO-1 或 ESP-WROVER 模組的 Espressif ESP32-DevKitC，以及 ESP-WROVER-KIT-VB。若要從合作夥伴裝置目錄中的 AWS 合作夥伴購買，請使用以下連結：

- ESP32-WROOM-32 DevKitC
- ESP32-SOLO-1
- ESP32-WROVER-KIT

FreeRTOS 支援這些版本的開發板。

如需這些主機板最新版本的詳細資訊，請參閱 Espressif 網站上的 ESP32-DevKitC V4 或 ESP-WROVER-KIT v4.1。

> **Note**
>
> 目前，ESP32-WROVER-KIT 和 ESP DevKitC 的 FreeRTOS 連接埠不支援對稱多處理 (SMP) 功能。

概觀

本教學課程將指引您完成下列步驟：

1. 將電路板連接至主機
2. 在主機機器上安裝軟體，以開發和偵錯微型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至二進位映像
4. 將應用程式二進位映像載入您的電路板，然後執行應用程式
5. 透過序列連線與在電路板上執行的應用程式互動，以用於監控和偵錯

先決條件

在 Espressif 電路板上開始使用 FreeRTOS 之前，您必須設定 AWS 您的帳戶和許可。

註冊 AWS 帳戶

如果您沒有 AWS 帳戶，請完成下列步驟來建立一個。

註冊 AWS 帳戶

1. 開啟 https://portal.aws.amazon.com/billing/signup。
2. 請遵循線上指示進行。

   註冊程序的一部分包括接聽電話或文字訊息，並在電話鍵盤上輸入驗證碼。

   當您註冊 時 AWS 帳戶，AWS 帳戶根使用者會建立 。根使用者有權存取該帳戶中的所有 AWS 服務 和資源。作為安全最佳實務，請將管理存取權指派給使用者，並且僅使用根使用者來執行需要根使用者存取權的任務。

AWS 會在註冊程序完成後傳送確認電子郵件給您。您可以隨時登錄 https://aws.amazon.com/ 並選擇我的帳戶，以檢視您目前的帳戶活動並管理帳戶。

建立具有管理存取權的使用者

註冊 後 AWS 帳戶，請保護 AWS 帳戶根使用者、啟用 AWS IAM Identity Center和建立管理使用者，以免將根使用者用於日常任務。

保護您的 AWS 帳戶根使用者

1. 選擇根使用者並輸入 AWS 帳戶 您的電子郵件地址，以帳戶擁有者AWS Management Console身分登入 。在下一頁中，輸入您的密碼。

   如需使用根使用者登入的說明，請參閱 AWS 登入 使用者指南中的以根使用者身分登入。
2. 若要在您的根使用者帳戶上啟用多重要素驗證 (MFA)。

   如需說明，請參閱《IAM 使用者指南》中的為您的 AWS 帳戶 根使用者 （主控台） 啟用虛擬 MFA 裝置。

建立具有管理存取權的使用者

1. 啟用 IAM Identity Center。

如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的啟用 AWS IAM Identity Center。

2. 在 IAM Identity Center 中，將管理存取權授予使用者。

   如需使用 IAM Identity Center 目錄 做為身分來源的教學課程，請參閱AWS IAM Identity Center 《 使用者指南》中的使用預設值設定使用者存取權 IAM Identity Center 目錄。

以具有管理存取權的使用者身分登入

- 若要使用您的 IAM Identity Center 使用者簽署，請使用建立 IAM Identity Center 使用者時傳送至您電子郵件地址的簽署 URL。

  如需使用 IAM Identity Center 使用者登入的說明，請參閱AWS 登入 《 使用者指南》中的登入 AWS 存取入口網站。

指派存取權給其他使用者

1. 在 IAM Identity Center 中，建立一個許可集來遵循套用最低權限的最佳實務。

   如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的建立許可集。

2. 將使用者指派至群組，然後對該群組指派單一登入存取權。

   如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的新增群組。

若要提供存取權，請新增權限至您的使用者、群組或角色：

- 中的使用者和群組 AWS IAM Identity Center：

  建立權限合集。請按照 AWS IAM Identity Center 使用者指南 中的 建立權限合集 說明進行操作。

- 透過身分提供者在 IAM 中管理的使用者：

  建立聯合身分的角色。遵循「IAM 使用者指南」的為第三方身分提供者 (聯合) 建立角色中的指示。

- IAM 使用者：
  - 建立您的使用者可擔任的角色。請按照「IAM 使用者指南」的為 IAM 使用者建立角色中的指示。
  - (不建議) 將政策直接附加至使用者，或將使用者新增至使用者群組。請遵循 IAM 使用者指南的新增許可到使用者 (主控台) 中的指示。

> **Note**
>
> FreeRTOS 202012.00 版本支援 ESP-IDF v3.3。如果您使用的是較新版本的 FreeRTOS，請遵循 IDFv4.2 的特定說明，網址為 ESP-IDF 4.2 版入門。

ESP-IDF 3.3 版入門

本節說明如何在 Espressif 硬體上使用 ESP-IDF v3.3。若要使用 ESP-IDF 4.2 版，請參閱 ESP-IDF 4.2 版入門。

設定 Espressif 硬體

如需設定 ESP32-DevKitC 開發板硬體的詳細資訊，請參閱 ESP32-DevKitC V4 入門指南。

如需設定 ESP-WROVER-KIT 開發板硬體的相關資訊，請參閱 ESP-WROVER-KIT V4.1 入門指南。

> **Important**
>
> 當您到達 Espressif 指南的入門區段時，請停止，然後遵循以下步驟。

設定開發環境

若要與您的主機板通訊，您必須下載並安裝工具鏈。

設定工具鏈

> **Note**
>
> ESP-IDF v3.3 不支援最新版本的 ESP32 編譯器。如果您已安裝最新版本的 ESP32 編譯器，則應將其解除安裝，並使用工具鏈安裝中包含的相容版本編譯器。請參閱本節中的連結。若要檢查編譯器的版本，請執行下列命令。
>
> ```
> xtensa-esp32-elf-gcc --version
> ```

若要設定工具鏈，請遵循主機機器作業系統的下列指示：

- 適用於 Windows 的工具鏈標準設定 (ESP-IDF v3.3)

- macOS 的工具鏈標準設定 (ESP-IDF v3.3)
- Linux 的工具鏈標準設定 (ESP-IDF v3.3)

> **Important**
>
> 當您達到後續步驟下的「取得 ESP-IDF」指示時，請停止，然後返回此頁面的指示。
> 如果您遵循後續步驟下的「取得 ESP-IDF」指示，現在或之前，您應該從系統清除`IDF_PATH`環境變數，然後再繼續。如果您遵循「取得 ESP-IDF」指示，則會自動設定此環境變數。

安裝 CMake

需要 CMake 建置系統才能建置此裝置的 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.org 下載最新版的 CMake。同時提供來源和二進位發佈。

如需搭配 FreeRTOS 使用 CMake 的詳細資訊，請參閱 搭配 FreeRTOS 使用 CMake 。

建立序列連線

若要在您的主機與 ESP32-DevKitC 之間建立序列連線，您必須安裝 CP210x USB to UART Bridge VCP 驅動程式。您可以從 Silicon Labs 下載這些驅動程式。

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。

下載並設定 FreeRTOS

設定環境後，您可以從 GitHub 下載 FreeRTOS。如需說明，請參閱 GitHub 網站上的 README.md 檔案。

設定 FreeRTOS 示範應用程式

1. 如果您正在執行 macOS 或 Linux，請開啟終端機提示。如果您正在執行 Windows，請開啟 `mingw32.exe`。MinGW 是原生 Microsoft Windows 應用程式的最小開發環境。

2. 若要確認您已安裝 Python 2.7.10 或更新版本，請執行

```
python --version
```

安裝的版本即會顯示。如果您沒有安裝 Python 2.7.10 或更新版本，您可以從 Python 網站安裝。

3. 您需要 AWS Command Line Interface (AWS CLI) 來執行 AWS IoT 命令。如果您正在執行 Windows，請使用 在 AWS CLI mingw32 環境中easy_install awscli安裝 。

如果您正在執行 macOS 或 Linux，請參閱安裝 AWS CLI。

4. AWS CLI 使用您的 AWS 存取金鑰 ID、私密存取金鑰和 AWS 區域執行aws configure和設定 。如需詳細資訊，請參閱設定 AWS CLI。

5. 使用下列命令來安裝適用於 Python (boto3) 的 AWS SDK：

- 在 Windows 的 mingw32 環境中，執行

```
easy_install boto3
```

- 在 macOS 或 Linux 上執行

```
pip install tornado nose --user
```

然後執行

```
pip install boto3 --user
```

FreeRTOS 包含`SetupAWS.py`指令碼，可讓您更輕鬆地設定要連線的 Espressif 電路板 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`

電腦上 *`freertos`* 目錄的完整路徑。請確定您使用斜線來指定此路徑。

`thing_name`

您要指派給代表電路板之 AWS IoT 物件的名稱。

`wifi_ssid`

您的 Wi-Fi 網路 SSID。

`wifi_password`

您 Wi-Fi 網路的密碼。

`wifi_security`

您 Wi-Fi 網路的安全類型。

以下是有效的安全類型：

- `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. 如果您正在執行 macOS 或 Linux，請開啟終端機提示。如果您正在執行 Windows，請開啟 `mingw32.exe`。
2. 導覽至 *freertos*`/tools/aws_config_quick_start`目錄並執行

```
python SetupAWS.py setup
```

指令碼會執行以下操作：

- 建立 IoT 實物、憑證及政策
- 將 IoT 政策連接到憑證，並將憑證連接到 AWS IoT 實物
- 使用 AWS IoT 端點、Wi-Fi SSID 和登入資料填入`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**
>
> 憑證為硬式編碼，僅供示範之用。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 的詳細資訊`SetupAWS.py`，請參閱 `README.md` *freertos*`/tools/aws_config_quick_start`目錄中的 。

建置、刷新和執行 FreeRTOS 示範專案

您可以使用 CMake 來產生建置檔案、使用 Make 建置應用程式二進位，以及 Espressif 的 IDF 公用程式以刷新您的主機板。

在 Linux 和 MacOS 上建置 FreeRTOS (ESP-IDF 3.3 版)

如果您使用的是 Windows，請參閱 在 Windows 上建置 FreeRTOS (ESP-IDF 3.3 版)。

使用 CMake 來產生建置檔案，然後使用 Make 來建置應用程式。

使用 CMake 來產生示範應用程式的建置檔案

1. 將目錄變更為 FreeRTOS 下載目錄的根目錄。
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) 堆疊做為預設網路堆疊。若要改用 FreeRTOS+TCP 網路堆疊，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，請將以下幾行新增至自訂 Wi-Fi 元件CMakeLists.txt的 CMake 相依性檔案 。
>
> ```
> # 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，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 (ESP-IDF 3.3 版)

在 Windows 上，您必須指定用於 CMake 的建置產生器。若未指定，CMake 則會依預設使用 Visual Studio。Espressif 正式推薦 Ninja 建構系統，因為它適用於Windows、Linux 和 MacOS。您必須在原生 Windows 環境 (例如 cmd 或 PowerShell) 中執行 CMake 命令。不支援在虛擬 Linux 環境 (例如 MSYS2 或 WSL) 中執行 CMake 命令。

使用 CMake 來產生建置檔案，然後使用 Make 來建置應用程式。

使用 CMake 來產生示範應用程式的建置檔案

1. 將目錄變更為 FreeRTOS 下載目錄的根目錄。
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。若要改用 FreeRTOS+TCP 網路堆疊，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，請將以下幾行新增至自訂 Wi-Fi 元件CMakeLists.txt的 CMake 相依性檔案 。

```
# 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 來建置應用程式。

```
ninja
```

或者，使用一般 CMake 界面來建置應用程式。

```
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。

快閃並執行 FreeRTOS (ESP-IDF v3.3)

本節說明如何在 Espressif 硬體上使用 ESP-IDF v3.3。若要使用 ESP-IDF 4.2 版，請參閱 ESP-IDF 4.2 版入門。

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，請前往 *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檔刷新至您的主機板，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
> ```

ESP-IDF 4.2 版入門

本節說明如何在 Espressif 硬體上使用 ESP-IDF v4.2。若要使用 ESP-IDF 3.3 版，請參閱 ESP-IDF 3.3 版入門。

> **Note**
>
> 本教學課程中的 Linux 命令要求您使用 Bash shell。

設定 Espressif 硬體

如需有關設定 ESP32-DevKitC 開發板硬體的資訊，請參閱 ESP32-DevKitC V4 入門指南。

如需有關設定 ESP-WROVER-KIT 開發板硬體的資訊，請參閱 ESP-WROVER-KIT V4.1 入門指南。

> **Important**
>
> 當您到達 Espressif 指南的入門區段時，請停止，然後返回此頁面的指示。

設定開發環境

若要與您的主機板通訊，您必須下載並安裝工具鏈。

設定工具鏈

若要設定工具鏈，請遵循主機機器作業系統的下列指示：

- Windows 工具鏈和先決條件 (ESP-IDF v4.2) 的標準設定
- macOS 的工具鏈先決條件標準設定 (ESP-IDF v4.2)
- Linux 的工具鏈先決條件標準設定 (ESP-IDF v4.2)

> **Important**
> 當您達到後續步驟下的「取得 ESP-IDF」指示時，請停止，然後返回此頁面的指示。

完成安裝 (Linux/MacOS)

ESP-IDF Windows 安裝程式會安裝所有必要工具。Linux 和 MacOS 平台需要額外的步驟來完成安裝。

1. 開啟命令列視窗
2. 導覽至 FreeRTOS 下載目錄，然後執行下列指令碼，為您的平台下載並安裝 espressif 工具鏈。

```
vendors/espressif/esp-idf/install.sh
```

3. 使用下列命令，將 ESP-IDF 工具鏈工具新增至終端機的路徑。

```
source vendors/espressif/esp-idf/export.sh
```

建立序列連線

若要在主機機器與 ESP32-DevKitC 之間建立序列連線，您必須安裝 CP210x USB 至 UART Bridge VCP 驅動程式。您可以從 Silicon Labs 下載這些驅動程式。

若要在主機機器與 ESP32-WROVER-KIT 之間建立序列連線，您必須安裝 FTDI 虛擬 COM 連接埠驅動程式。您可以從 FTDI 下載此驅動程式。

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。

下載並設定 FreeRTOS

設定環境後，您可以從 GitHub 下載 FreeRTOS。如需說明，請參閱 GitHub 網站上的 README.md 檔案。

設定 FreeRTOS 示範應用程式

1. 如果您正在執行 macOS 或 Linux，請開啟終端機提示。如果您正在執行 Windows，請開啟 mingw32.exe。(MinGW 是原生 Microsoft Windows 應用程式的最小開發環境。)
2. 若要確認您已安裝 Python3，請執行

   ```
   python --version
   ```

   安裝的版本即會顯示。如果您沒有安裝 Python 3.0.1 或更新版本，您可以從 Python 網站安裝。
3. 您需要 AWS Command Line Interface (AWS CLI) 來執行 AWS IoT 命令。如果您正在執行 Windows，請使用 在 AWS CLI mingw32 環境中easy_install awscli安裝 。

   如果您正在執行 macOS 或 Linux，請參閱安裝 AWS CLI。
4. 執行

   ```
   aws configure
   ```

   並使用 AWS CLI 您的 AWS 存取金鑰 ID、私密存取金鑰和預設 AWS 區域來設定 。如需詳細資訊，請參閱設定 AWS CLI。
5. 使用下列命令來安裝適用於 Python (boto3) 的 AWS SDK：

   - 在 Windows 的 mingw32 環境中，執行

     ```
     easy_install boto3
     ```

   - 在 macOS 或 Linux 上執行

     ```
     pip install tornado nose --user
     ```

     然後執行

     ```
     pip install boto3 --user
     ```

FreeRTOS 包含SetupAWS.py指令碼，可讓您更輕鬆地設定要連線的 Espressif 電路板 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

電腦上 *freertos* 目錄的完整路徑。請確定您使用斜線來指定此路徑。

thing_name

您要指派給代表電路板之 AWS IoT 物件的名稱。

wifi_ssid

您的 Wi-Fi 網路 SSID。

wifi_password

您 Wi-Fi 網路的密碼。

wifi_security

您 Wi-Fi 網路的安全類型。

以下是有效的安全類型：

- 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. 如果您正在執行 macOS 或 Linux，請開啟終端機提示。如果您正在執行 Windows，請開啟 mingw32.exe 檔案。
2. 導覽至 *freertos*/tools/aws_config_quick_start目錄並執行

```
python SetupAWS.py setup
```

指令碼會執行以下操作：

- 建立 IoT 實物、憑證及政策
- 將 IoT 政策連接到憑證，並將憑證連接到 AWS IoT 實物
- 使用 AWS IoT 端點、Wi-Fi SSID 和登入資料填入`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**
>
> 憑證為硬式編碼，僅供示範之用。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 的詳細資訊`SetupAWS.py`，請參閱 `README.md` *`freertos`*`/tools/aws_config_quick_start`目錄中的 。

使用 idf.py 指令碼建置、刷新和執行 FreeRTOS 示範專案

您可以使用 Espressif 的 IDF 公用程式來產生建置檔案、建置應用程式二進位檔，以及刷新您的主機板。

在 Windows、Linux 和 MacOS (ESP-IDF v4.2) 上建置和刷新 FreeRTOS MacOS

使用 `idf.py`指令碼建置專案，並將二進位檔刷入您的裝置。

> **Note**
>
> 有些設定可能需要搭配 使用連接埠選項`"-p` *`port-name`*`"idf.py`來指定正確的連接埠，如下列範例所示。
>
> ```
> idf.py -p /dev/cu.usbserial-00101301B flash
> ```

建置和刷新專案

1. 導覽至 FreeRTOS 下載目錄的根目錄。
2. 在命令列視窗中，輸入下列命令，將 ESP-IDF 工具新增至終端機的 PATH。

   Windows

   ```
   vendors\espressif\esp-idf\export.bat
   ```

Linux/MacOS

```
source vendors/espressif/esp-idf/export.sh
```

3. 在 build目錄中設定 cmake，並使用下列命令建置韌體映像。

```
idf.py -DVENDOR=espressif -DBOARD=esp32_wrover_kit -DCOMPILER=xtensa-esp32 build
```

您應該會看到類似以下的輸出。

```
Running cmake in directory /path/to/hello_world/build
Executing "cmake -G Ninja --warn-uninitialized /path/to/hello_world"...
Warn about uninitialized values.
-- Found Git: /usr/bin/git (found version "2.17.0")
-- Building empty aws_iot component due to configuration
-- Component names: ...
-- Component paths: ...

... (more lines of build system output)

[527/527] Generating hello-world.bin
esptool.py v2.3.1

Project build complete. To flash, run this command:
../../../components/esptool_py/esptool/esptool.py -p (PORT) -b 921600
 write_flash --flash_mode dio --flash_size detect --flash_freq 40m 0x10000 build/
hello-world.bin  build 0x1000 build/bootloader/bootloader.bin 0x8000 build/
partition_table/partition-table.bin
or run 'idf.py -p PORT flash'
```

如果沒有錯誤，組建將產生韌體二進位 .bin 檔案。

4. 使用以下命令清除開發板的快閃記憶體。

```
idf.py erase_flash
```

5. 使用idf.py指令碼將應用程式二進位檔刷新到您的主機板。

```
idf.py flash
```

6. 使用下列命令來監控主機板序列埠的輸出。

```
idf.py monitor
```

> **Note**
>
> 您可以在下列範例中結合這些命令，例如 。
>
> ```
> idf.py erase_flash flash monitor
> ```

> **Note**
>
> 對於某些主機機器設定，您必須在刷新電路板時指定連接埠，如下列範例所示。
>
> ```
> idf.py erase_flash flash monitor -p /dev/ttyUSB1
> ```

使用 CMake 建置和 Flash FreeRTOS

除了 IDF 開發套件提供的`idf.py`指令碼來建置和執行程式碼之外，您也可以使用 CMake 建置專案。目前，它支援 Unix Makefiles 或 Ninja 組建系統。

建置和刷新專案

1. 在命令列視窗中，導覽至 FreeRTOS 下載目錄的根目錄。
2. 執行下列指令碼，將 ESP-IDF 工具新增至 shell 的 PATH。

   Windows

   ```
   vendors\espressif\esp-idf\export.bat
   ```

   Linux/MacOS

   ```
   source vendors/espressif/esp-idf/export.sh
   ```
3. 輸入下列命令來產生建置檔案。

   使用 Unix Makefiles

```
cmake -DVENDOR=espressif -DBOARD=esp32_wrover_kit -DCOMPILER=xtensa-esp32 -S . -
B ./YOUR_BUILD_DIRECTORY -DAFR_ENABLE_ALL_MODULES=1 -DAFR_ENABLE_TESTS=0
```

使用 Ninja

```
cmake -DVENDOR=espressif -DBOARD=esp32_wrover_kit -DCOMPILER=xtensa-esp32 -S . -
B ./YOUR_BUILD_DIRECTORY -DAFR_ENABLE_ALL_MODULES=1 -DAFR_ENABLE_TESTS=0 -GNinja
```

4. 清除快閃記憶體，然後刷新電路板。

使用 Unix Makefiles

```
make -C ./YOUR_BUILD_DIRECTORY erase_flash
```

```
make -C ./YOUR_BUILD_DIRECTORY flash
```

使用 Ninja

```
ninja -C ./YOUR_BUILD_DIRECTORY erase_flash
```

```
ninja -C ./YOUR_BUILD_DIRECTORY flash
```

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 MQTT 用戶端訂閱 AWS IoT MQTT 主題

1. 導覽至 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 ***your-thing-name*/example/topic**，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。

若要跨低功耗藍牙執行 FreeRTOS 示範專案，您必須在 iOS 或 Android 行動裝置上執行 FreeRTOS 低功耗藍牙行動開發套件示範應用程式。

設定 FreeRTOS 低功耗藍牙行動開發套件示範應用程式

1. 遵循適用於 FreeRTOS 藍牙裝置的 Mobile SDKs 中的指示，在主機電腦上下載並安裝適用於行動平台的 SDK。
2. 遵循 FreeRTOS 低功耗藍牙行動開發套件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。

在 ESP32 的 CMake 專案中使用 FreeRTOS

如果您想要在自己的 CMake 專案中使用 FreeRTOS，您可以將它設定為子目錄，並與應用程式一起建置。首先，從 GitHub 取得 FreeRTOS 的副本。您也可以使用下列命令將其設定為 Git 子模組，以便未來更容易更新。

```
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 -- freertos
```

```
# Commit the submodule change because it is pointing to a different revision now.
git add freertos
```

```
git commit -m "Update FreeRTOS to a new release"
```

如果您的專案具有下列目錄結構：

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

```
- CMakeLists.txt
```

然後，以下是最上層CMakeLists.txt檔案的範例，可用於搭配 FreeRTOS 建置您的應用程式。

ESP-IDF v3.3

```
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)
```

ESP-IDF v4.2

```
cmake_minimum_required(VERSION 3.13)

project(freertos_examples)

# Pull in ESP-IDF defined CMake build infrastructure logic to configure the project,
 discover all the components, etc
set(esp_idf_dir "${CMAKE_CURRENT_LIST_DIR}/freertos/vendors/espressif/esp-idf")
include(${esp_idf_dir}/tools/cmake/idf.cmake)

# Tell IDF build to link against this target.
set(IDF_PROJECT_EXECUTABLE my_app)
get_filename_component(
    IDF_EXECUTABLE_SRCS
    "src/main.c" ABSOLUTE
    )
```

若要建置專案，請執行以下 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
```

若要將應用程式刷新到您的主機板，請執行下列命令。

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

使用 FreeRTOS 中的元件

執行 CMake 之後，您可以在摘要輸出中找到所有可用的元件。它看起來應該類似下列範例。

```
====================Configuration for FreeRTOS====================
  Version:                  201910.00
  Git version:              201910.00-388-gcb3612cb7

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, tinycbor
  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:
===========================================================================
```

您可以從Modules to build清單中參考任何元件。若要將它們連結到您的應用程式，請將AFR::命名空間放在名稱前面，例如 AFR::mqtt、 AFR::ota等。

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，假設您想要新增名為 的元件example_component，而您的專案看起來像這樣

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

以下是 元件的 CMakeLists.txt 檔案範例。

ESP-IDF v3.3

```
# 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/example_component.c)

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

```
register_component()
```

ESP-IDF v4.2

```
# Use idf_component_register to add this component, this will define a CMake library
 target.
# Specify component sources in SRCS.
# Specify component include directories in INCLUDE_DIRS.
idf_component_register(SRCS "src/foo.c"
                       INCLUDE_DIRS "include")
# 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)
```

您還可以使用標準 CMake 函數 `target_link_libraries` 指定相依性。請注意，元件的目標名稱存放在 ESP-IDF 所定義的變數 `COMPONENT_TARGET` 中。

ESP-IDF v3.3

```
# 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)
```

ESP-IDF v4.2

```
# add this component, this will define a CMake library target.
idf_component_register(SRCS "src/foo.c"
                       INCLUDE_DIRS "include")

# 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 元件，這可透過設定兩個變數 COMPONENT_REQUIRES和 來完成COMPONENT_PRIV_REQUIRES。請參閱 ESP-IDF 程式設計指南 v4.2 中的建置系統 (CMake)。

ESP-IDF 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)
```

ESP-IDF v4.2

```
# If the dependencies are from ESP-IDF, use these 2 variables (REQUIRES,
 PRIV_REQUIRES)
#   in the idf_component_register call: (Refer here for information on the
#   idf_component_register function: https://docs.espressif.com/projects/esp-idf/en/
v4.2/esp32/api-guides/build-system.html#cmake-component-register)
idf_component_register(SRCS "src/foo.c"
                       INCLUDE_DIRS "include"
                       REQUIRES log
                       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 之前的任何位置新增以下行add_subdirectory(freertos)。

ESP-IDF v3.3

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

ESP-IDF v4.2

```
# Add some extra components. IDF_EXTRA_COMPONENT_DIRS is a variable used by ESP-IDF
# to collect extra components.
get_filename_component(
```

```
    EXTRA_COMPONENT_DIRS
    "components/example_component" ABSOLUTE
)
idf_build_component(${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

目前沒有明確定義的方法來重新定義 FreeRTOS 來源樹狀目錄以外的組態。依預設，CMake 將尋找 *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/ 目錄。然後，在您的頂層CMakeLists.txt檔案中，在 之前新增此行，add_subdirectory(freertos)讓編譯器先搜尋此目錄。

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

為 ESP-IDF 提供您自己的 sdkconfig

如果你想要提供自己的 sdkconfig.default，則可以從命令行設定 CMake 變數 IDF_SDKCONFIG_DEFAULTS：

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

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

Summary

如果你有一個專案具有名為 example_component 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。

ESP-IDF v3.3

```
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/example_component" ABSOLUTE
)
list(APPEND IDF_EXTRA_COMPONENT_DIRS ${EXTRA_COMPONENT_DIRS})

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

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

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

ESP-IDF v4.2

```
cmake_minimum_required(VERSION 3.13)

project(freertos_examples)
```

```
# Pull in ESP-IDF defined CMake build infrastructure logic to configure the project,
 discover all the components, etc
set(esp_idf_dir "${CMAKE_CURRENT_LIST_DIR}/freertos/vendors/espressif/esp-idf")
include(${esp_idf_dir}/tools/cmake/idf.cmake)
# Tell IDF build to link against this target.
set(IDF_PROJECT_EXECUTABLE my_app)
get_filename_component(
    IDF_EXECUTABLE_SRCS
    "src/main.c" ABSOLUTE
    )
# Add some extra components. IDF_EXTRA_COMPONENT_DIRS is an variable used by ESP-IDF
# to collect extra components.
get_filename_component(
    EXTRA_COMPONENT_DIRS
    "components/foo" ABSOLUTE
)
idf_build_component(${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)
```

如需使用 CMake 建置系統的詳細資訊，請參閱 Espressif API 指南中的建置系統

故障診斷

- 如果您正在執行 macOS 且作業系統無法辨識 ESP-WROVER-KIT，請確定您未安裝 D2XX 驅動程式。若要解除安裝這些驅動程式，請遵循 FTDI Drivers Installation Guide for macOS X 中的說明。
- 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，它可協助您在應用程式停止運作時取得一些有意義的回溯。如需詳細資訊，請參閱 Espressif 網站上的自動地址解碼。
- 您也可以啟用 GDBstub 與 gdb 通訊，而不需要任何特殊的 JTAG 硬體。如需詳細資訊，請參閱 Espressif 網站上的使用 GDBStub 啟動 GDB。

- 如需在需要 JTAG 硬體型偵錯時設定 OpenOCD 型環境的相關資訊，請參閱 Espressif 網站上的 JTAG Debugging。
- 如果pyserial無法在 macOS pip 上使用 安裝 ，請從 pyserial 網站下載。
- 如果電路板持續重設，請嘗試在終端機上輸入下列命令來清除快閃記憶體。

```
make erase_flash
```

- 如果您在執行 idf_monitor.py 時看到錯誤，請使用 Python 2.7。
- 從 ESP-IDF 取得的必要程式庫包含在 FreeRTOS 中，因此不需要從外部下載。如果已設定IDF_PATH環境變數，建議您在建置 FreeRTOS 之前將其清除。
- 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
```

- 如果您的裝置無法連線至 AWS IoT，請開啟 aws_clientcredential.h 檔案，並確認檔案中已正確定義組態變數。 clientcredentialMQTT_BROKER_ENDPOINT[]看起來應該像 *1234567890123*-ats.iot.*us-east-1*.amazonaws.com。
- 如果您遵循 在 ESP32 的 CMake 專案中使用 FreeRTOS 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。若要新增這些項目，請使用標準 CMake 函數 target_link_libraries 更新 CMakeLists.txt 檔案 (在根目錄下)。

- ESP-IDF v4.2 支援使用 xtensa-esp32-elf-gcc 8.2.0. 工具鏈。如果您使用的是舊版 Xtensa 工具鏈，請下載必要的版本。

- 如果您看到有關 ESP-IDF v4.2 未滿足的 Python 相依性的錯誤日誌，如下所示。

```
The following Python requirements are not satisfied:
click>=5.0
pyserial>=3.0
future>=0.15.2
pyparsing>=2.0.3,<2.4.0
pyelftools>=0.22
gdbgui==0.13.2.0
pygdbmi<=0.9.0.2
reedsolo>=1.5.3,<=1.5.4
bitstring>=3.1.6
ecdsa>=0.16.0
```

```
Please follow the instructions found in the "Set up the tools" section of ESP-IDF
 Getting Started Guide
```

使用下列 Python 命令在您的平台上安裝 python 相依性：

```
root/vendors/espressif/esp-idf/requirements.txt
```

如需更多故障診斷資訊，請參閱 故障診斷入門。

除錯

Espressif ESP32-DevKitC 和 ESP-WROVER-KIT (ESP-IDF v3.3) 上的偵錯程式碼

本節說明如何使用 ESP-IDF v3.3 偵錯 Espressif 硬體。若要使用 ESP-IDF 4.2 版進行偵錯，請參閱 Espressif ESP32-DevKitC 和 ESP-WROVER-KIT (ESP-IDF v4.2) 上的偵錯程式碼。

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 FTDI C232HM-DDHSL-0 纜線，這些是 ESP32 DevkitC 的連線。

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

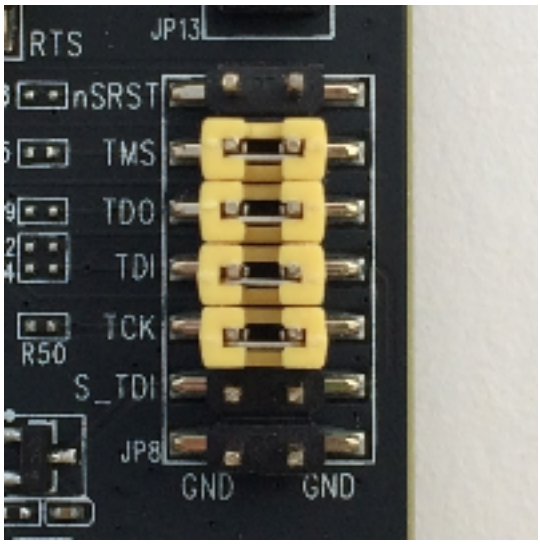
ESP-WROVER-KIT JTAG 設定

對於 FTDI C232HM-DDHSL-0 纜線，這些是 ESP32-WROVER-KIT 的連線。

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
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，請參閱資料表中的「C232HM MPSSE 纜線連接和機械詳細資訊」一節。

若要在 ESP-WROVER-KIT 上啟用 JTAG，請在 TMS、TDO、TDI、TCK 和 S_TDI 接腳上放置跳線，如下所示。



在 Windows 上偵錯 (ESP-IDF v3.3)

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Espressif ESP32-DevKitC 和 ESP-WROVER-KIT (ESP-IDF v3.3) 上的偵錯程式碼中所述進行。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 FTDI C232HM-DDHSL-0 的 Windows 驅動程式。

3. 在 Details (詳細資訊) 索引標籤中，選擇 Property (屬性)，然後選擇 Hardware IDs (硬體 ID)。您應該會在 值 欄位中看到類似的情況。

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

在此範例中，廠商 ID 為 0403，產品 ID 為 6014。

確認這些 ID 符合 `projects/espressif/esp32/make/aws_demos/esp32_devkitj_v1.cfg` 中的 ID。IDs 是在以 開頭的行中指定，`ftdi_vid_pid`後面接著廠商 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. 執行 `zadig.exe`。從 Options (選項) 功能表中，選擇 List All Devices (列出所有裝置)。
   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下的下拉式清單中，選擇箭頭，然後選擇安裝驅動程式。選擇 Replace Driver (取代驅動程式)。
7. 開啟命令提示字元，導覽至 `projects/espressif/esp32/make/aws_demos`並執行下列命令。

   對於 ESP32-WROOM-32 和 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. 開啟新的命令提示，然後導覽至您的 msys32 目錄，並執行 mingw32.exe。
9. 在 mingw32 終端機中，導覽至 projects/espressif/esp32/make/aws_demos並執行

```
make flash monitor
```

10. 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。示範開始執行時，執行 xtensa-esp32-elf-gdb -x gdbinit build/aws_demos.elf。程式應該會在 main 函數中停止。

> **Note**
>
> ESP32 支援最多兩個中斷點。

在 macOS 上偵錯 (ESP-IDF v3.3)

1. 下載適用於 macOS 的 FTDI 驅動程式。
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. 使用下列命令在 macOS libusb上安裝 。

```
brew install libusb
```

5. 使用下列命令卸載序列連接埠驅動程式。

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

6. 如果您執行的 macOS 版本高於 10.9，請使用下列命令來卸載 Apple FTDI 驅動程式。

```
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。它列出連接的 USB 裝置。

   ```
   system_profiler SPUSBDataType
   ```

   的輸出system_profiler應如下所示。

   ```
   DEVICE:

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

8. 開啟 projects/espressif/esp32/make/aws_demos/esp32_devkitj_v1.cfg 檔案。裝置的廠商 ID 和產品 ID 是指定在開頭為 ftdi_vid_pid 的一行中。變更 ID 以符合來自前一個步驟中 system_profiler 輸出的 ID。
9. 開啟終端機視窗，導覽至 projects/espressif/esp32/make/aws_demos，然後使用下列命令來執行 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 序列連接埠驅動程式。

    ```
    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，然後執行下列命令。

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

程式應該會在 `main()` 停止。

Linux 上的偵錯 (ESP-IDF v3.3)

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辨識電路板的 USB 連接埠。您應該會看到類似以下的輸出。

```
$ls -l /dev/ttyUSB*
crw-rw----    1    root    dialout    188,    0    Jul    10    19:04    /dev/
ttyUSB0
crw-rw----    1    root    dialout    188,    1    Jul    10    19:04    /dev/
ttyUSB1
```

4. 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。確定群組擁有者已從 變更為 `dialout` `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。另一個界面會路由至 ESP32 的序列連接埠 (UART)，並用於將程式碼上傳至 ESP32 的快閃記憶體。

5. 在終端機視窗中，導覽至 `projects/espressif/esp32/make/aws_demos`，然後使用下列命令來執行 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
```

6. 開啟另一個終端機，導覽至 `projects/espressif/esp32/make/aws_demos`，然後執行下列命令。

```
make flash monitor
```

7. 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-DevKitC 和 ESP-WROVER-KIT (ESP-IDF v4.2) 上的偵錯程式碼

本節說明如何使用 ESP-IDF v4.2 偵錯 Espressif 硬體。若要使用 ESP-IDF v3.3 進行偵錯，請參閱 Espressif ESP32-DevKitC 和 ESP-WROVER-KIT (ESP-IDF v3.3) 上的偵錯程式碼。

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 FTDI C232HM-DDHSL-0 纜線，這些是 ESP32 DevkitC 的連線。

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

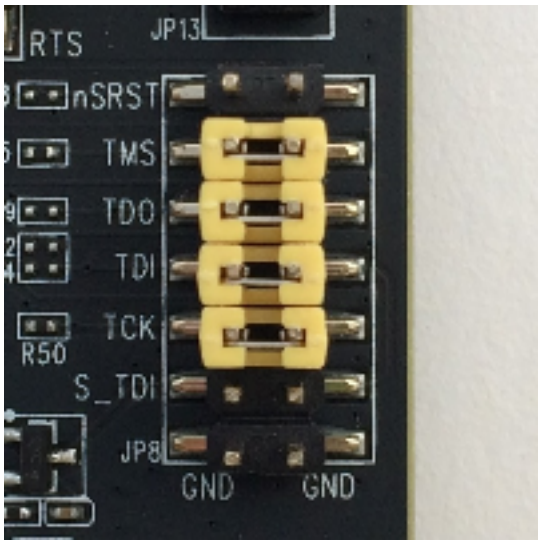
ESP-WROVER-KIT JTAG 設定

對於 FTDI C232HM-DDHSL-0 纜線，這些是 ESP32-WROVER-KIT 的連線。

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
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，請參閱資料表中的「C232HM MPSSE 纜線連接和機械詳細資訊」一節。

若要在 ESP-WROVER-KIT 上啟用 JTAG，請在 TMS、TDO、TDI、TCK 和 S_TDI 接腳上放置跳線，如下所示。



在 Windows 上偵錯 (ESP-IDF v4.2)

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Espressif ESP32-DevKitC 和 ESP-WROVER-KIT (ESP-IDF v4.2) 上的偵錯程式碼中所述進行。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 FTDI C232HM-DDHSL-0 的 Windows 驅動程式。

3. 在 Details (詳細資訊) 索引標籤中，選擇 Property (屬性)，然後選擇 Hardware IDs (硬體 ID)。您應該會在 值 欄位中看到類似的情況。

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

在此範例中，廠商 ID 為 0403，產品 ID 為 6014。

確認這些 ID 符合 `projects/espressif/esp32/make/aws_demos/esp32_devkitj_v1.cfg` 中的 ID。IDs 是在以 開頭的行中指定，`ftdi_vid_pid`後面接著廠商 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. 執行 `zadig.exe`。從 Options (選項) 功能表中，選擇 List All Devices (列出所有裝置)。
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下的下拉式清單中，選擇箭頭，然後選擇安裝驅動程式。選擇 Replace Driver (取代驅動程式)。

7. 開啟命令提示字元，導覽至 FreeRTOS 下載目錄的根目錄，然後執行下列命令。

```
idf.py openocd
```

將此命令提示保持開啟。

8. 開啟新的命令提示字元，導覽至 FreeRTOS 下載目錄的根目錄，然後執行

```
idf.py flash monitor
```

9. 開啟另一個命令提示字元，導覽至 FreeRTOS 下載目錄的根目錄，並等待示範開始在您的電路板上執行。執行時，請執行

```
idf.py gdb
```

程式應該會在 `main` 函數中停止。

> **Note**
>
> ESP32 支援最多兩個中斷點。

在 macOS 上偵錯 (ESP-IDF v4.2)

1. 下載適用於 macOS 的 FTDI 驅動程式。
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. 使用下列命令在 macOS `libusb`上安裝 。

```
brew install libusb
```

5. 使用下列命令卸載序列連接埠驅動程式。

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

6. 如果您執行的 macOS 版本高於 10.9，請使用下列命令來卸載 Apple FTDI 驅動程式。

```
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。它列出連接的 USB 裝置。

   ```
   system_profiler SPUSBDataType
   ```

   的輸出system_profiler應如下所示。

   ```
   DEVICE:

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

8. 開啟 projects/espressif/esp32/make/aws_demos/esp32_devkitj_v1.cfg 檔案。裝置的廠商 ID 和產品 ID 是指定在開頭為 ftdi_vid_pid 的一行中。變更 ID 以符合來自前一個步驟中 system_profiler 輸出的 ID。
9. 開啟終端機視窗，導覽至 FreeRTOS 下載目錄的根目錄，然後使用下列命令來執行 OpenOCD。

   ```
   idf.py openocd
   ```

   保持此終端機視窗開啟。
10. 開啟新的終端機，並使用下列命令載入 FTDI 序列連接埠驅動程式。

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

11. 導覽至 FreeRTOS 下載目錄的根目錄，然後執行

    ```
    idf.py flash monitor
    ```

12. 開啟另一個新的終端機，導覽至 FreeRTOS 下載目錄的根目錄，然後執行

    ```
    idf.py gdb
    ```

    程式應該會在 main 停止。

Linux 上的偵錯 (ESP-IDF v4.2)

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辨識電路板的 USB 連接埠。您應該會看到類似以下的輸出。

```
$ls -l /dev/ttyUSB*
crw-rw----    1    root    dialout    188,    0    Jul    10    19:04    /dev/
ttyUSB0
crw-rw----    1    root    dialout    188,    1    Jul    10    19:04    /dev/
ttyUSB1
```

4. 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。確定群組擁有者已從 變更為 `dialout` `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。另一個界面會路由至 ESP32 的序列連接埠 (UART)，並用於將程式碼上傳至 ESP32 的快閃記憶體。

5. 在終端機視窗中，導覽至 FreeRTOS 下載目錄的根目錄，然後使用下列命令來執行 OpenOCD。

```
idf.py openocd
```

6. 開啟另一個終端機，導覽至 FreeRTOS 下載目錄的根目錄，然後執行下列命令。

```
idf.py flash monitor
```

7. 開啟另一個終端機，導覽 FreeRTOS 下載目錄的根目錄，然後執行下列命令：

```
idf.py gdb
```

   程式應該會在 `main()` 中停止。

Espressif ESP32-WROOM-32SE 入門

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

請按照本教學課程進行，以開始使用 Espressif ESP32-WROOM-32SE。若要從合作夥伴裝置目錄中的 AWS 合作夥伴購買，請參閱 ESP32-WROOM-32SE。

> **Note**
>
> 目前，ESP32-WROOM-32SE 的 FreeRTOS 連接埠不支援對稱多處理 (SMP) 功能。

概觀

本教學課程將指引您完成下列步驟：

1. 將電路板連接至主機
2. 在主機機器上安裝軟體，以開發和偵錯微型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至二進位映像
4. 將應用程式二進位映像載入您的主機板，然後執行應用程式
5. 使用序列連線監控和偵錯執行中的應用程式

先決條件

在 Espressif 電路板上開始使用 FreeRTOS 之前，您必須設定 AWS 您的帳戶和許可。

註冊 AWS 帳戶

如果您沒有 AWS 帳戶，請完成下列步驟來建立一個。

註冊 AWS 帳戶

1. 開啟 https://portal.aws.amazon.com/billing/signup。

2. 請遵循線上指示進行。

   註冊程序的一部分包括接聽電話或文字訊息，並在電話鍵盤上輸入驗證碼。

   當您註冊 時 AWS 帳戶，AWS 帳戶根使用者會建立 。根使用者有權存取該帳戶中的所有 AWS 服務 和資源。作為安全最佳實務，請將管理存取權指派給使用者，並且僅使用根使用者來執行需要根使用者存取權的任務。

AWS 會在註冊程序完成後傳送確認電子郵件給您。您可以隨時登錄 https://aws.amazon.com/ 並選擇我的帳戶，以檢視您目前的帳戶活動並管理帳戶。

建立具有管理存取權的使用者

註冊 後 AWS 帳戶，請保護 AWS 帳戶根使用者、啟用 AWS IAM Identity Center和建立管理使用者，以免將根使用者用於日常任務。

保護您的 AWS 帳戶根使用者

1. 選擇根使用者並輸入 AWS 帳戶 您的電子郵件地址，以帳戶擁有者AWS Management Console身分登入 。在下一頁中，輸入您的密碼。

   如需使用根使用者登入的說明，請參閱 AWS 登入 使用者指南中的以根使用者身分登入。

2. 若要在您的根使用者帳戶上啟用多重要素驗證 (MFA)。

   如需說明，請參閱《IAM 使用者指南》中的為您的 AWS 帳戶 根使用者 （ 主控台 ） 啟用虛擬 MFA 裝置。

建立具有管理存取權的使用者

1. 啟用 IAM Identity Center。

   如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的啟用 AWS IAM Identity Center。

2. 在 IAM Identity Center 中，將管理存取權授予使用者。

   如需使用 IAM Identity Center 目錄 做為身分來源的教學課程，請參閱AWS IAM Identity Center《 使用者指南》中的使用預設值設定使用者存取 IAM Identity Center 目錄。

以具有管理存取權的使用者身分登入

- 若要使用您的 IAM Identity Center 使用者簽署，請使用建立 IAM Identity Center 使用者時傳送至您電子郵件地址的簽署 URL。

  如需使用 IAM Identity Center 使用者登入的說明，請參閱AWS 登入 《 使用者指南》中的登入 AWS 存取入口網站。

指派存取權給其他使用者

1. 在 IAM Identity Center 中，建立一個許可集來遵循套用最低權限的最佳實務。

   如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的建立許可集。
2. 將使用者指派至群組，然後對該群組指派單一登入存取權。

   如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的新增群組。

若要提供存取權，請新增權限至您的使用者、群組或角色：

- 中的使用者和群組 AWS IAM Identity Center：

  建立權限合集。請按照 AWS IAM Identity Center 使用者指南 中的 建立權限合集 說明進行操作。
- 透過身分提供者在 IAM 中管理的使用者：

  建立聯合身分的角色。遵循「IAM 使用者指南」的為第三方身分提供者 (聯合) 建立角色中的指示。
- IAM 使用者：
  - 建立您的使用者可擔任的角色。請按照「IAM 使用者指南」的為 IAM 使用者建立角色中的指示。
  - (不建議) 將政策直接附加至使用者，或將使用者新增至使用者群組。請遵循 IAM 使用者指南的新增許可到使用者 (主控台) 中的指示。

> **Note**
>
> FreeRTOS 202012.00 版本支援 ESP-IDF v3.3。如果您使用的是較新版本的 FreeRTOS，請遵循 中 IDFv4.2 的特定說明ESP-IDF 4.2 版入門。

ESP-IDF 3.3 版入門

本節說明如何在 Espressif 硬體上使用 ESP-IDF v3.3。若要使用 ESP-IDF 4.2 版，請參閱 ESP-IDF 4.2 版入門。

設定 Espressif 硬體

如需設定 ESP32-WROOM-32SE 開發板硬體的相關資訊，請參閱 ESP32-DevKitC 入門指南。
ESP32-WROOM-32SE

> **Important**
>
> 當您到達 Espressif 指南的入門區段時，請停止 ，然後遵循以下步驟。

設定開發環境

若要與您的主機板通訊，您必須下載並安裝工具鏈。

設定工具鏈

> **Note**
>
> ESP-IDF v3.3 (FreeRTOS 使用的版本） 不支援最新版本的 ESP32 編譯器。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
> ```

若要設定工具鏈，請依照您主機作業系統的指示進行：

- 適用於 Windows 的工具鏈標準設定 (ESP-IDF v3.3)
- macOS 的工具鏈標準設定 (ESP-IDF v3.3)
- Linux 的工具鏈標準設定 (ESP-IDF v3.3)

> **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。
> 如果您遵循後續步驟下的「取得 ESP-IDF」指示，現在或先前，您應該從系統清除`IDF_PATH`環境變數，然後再繼續。如果您遵循「取得 ESP-IDF」指示，則會自動設定此環境變數。

安裝 CMake

需要 CMake 建置系統才能建置此裝置的 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.org 下載最新版的 CMake。提供來源和二進位發佈。

如需搭配 FreeRTOS 使用 CMake 的詳細資訊，請參閱 搭配 FreeRTOS 使用 CMake 。

建立序列連線

1. 若要在您的主機與 ESP32-WROOM-32SE 之間建立序列連線，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。

下載並設定 FreeRTOS

設定環境後，您可以從 GitHub 下載 FreeRTOS。如需說明，請參閱 GitHub 網站上的 README.md 檔案。

> **Important**
>
> ATECC608A 裝置具有一次性初始化，會在專案第一次執行時 （呼叫 期間） 鎖定在裝置上`C_InitToken`。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. 依照中的步驟設定 FreeRTOS 示範專案設定 FreeRTOS 示範。略過最後一個步驟 改為格式化您的 AWS IoT 登入資料，並遵循以下步驟。

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，然後開啟 README.md 檔案。
3. 遵循 README.md 檔案中的指示，佈建您的裝置。這些步驟如下：
   a. 使用 建立並註冊憑證授權機構 AWS
   b. 在 ATECC608A 上產生您的金鑰，並匯出公有金鑰和裝置序號
   c. 產生裝置的憑證，並向 註冊該憑證 AWS
4. 按照開發人員模式金鑰佈建的指示，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 來產生建置檔案、使用 Make 建置應用程式二進位，以及 Espressif 的 IDF 公用程式以刷新您的主機板。

在 Linux 或 MacOS 上建置 FreeRTOS (ESP-IDF 3.3 版)

如果您使用的是 Windows，您可以跳到在 Windows 上建置 FreeRTOS (ESP-IDF 3.3 版) 。

使用 CMake 來產生建置檔案，然後使用 Make 來建置應用程式。

使用 CMake 來產生示範應用程式的建置檔案

1. 導覽至 FreeRTOS 下載目錄的根目錄。
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
```

> **備註**
>
> 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 網路堆疊，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，請將以下幾行新增至自訂 Wi-Fi 元件CMakeLists.txt的 CMake 相依性檔案 。

```
# 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. 導覽至 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 (ESP-IDF 3.3 版)

在 Windows 上，您必須指定用於 CMake 的建置產生器。若未指定，CMake 則會依預設使用 Visual Studio。Espressif 官方建議使用 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 PowerShell) 中執行 CMake 命令。不支援在虛擬 Linux 環境 (例如 MSYS2 或 WSL) 中執行 CMake 命令。

使用 CMake 來產生建置檔案，然後使用 Make 來建置應用程式。

使用 CMake 來產生示範應用程式的建置檔案

1. 導覽至 FreeRTOS 下載目錄的根目錄。
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 網路堆疊，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，請將以下幾行新增至自訂 Wi-Fi 元件CMakeLists.txt的 CMake 相依性檔案 。
>
> ```
> # Add a dependency on the bluetooth espressif component to the common
>  component
> set(COMPONENT_REQUIRES lwip)
> ```

建置應用程式

1. 導覽至 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 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。

快閃並執行 FreeRTOS (ESP-IDF v3.3)

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
```

使用下列命令來檢視日誌：

```
./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
> ```

ESP-IDF 4.2 版入門

本節說明如何在 Espressif 硬體上使用 ESP-IDF 4.2 版。若要使用 ESP-IDF 3.3 版，請參閱 ESP-IDF 3.3 版入門。

> **Note**
>
> 本教學課程中的 Linux 命令要求您使用 Bash shell。

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 V4 入門指南。

> **Important**
>
> 當您到達 Espressif 指南的入門區段時，請停止，然後遵循以下步驟。

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 工具鏈和先決條件 (ESP-IDF v4.2) 的標準設定
- macOS 的工具鏈先決條件標準設定 (ESP-IDF v4.2)
- Linux 的工具鏈先決條件標準設定 (ESP-IDF v4.2)

> **Important**
>
> 當您達到後續步驟下的「取得 ESP-IDF」指示時，請停止，然後返回此頁面的指示。

完成安裝 (Linux/Mac)

ESP-IDF Windows 安裝程式會安裝所有必要的工具。Linux 和 MAC OSX 平台需要額外的步驟才能完成安裝。

開啟命令列視窗，導覽至 FreeRTOS 下載目錄，然後執行下列指令碼，為您的平台下載並安裝 espressif 工具鏈。

```
vendors/espressif/esp-idf/install.sh
```

接著，使用下列命令將 ESP-IDF 工具鏈工具新增至終端機的路徑。

```
source vendors/espressif/esp-idf/export.sh
```

建立序列連線

1. 若要在您的主機與 ESP32-WROOM-32SE 之間建立序列連線，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。

下載並設定 FreeRTOS

設定環境後，您可以從 GitHub 下載 FreeRTOS。如需說明，請參閱 GitHub 網站上的 README.md 檔案。

> **Important**
>
> ATECC608A 裝置具有一次性初始化，會在專案第一次執行時（呼叫 期間）鎖定在裝置上`C_InitToken`。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. 依照中的步驟設定 FreeRTOS 示範專案設定 FreeRTOS 示範。當您進入最後一個步驟時，若要格式化您的 AWS IoT 登入資料 STOP，請改為遵循下列步驟。
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。
3. 遵循 `README.md` 檔案中的指示，佈建您的裝置。這些步驟如下：
   a. 使用 建立並註冊憑證授權機構 AWS
   b. 在 ATECC608A 上產生您的金鑰，並匯出公有金鑰和裝置序號
   c. 產生裝置的憑證，並向 註冊該憑證 AWS
4. 按照開發人員模式金鑰佈建的指示，將憑證授權機構憑證和裝置憑證載入至裝置。

使用 idf.py 指令碼建置、刷新和執行 FreeRTOS 示範專案

您可以使用 Espressif 的 IDF 公用程式來產生建置檔案、建置應用程式二進位檔，以及刷新您的主機板。

在 Windows、Linux 和 MacOS (ESP-IDF v4.2) 上建置和刷新 FreeRTOS MacOS

使用 `idf.py`指令碼建置專案，並將二進位檔刷入您的裝置。

> **Note**
>
> 有些設定可能需要搭配 使用連接埠選項"-p *port-name*"idf.py來指定正確的連接埠，如下列範例所示。
>
> ```
> idf.py -p /dev/cu.usbserial-00101301B flash
> ```

建置和刷新專案

1. 導覽至 FreeRTOS 下載目錄的根目錄。
2. 在命令列視窗中，輸入下列命令，將 ESP-IDF 工具新增至終端機的 PATH：

   Windows

   ```
   vendors\espressif\esp-idf\export.bat
   ```

   Linux/MacOS

   ```
   source vendors/espressif/esp-idf/export.sh
   ```

3. 在 build目錄中設定 cmake，並使用下列命令建置韌體映像。

   ```
   idf.py -DVENDOR=espressif -DBOARD=esp32_ecc608a_devkitc -DCOMPILER=xtensa-esp32
    build
   ```

   您應該會看到如下範例的輸出。

   ```
   Running cmake in directory /path/to/hello_world/build
   Executing "cmake -G Ninja --warn-uninitialized /path/to/hello_world"...
   Warn about uninitialized values.
   -- Found Git: /usr/bin/git (found version "2.17.0")
   -- Building empty aws_iot component due to configuration
   -- Component names: ...
   -- Component paths: ...

   ... (more lines of build system output)

   [527/527] Generating hello-world.bin
   ```

```
esptool.py v2.3.1

Project build complete. To flash, run this command:
../../../components/esptool_py/esptool/esptool.py -p (PORT) -b 921600
 write_flash --flash_mode dio --flash_size detect --flash_freq 40m 0x10000 build/
hello-world.bin  build 0x1000 build/bootloader/bootloader.bin 0x8000 build/
partition_table/partition-table.bin
or run 'idf.py -p PORT flash'
```

如果沒有錯誤，組建將產生韌體二進位 .bin 檔案。

4. 使用下列命令清除開發板的快閃記憶體。

```
idf.py erase_flash
```

5. 使用idf.py指令碼將應用程式二進位檔刷新到您的主機板。

```
idf.py flash
```

6. 使用下列命令監控主機板序列連接埠的輸出。

```
idf.py monitor
```

> **Note**
>
> 您可以合併這些命令，如下列範例所示。
>
> ```
> idf.py erase_flash flash monitor
> ```

> **Note**
>
> 對於某些主機機器設定，您必須在刷新電路板時指定連接埠，如下列範例所示。
>
> ```
> idf.py erase_flash flash monitor -p /dev/ttyUSB1
> ```

使用 CMake 建置和 Flash FreeRTOS

除了使用 IDF 開發套件提供的 idf.py 指令碼來建置和執行程式碼，您也可以使用 CMake 建置專案。目前支援 Unix Makefile 和 Ninja 組建系統。

建置和刷新專案

1. 在命令列視窗中，導覽至 FreeRTOS 下載目錄的根目錄。
2. 執行下列指令碼，將 ESP-IDF 工具新增至 shell 的 PATH。

   Windows

   ```
   vendors\espressif\esp-idf\export.bat
   ```

   Linux/MacOS

   ```
   source vendors/espressif/esp-idf/export.sh
   ```
3. 輸入下列命令以產生建置檔案。

   使用 Unix Makefiles

   ```
   cmake -DVENDOR=espressif -DBOARD=esp32_plus_ecc608a_devkitc -DCOMPILER=xtensa-esp32
    -S . -B ./YOUR_BUILD_DIRECTORY -DAFR_ENABLE_ALL_MODULES=1 -DAFR_ENABLE_TESTS=0
   ```

   使用 Ninja

   ```
   cmake -DVENDOR=espressif -DBOARD=esp32_plus_ecc608a_devkitc -DCOMPILER=xtensa-esp32
    -S . -B ./YOUR_BUILD_DIRECTORY -DAFR_ENABLE_ALL_MODULES=1 -DAFR_ENABLE_TESTS=0 -
   GNinja
   ```
4. 清除快閃記憶體，然後刷新電路板。

   使用 Unix Makefiles

   ```
   make -C ./YOUR_BUILD_DIRECTORY erase_flash
   ```

   ```
   make -C ./YOUR_BUILD_DIRECTORY flash
   ```

   使用 Ninja

```
ninja -C ./YOUR_BUILD_DIRECTORY erase_flash
```

```
ninja -C ./YOUR_BUILD_DIRECTORY flash
```

在 AWS 雲端監控 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 MQTT 用戶端訂閱 AWS IoT 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 ***your-thing-name*/example/topic**，然後選擇訂閱主題。

其他資訊

如需使用和故障診斷 Espressif ESP32 電路板的詳細資訊，請參閱下列主題：

- 在 ESP32 的 CMake 專案中使用 FreeRTOS
- 故障診斷
- 除錯

Infineon XMC4800 IoT Connectivity Kit 入門

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

本教學課程提供 Infineon XMC4800 IoT 連線套件入門指示。如果您沒有 Infineon XMC4800 IoT 連線套件，請造訪 AWS Partner Device Catalog 向我們的合作夥伴購買。

如果您想要開啟電路板的序列連接以檢視記錄和除錯資訊，則除了 XMC4800 IoT 連線套件以外，您還需要 3.3V USB/Serial 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，您必須設定 AWS IoT 和 FreeRTOS 下載，將您的裝置連線至 AWS 雲端。如需說明，請參閱 首要步驟。在本教學課程中，FreeRTOS 下載目錄的路徑稱為 *freertos*。

概觀

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. 跨編譯 FreeRTOS 示範應用程式至二進位映像。
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 開發環境來程式設計 XMC4800。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 電路板的內建除錯探查通訊，您需要 J-Link 軟體和文件套件中隨附的驅動程式。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 示範應用程式可在 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 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。

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. 使用 USB 纜線將 XMC4800 IoT 連線套件接到您的電腦。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。
2. 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。
3. 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)。
4. 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)。
5. 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 AWS IoT 主控台中，步驟 4-5 的 MQTT 用戶端應會顯示裝置傳送的 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 CMake 建置 FreeRTOS 示範

如果您不想使用 IDE for FreeRTOS 開發，您也可以使用 CMake 來建置和執行您已開發的示範應用程式，或使用第三方程式碼編輯器和偵錯工具來執行這些應用程式。

> **Note**
>
> 此節說明在 Windows 上透過 MingW 作為原生建置系統以使用 CMake。如需有關搭配其他作業系統和選項以使用 CMake 的詳細資訊，請參閱 搭配 FreeRTOS 使用 CMake 。(MinGW 是本機 Microsoft Windows 應用程式的簡約開發環境。)

使用 CMake 建置 FreeRTOS 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。

   如需詳細說明，請參閱 CMake 先決條件。

3. 建立資料夾以包含產生的建置檔案 (*build-folder*)。

4. 將目錄變更為 FreeRTOS 下載目錄 (*freertos*)，並使用下列命令產生建置檔案：

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

5. 將目錄變更到建置目錄 (*build-folder*)，然後使用下列命令來建置二進位：

   ```
   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 刷新並執行映像。

   a. 使用下列命令以從建置目錄 (*build-folder*) 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 JLNIK 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 Cloud 的訊息。

使用 MQTT 用戶端訂閱 AWS IoT 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 ***your-thing-name*/example/topic**，然後選擇訂閱主題。

故障診斷

如果您尚未進行，請務必設定 AWS IoT 和您的 FreeRTOS 下載，將您的裝置連線至 AWS 雲端。如需說明，請參閱 首要步驟。

如需 FreeRTOS 入門的一般疑難排解資訊，請參閱故障診斷入門。

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

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

本教學提供 Infineon OPTIGA Trust X 安全元素和 XMC4800 IoT Connectivity Kit 的入門說明。與 Infineon XMC4800 IoT Connectivity Kit 入門 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 連線套件，請造訪 AWS 合作夥伴裝置目錄，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。

     請造訪 AWS 合作夥伴裝置目錄，向我們的合作夥伴購買它們。
   - 個人化面板 - Infineon OPTIGA 個人化面板。
   - 轉接器板 - Infineon MyIoT Adapter。

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。（其中一個步驟要求您從主機板的序列偵錯輸出複製公有金鑰，並將其貼到檔案。）若要這樣做，除了 XMC4800 IoT Connectivity Kit 之外，您還需要 3.3V USB/序列轉換器。已知此示範使用 JBtek EL-PN-47310126 US/序列轉換器。您也需要三個male-to-male跳線（用於接收 (RX)、傳輸 (TX) 和接地 (GND)），將序列纜線連接到 Infineon MyIoT 轉接器板。

開始之前，您必須設定 AWS IoT 和 FreeRTOS 下載，將您的裝置連線至 AWS 雲端。如需說明，請參閱 選項 #2：產生內建私有金鑰。在本教學課程中，FreeRTOS 下載目錄的路徑稱為 *freertos*。

概觀

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 示範應用程式交叉編譯至二進位映像。
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 開發環境來程式設計 XMC4800。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 的內建偵錯探查通訊，J-Link 軟體和文件套件中需要包含驅動程式。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/序列轉換器纜線連接至 Infineon Shield2Go Adapter。這可讓面板使用您在開發機器上檢視的格式，傳送記錄和偵錯資訊。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 Cloud 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 MQTT 用戶端訂閱 AWS IoT 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 ***your-thing-name*****/example/topic**，然後選擇訂閱主題。

建置並執行 FreeRTOS 示範專案

將 FreeRTOS 示範匯入 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：產生內建私有金鑰中的擷取公用金鑰步驟。完成所有步驟後，請前往 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 CMake 建置 FreeRTOS 示範

此節說明在 Windows 上透過 MingW 作為原生建置系統以使用 CMake。如需有關搭配其他作業系統和選項以使用 CMake 的詳細資訊，請參閱 搭配 FreeRTOS 使用 CMake 。(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 進行 FreeRTOS 開發，您可以使用 CMake 來建置和執行您已使用第三方程式碼編輯器和偵錯工具開發的示範應用程式。

使用 CMake 建置 FreeRTOS 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**
      >
      > 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。

   b. 開啟下載的工具鏈安裝程式，然後依照精靈中的指示進行。

   c. 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。

   如需詳細說明，請參閱 CMake 先決條件。

3. 建立資料夾以包含產生的建置檔案 (*build-folder*)。

4. 將目錄變更為 FreeRTOS 下載目錄 (*freertos*)，並使用下列命令產生建置檔案：

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

5. 將目錄變更到建置目錄 (*build-folder*)，然後使用下列命令來建置二進位：

```
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 刷新並執行映像。

   a. 使用下列命令以從建置目錄 (*build-folder*) 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 JLNIK 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看到應用程式日誌。繼續執行選項 #2：產生內建私有金鑰中的擷取公用金鑰步驟。完成所有步驟後，請前往 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。

故障診斷

如需一般疑難排解資訊，請參閱故障診斷入門。

MW32x AWS IoT 入門套件入門

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

AWS IoT 入門套件是以 NXP 最新整合的 Cortex M4 微控制器 88MW320/88MW322 為基礎的開發套件，該微控制器在單一微控制器晶片上整合 802.11b/g/n Wi-Fi。開發套件已通過 FCC 認證。如需詳細資訊，請參閱AWS 合作夥伴裝置目錄，向我們的合作夥伴購買。88MW320/88MW322 模組也經過 FCC 認證，可用於自訂和大量銷售。

本入門指南說明如何在主機電腦上將應用程式與軟體開發套件交叉編譯，然後使用軟體開發套件隨附的工具，將產生的二進位檔案載入主機板。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序列主控台進行偵錯或與其互動。

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，但這些平台並未獲得正式支援。您必須擁有許可，才能在主機平台上安裝軟體。建置 SDK 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 SDK。SDK 利用最新版本的工具鏈來最佳化映像佔用空間，並將更多功能納入較少的空間。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。開發套件已使用無線微控制器示範專案韌體預先更新。

主題

- 設定您的硬體
- 設定開發環境
- 建置並執行 FreeRTOS 示範專案
- 除錯
- 故障診斷

設定您的硬體

使用迷你 USB 轉 USB 纜線，將 MW32x 主機板連接至您的筆記型電腦。將 mini-USB 纜線連接到主機板上唯一的 mini-USB 連接器。您不需要變更跳線器。

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。

此 USB 連線提供下列項目：

- 主控台存取主機板。虛擬 tty/com 連接埠已向開發主機註冊，可用於存取 主控台。

- JTAG 對主機板的存取。這可用於將韌體映像載入或卸載至主機板的 RAM 或快閃記憶體，或用於除錯。

設定開發環境

為了開發目的，最低需求是 ARM 工具鏈和與 SDK 綁定的工具。下列各節提供 ARM 工具鏈設定的詳細資訊。

GNU 工具鏈

軟體開發套件正式支援 GCC Compiler 工具鏈。GNU ARM 的跨編譯工具鏈可在 GNU Arm Embedded Toolchain 4.9-2015-q3-update 中取得。

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 上使用，並且可以從 Makefiles 叫用。Makefiles 也假設 GNU 工具鏈二進位檔的檔案名稱以 開頭`arm-none-eabi-`。

GCC 工具鏈可與 GDB 搭配使用，以使用 OpenOCD 進行偵錯 （ 搭配 SDK)。這提供與 JTAG 界面的軟體。

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。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。請確定目錄名稱中沒有空格。
3. 使用下列命令來解壓縮 檔案。

   ```
   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 /home/*user-name*目錄中`.profile`的檔案結尾附加以下行。

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

> **Note**
>
> 較新的 Ubuntu 分佈可能隨附 GCC Cross Compiler 的 Debian 版本。如果是這樣，您必須移除原生的跨編譯器，並遵循上述設定程序。

使用 Linux 開發主機

您可以使用任何現代 Linux 桌面發行版本，例如 Ubuntu 或 Fedora。不過，我們建議您升級至最新版本。下列步驟已經過驗證，可在 Ubuntu 16.04 上運作，並假設您正在使用該版本。

安裝套件

軟體開發套件包含指令碼，可讓您在新設定 Linux 機器上快速設定開發環境。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、Ncurses、python 和 latex。在本節中，一般目錄名稱會*amzsdk_bundle-x.y.z*指出 AWS SDK 根目錄。實際目錄名稱可能不同。您必須具有根權限。

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

```
./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`指令碼來刷新主機板的 NAND，如下所述。同樣地， `ramload`操作會使用`ramload.py`指令碼，將韌體映像直接從主機複製到微控制器的 RAM，而不會刷新 NAND。

您可以設定 Linux 開發主機來執行 `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**
>
> 您必須以這種方式設定 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 纜線插入 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命令後看到類似下列的訊息。

```
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，此名稱名為 "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，以在設定模式中執行 minicom。

```
minicom –s
```

在 minicom 中，導覽至序列連接埠設定並擷取下列設定。

```
| 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 Enter 鍵。這會在畫面上顯示雜湊 (#)。

> **Note**
>
> 開發板包含 FTDI 矽裝置。FTDI 裝置會公開主機的兩個 USB 介面。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 是一種軟體，可為內嵌目標裝置提供偵錯、系統內程式設計和邊界掃描測試。

需有 OpenOCD 0.9 版。Eclipse 功能也需要此功能。如果您的 Linux 主機上安裝了舊版，例如 0.7 版，請使用您目前使用的 Linux 發行版本適用的命令來移除該儲存庫。

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 原始碼。

安裝 OpenOCD

1. 執行下列命令來安裝 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 並導覽至您解壓縮它的目錄。
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 opencd。

   ```
   make install
   ```

設定 Eclipse

> **Note**
>
> 本節假設您已完成 中的步驟 避免 sudo

Eclipse 是應用程式開發和偵錯的偏好 IDE。它提供豐富、易於使用的 IDE 與整合偵錯支援，包括執行緒感知偵錯。本節說明支援的所有開發主機的通用 Eclipse 設定。

設定 Eclipse

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

   Eclipse 要求您安裝 JRE。建議您先安裝此 ，但可以在安裝 Eclipse 之後安裝。JRE 版本 (32/64 位元） 必須符合 Eclipse 版本 (32/64 位元)。您可以從 Oracle 網站上的 Java SE Runtime Environment 8 Downloads 下載 JRE。
2. 從 http://www.eclipse.org：// 下載並安裝「適用於 C/C++ 開發人員的 Eclipse IDE」。支援 Eclipse 4.9.0 版或更新版本。安裝只需要您擷取下載的封存。您可以執行平台特定的 Eclipse 可執行檔來啟動應用程式。

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。

本主題涵蓋兩個選項。

佈建中

- 視您使用的是測試或示範應用程式而定，請在下列其中一個檔案中設定佈建資料：
  - `./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 和密碼應以雙引號括住。

使用命令列建置和執行 FreeRTOS 示範

1. 使用以下命令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$
marvell@pe-lt586:amzsdk$ cmake -DVENDOR=marvell -DBOARD=mw300_rd -DCOMPILER=arm-gcc -S . -
Bbuild -DAFR_ENABLE_TESTS=0

====================Configuration for Amazon FreeRTOS====================
  Version:                  v1.2.4
  Git version:              AMZSDK_V1.2.r6.p1-12-gdd17d10

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

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

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

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

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

2. 導覽至建置目錄。

```
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所示相同的輸出：

4. 使用以下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 aws_demos project和 之間切換時，請執行 cmake命令aws_tests project。

5. 將韌體映像寫入開發板的快閃記憶體。重設開發板後，韌體將會執行。您必須先建置 SDK，才能將映像刷新到微控制器。

   a. 在您刷新韌體映像之前，請使用常見元件配置和Boot2 準備開發板的快閃記憶體。使用以下命令。

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

   flashprog 命令會啟動下列項目：

   - 配置 – 系統會先指示 flashprog 公用程式將配置寫入快閃記憶體。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 使用的開機載入器。flashprog 命令也會將開機載入器寫入快閃記憶體。開機載入器的任務是在快閃記憶體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: 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. 韌體會使用 Wi-Fi 晶片組做為其功能，而 Wi-Fi 晶片組本身的韌體也必須存在於快閃記憶體中。您可以使用 flashprog.py公用程式，以與刷新 Boot2 開機載入器和 MCU 韌體相同的方式刷新 Wi-Fi 韌體。使用下列命令來刷新 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日誌。

e. 若要執行測試應用程式，請刷新位於相同目錄的`aws_tests.bin`二進位檔案。

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

您的命令輸出應類似於下圖所示的輸出。

```
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. 在您刷新韌體並重設電路板後，示範應用程式應啟動，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 a2wtm15blvjjs8-ats.iot.us-east-2.amazonaws.com
7 11668 [MQTTEcho] MQTT echo test echoing task created.
8 11961 [MQTTEcho] MQTT Echo demo subscribed to freertos/demos/echo
9 12248 [MQTTEcho] Echo successfully published 'Hello World 0'
10 12591 [Echoing] Message returned with ACK: 'Hello World 0 ACK'
11 17633 [MQTTEcho] Echo successfully published 'Hello World 1'
12 17927 [Echoing] Message returned with ACK: 'Hello World 1 ACK'
13 22953 [MQTTEcho] Echo successfully published 'Hello World 2'
14 23276 [Echoing] Message returned with ACK: 'Hello World 2 ACK'
15 28245 [MQTTEcho] Echo successfully published 'Hello World 3'
16 28575 [Echoing] Message returned with ACK: 'Hello World 3 ACK'
17 33542 [MQTTEcho] Echo successfully published 'Hello World 4'
18 33980 [Echoing] Message returned with ACK: 'Hello World 4 ACK'
19 38823 [MQTTEcho] Echo successfully published 'Hello World 5'
20 39279 [Echoing] Message returned with ACK: 'Hello World 5 ACK'
21 44139 [MQTTEcho] Echo successfully published 'Hello World 6'
22 44501 [Echoing] Message returned with ACK: 'Hello World 6 ACK'
23 49516 [MQTTEcho] Echo successfully published 'Hello World 7'
24 50270 [Echoing] Message returned with ACK: 'Hello World 7 ACK'
25 54796 [MQTTEcho] Echo successfully published 'Hello World 8'
26 55129 [Echoing] Message returned with ACK: 'Hello World 8 ACK'
27 60080 [MQTTEcho] Echo successfully published 'Hello World 9'
28 60389 [Echoing] Message returned with ACK: 'Hello World 9 ACK'
29 65378 [MQTTEcho] Echo successfully published 'Hello World 10'
30 65998 [Echoing] Message returned with ACK: 'Hello World 10 ACK'
31 70658 [MQTTEcho] Echo successfully published 'Hello World 11'
32 70964 [Echoing] Message returned with ACK: 'Hello World 11 ACK'
33 75958 [MQTTEcho] MQTT echo demo finished.
34 75958 [MQTTEcho] ----Demo finished----
```

7. （選用）做為測試映像的替代方法，請使用 flashprog 公用程式，將微型控制器映像直接從主機複製到微型控制器 RAM。映像不會在快閃記憶體中複製，因此在您重新啟動微型控制器後將會遺失。

   將韌體映像載入 SRAM 是更快的操作，因為它會立即啟動執行檔案。此方法主要用於反覆開發。

   使用以下命令將韌體載入 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
   ```

   命令輸出如下圖所示。

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

當命令執行完成時，您應該會看到示範應用程式的日誌。

使用 Eclipse IDE 建置和執行 FreeRTOS 示範

1. 設定 Eclipse 工作區之前，您必須執行 cmake命令。

   執行下列命令以使用 aws_demos Eclipse 專案。

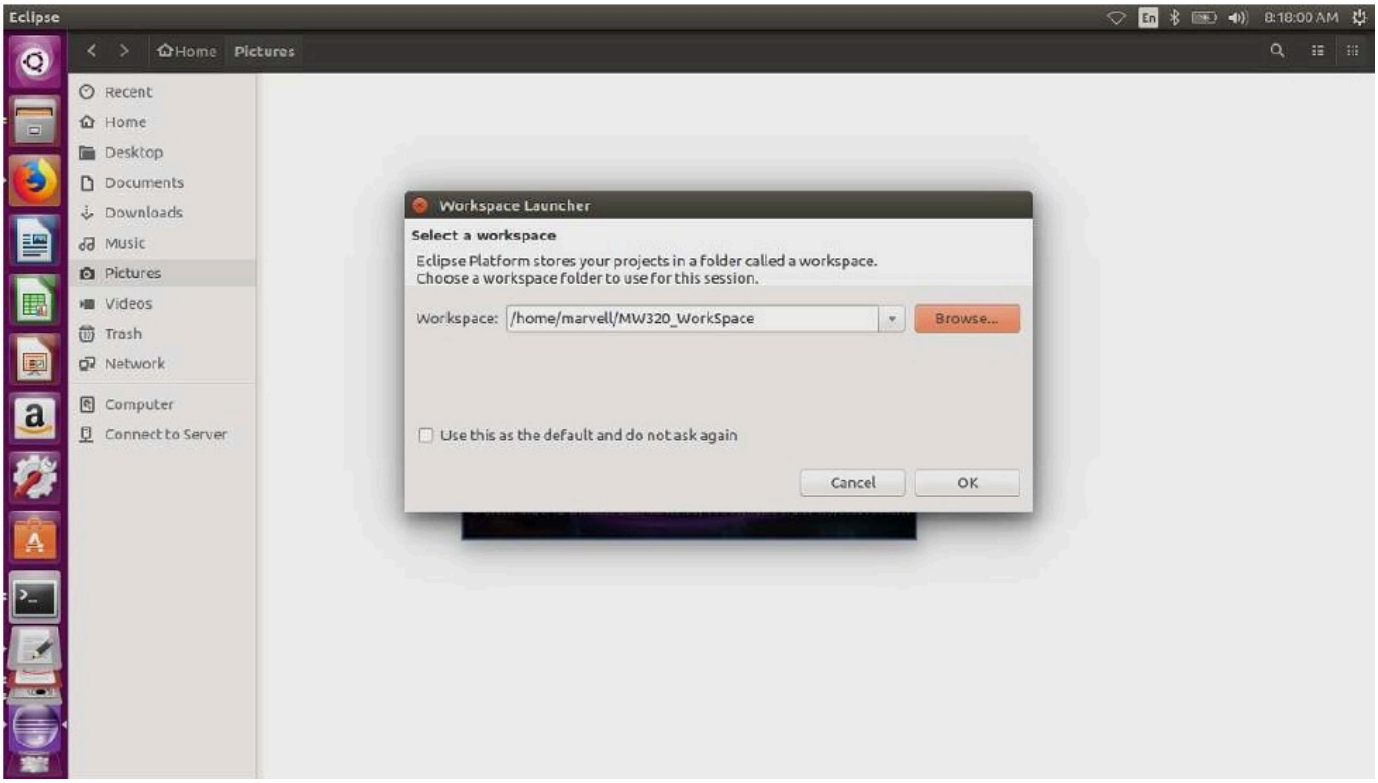
   ```
   cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S .  -Bbuild -
   DAFR_ENABLE_TESTS=0
   ```

   執行下列命令以使用 aws_tests Eclipse 專案。

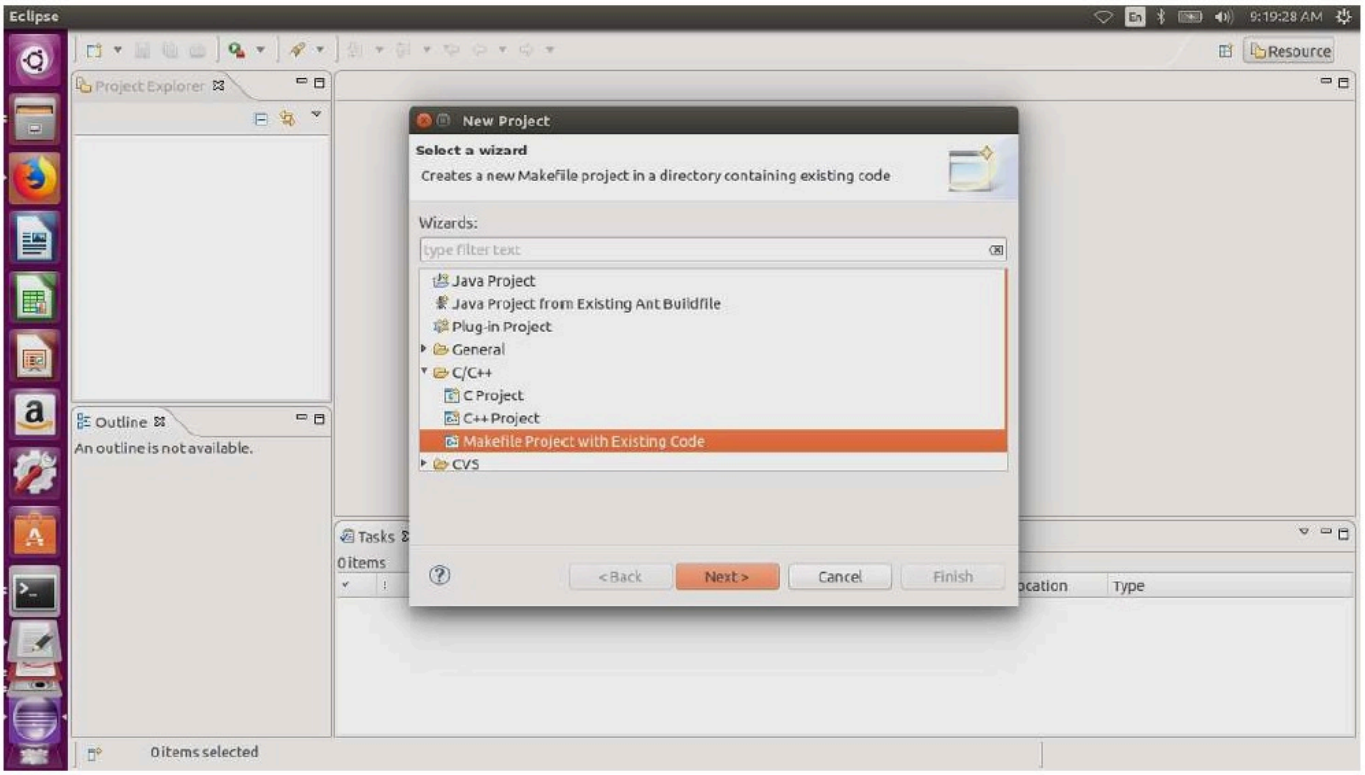
   ```
   cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S .  -Bbuild -
   DAFR_ENABLE_TESTS=1
   ```

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

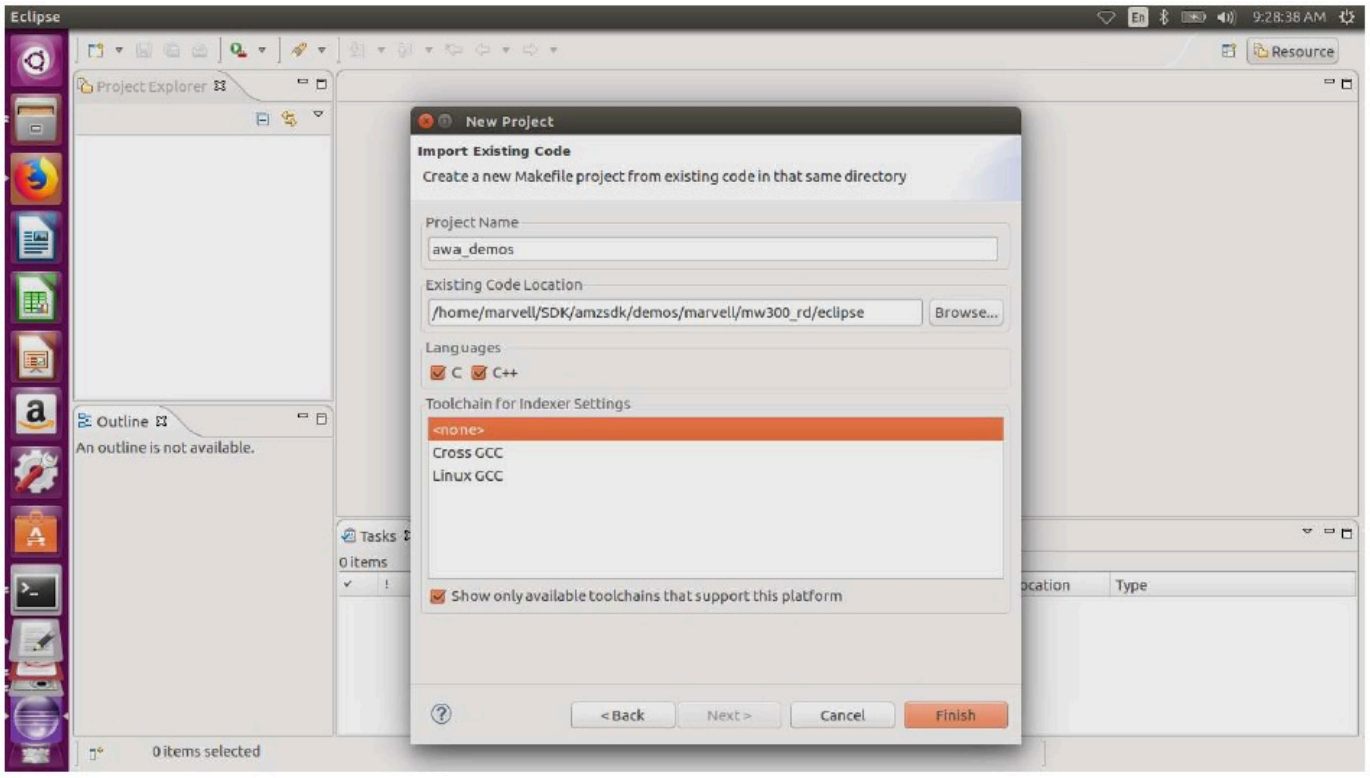
2. 開啟 Eclipse，並在出現提示時選擇您的 Eclipse 工作區，如下圖所示。



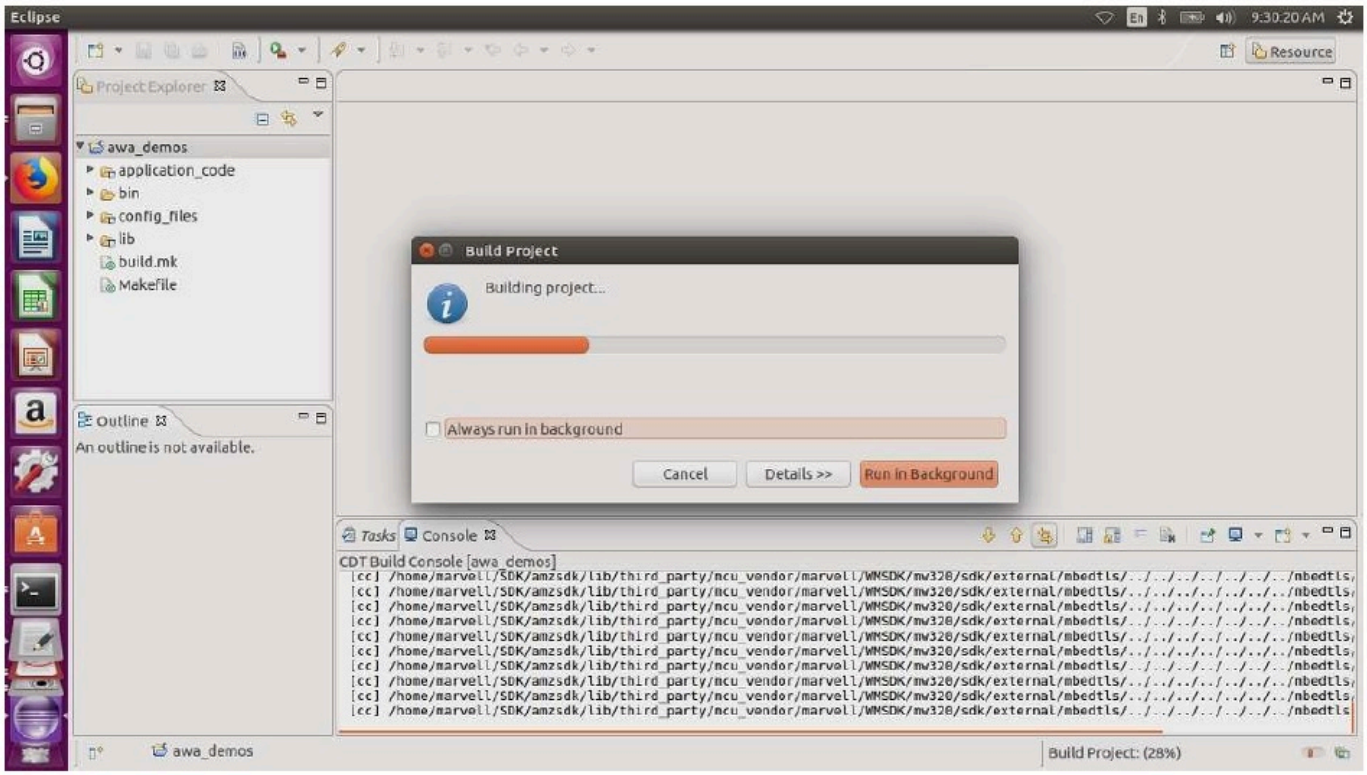
3. 選擇 選項以建立 Makefile 專案：使用現有程式碼，如下圖所示。



4. 選擇瀏覽，指定現有程式碼的目錄，然後選擇完成。



5. 在導覽窗格中，選擇專案總管中的 aws_demos。在 aws_demos 上按一下滑鼠右鍵以開啟選單，然後選擇建置。



如果建置成功，會產生 build/cmake/vendors/marvell/mw300_rd/aws_demos.bin 檔案。

6. 使用命令列工具來刷新配置檔案 (layout.txt)、Boot2 二進位檔 ()boot2.bin、MCU 韌體二進位檔 (aws_demos.bin) 和 Wi-Fi 韌體。

   a. 刷新韌體映像之前，請先使用常用元件、配置和Boot2 準備開發板的快閃記憶體。使用以下命令。

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

   flashprog 命令會啟動下列項目：

   - 配置 – 系統會先指示 flashprog 公用程式將配置寫入快閃記憶體。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 使用的開機載入器。flashprog 命令也會將開機載入器寫入快閃記憶體。開機載入器的任務是在快閃記憶體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: 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. 韌體會使用 Wi-Fi 晶片組做為其功能，而 Wi-Fi 晶片組本身的韌體也必須存在於快閃記憶體中。您可以使用 `flashprog.py`公用程式，以與刷新 boot2 boot-loader 和 MCU 韌體相同的方式刷新 Wi-Fi 韌體。使用下列命令來刷新 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日誌。

e. 若要執行測試應用程式，請刷新位於相同目錄的`aws_tests.bin`二進位檔案。

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

您的命令輸出應類似於下圖所示的輸出。

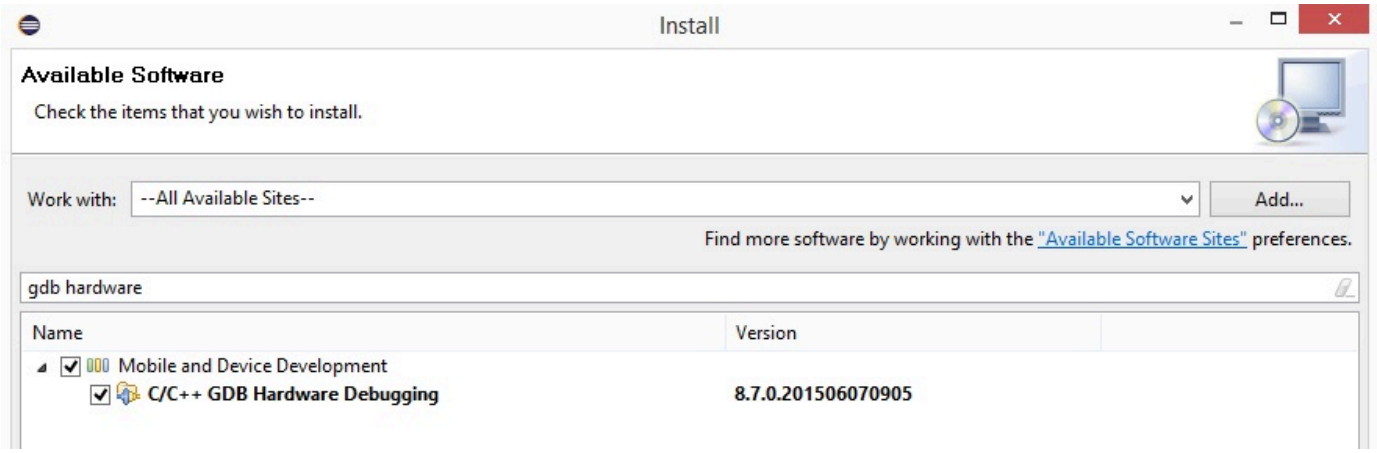
```
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00007f14 msp: 0x20001000
29088 bytes written at address 0x00100000
downloaded 29088 bytes in 0.245499s (115.708 KiB/s)
verified 29088 bytes in 0.350231s (81.107 KiB/s)
semihosting is enabled

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...
```

除錯

- 啟動 Eclipse，然後選擇說明，然後選擇安裝新軟體。在使用功能表中，選擇所有可用的網站。輸入篩選條件文字 `GDB Hardware`。選取 C/C++ GDB 硬體偵錯選項並安裝外掛程式。



故障診斷

網路問題

檢查您的網路憑證。請參閱 中的「佈建」建置並執行 FreeRTOS 示範專案。

啟用其他日誌

- 啟用主機板特定日誌。

  在 `main.c` 檔案中的 函數`wmstdio_init(UART0_ID, 0)`中啟用對 `prvMiscInitialization`的呼叫，以進行測試或示範。

- 啟用 Wi-Fi 日誌

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

使用 GDB

我們建議您使用與 SDK 一起封裝的 `arm-none-eabi-gdb`和 `gdb`命令檔案。導覽至 目錄。

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

執行下列命令（在單行上）以連線至 GDB。

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

MediaTek MT7697Hx 開發套件入門

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

本教學課程提供 MediaTek MT7697Hx 開發套件入門指示。如果您沒有 MediaTek MT7697Hx 開發套件，請造訪 AWS Partner Device Catalog 向我們的合作夥伴購買。

開始之前，您必須設定 AWS IoT 和 FreeRTOS 下載，將您的裝置 AWS 連線至 雲端。如需說明，請參閱 首要步驟。在本教學課程中，FreeRTOS 下載目錄的路徑稱為 *freertos*。

概觀

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. 跨編譯 FreeRTOS 示範應用程式至二進位映像。
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

在設定您的環境之前，請將您的電腦連接至 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 微控制器開發套件 (MDK) 在您的主機板上設定、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。確定您安裝 MediaTek 裝置套件 (MT76x7 系列)。

建立序列連線

若要與 MediaTek MT7697Hx 開發套件建立序列連線，您必須安裝 Arm Mbed Windows 序列連接埠驅動程式。您可以從 Mbed 下載驅動程式。遵循 Windows serial driver (Windows 序列驅動程式) 頁面上的步驟，來下載並安裝 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。如需偵錯，您可以使用終端機公用程式工具 (例如 HyperTerminal 或 TeraTerm)，將工作階段開啟至連接埠。

> **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 建置和執行 FreeRTOS 示範專案

在 Keil µVision 中建置 FreeRTOS 示範專案

1. 從開始功能表中，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. 在目標 '**aws_demo**' 的選項中，選擇偵錯索引標籤。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. 在 MediaTek MT7697Hx 開發套件上，選擇 RESET (重設) 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 Cloud 的訊息。

使用 MQTT 用戶端訂閱 AWS IoT 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 ***your-thing-name*****example/topic**，然後選擇訂閱主題。

故障診斷

在 Keil µVision 中偵錯 FreeRTOS 專案

目前，您必須先編輯 Keil µVision 隨附的 MediaTek 套件，然後才能針對使用 Keil µVision 的 MediaTek 偵錯 FreeRTOS 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 中重建示範專案。
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. 在 MediaTek MT7697Hx 開發套件上，按下 RESET (重設) 按鈕。
9. 從 µVision 功能表中，選擇偵錯，然後選擇開始/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。

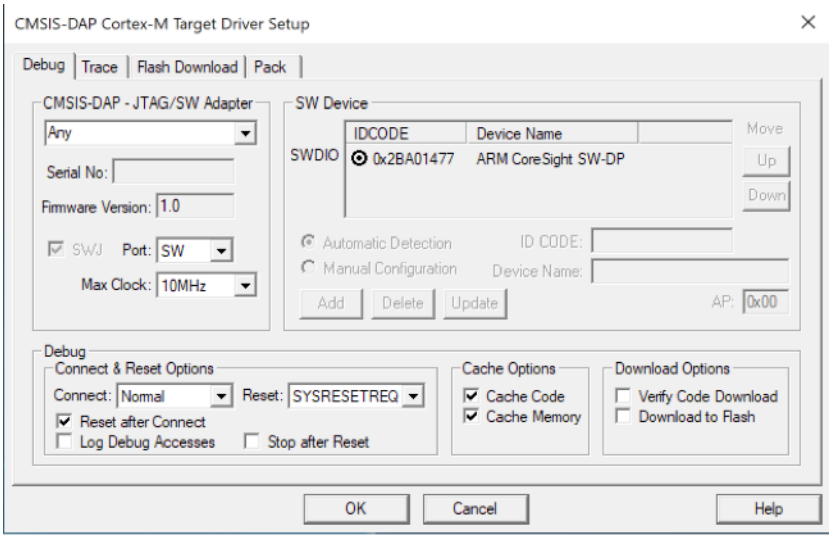
故障診斷 IDE 除錯器設定

如果您無法除錯應用程式，您的除錯器設定可能不正確。

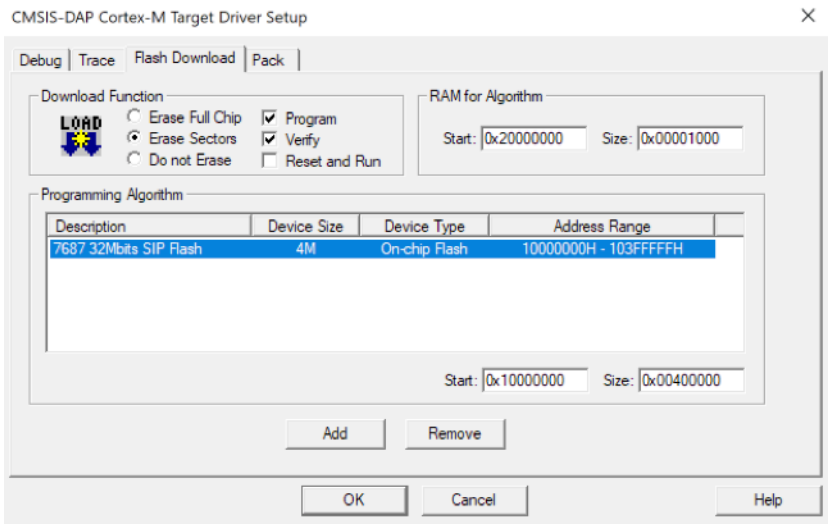
驗證您的除錯器設定是否正確

1. 開啟 Keil µVision。
2. 在aws_demos專案上按一下滑鼠右鍵，選擇選項，然後在公用程式索引標籤下，選擇「-- 使用偵錯驅動程式 --」旁的設定。

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：



如需開始使用 FreeRTOS 的一般疑難排解資訊，請參閱故障診斷入門。

Microchip Curiosity PIC32MZ EF 入門

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

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，您必須設定 AWS IoT 和 FreeRTOS 下載，將您的裝置連線至 AWS 雲端。如需說明，請參閱 首要步驟。

> **Important**
>
> - 在本主題中，FreeRTOS 下載目錄的路徑稱為 *`freertos`*。
> - *`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。長 FreeRTOS 下載目錄路徑可能會導致建置失敗。

概觀

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. 跨編譯 FreeRTOS 示範應用程式至二進位映像。
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 Wi-Fi 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。
   - 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。您也必須定義前置處理器巨集 `PIC32_USE_ETHERNET`。
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 資訊表。

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，您可以依照下列步驟，使用整合式 PICkit On Board (PKOB) 程式設計器/偵錯器，使用基本偵錯來評估和執行 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。您也必須定義前置處理器巨集 `PIC32_USE_ETHERNET`。

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 專案是以 MPLAB Harmony v2 為基礎。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 AWS 參考整合 v202007.00 目前僅支援 MPLabv5.35。MPLabv5.40 支援舊版 FreeRTOS AWS 參考整合。 MPLabv5.40.

   MPLabv5.35 下載

   - 適用於 Windows 的 MPLAB X 整合式開發環境
   - 適用於 macOS 的 MPLAB X 整合式開發環境
   - Linux 的 MPLAB X 整合式開發環境

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
- 停止位元：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。

若要使用乙太網路進行連線，您必須定義前置處理器巨集 `PIC32_USE_ETHERNET`。

使用乙太網路透過 MPLAB IDE 進行連線

1. 在 MPLAB IDE 中，在專案上按一下滑鼠右鍵，然後選擇屬性。
2. 在專案屬性對話方塊中，選擇 ***compiler-name***（全域選項）以展開它，然後選擇 ***compiler-name***-gcc。
3. 針對選項類別，選擇預先處理和訊息，然後將PIC32_USE_ETHERNET字串新增至預先處理器巨集。

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 for FreeRTOS 開發，您也可以使用 CMake 來建置和執行您已開發的示範應用程式，或使用第三方程式碼編輯器和偵錯工具來執行這些應用程式。

使用 CMake 建置 FreeRTOS 示範

1. 建立目錄以包含產生的建置檔案，例如 *build-directory*。
2. 使用以下命令從原始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 Hexmate 和工具鏈二進位檔的正確路徑，例如 C:\Program Files (x86)\Microchip\MPLABX\v5.35\mplab_platform\bin和 C:\Program Files \Microchip\xc32\v2.40\bin 路徑。

3. 將目錄變更為建置目錄 (*build-directory*)，然後從make該目錄執行。

如需詳細資訊，請參閱搭配 FreeRTOS 使用 CMake 。

若要使用乙太網路進行連線，您必須定義前置處理器巨集 `PIC32_USE_ETHERNET`。

在雲端監控 MQTT 訊息

您可以使用 AWS IoT 主控台中的 MQTT 用戶端來監控裝置傳送至 AWS Cloud 的訊息。

使用 MQTT 用戶端訂閱 AWS IoT 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 ***your-thing-name*/example/topic**，然後選擇訂閱主題。

故障診斷

如需故障診斷資訊，請參閱故障診斷入門。

Nordic nRF52840-DK 入門

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

本教學課程提供 Nordic nRF52840-DK 的入門說明。如果您沒有 Nordic nRF52840-DK，請造訪 AWS Partner Device Catalog 向我們的合作夥伴購買。

開始之前，您需要設定 AWS IoT 和 Amazon Cognito for FreeRTOS 低功耗藍牙。

若要執行 FreeRTOS 低功耗藍牙示範，您也需要具有藍牙和 Wi-Fi 功能的 iOS 或 Android 行動裝置。

> **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。

概觀

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. 跨編譯 FreeRTOS 示範應用程式至二進位映像。
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 Segger Embedded Studio 做為 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 低功耗藍牙行動開發套件示範應用程式

若要跨低功耗藍牙執行 FreeRTOS 示範專案，您需要在行動裝置上執行 FreeRTOS 低功耗藍牙行動開發套件示範應用程式。

設定 FreeRTOS 低功耗藍牙行動 SDK 示範應用程式

1. 按照 適用於 FreeRTOS 藍牙裝置的 Mobile SDKs 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 低功耗藍牙行動 SDK 示範應用程式 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 中的指示來設定屬性。
4. 從頂端功能表中，選擇工具、終端機模擬器、連線**###** (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。

下載並設定 FreeRTOS

設定硬體和環境之後，您可以下載 FreeRTOS。

下載 FreeRTOS

若要下載 Nordic nRF52840-DK 的 FreeRTOS，請前往 FreeRTOS GitHub 頁面並複製儲存庫。如需說明，請參閱 README.md 檔案。

> **Important**
>
> - 在本主題中，FreeRTOS 下載目錄的路徑稱為 *freertos*。
> - *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。長 FreeRTOS 下載目錄路徑可能會導致建置失敗。

設定專案

若要執行示範，您需要設定專案來使用 AWS IoT。若要設定專案以使用 AWS IoT，您的裝置必須註冊為 AWS IoT 物件。您應該會在 設定 AWS IoT 和 Amazon Cognito for FreeRTOS 低功耗藍牙 時註冊您的裝置。

設定您的 AWS IoT 端點

1. 登入 AWS IoT 主控台。
2. 在導覽窗格中，選擇設定。

   您的 AWS IoT 端點會出現在端點文字方塊中。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)。記下裝置的 AWS IoT 物件名稱。
4. 在您的 AWS IoT 端點和實 AWS IoT 物名稱準備好時，在 IDE *freertos*/demos/include/aws_clientcredential.h中開啟 ，並指定下列#define常數的值：

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

啟用示範

1. 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。
2. 開啟 vendors/nordic/boards/nrf52840-dk/aws_demos/config_files/aws_demos_config.h，然後定義 CONFIG_MQTT_DEMO_ENABLED。
3. 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，您就可以在電路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 中的aws_demos示範專案上按一下滑鼠右鍵，然後選擇建置。

> **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. 遵循 MQTT over Bluetooth Low Energy Demo Application 的指示，使用 FreeRTOS Bluetooth Low Energy Mobile SDK 示範應用程式做為行動 MQTT 代理來完成示範。

故障診斷

如需 FreeRTOS 入門的一般疑難排解資訊，請參閱故障診斷入門。

Nuvoton NuMaker-IoT-M487 入門

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

本教學課程提供 Nuvoton NuMaker-IoT-M487 開發面板入門的指示。系列微控制器， 和 包含內建 RJ45 乙太網路和 Wi-Fi 模組。如果您沒有 Nuvoton NuMaker-IoT-M487，請造訪 AWS Partner Device Catalog 向我們的合作夥伴購買。

開始之前，您必須設定 AWS IoT 和 FreeRTOS 軟體，將開發板連接到 AWS 雲端。如需說明，請參閱首要步驟。在本教學課程中，FreeRTOS 下載目錄的路徑稱為 *freertos*。

概觀

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 示範應用程式交叉編譯至二進位映像。
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 MCU 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_V3.06.7215r（或最新版本），其位於 Nuvoton 開發工具頁面。

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 故障診斷 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 Cloud 的訊息。

使用 MQTT 用戶端訂閱 AWS IoT 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 ***your-thing-name*****/example/topic**，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))。

   您應該會在 AWS IoT 主控台的 MQTT 用戶端中看到裝置傳送的 MQTT 訊息。

搭配 FreeRTOS 使用 CMake

您也可以使用 CMake 來建置和執行 FreeRTOS 示範應用程式，或您使用第三方程式碼編輯器和偵錯工具開發的應用程式。

確定您已安裝 CMake 建置系統。請按照 搭配 FreeRTOS 使用 CMake 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 Cloud 的訊息。

使用 MQTT 用戶端訂閱 AWS IoT 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 ***your-thing-name*/example/topic**，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. 在主機機器上，開啟命令提示字元，然後導覽至 *freertos* 資料夾。
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：

     導覽至包含 FreeRTOS 示範專案來源檔案的目錄。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：

     導覽至包含 FreeRTOS 示範專案來源檔案的目錄。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
   ```

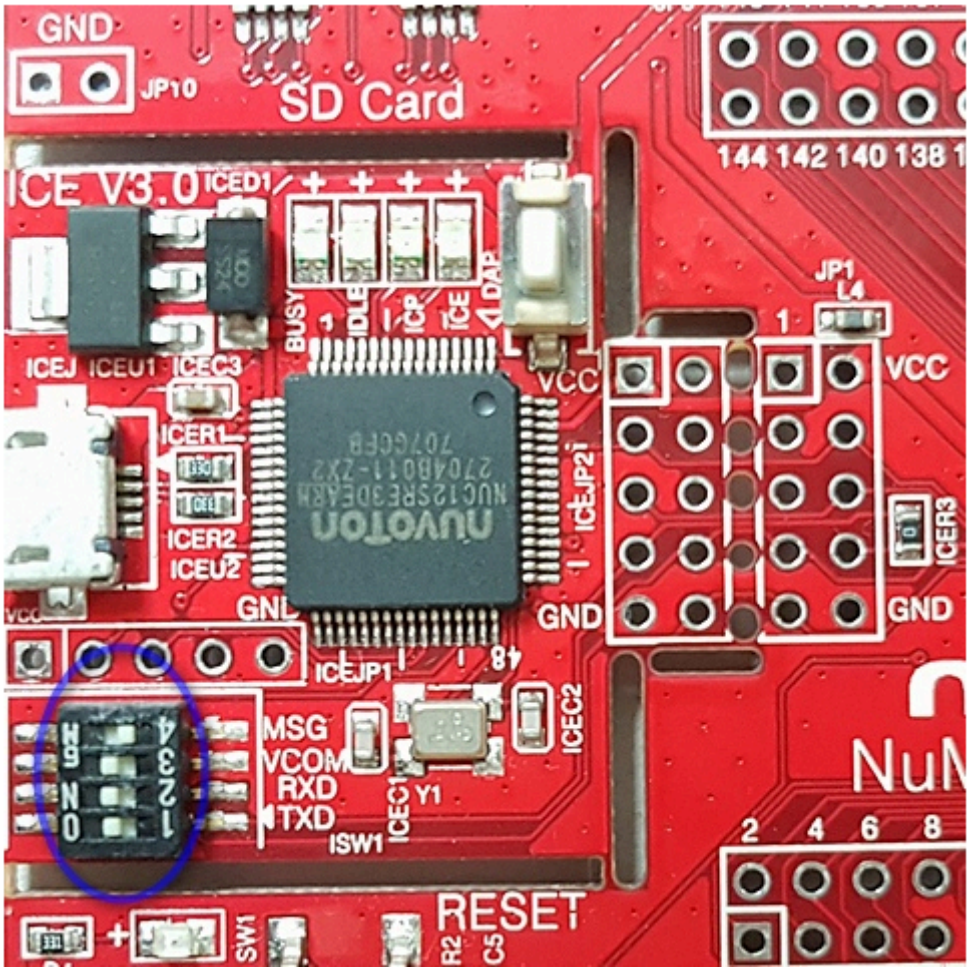
   此時，二進位檔案 aws_demos.bin 應該位於 *BUILD_FOLDER*/vendors/Nuvoton/boards/numaker_iot_m487_wifi 資料夾。

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 故障診斷)。
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 MQTT 用戶端訂閱 AWS IoT MQTT 主題，您應該會在 AWS IoT 主控台中看到裝置傳送的 MQTT 訊息。

故障診斷

- 如果您的 Windows 無法辨識裝置 VCOM，請從連結 Nu-Link USB Driver v1.6 安裝 NuMaker windows 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。



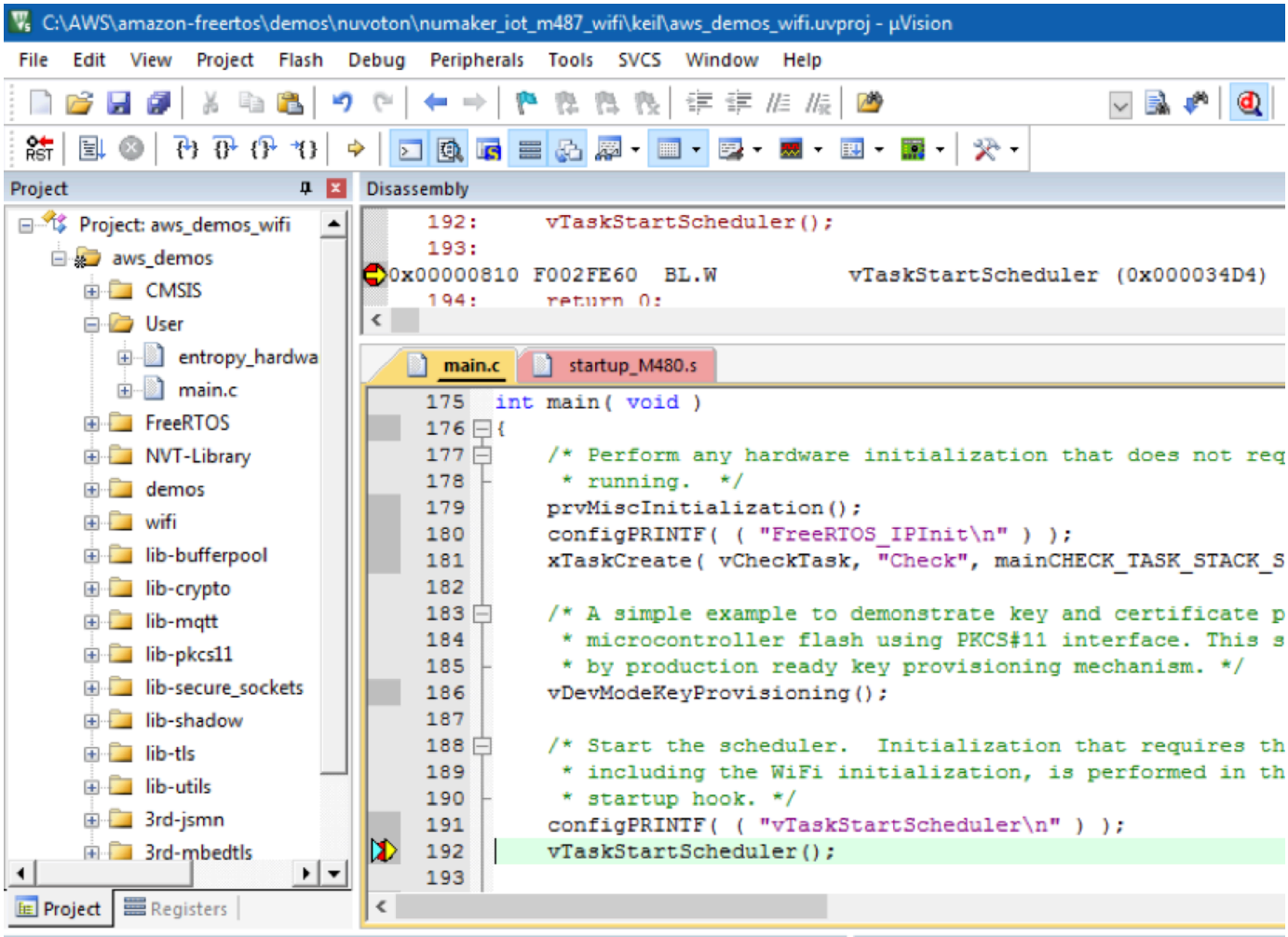
如果您在設定開發環境或連接到主機板時遇到問題，請聯絡 Nuvoton。

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 中建置 FreeRTOS 示範專案建置並執行 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。µVision 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：

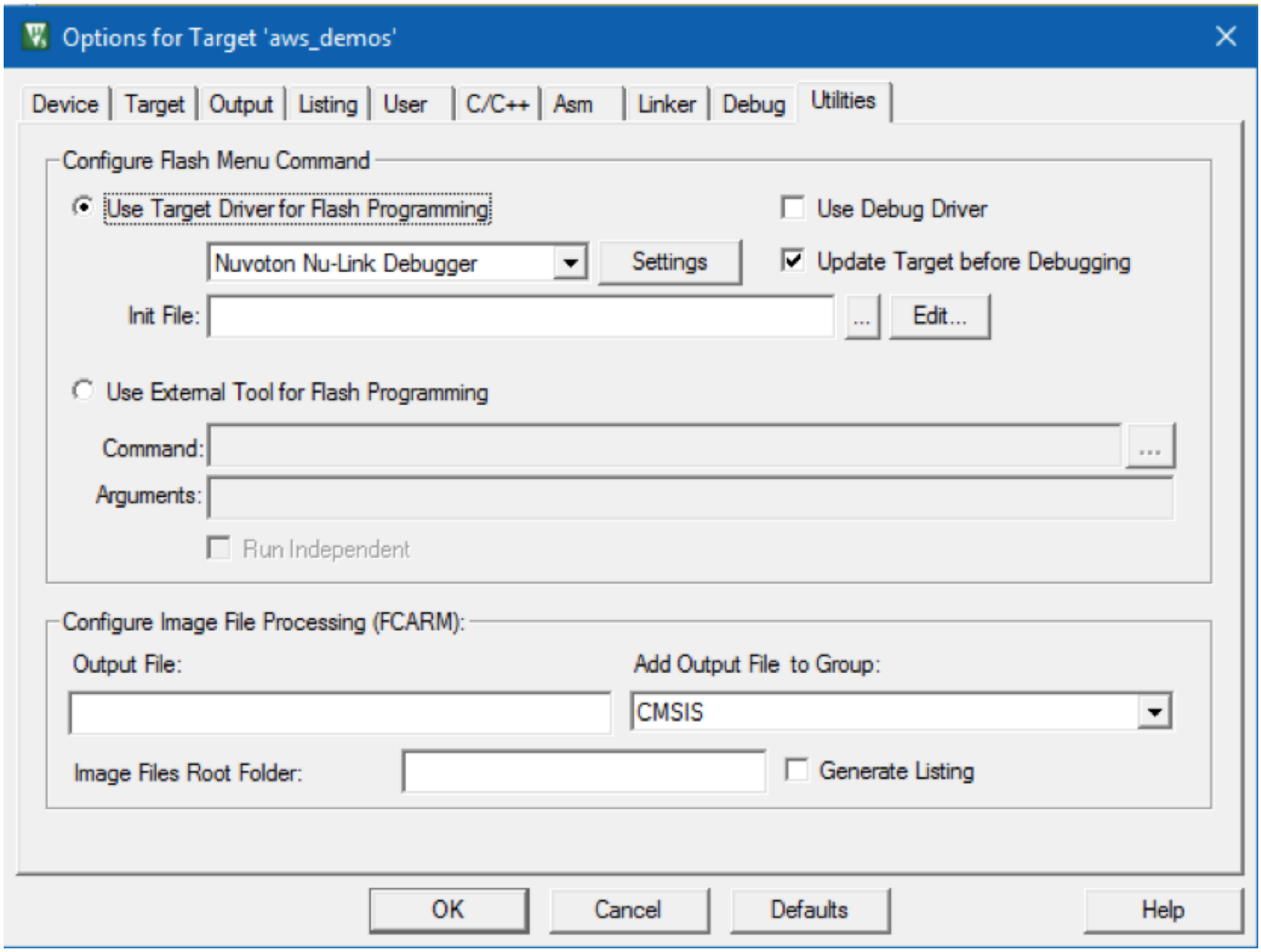


故障診斷 µVision 偵錯設定

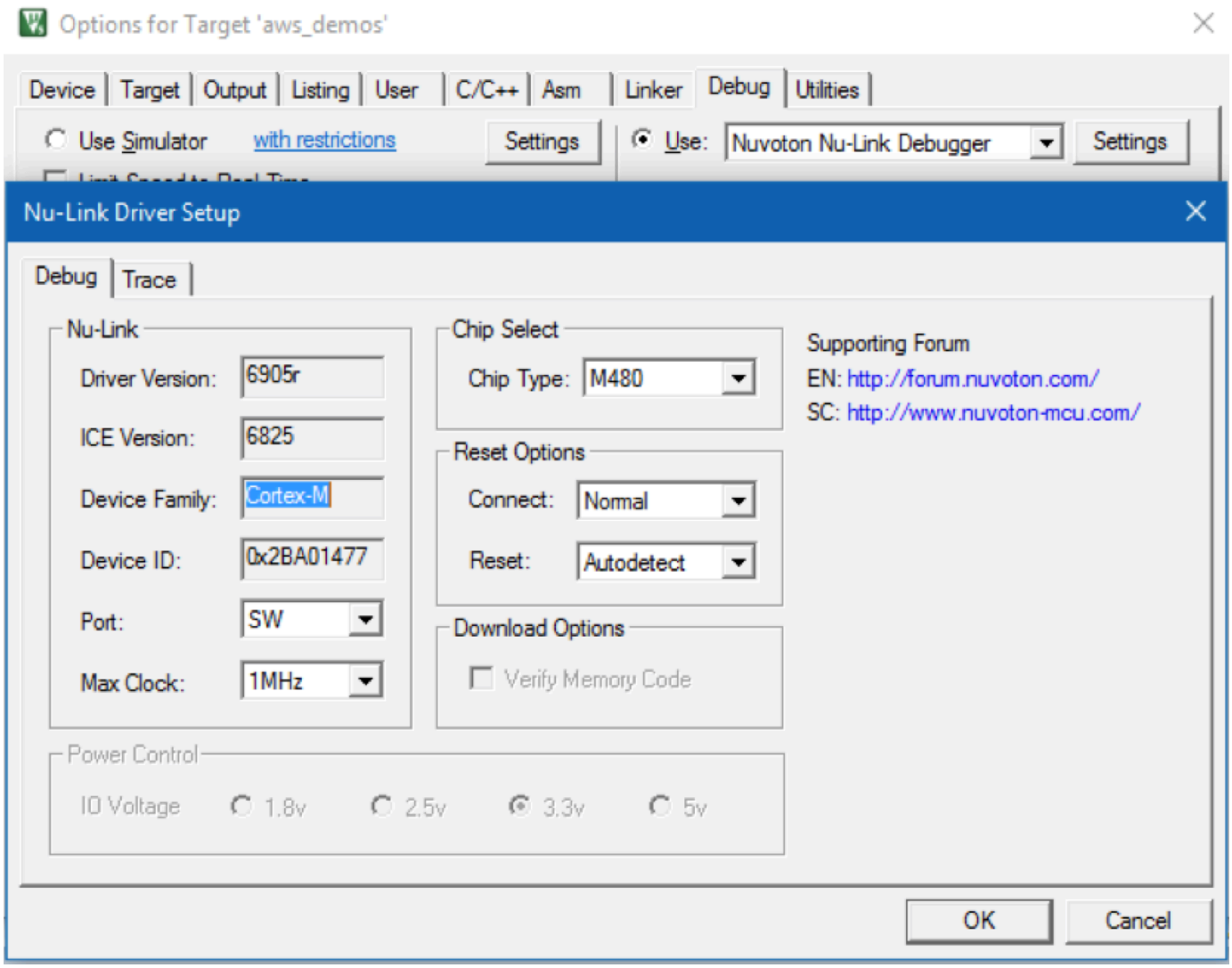
如果您在偵錯應用程式時遇到問題，請檢查您的偵錯設定是否在 Keil µVision 中正確設定。

若要驗證「μVision」偵錯設定是否正確

1. 開啟 Keil μVision。
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 Module 入門

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

本教學課程提供 NXP LPC54018 IoT 模組的入門指示。如果您沒有 NXP LPC54018 IoT 模組，請造訪 AWS Partner Device Catalog 向我們的合作夥伴購買。使用 USB 纜線將 NXP LPC54018 IoT Module 接到您的電腦。

開始之前，您必須設定 AWS IoT 和您的 FreeRTOS 下載，將您的裝置 AWS 連線至 雲端。如需說明，請參閱 首要步驟。在本教學課程中，FreeRTOS 下載目錄的路徑稱為 *freertos*。

概觀

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. 跨編譯 FreeRTOS 示範應用程式至二進位映像。
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 偵錯器來啟動和偵錯在 NXP LPC54018 主機板上執行的程式碼。FreeRTOS 已使用 OM40006 IoT 模組進行測試。如需支援的偵錯工具的詳細資訊，請參閱 OM40007 LPC54018 IoT 模組產品頁面提供的 NXP LPC54018 IoT 模組使用手冊。 OM40007 LPC54018 IoT

1. 如果您使用的是 OM40006 IoT 模組除錯器，請使用轉換器纜線，才能從除錯器 20 pin 的連接器接到 NXP IoT 模組的 10 pin 連接器。
2. 使用 mini-USB 對 USB 纜線，將 NXP LPC54018 和 OM40006 IoT 模組除錯器接到電腦的 USB 連接埠。

設定開發環境

FreeRTOS 支援 NXP LPC54018 IoT 模組的兩個 IDEs：IAR Embedded Workbench 和 MCUXpresso。

開始之前，請先安裝其中一種 IDE。

安裝 IAR Embedded Workbench for ARM

1. 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。

2. 執行安裝程式並遵循提示。
3. 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)。
4. 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

1. 從 NXP 下載並執行 MCUXpresso 安裝程式。

   > **Note**
   >
   > 支援 10.3.x 版和更新版本。

2. 瀏覽至 MCUXpresso SDK，然後選擇 Build your SDK (建置您的軟體開發套件)。

   > **Note**
   >
   > 支援 2.5 版和更新版本。

3. 選擇 Select Development Board (選取電路板)。
4. 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`**。
5. 在 Boards (電路板) 中，選擇 LPC54018-IoT-Module (LPC54018 IoT 模組)。
6. 驗證硬體詳細資訊，然後選擇 Build MCUXepresso SDK (建置 MCUXepresso 軟體開發套件)。
7. 已建置使用 MCUXpresso IDE 的 Windows 軟體開發套件。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. 啟動 MCUXpresso IDE，然後選擇 Installed SDKs (已安裝的軟體開發套件) 標籤。
9. 將已下載的軟體開發套件封存檔案拖放到 Installed SDKs (已安裝的軟體開發套件) 視窗中。

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 IDE

將 FreeRTOS 範例程式碼匯入 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.eww。
3. 在 Project (專案) 功能表中，選擇 Rebuild All (全部重建)。

將 FreeRTOS 範例程式碼匯入 MCUXpresso IDE

1. 開啟 MCUXpresso，並從 File (檔案) 功能表中，選擇 Open Projects From File System (從檔案系統開啟專案)。
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

使用 IAR Embedded 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。

使用 MCUxpresso IDE 執行 FreeRTOS 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，請按一下重新啟動除錯按鈕
>
>
>
> 以重新啟動除錯工作階段。(這是必要的步驟，因為適用於 NXP54018 IoT 模組的 MCUXpresso 除錯器有錯誤)。

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)。

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 Cloud 的訊息。

使用 MQTT 用戶端訂閱 AWS IoT 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 ***your-thing-name*****/example/topic**，然後選擇訂閱主題。

故障診斷

如需 FreeRTOS 入門的一般疑難排解資訊，請參閱故障診斷入門。

Renesas Starter Kit+ for RX65N-2MB 入門

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

本教學課程提供 RX65N-2MB 的 Renesas Starter Kit+ 入門指示。如果您沒有 Renesas RSK+ for RX65N-2MB，請造訪 AWS Partner Device Catalog，並向我們的合作夥伴購買。

開始之前，您必須設定 AWS IoT 和您的 FreeRTOS 下載，將您的裝置 AWS 連線至 雲端。如需說明，請參閱 首要步驟。在本教學課程中，FreeRTOS 下載目錄的路徑稱為 *freertos*。

概觀

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. 跨編譯 FreeRTOS 示範應用程式至二進位映像。
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 Renesas 硬體

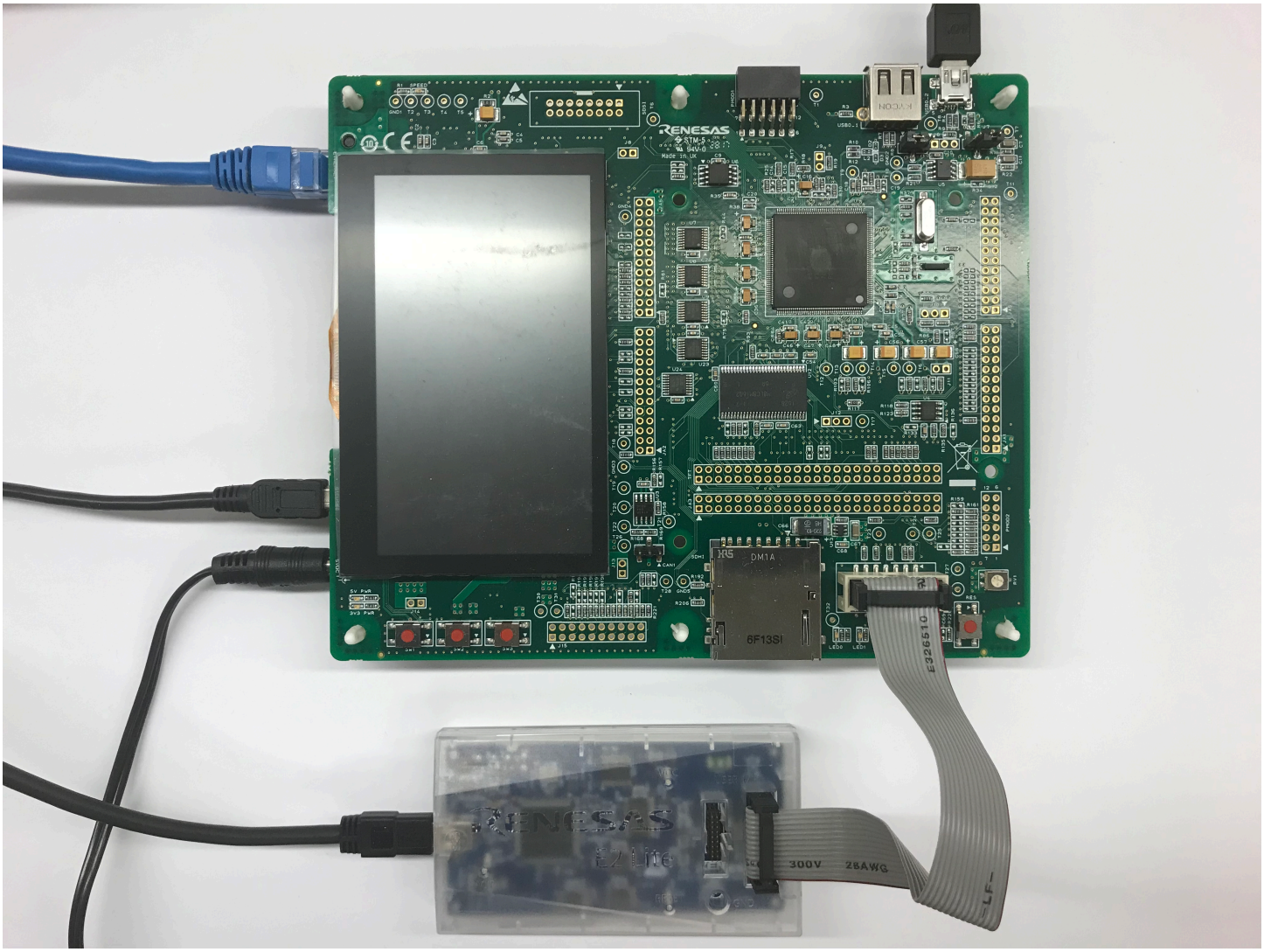
設定 RSK+ for RX65N-2MB

1. 將正極 +5V 電源整流器連接到 RSK+RX65N-2MB 的 PWR 連接器。
2. 將您的電腦連線至 RSK+ for RX65N-2MB 上的 USB 2.0 FS 連接埠。
3. 將您的電腦連線至 RSK+ for RX65N-2MB 上的 USB 序列連接埠。
4. 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

1. 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 e^{2}studio IDE 和 CC-RX 編譯器。

> **Note**
>
> 支援 Renesas e^{2}studio IDE 和 CC-RX 編譯器的只有 Windows 7、8 及 10 作業系統。

下載並安裝 e^{2}studio

1. 前往 Renesas e^{2}studio 安裝程式下載頁面，並下載離線安裝程式。
2. 您將會導向至 Renesas 登入頁面。

如果您有 Renesas 帳戶，請輸入您的登入憑證，然後選擇登入。

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 e^{2}studio 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，請參閱 Renesas 網站上的 e^{2}studio。

下載並安裝 RX Family C/C++ 編譯器套件

1. 前往 RX 系列 C/C++ 編譯器套件下載頁面，並下載 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 e^{2}studio 中建置 FreeRTOS 示範

匯入並在 e^{2}studio 中建置示範

1. 從開始功能表啟動 e^{2}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 e^{2}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 e^{2}studio，將會顯示 Code Generator Registration (程式碼產生器註冊) 視窗。選擇確定。
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。在請重新啟動 e^{2}studio 以使用 Code Generator 下，選擇確定。
6. 重新啟動 e^{2}studio 視窗隨即出現。選擇確定。
7. e^{2}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 e^{2}studio 歡迎畫面上，選擇前往 e^{2}studio 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 e^{2}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。

   您可能會看到防火牆針對 `e2-server-gdb.exe` 發出的警告。請檢查 Private networks, such 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. e^{2}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 Cloud 的訊息。

使用 MQTT 用戶端訂閱 AWS IoT 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 ***your-thing-name*/example/topic**，然後選擇訂閱主題。

如需 Renesas 發佈的最新專案，請參閱 GitHub 上`amazon-freertos`儲存庫的`renesas-rx`叉。

故障診斷

如需 FreeRTOS 入門的一般疑難排解資訊，請參閱故障診斷入門。

IoT 節點用的 STMicroelectronics STM32L4 探索套件入門

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

本教學課程提供 STMicroelectronics STM32L4 Discovery Kit IoT Node 的入門指示。如果您還沒有 STMicroelectronics STM32L4 探索套件 IoT 節點，請造訪 AWS 合作夥伴裝置目錄，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，請參閱 STM32L4 Discovery kit IoT node, low-power wireless, Bluetooth Low Energy, 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，您必須設定 AWS IoT 和 FreeRTOS 下載，將您的裝置連線至 AWS 雲端。如需說明，請參閱 首要步驟。在本教學課程中，FreeRTOS 下載目錄的路徑稱為 *freertos*。

概觀

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. 跨編譯 FreeRTOS 示範應用程式至二進位映像。
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. 瀏覽至 OpenSTM32.org。
2. 在 OpenSTM32 網頁上註冊。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。

如果您在安裝時遇到問題，請參閱 System Workbench 網站的常見問答集。

建置並執行 FreeRTOS 示範專案

將 FreeRTOS 示範匯入 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 纜線將 IoT 節點用的 STMicroelectronics STM32L4 探索套件接到您的電腦。

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)。

搭配 FreeRTOS 使用 CMake

如果您不想使用 IDE for FreeRTOS 開發，您也可以使用 CMake 來建置和執行您已開發的示範應用程式，或使用第三方程式碼編輯器和偵錯工具來執行這些應用程式。

首先，建立資料夾以包含產生的建置檔案 (*`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
```

即使 `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
```

如需搭配 FreeRTOS 使用 CMake 的詳細資訊，請參閱 搭配 FreeRTOS 使用 CMake 。

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 Cloud 的訊息。

使用 MQTT 用戶端訂閱 AWS IoT 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 ***your-thing-name*/example/topic**，然後選擇訂閱主題。

故障診斷

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

若要下載最新的 Wi-Fi 韌體，請參閱 STM32L4 Discovery kit IoT node, low-power wireless, Bluetooth Low Energy, 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。

Texas Instruments CC3220SF-LAUNCHXL 入門

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

本教學課程提供 Texas Instruments CC3220SF-LAUNCHXL 入門指示。如果您沒有 Texas Instruments (TI) CC3220SF-LAUNCHXL 開發套件，請造訪 AWS Partner Device Catalog 向我們的合作夥伴購買。

開始之前，您必須設定 AWS IoT 和您的 FreeRTOS 下載，將您的裝置 AWS 連線至 雲端。如需說明，請參閱 首要步驟。在本教學課程中，FreeRTOS 下載目錄的路徑稱為 *freertos*。

概觀

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. 跨編譯 FreeRTOS 示範應用程式至二進位映像。
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

請依照下列步驟設定您的開發環境，以開始使用 FreeRTOS。

請注意，FreeRTOS 支援 TI CC3220SF-LAUNCHXL 開發套件的兩個 IDEs：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

1. 瀏覽至 TI Code Composer Studio。
2. 下載適用於主機機器平台 (Windows、macOS 或 Linux 64 位元) 的離線安裝程式。
3. 解壓縮並執行離線安裝程式。依照提示進行。
4. 針對 Product Families to Install (要安裝的產品系列)，選擇 SimpleLink Wi-Fi CC32xx Wireless MCUs (SimpleLink Wi-Fi CC32xx 無線 MCU)。
5. 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，請參閱 TI Development Tools Support、Code Composer Studio FAQs 和 Troubleshooting CCS。

安裝 IAR Embedded Workbench

1. 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。
2. 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 軟體開發套件

安裝 SimpleLink 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。

安裝 Uniflash

安裝 Uniflash。CCS Uniflash 是一種獨立工具，可用來對 TI MCU 上的晶片內建快閃記憶體進行程式設計。Uniflash 提供 GUI、命令列和指令碼界面。

安裝最新的 Service Pack

1. 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。
2. 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，請從新組態下的電路板清單中選擇 CC3220SF-LAUNCHXL，然後選擇啟動映像建立器。
3. 選擇新專案。
4. 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)。
5. 在 Uniflash 應用程式視窗的右側，選擇 Connect (連線)。
6. 選擇左側欄中的 Advanced (進階)、Files (檔案)，然後選擇 Service Pack (Service Pack)。
7. 選擇 Browse (瀏覽)，然後導覽至您安裝 CC3220SF SimpleLink 軟體開發套件的位置。此服務套件位於 `ti/simplelink_cc32xx_sdk_`*`VERSION`*`/tools/cc32xx_tools/servicepack-cc3x20/sp_`*`VERSION`*`.bin`。
8. 選擇 Burn (燒錄)

   ( )

   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：

- 設定中所述的 FreeRTOS 示範應用程式設定 FreeRTOS 示範。
- 使用 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 TI Code Composer

1. 開啟 TI Code Composer，然後選擇 OK (確定) 以接受預設的工作空間名稱。

2. 在 Getting Started (入門) 頁面上，選擇 Import Project (匯入專案)。
3. 在 Select search-directory (選取搜尋目錄) 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)。
4. 在 Project Explorer (專案瀏覽器) 中，按兩下 aws_demos 將專案設為作用中。
5. 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

1. 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，請參閱 SimpleLink Wi-Fi CC3x20, CC3x3x 網路處理器使用者指南。
2. 使用 USB 纜線將 Texas Instruments CC3220SF-LAUNCHXL 接到您的電腦。
3. 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)。
4. 在 TI Code Composer 的 Run (執行) 中，選擇 Debug (除錯)。
5. 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 IAR Embedded Workbench

1. 開啟 IAR Embedded Workbench，選擇 File (檔案)，然後選擇 Open Workspace (開啟工作空間)。
2. 導覽至 `projects/ti/cc3220_launchpad/iar/aws_demos`，並選擇 aws_demos.eww，然後選擇 OK (確定)。
3. 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

1. 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，請參閱 SimpleLink Wi-Fi CC3x20, CC3x3x 網路處理器使用者指南。
2. 使用 USB 纜線將 Texas Instruments CC3220SF-LAUNCHXL 接到您的電腦。
3. 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)。

4. 從 Project (專案) 功能表中，選擇 Download and Debug (下載並除錯)。如果顯示 "Warning: Failed to initialize EnergyTrace" (警告：無法初始化 EnergyTrace)，您可以將其忽略。如需 EnergyTrace 的詳細資訊，請參閱 MSP EnergyTrace Technology。
5. 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)。

搭配 FreeRTOS 使用 CMake

如果您不想使用 IDE for FreeRTOS 開發，您也可以使用 CMake 來建置和執行您已開發的示範應用程式，或使用第三方程式碼編輯器和偵錯工具來執行這些應用程式。

使用 CMake 建置 FreeRTOS 示範

1. 建立資料夾以包含產生的建置檔案 (*build-folder*)。
2. 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，請參閱搭配 FreeRTOS 使用 CMake 。

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 Cloud 的訊息。

使用 MQTT 用戶端訂閱 AWS IoT 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 ***your-thing-name*/example/topic**，然後選擇訂閱主題。

故障診斷

如果您在 AWS IoT 主控台的 MQTT 用戶端中看不到訊息，您可能需要設定主機板的偵錯設定。

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。

Windows 裝置模擬器入門

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

本教學課程提供開始使用 FreeRTOS Windows Device Simulator 的說明。

開始之前，您必須設定 AWS IoT 和 FreeRTOS 下載，將您的裝置連線至 AWS 雲端。如需說明，請參閱 首要步驟。在本教學課程中，FreeRTOS 下載目錄的路徑稱為 *freertos*。

FreeRTOS 會以 zip 檔案發行，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**
>
> 此教學課程無法在 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。您可以在具有 C++ 元件的桌面開發的選用區段下選擇此選項。

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 CMake 型建置系統來建置 FreeRTOS 專案，請安裝最新版本的 CMake。FreeRTOS 需要 CMake 3.13 版或更新版本。

建置並執行 FreeRTOS 示範專案

您可以使用 Visual Studio 或 CMake 來建置 FreeRTOS 專案。

使用 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。

使用 CMake 建置和執行 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 原始程式碼目錄。

   在 GUI 中，為其中 為原始程式碼設定 FreeRTOS 原始程式碼目錄 (*freertos*)。

   設 *freertos*/build 為建置二進位程式碼的位置。
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

   Type

   STRING

   Value

   pc.windows

   描述

   (選用)
3. 選擇設定。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。

如需搭配 FreeRTOS 使用 CMake 的詳細資訊，請參閱 搭配 FreeRTOS 使用 CMake 。

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 Cloud 的訊息。

使用 MQTT 用戶端訂閱 AWS IoT 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 ***your-thing-name*/example/topic**，然後選擇訂閱主題。

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。

Xilinx Avnet MicroZed Industrial IoT 入門套件

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

本教學課程提供 Xilinx Avnet MicroZed Industrial IoT 套件入門指示。如果您沒有 Xilinx Avnet MicroZed 工業 IoT 套件，請造訪 AWS Partner Device Catalog 向我們的合作夥伴購買。

開始之前，您必須設定 AWS IoT 和 FreeRTOS 下載，將您的裝置連線至 AWS 雲端。如需說明，請參閱 首要步驟。在本教學課程中，FreeRTOS 下載目錄的路徑稱為 *freertos*。

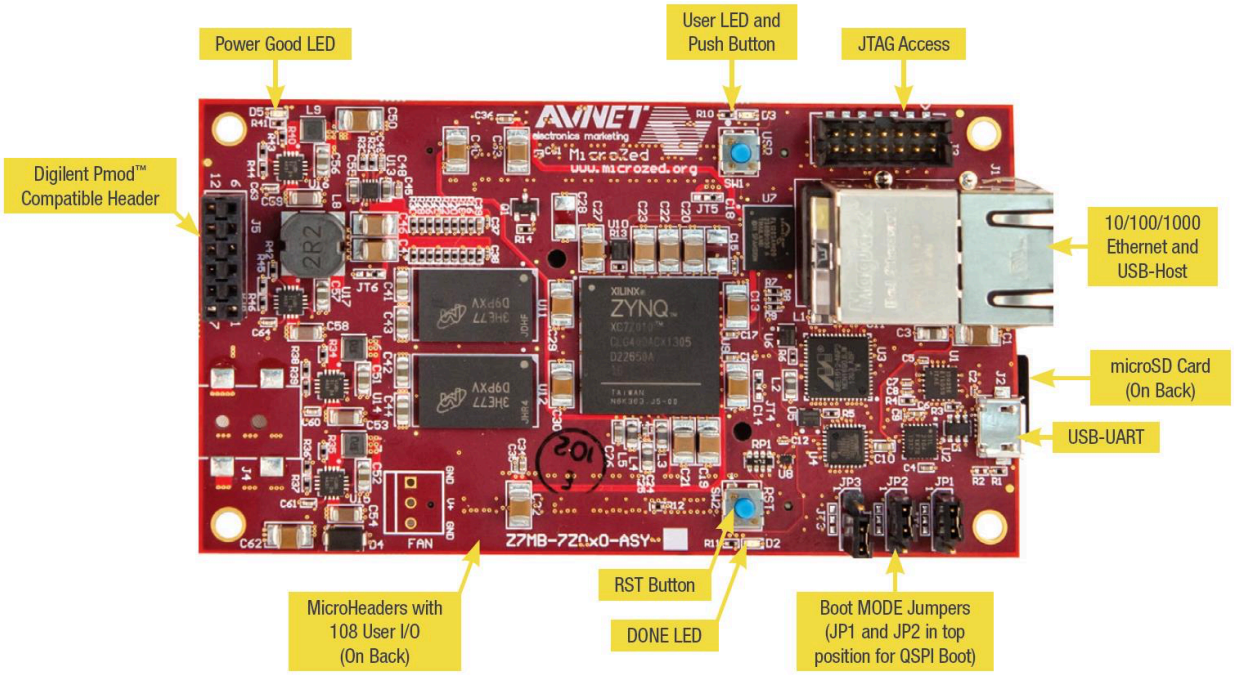
概觀

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. 跨編譯 FreeRTOS 示範應用程式至二進位映像。
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. 將您的電腦連線至 MicroZed 主機板上的 USB-UART 連接埠。
2. 將您的電腦連線至 MicroZed 主機板上的 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線至 MicroZed 主機板上的乙太網路和 USB 主機連接埠。

設定開發環境

若要設定 MicroZed 套件的 FreeRTOS 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. 移至 Software Development Kit Standalone 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 Xilinx 帳戶，請輸入您的登入憑證，然後選擇登入。

   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 連線，請手動安裝 CP210x USB-to-UART Bridge VCP 驅動程式。如需指示，請參閱 Silicon Labs CP210x USB-to-UART Installation Guide。

如需 XSDK 的詳細資訊，請參閱 Xilink 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 (選取根目錄)，然後輸入示範專案的根目錄：*freertos*/projects/xilinx/microzed/xsdk/aws_demos。若要瀏覽目錄，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，對於 Name (名稱)，輸入 **AFR_ROOT**。對於 Value (數值)，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)，來儲存變數並關閉 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。

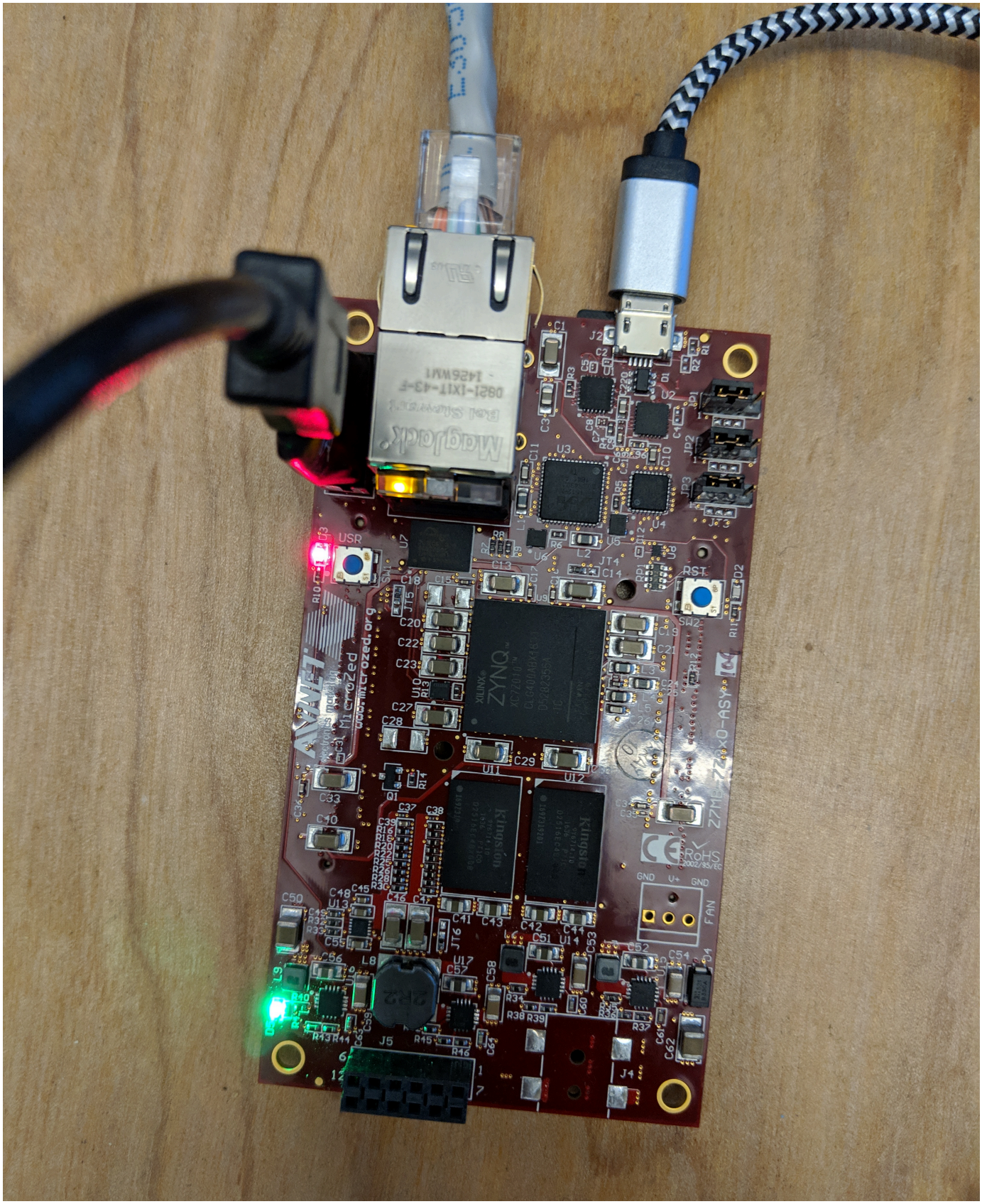
JTAG 偵錯

1. 將您的 MicroZed 主機板的開機模式跳接器設定為 JTAG 開機模式。

2. 將您的 MicroSD 卡直接插入至位於 USB UART 連接埠下方的 MicroSD 卡槽。

> **Note**
>
> 在偵錯之前，請務必備份 MicroSD 卡上您具有的任何內容。

您的主機板看起來如下：


USR
RST
ZYNQ
XILINX
Kingston
Kingston
GND
V+
GND
FAN
RoHS

3. 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)。
4. 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 MicroSD 卡或 QSPI 快閃記憶體啟動 MicroZed 主機板，以執行 FreeRTOS 示範專案。MicroSD 如需說明，請參閱 產生 FreeRTOS 示範專案的開機映像 與 執行 FreeRTOS 示範專案。

執行 FreeRTOS 示範專案

若要執行 FreeRTOS 示範專案，您可以從 MicroSD 卡或 QSPI 快閃記憶體啟動 MicroZed 主機板。MicroSD

當您設定 MicroZed 主機板執行 FreeRTOS 示範專案時，請參閱 中的圖表設定 MicroZed 硬體。確定您已將 MicroZed 主機板連接至電腦。

從 MicroSD 卡啟動 FreeRTOS 專案

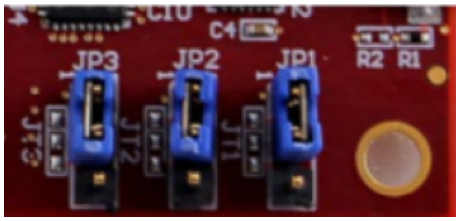
格式化 Xilinx MicroZed Industrial IoT 套件隨附的 MicroSD 卡。

1. 將 `BOOT.bin` 檔案複製到 MicroSD 卡。
2. 將此卡直接插入至在 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：

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 (連線)，選擇 MicroZed 硬體伺服器，以連接主機板與主機電腦。

> **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**
>
> 務必備份 MicroSD 卡上您具有的任何內容。

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

您可以使用 AWS IoT 主控台中的 MQTT 用戶端來監控裝置傳送至 AWS Cloud 的訊息。

使用 MQTT 用戶端訂閱 AWS IoT 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 ***your-thing-name*****/example/topic**，然後選擇訂閱主題。

故障診斷

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。

FreeRTOS Over-the-Air更新

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

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 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 FreeRTOS 任務，以訂閱 AWS IoT 任務的通知主題，並接聽更新訊息。當有可用的更新時，OTA 代理程式會根據您選擇的設定，將請求發佈至 AWS IoT ，並使用 HTTP 或 MQTT 通訊協定接收更新。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。如果您不使用 FreeRTOS OTA 更新示範應用程式，則必須將 整合OTA 代理程式程式庫到您自己的應用程式中，以取得韌體更新功能。

FreeRTOS over-the-air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，依用途、擁有者或環境）分類 AWS IoT 資源。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

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

若要使用無線 (OTA) 更新，請執行下列動作：

- 檢查使用 HTTP 進行 OTA 更新的先決條件或使用 MQTT 進行 OTA 更新的先決條件。
- 建立 Amazon S3 儲存貯體以存放您的更新.
- 建立 OTA 更新服務角色.
- 建立 OTA 使用者政策.
- 建立程式碼簽署憑證.
- 如果您使用的是 的程式碼簽署 AWS IoT，授予 程式碼簽署的存取權 AWS IoT。
- 使用 OTA 程式庫下載 FreeRTOS .

建立 Amazon S3 儲存貯體以存放您的更新

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

OTA 更新檔案存放在 Amazon S3 儲存貯體中。

如果您使用的是 Code Signing for AWS IoT，您用來建立程式碼簽署任務的命令會採用來源儲存貯體（未簽署韌體映像所在的位置）和目的地儲存貯體（寫入已簽署韌體映像的位置）。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 GUID，因此原始檔案不會遭到覆寫。

建立 Amazon S3 儲存貯體

1. 至 https://console.aws.amazon.com/s3/ 登入 Amazon S3 主控台。

2. 選擇建立儲存貯體。
3. 輸入儲存貯體名稱。
4. 在封鎖公開存取的儲存貯體設定下，保持選取封鎖所有公開存取，以接受預設許可。
5. 在儲存貯體版本控制下，選取啟用，將所有版本保留在相同的儲存貯體中。
6. 選擇建立儲存貯體。

如需 Amazon S3 的詳細資訊，請參閱 Amazon Simple Storage Service 使用者指南。

建立 OTA 更新服務角色

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

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：//https://console.aws.amazon.com/iam/。
2. 從導覽窗格中，選擇 Roles (角色)。
3. 選擇建立角色。
4. 在選取可信任執行個體類型下，選取 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: Permissions (下一步：許可)。
8. 選擇 Next: Tags (下一步：標籤)。
9. 選擇下一步：檢閱。
10. 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，輸入角色的名稱，然後從清單中選擇。
2. 選擇連接政策。
3. 在搜尋方塊中，輸入「AmazonFreeRTOSOTAUpdate」，AmazonFreeRTOSOTAUpdate從篩選政策清單中選擇，然後選擇連接政策以將政策連接至您的服務角色。

將必要的 IAM 許可新增至 OTA 服務角色

1. 在 IAM 主控台頁面上的搜尋方塊中，輸入角色的名稱，然後從清單中選擇。
2. 選擇 Add inline policy (新增內嵌政策)。
3. 選擇 JSON 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*# your_account_id* 取代為 AWS 您的帳戶 ID，並將 *your_role_name* 取代為 OTA 服務角色的名稱。

5. 選擇檢閱政策。
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。如果是這樣， AWS 受管政策`AmazonFreeRTOSOTAUpdate`已包含必要的許可。

將必要的 Amazon S3 許可新增至您的 OTA 服務角色

1. 在 IAM 主控台頁面上的搜尋方塊中，輸入角色的名稱，然後從清單中選擇。
2. 選擇 Add inline policy (新增內嵌政策)。
3. 選擇 JSON 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/*"
            ]
        }
    ]
}
```

   此政策授予您的 OTA 服務角色讀取 Amazon S3 物件的許可。確保您將 *example-bucket* 取代為儲存貯體的名稱。

5. 選擇檢閱政策。
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

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

您必須授予 IAM 使用者執行over-the-air更新的權限。您的 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 中的憑證 AWS Certificate Manager。
- 存取 AWS IoT MQTT 型檔案交付功能。
- 存取 FreeRTOS OTA 更新。
- 存取 AWS IoT 任務。
- 存取 IAM。
- 的存取碼簽署 AWS IoT。請參閱 授予 程式碼簽署的存取權 AWS IoT。
- 列出 FreeRTOS 硬體平台。
- 標記和取消標記 AWS IoT 資源。

若要提供存取權，請新增權限至您的使用者、群組或角色：

- 中的使用者和群組 AWS IAM Identity Center：

  建立權限合集。請按照 AWS IAM Identity Center 使用者指南 中的 建立權限合集 說明進行操作。
- 透過身分提供者在 IAM 中管理的使用者：

  建立聯合身分的角色。遵循「IAM 使用者指南」的為第三方身分提供者 (聯合) 建立角色中的指示。
- IAM 使用者：
  - 建立您的使用者可擔任的角色。請按照「IAM 使用者指南」的為 IAM 使用者建立角色中的指示。
  - (不建議) 將政策直接附加至使用者，或將使用者新增至使用者群組。請遵循 IAM 使用者指南的新增許可到使用者 (主控台) 中的指示。

建立程式碼簽署憑證

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

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。基於測試目的，您可以建立自我簽署憑證和私有金鑰。對於生產環境，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。下列各節說明如何為不同的 FreeRTOS 合格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
- 為 Microchip Curiosity PIC32MZEF 建立程式碼簽署憑證
- 為 Espressif ESP32 建立程式碼簽署憑證
- 為 Nordic nrf52840-dk 建立程式碼簽署憑證
- 為 FreeRTOS Windows 模擬器建立程式碼簽署憑證
- 為自訂硬體建立程式碼簽署憑證

為 Texas Instruments CC3220SF-LAUNCHXL 建立程式碼簽署憑證

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

SimpleLink Wi-Fi CC3220SF Wireless Microcontroller Launchpad Development Kit 支援兩種韌體程式碼簽署的憑證鏈：

- 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)

  遊樂場憑證鏈可讓您使用自我簽署的程式碼簽署憑證來嘗試 OTA 更新。

使用 AWS Command Line Interface 將程式碼簽署憑證、私有金鑰和憑證鏈匯入 AWS Certificate Manager。如需詳細資訊，請參閱AWS Command Line Interface 《使用者指南》中的安裝 AWS CLI。

下載並安裝最新版的 SimpleLink CC3220 開發套件。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_`*`version`*`/tools/cc32xx_tools/certificate-playground` (macOS)

SimpleLink CC3220 SDK 中的憑證格式為 DER 格式。若要建立自我簽署的程式碼簽署憑證，您必須將其轉換為 PEM 格式。

請依照下列步驟建立連結至 Texas Instruments 遊樂場憑證階層的程式碼簽署憑證，並符合 AWS Certificate Manager 和 Code Signing for AWS IoT criteria。

> **Note**
>
> 若要建立程式碼簽署的憑證，請在您的機器上安裝 OpenSSL。在安裝 OpenSSL 後，請確保將 `openssl` 指派給命令提示字元或終端機環境中的 OpenSSL 可執行檔。

建立自我簽署的程式碼簽署憑證

1. 使用管理員許可來開啟命令提示字元或終端機。
2. 在您的工作目錄中，使用以下文字來建立名為 `cert_config.txt` 的檔案。將 *`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、私有金鑰和憑證鏈匯入至 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**
>
> 此步驟的編寫假設您將使用適用於 的程式碼簽署 AWS IoT 來簽署韌體映像。雖然 AWS IoT 建議使用適用於 的程式碼簽署，但您可以手動簽署韌體映像。

為 Microchip Curiosity PIC32MZEF 建立程式碼簽署憑證

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

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。在安裝 OpenSSL 後，請確保將 `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 將程式碼簽署憑證、私有金鑰和憑證鏈匯入至其中 AWS Certificate Manager。如需安裝 的詳細資訊 AWS CLI，請參閱安裝 AWS CLI。

1. 在您的工作目錄中，使用以下文字來建立名為 cert_config.txt 的檔案。將 *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
```

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、私有金鑰和憑證鏈匯入至 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**
>
> 此步驟的編寫假設您將使用適用於 的程式碼簽署 AWS IoT 來簽署韌體映像。雖然 AWS IoT 建議使用適用於 的程式碼簽署，但您可以手動簽署韌體映像。

為 Espressif ESP32 建立程式碼簽署憑證

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

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的憑證，請在您的機器上安裝 OpenSSL。在安裝 OpenSSL 後，請確保將 `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 將程式碼簽署憑證、私有金鑰和憑證鏈匯入至其中 AWS Certificate Manager。如需安裝 的詳細資訊 AWS CLI，請參閱安裝 AWS CLI。

1. 在您的工作目錄中，使用以下文字來建立名為 `cert_config.txt` 的檔案。將 *`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
```

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、私有金鑰和憑證鏈匯入至 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**
>
> 此步驟的編寫假設您將使用適用於 的程式碼簽署 AWS IoT 來簽署韌體映像。雖然 AWS IoT 建議使用適用於 的程式碼簽署，但您可以手動簽署韌體映像。

為 Nordic nrf52840-dk 建立程式碼簽署憑證

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

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的憑證，請在您的機器上安裝 OpenSSL。在安裝 OpenSSL 後，請確保將 `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 將程式碼簽署憑證、私有金鑰和憑證鏈匯入至其中 AWS Certificate Manager。如需安裝 的資訊 AWS CLI，請參閱安裝 AWS CLI。

1. 在您的工作目錄中，使用以下文字來建立名為 `cert_config.txt` 的檔案。將 *`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
```

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、私有金鑰和憑證鏈匯入至 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**
>
> 此步驟的編寫假設您將使用適用於 的程式碼簽署 AWS IoT 來簽署韌體映像。雖然 AWS IoT 建議使用適用於 的程式碼簽署，但您可以手動簽署韌體映像。

為 FreeRTOS Windows 模擬器建立程式碼簽署憑證

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

FreeRTOS Windows 模擬器需要具有 ECDSA P-256 金鑰和 SHA-256 雜湊的程式碼簽署憑證，才能執行 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。在安裝 OpenSSL 後，請確保將 `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 將程式碼簽署憑證、私有金鑰和憑證鏈匯入至其中 AWS Certificate Manager。如需安裝 的詳細資訊 AWS CLI，請參閱安裝 AWS CLI。

1. 在您的工作目錄中，使用以下文字來建立名為 `cert_config.txt` 的檔案。將 *`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
```

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、私有金鑰和憑證鏈匯入至 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**
>
> 此步驟的編寫假設您將使用適用於 的程式碼簽署 AWS IoT 來簽署韌體映像。雖然 AWS IoT 建議使用適用於 的程式碼簽署，但您可以手動簽署韌體映像。

為自訂硬體建立程式碼簽署憑證

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

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 將程式碼簽署憑證、私有金鑰和憑證鏈匯入 AWS Certificate Manager。如需安裝 的詳細資訊 AWS CLI，請參閱安裝 AWS CLI。

建立程式碼簽署憑證之後，您可以使用 AWS CLI 將其匯入 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和金鑰大小。如需詳細資訊，請參閱匯入憑證的事前準備。如需 ACM 的詳細資訊，請參閱將憑證匯入 AWS Certificate Manager。

您必須將程式碼簽署憑證的內容複製、貼上並格式化至您稍後下載的 FreeRTOS 程式碼中的`aws_ota_codesigner_certificate.h`檔案。

授予 程式碼簽署的存取權 AWS IoT

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

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。您可以手動簽署更新，也可以使用適用於 的程式碼簽署 AWS IoT 來簽署程式碼。若要使用 Code Signing for FreeRTOS，您必須授予 IAM 使用者帳戶對 Code Signing for FreeRTOS 的存取權。

若要提供存取權，請新增權限至您的使用者、群組或角色：

- 中的使用者和群組 AWS IAM Identity Center：

  建立權限合集。請按照 AWS IAM Identity Center 使用者指南 中的 建立權限合集 說明進行操作。
- 透過身分提供者在 IAM 中管理的使用者：

  建立聯合身分的角色。遵循「IAM 使用者指南」的為第三方身分提供者 (聯合) 建立角色中的指示。
- IAM 使用者：
  - 建立您的使用者可擔任的角色。請按照「IAM 使用者指南」的為 IAM 使用者建立角色中的指示。
  - (不建議) 將政策直接附加至使用者，或將使用者新增至使用者群組。請遵循 IAM 使用者指南的新增許可到使用者 (主控台) 中的指示。

使用 OTA 程式庫下載 FreeRTOS

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

您可以從 GitHub 複製或下載 FreeRTOS。如需說明，請參閱 README.md 檔案。

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。

使用 MQTT 進行 OTA 更新的先決條件

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

本節描述使用 MQTT 執行無線 (OTA) 更新的一般需求。

若要提供存取權，請新增權限至您的使用者、群組或角色：

- 中的使用者和群組 AWS IAM Identity Center：

  建立權限合集。請按照 AWS IAM Identity Center 使用者指南 中的 建立權限合集 說明進行操作。
- 透過身分提供者在 IAM 中管理的使用者：

  建立聯合身分的角色。遵循「IAM 使用者指南」的為第三方身分提供者 (聯合) 建立角色中的指示。
- IAM 使用者：
  - 建立您的使用者可擔任的角色。請按照「IAM 使用者指南」的為 IAM 使用者建立角色中的指示。
  - (不建議) 將政策直接附加至使用者，或將使用者新增至使用者群組。請遵循 IAM 使用者指南的新增許可到使用者 (主控台) 中的指示。

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）。
- 需要 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。

組態

從 201912.00 版開始，FreeRTOS OTA 可以使用 HTTP 或 MQTT 通訊協定，將韌體更新映像從 傳輸 AWS IoT 到 裝置。如果您在 FreeRTOS 中建立 OTA 更新時指定這兩個通訊協定，則每個裝置都會決定用於傳輸映像的通訊協定。如需詳細資訊，請參閱使用 HTTP 進行 OTA 更新的先決條件。

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

使用 MQTT 接收 OTA 更新的每個裝置都必須在 中註冊為物件， AWS IoT 而且物件必須具有類似此處列出的附加政策。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/*"
            ]
        }
    ]
}
```

備註

- `iot:Connect` 許可允許您的裝置 AWS IoT 透過 MQTT 連線至 。
- AWS IoT 任務 (`.../jobs/*`) 主題的 `iot:Subscribe`和 `iot:Publish`許可允許連線的裝置接收任務通知和任務文件，以及發佈任務執行的完成狀態。
- OTA 串流 (`.../streams/*`) AWS IoT 主題的 `iot:Subscribe`和 `iot:Publish`許可允許連線的裝置從中擷取 OTA 更新資料 AWS IoT。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 在這些主題上發佈訊息到連線的裝置。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

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

本節描述使用 HTTP 執行無線 (OTA) 更新的一般需求。從 201912.00 版開始，FreeRTOS OTA 可以使用 HTTP 或 MQTT 通訊協定，將韌體更新映像從 傳輸 AWS IoT 到 裝置。

> **Note**
>
> - 雖然 HTTP 通訊協定可能用於傳輸韌體映像，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、任務文件和執行狀態更新。
> - 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 原始程式碼的標頭檔案。

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）。
- FreeRTOS 版本 201912.00 或更新版本需要變更 OTA 通訊協定的組態，才能透過 HTTP 啟用 OTA 資料傳輸。

組態

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

1. 將 configENABLED_DATA_PROTOCOLS 變更為 OTA_DATA_OVER_HTTP。
2. 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，使用 FreeRTOS `xPortGetFreeHeapSize` API 計算。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/*"
            ]
        }
    ]
}
```

備註

- `iot:Connect` 許可允許您的裝置透過 MQTT 連接至 AWS IoT 。
- AWS IoT 任務 (`.../jobs/*`) 主題的 `iot:Subscribe`和 `iot:Publish`許可允許連線的裝置接收任務通知和任務文件，以及發佈任務執行的完成狀態。
- `iot:Receive` 許可允許 AWS IoT Core 在這些主題上發佈訊息到目前連線的裝置。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

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

本節包含使用 OTA 更新在執行 FreeRTOS 的裝置上更新韌體的教學課程。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。當您自動化 OTA 更新任務、使用大量裝置，或使用不符合 FreeRTOS 資格的裝置時， AWS CLI 非常有用。如需 FreeRTOS 合格裝置的詳細資訊，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. 如果您要手動簽署韌體，請簽署已簽署的韌體映像，然後上傳至 Amazon S3 儲存貯體。如果您使用 Code Signing for AWS IoT，請將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 代理程式會收到更新的韌體映像，並驗證新映像的數位簽章、檢查總和和和版本編號。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。如果您的裝置未執行 FreeRTOS，您必須實作在裝置上執行的 OTA 代理程式。

安裝初始韌體

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

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。如果您未執行 FreeRTOS，請略過此步驟。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
- 在 Microchip Curiosity PIC32MZEF 上安裝初始版本韌體
- 在 Espressif ESP32 上安裝初始版本韌體

- 在 Nordic nRF52840 DK 上安裝韌體的初始版本
- Windows 模擬器上的初始韌體
- 在自訂電路板上安裝初始版本韌體

在 Texas Instruments CC3220SF-LAUNCHXL 上安裝初始版本韌體

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

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 上下載、建置、刷新和執行 FreeRTOS OTA 示範 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) 下方，選擇 Browse (瀏覽)，選擇 simplelink_cc32xx_sdk_***version***/tools/cc32xx_tools/certificate-playground/certcatalogPlayGround20160911.lst，然後選擇 Open (開啟)。
19. 在 Signature Source File (簽章來源檔案) 下方，選擇 Browse (瀏覽)，選擇 simplelink_cc32xx_sdk_***version***/tools/cc32xx_tools/certificate-playground/certcatalogPlayGround20160911.lst.signed_3220.bin，然後選擇 Open (開啟)。
20. 選擇

    按鈕以儲存您的專案。
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

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

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 上下載、建置、刷新和執行 FreeRTOS OTA 示範 中所述。

將示範應用程式燒入您的電路板

1. 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

   。
3. 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] vDHCPProcess: offer ac140a0eip
                                                    1 36297 [IP-task] vDHCPProcess:
 offer ac140a0eip
                2 36297 [IP-task]
```

```
IP Address: 172.20.10.14
3 36297 [IP-task] Subnet Mask: 255.255.255.240
4 36297 [IP-task] Gateway Address: 172.20.10.1
5 36297 [IP-task] DNS Server Address: 172.20.10.1


6 36299 [OTA] OTA demo version 0.9.2
7 36299 [OTA] Creating MQTT Client...
8 36299 [OTA] Connecting to broker...
9 38673 [OTA] Connected to broker.
10 38793 [OTA Task] [prvSubscribeToJobNotificationTopics] OK: $aws/things/
devthingota/jobs/$next/get/accepted
11 38863 [OTA Task] [prvSubscribeToJobNotificationTopics] OK: $aws/things/
devthingota/jobs/notify-next
12 38863 [OTA Task] [OTA_CheckForUpdate] Request #0
13 38964 [OTA] [OTA_AgentInit] Ready.
14 38973 [OTA Task] [prvParseJSONbyModel] Extracted parameter [ clientToken:
 0:devthingota ]
15 38973 [OTA Task] [prvParseJSONbyModel] parameter not present: execution
16 38973 [OTA Task] [prvParseJSONbyModel] parameter not present: jobId
17 38973 [OTA Task] [prvParseJSONbyModel] parameter not present: jobDocument
18 38973 [OTA Task] [prvParseJSONbyModel] parameter not present: streamname
19 38973 [OTA Task] [prvParseJSONbyModel] parameter not present: files
20 38975 [OTA Task] [prvParseJSONbyModel] parameter not present: filepath
21 38975 [OTA Task] [prvParseJSONbyModel] parameter not present: filesize
22 38975 [OTA Task] [prvParseJSONbyModel] parameter not present: fileid
23 38975 [OTA Task] [prvParseJSONbyModel] parameter not present: certfile
24 38975 [OTA Task] [prvParseJSONbyModel] parameter not present: sig-sha256-ecdsa
25 38975 [OTA Task] [prvParseJobDoc] Ignoring job without ID.
26 38975 [OTA Task] [prvOTA_Close] Context->0x8003b620
27 38975 [OTA Task] [prvPAL_Abort] Abort - OK
28 39964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
29 40964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
30 41964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
31 42964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
32 43964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
33 44964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
34 45964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
35 46964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
36 47964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
```

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

1. 確定您已安裝從 Source Forge 取得的 SRecord 工具。驗證包含 `srec_cat` 和 `srec_info` 程式的目錄位於您的系統路徑。
2. 更新原廠映像的 OTA 序號及應用程式版本。
3. 建置 `aws_demos` 專案。
4. 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`：應用程式的二進位檔案。
- `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.factory.unified.hex`。

5. 使用 MPLab IPE 工具將產生的 hex 檔案編寫到裝置上。
6. 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

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

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。嘗試 OTA 更新之前，建議您執行 FreeRTOS 入門中所述的 MQTT 示範專案，以確保您的主機板和工具鏈設定正確。

刷新主機板的初始原廠映像

1. 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。
2. 將您在 OTA 更新先決條件 產生的 SHA-256/ECDSA PEM 格式的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";
```

3. 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 00003BC0 (14 KiB): D/IRAM
I (393) heap_init: At 3FFE4350 len 0001BCB0 (111 KiB): D/IRAM
I (400) heap_init: At 4008DD9C len 00012264 (72 KiB): IRAM
I (406) cpu_start: Pro cpu start user code
I (88) cpu_start: Starting scheduler on PRO CPU.
I (113) wifi: wifi firmware version: f79168c
I (113) wifi: config NVS flash: enabled
I (113) wifi: config nano formating: disabled
I (113) system_api: Base MAC address is not set, read default base MAC address from
 BLK0 of EFUSE
I (123) system_api: Base MAC address is not set, read default base MAC address from
 BLK0 of EFUSE
I (133) wifi: Init dynamic tx buffer num: 32
I (143) wifi: Init data frame dynamic rx buffer num: 32
I (143) wifi: Init management frame dynamic rx buffer num: 32
I (143) wifi: wifi driver task: 3ffc73ec, prio:23, stack:4096
I (153) wifi: Init static rx buffer num: 10
I (153) wifi: Init dynamic rx buffer num: 32
I (163) wifi: wifi power manager task: 0x3ffcc028 prio: 21 stack: 2560
0 6 [main] WiFi module initialized. Connecting to AP <Your_WiFi_SSID>...
I (233) phy: phy_version: 383.0, 79a622c, Jan 30 2018, 15:38:06, 0, 0
I (233) wifi: mode : sta (30:ae:a4:80:0a:04)
I (233) WIFI: SYSTEM_EVENT_STA_START
I (363) wifi: n:1 0, o:1 0, ap:255 255, sta:1 0, prof:1
I (1343) wifi: state: init -> auth (b0)
I (1343) wifi: state: auth -> assoc (0)
I (1353) wifi: state: assoc -> run (10)
I (1373) wifi: connected with <Your_WiFi_SSID>, channel 1
I (1373) WIFI: SYSTEM_EVENT_STA_CONNECTED
1 302 [IP-task] vDHCPProcess: offer c0a86c13ip
I (3123) event: sta ip: 192.168.108.19, mask: 255.255.224.0, gw: 192.168.96.1
I (3123) WIFI: SYSTEM_EVENT_STA_GOT_IP
2 302 [IP-task] vDHCPProcess: offer c0a86c13ip
3 303 [main] WiFi Connected to AP. Creating tasks which use network...
4 304 [OTA] OTA demo version 0.9.6
5 304 [OTA] Creating MQTT Client...
6 304 [OTA] Connecting to broker...
I (4353) wifi: pm start, type:0

I (8173) PKCS11: Initializing SPIFFS
I (8183) PKCS11: Partition size: total: 52961, used: 0
7 1277 [OTA] Connected to broker.
```

```
8 1280 [OTA Task] [prvSubscribeToJobNotificationTopics] OK: $aws/things/
<Your_Thing_Name>/jobs/$next/get/accepted
I (12963) ota_pal: prvPAL_GetPlatformImageState
I (12963) esp_ota_ops: [0] aflags/seq:0x2/0x1, pflags/seq:0xffffffff/0x0
9 1285 [OTA Task] [prvSubscribeToJobNotificationTopics] OK: $aws/things/
<Your_Thing_Name>/jobs/notify-next
10 1286 [OTA Task] [OTA_CheckForUpdate] Request #0
11 1289 [OTA Task] [prvParseJSONbyModel] Extracted parameter [ clientToken:
 0:<Your_Thing_Name> ]
12 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: execution
13 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: jobId
14 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: jobDocument
15 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: afr_ota
16 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: streamname
17 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: files
18 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: filepath
19 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: filesize
20 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: fileid
21 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: certfile
22 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: sig-sha256-ecdsa
23 1289 [OTA Task] [prvParseJobDoc] Ignoring job without ID.
24 1289 [OTA Task] [prvOTA_Close] Context->0x3ffbb4a8
25 1290 [OTA] [OTA_AgentInit] Ready.
26 1390 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
27 1490 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
28 1590 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
29 1690 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
[ ... ]
```

4. ESP32 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+] 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

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

本指南是在您已執行 Nordic nRF52840-DK 入門和無線更新事前準備所述步驟的前提下撰寫而成。嘗試 OTA 更新之前，建議您執行 FreeRTOS 入門中所述的 MQTT 示範專案，以確保您的主機板和工具鏈設定正確。

刷新主機板的初始原廠映像

1. 打開 *freertos*/vendors/nordic/boards/nrf52840-dk/aws_demos/config_files/aws_demo_config.h.
2. 使用 #define democonfigOTA_UPDATE_DEMO_ENABLED 取代 #define CONFIG_MQTT_DEMO_ENABLED。
3. 在選取 OTA 更新示範的情況下，請遵循 Nordic nRF52840-DK 入門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

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

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 應用程式的一部分，其中也包含了韌體。

在自訂電路板上安裝初始版本韌體

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

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。

請務必在 FreeRTOS 專案或裝置上包含您的程式碼簽署憑證、私有金鑰和憑證信任鏈。

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

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

FreeRTOS 隨附的 OTA 代理程式會檢查任何更新的版本，並只在比現有韌體版本更新時才進行安裝。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. 若您使用的是 Microchip Curiosity PIC32MZEF，請增加 `vendors/microchip/boards/curiosity_pic32mzef/bootloader/bootloader/utility/user-config/ota-descriptor.config` 中的 OTA 序號。OTA 序號應在每次產生新的 OTA 映像時增加。
4. 重新建置專案。

您必須將韌體更新複製到您建立的 Amazon S3 儲存貯體，如中所述建立 Amazon S3 儲存貯體以存放您的更新。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）

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

1. 在 AWS IoT 主控台的導覽窗格中，選擇管理，然後選擇任務。
2. 選擇建立。
3. 在建立 FreeRTOS Over-the-Air(OTA) 更新任務下，選擇建立 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**
   >
   > 只有符合 FreeRTOS 資格的硬體平台會顯示在此清單中。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。如果您使用的是不合格的平台，而且使用 ECDSA P-256 SHA-256 以外的密碼套件進行簽署，則可以使用 的程式碼簽署 AWS IoT，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。

   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 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 更新便會在身分驗證期間發生 `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 Amazon S3 儲存貯體的清單。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 示範專案會以預設名稱 `mplab.production.bin` 及 `mplab.production.ota.bin` 產生兩個二進位映像。請在上傳 OTA 更新的映像時使用第二個檔案。

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. 在檔案類型下，輸入 0-255 範圍內的整數值。您輸入的檔案類型將新增至交付至 MCU 的任務文件。MCU 韌體/軟體開發人員對如何處理此值擁有完整擁有權。可能的情況包括具有次要處理器的 MCU，其韌體可以獨立於主要處理器進行更新。當裝置收到 OTA 更新任務時，可以使用 檔案類型來識別更新的目標處理器。
13. 在 IAM role for OTA update job (OTA 更新任務的 IAM 角色) 下方，根據建立 OTA 更新服務角色中的指示選擇角色。
14. 選擇下一步。
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。此路徑名稱因平台而異。
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. 在選取 Amazon S3 中的韌體映像下，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，狀態為 IN PROGRESS。

> **Note**
>
> - AWS IoT 主控台不會自動更新任務的狀態。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。

使用 建立 OTA 更新 AWS CLI

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

當您使用 AWS CLI 建立 OTA 更新時，您會：

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 的程式碼簽署 AWS IoT，也可以自行簽署韌體更新。如需程式碼簽署支援的密碼編譯簽署和雜湊演算法清單 AWS IoT，請參閱 SigningConfigurationOverrides。如果您想要使用 Code Signing for 不支援的密碼編譯演算法 AWS IoT，您必須先簽署韌體二進位檔，才能將其上傳至 Amazon S3。

使用 的程式碼簽署來簽署您的韌體映像 AWS IoT

若要使用 Code Signing for 來簽署韌體映像 AWS IoT，您可以使用其中一個 AWS SDKs或命令列工具。如需 程式碼簽署的詳細資訊 AWS IoT，請參閱 程式碼簽署 AWS IoT。

安裝和設定程式碼簽署工具之後，請將未簽署的韌體映像複製到 Amazon S3 儲存貯體，並使用下列 AWS 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`：您 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。您可以瀏覽至 Amazon S3 主控台、選擇儲存貯體，然後在頁面頂端的版本旁選擇顯示，即可找到它。

`destination`

S3 儲存貯體中已簽署韌體將複製到其裝置上的目的地。此參數的格式與 `source` 參數相同。

`signing-material`

您程式碼簽署憑證的 ARN。當您將憑證匯入 ACM 時，會產生此 ARN。

`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**
>
> 當您在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。

簽署任務會啟動已簽署的韌體映像，並將其寫入目的地 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。您可以瀏覽至 Amazon S3 主控台並選擇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 程式碼簽署的詳細資訊 AWS IoT，請參閱 程式碼簽署 AWS IoT。

手動簽署您的韌體映像

以數位方式簽署您的韌體映像，並將已簽章的韌體映像上傳至您的 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 服務可讓您使用儲存在 Amazon S3 中不同位置的多個資料片段來執行 OTA 更新。

建立 AWS IoT OTA 更新時，您也可以建立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`

存放未簽署韌體映像的 Amazon S3 儲存貯體。

`key`

Amazon S3 儲存貯體中已簽署韌體映像的檔案名稱。您可以查看儲存貯體的內容，在 Amazon S3 主控台中找到此值。

如果您使用 Code Signing for AWS IoT，則檔案名稱是由 Code Signing for 產生的 GUID AWS IoT。

使用 create-stream AWS 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
```

以下是 create-stream AWS 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`

存放已簽署韌體映像的 Amazon S3 儲存貯體。

`key`

已簽署韌體映像的鍵 (檔案名稱)。

當您使用 的程式碼簽署時 AWS IoT，此金鑰是 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`

也授予儲存韌體映像之 Amazon S3 儲存貯體存取權的 OTA 服務角色。

若要尋找已簽章韌體映像的 Amazon S3 物件金鑰，請使用 aws signer describe-signing-job --job-id ***my-job-id***命令，其中 `my-job-id`是 create-signing-job AWS CLI 命令顯示的任務 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 AWS CLI 命令來建立 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",
          "signatureAlgorithm": "string"
        }
      },
      "attributes": {
        "string": "string"
      }
    }
  ],
```

```
  "roleArn": "string",
  "additionalParameters": {
    "string": "string"
  },
  "tags": [
    {
      "Key": "string",
      "Value": "string"
    }
  ]
}
```

cli-input-json 欄位

名稱	Type	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	列出	將目標鎖定於接收 OTA 更新的裝置。
protocols	列出	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

名稱	Type	描述
		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。
exponentialRate		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。
baseRatePerMinute	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。
rateIncreaseCriteria		任務啟動增加推展速率的條件。 AWS IoT 最多支援十進位後的 1 位數（例如 1.5，但不支援 1.55)。
numberOfNotifiedThings	integer (下限：1)	收到此物件數量通知時，它會啟動推展率的增加。
numberOfSucceededThings	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。

名稱	Type	描述
awsJobPresignedUrlConfig		預先簽章的 URL 之組態資訊。
expiresInSec	long	預先簽章的 URL 有效時間 (以秒為單位)。有效值為 60 - 3600，預設值為 1800 秒。收到任務文件的請求時，就會產生預先簽章的 URL。
awsJobAbortConfig		決定任務中止發生之時間和方式的準則。
abortCriteriaList	列出	決定中止任務之時間和方式的準則清單。
failureType	string	可啟動任務中止的任務執行失敗類型。 enum: FAILED | REJECTED | TIMED_OUT | ALL
action	string	啟動任務中止所要採取的任務動作類型。 enum: CANCEL
minNumberOfExecutedThings	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。
awsJobTimeoutConfig		指定每個裝置必須完成其任務執行的時間量。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	Type	描述
inProgressTimeoutInMinutes	long	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。
files	列出	OTA 更新所串流的檔案。
fileName	string	檔案名稱。
fileType	integer 範圍 - 上限：255；下限：0	您可以在任務文件中包含的整數值，可讓裝置識別從雲端收到的檔案類型。
fileVersion	string	檔案版本。
fileLocation		已更新韌體的位置。
stream		包含 OTA 更新的串流。
streamId	string (上限：128；下限：1)	串流 ID。
fileId	integer (上限：255；下限：0)	與串流建立關聯的檔案 ID。
s3Location		S3 中已更新韌體的位置。

名稱	Type	描述
bucket	string (下限：1)	S3 儲存貯體。
key	string (下限：1)	S3 金鑰。
version	string	S3 儲存貯體版本。
codeSigning		檔案的代碼簽署方式。
awsSignerJobId	string	用於簽署檔案所建立的 AWSSignerJob ID。
startSigningJobParameter		描述程式碼簽署任務。
signingProfileParameter		描述程式碼簽署描述檔。
certificateArn	string	憑證 ARN。
platform	string	您裝置的硬體平台。
certificatePathOnDevice	string	程式碼簽署憑證在您裝置上的位置。
signingProfileName	string	程式碼簽署描述檔名稱。
destination		撰寫程式碼簽章檔案的位置。
s3Destination		描述 S3 中已更新韌體的位置。
bucket	string (下限：1)	內含已更新韌體的 S3 儲存貯體。

名稱	Type	描述
prefix	string	S3 字首。
customCodeSigning		代碼簽署檔案的自訂方式。
signature		檔案的簽章。
inlineDocument	blob	以 base64 編碼呈現的二進位代碼簽署簽章。
certificateChain		憑證鏈。
certificateName	string	憑證名稱。
inlineDocument	string	以 base64 編碼呈現的二進位代碼簽署憑證鏈。
hashAlgorithm	string	用於以代碼簽署檔案的雜湊演算法。
signatureAlgorithm	string	用於以代碼簽署檔案的簽章演算法。
attributes	映射	名稱/屬性對清單。
roleArn	string (上限：2048；下限：20)	授予 Amazon S3、 AWS IoT 工作和 AWS 程式碼簽署資源 AWS IoT 存取權的 IAM 角色，以建立 OTA 更新任務。
additionalParameters	映射	其他 OTA 更新參數清單，以名稱/值對表示。
tags	列出	用於管理更新的中繼資料。
Key	string (上限：128；下限：1)	標籤的金鑰。

名稱	Type	描述
Value	string (上限：256；下限：1)	標籤的值。

輸出

```
{
  "otaUpdateId": "string",
  "awsIotJobId": "string",
  "otaUpdateArn": "string",
  "awsIotJobArn": "string",
  "otaUpdateStatus": "string"
}
```

AWS CLI 輸出欄位

名稱	Type	描述
otaUpdateId	string (上限：128；下限：1)	OTA 更新 ID。
awsIotJobId	string	與 OTA 更新相關聯的 AWS IoT 任務 ID。
otaUpdateArn	string	OTA 更新 ARN。
awsIotJobArn	string	與 OTA 更新相關聯的 AWS IoT 任務 ARN。
otaUpdateStatus	string	OTA 更新狀態。 列舉：CREATE_PENDING | CREATE_IN_PROGRESS | CREATE_COMPLETE | CREATE_FAILED

以下是傳入使用 Code Signing 之 create-ota-update命令的 JSON 檔案範例 AWS IoT。

```
[
  {
    "fileName": "firmware.bin",
    "fileType": 1,
    "fileLocation": {
      "stream": {
        "streamId": "004",
        "fileId":123
      }
    },
    "codeSigning": {
      "awsSignerJobId": "48c67f3c-63bb-4f92-a98a-4ee0fbc2bef6"
    }
  }
]
```

以下是傳入 create-ota-update AWS CLI 命令的 JSON 檔案範例，該命令使用內嵌檔案來提供自訂程式碼簽署資料。

```
[
  {
    "fileName": "firmware.bin",
    "fileType": 1,
    "fileLocation": {
      "stream": {
        "streamId": "004",
        "fileId": 123
      }
    },
    "codeSigning": {
      "customCodeSigning":{
        "signature":{
          "inlineDocument":"your_signature"
        },
        "certificateChain": {
          "certificateName": "your_certificate_name",
          "inlineDocument":"your_certificate_chain"
        },
        "hashAlgorithm":"your_hash_algorithm",
        "signatureAlgorithm":"your_signature_algorithm"
      }
```

```
    }
  }
]
```

以下是傳入 create-ota-update AWS CLI 命令的 JSON 檔案範例，允許 FreeRTOS OTA 啟動程式碼簽署任務並建立程式碼簽署設定檔和串流。

```
[
  {
    "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"
          }
        }
      }
    }
  }
]
```

以下是傳入 create-ota-update AWS CLI 命令的 JSON 檔案範例，該命令會建立 OTA 更新，以使用現有設定檔啟動程式碼簽署任務，並使用指定的串流。

```
[
  {
    "fileName": "your_firmware_path_on_device",
```

```
    "fileType": 1,
    "fileVersion": "1",
    "fileLocation": {
      "s3Location": {
        "bucket": "your_s3_bucket_name",
        "key": "your_object_key",
        "version": "your_S3_object_version"
      }
    },
    "codeSigning":{
      "startSigningJobParameter":{
        "signingProfileName": "your_unique_profile_name",
        "destination": {
          "s3Destination": {
            "bucket": "your_destination_bucket"
          }
        }
      }
    }
  }
]
```

以下是傳入 create-ota-update AWS CLI 命令的 JSON 檔案範例，允許 FreeRTOS OTA 建立具有現有程式碼簽署任務 ID 的串流。

```
[
  {
    "fileName": "your_firmware_path_on_device",
    "fileType": 1,
    "fileVersion": "1",
    "codeSigning":{
      "awsSignerJobId": "your_signer_job_id"
    }
  }
]
```

以下是傳入 create-ota-update AWS CLI 命令以建立 OTA 更新的 JSON 檔案範例。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 AWS CLI 命令來取得所有 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 AWS CLI 命令來取得 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:region:123456789012:otaupdate/ota-update-001",
        "creationDate": 1575414146.286,
        "lastModifiedDate": 1575414149.091,
        "targets": [
            "arn:aws:iot:region:123456789012:thing/myDevice"
        ],
        "protocols": [ "HTTP" ],
        "awsJobExecutionsRolloutConfig": {
            "maximumPerMinute": 0
        },
        "awsJobPresignedUrlConfig": {
            "expiresInSec": 1800
        },
        "targetSelection": "SNAPSHOT",
        "otaUpdateFiles": [
            {
                "fileName": "my_firmware.bin",
                "fileType": 1,
                "fileLocation": {
                    "s3Location": {
                        "bucket": "my-bucket",
                        "key": "my_firmware.bin",
```

```
                        "version": "AvP3bfJC9gyqnwoxPHuTqM5GWENt4iii"
                    }
                },
                "codeSigning": {
                    "awsSignerJobId": "b7a55a54-fae5-4d3a-b589-97ed103737c2",
                    "startSigningJobParameter": {
                        "signingProfileParameter": {},
                        "signingProfileName": "my-profile-name",
                        "destination": {
                            "s3Destination": {
                                "bucket": "some-ota-bucket",
                                "prefix": "SignedImages/"
                            }
                        }
                    },
                    "customCodeSigning": {}
                }
            }
        ],
        "otaUpdateStatus": "CREATE_COMPLETE",
        "awsIotJobId": "AFR_OTA-ota-update-001",
        "awsIotJobArn": "arn:aws:iot:region:123456789012:job/AFR_OTA-ota-update-001"
    }
}
```

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，請參閱描述任務執行。

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 AWS IoT 的任務。

刪除 OTA 串流

當您建立使用 MQTT 的 OTA 更新時，您可以使用命令列或 AWS IoT 主控台來建立串流，將韌體分成區塊，以便透過 MQTT 傳送。您可以使用 delete-stream AWS CLI 命令刪除此串流，如下列範例所示。

```
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 AWS IoT 的任務。
- 要更新之每個裝置 AWS IoT 的任務執行。

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。

刪除為 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 AWS 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，來刪除並未處於終止狀態的任務。如需詳細資訊，請參閱 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
```

如同 delete-job AWS CLI 命令，您可以將 `--force` 參數傳遞至 delete-job-execution，以強制刪除任務執行。如需詳細資訊，請參閱 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。

OTA Update Manager 服務

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

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 串流和程式碼簽署。
- 取得 OTA 更新的相關資訊。
- 列出與 AWS 您的帳戶相關聯的所有 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：CONTINUOUS 或 SNAPSHOT。如需您需要的更新類型的討論，請參閱 AWS IoT 開發人員指南的任務https://docs.aws.amazon.com/iot/latest/developerguide/iot-jobs.html一節。
- 用來執行 OTA 更新的通訊協定：[MQTT]、[HTTP] 或 [MQTT, HTTP]。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 Amazon S3、 AWS IoT 工作和 AWS 程式碼簽署資源 AWS IoT 存取權的 IAM 角色，以建立 OTA 更新任務。
- 選擇性的使用者定義名稱/值對清單。

OTA 更新旨在更新裝置韌體，但您可以使用它們將您想要的任何檔案傳送到一或多個註冊的裝置 AWS IoT。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 Code Signing for FreeRTOS 來簽署韌體更新，也可以使用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更新。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，則必須實作自己的 OTA 代理程式來接聽和下載更新，並執行任何安裝操作。

將 OTA 代理程式整合到您的應用程式

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

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。若要熟悉如何使用物件、登入資料、程式碼簽署憑證、佈建裝置和 OTA 更新任務來設定 AWS 您的帳戶 AWS IoT ，請參閱 FreeRTOS 先決條件。

連線管理

OTA 代理程式會將 MQTT 通訊協定用於涉及 AWS IoT 服務的所有控制通訊操作，但不會管理 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 示範

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。在此範例中，我們將示範設定為使用預設的 OTA 完成回呼，並每秒傳回一些統計資料。為求簡化，我們會從此示範刪掉一下詳細資訊。

OTA 示範也會透過監控中斷連線回呼並重新建立連線來示範 MQTT 連線管理。發生中斷連線時，示範會先暫停 OTA 代理程式操作，然後嘗試重新建立 MQTT 連線。MQTT 重新連線嘗試的延遲時間會以指數方式增加到最大值，也會新增抖動。如果重新建立連線，OTA 代理程式會繼續其操作。

如需使用 AWS IoT MQTT 代理程式的工作範例，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 更新任務並每秒輸出一次統計資料的迴圈。
- 如果 MQTT 中斷連線，請暫停 OTA 代理程式操作。
- 請嘗試以指數延遲和抖動再次連線。

- 如果重新連線，請恢復 OTA 代理程式操作。
- 如果客服人員停止，請延遲一秒鐘，然後嘗試重新連線。

針對 OTA 完成事件使用自訂回撥

先前範例App_OTACompleteCallback用作 OTA 完成事件的回呼處理常式。（請參閱 OTA_AgentInit API 呼叫的第三個參數。）App_OTACompleteCallback 已在 中定義*freertos*/demos/ota/aws_iot_ota_update_demo.c。若您希望實作完成事件的自訂處理，您必須將您回撥處理常式的函數地址傳遞給 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

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

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 更新流量會通過 AWS IoT 裝置閘道並使用 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 的程式碼簽署 AWS IoT 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

透過控制平面 API 進行的每個 API 呼叫都會經過標準 IAM Signature 第 4 版身分驗證和授權。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，您必須提供讀取許可給 AWS IoT 服務主體，以便在串流期間存取存放在 Amazon S3 中的韌體更新。

的程式碼簽署 AWS IoT

AWS IoT 主控台使用適用於 的程式碼簽署 AWS IoT，為 支援的任何裝置自動簽署您的韌體映像 AWS IoT。

的程式碼簽署 AWS IoT 會使用您匯入 ACM 的憑證和私有金鑰。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機構 (CA) 取得憑證。

程式碼簽署憑證使用 X.509 第 3 版`Key Usage`和`Extended Key Usage`擴充功能。`Key Usage` 延伸模組設為 `Digital Signature`，`Extended Key Usage` 延伸模組設為 `Code Signing`。如需有關簽署程式碼映像的詳細資訊，請參閱適用於開發人員的 AWS IoT 程式碼簽署指南和適用於 AWS IoT API 的程式碼簽署參考。

> **Note**
>
> 您可以從適用於 Amazon Web Services 的工具下載 AWS IoT 適用於 SDK 的程式碼簽署。https://aws.amazon.com/tools/

OTA 故障診斷

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

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
- 使用 AWS IoT 記錄 OTA API 呼叫 AWS CloudTrail
- 使用 取得 CreateOTAUpdate 失敗詳細資訊 AWS CLI
- 使用 取得 OTA 失敗代碼 AWS CLI
- 故障診斷多個裝置的 OTA 更新
- 故障診斷 Texas Instruments CC3220SF Launchpad 的 OTA 更新

設定 CloudWatch Logs 以進行 OTA 更新

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

OTA Update 服務支援使用 Amazon CloudWatch 進行記錄。您可以使用 AWS IoT 主控台來啟用和設定 OTA 更新的 Amazon CloudWatch 記錄。如需詳細資訊，請參閱 Cloudwatch Logs。

若要啟用記錄，您必須建立 IAM 角色並設定 OTA 更新記錄。

> **Note**
>
> 啟用 OTA 更新記錄之前，請確定您了解 CloudWatch Logs 存取許可。具有 CloudWatch Logs 存取權限的使用者，皆可查看偵錯資訊。如需資訊，請參閱 Amazon CloudWatch Logs 身分驗證與存取控制。

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. 在設定角色下，選擇建立新的 以建立用於記錄的 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。
- 任務 AWS IoT 已建立。
- AWS IoT 任務已完成。
- 已建立串流。

您可以在 CloudWatch 主控台中檢視日誌。

在 CloudWatch Logs 中檢視 OTA 更新

1. 從導覽窗格中，選擇 Logs (日誌)。
2. 在 Log Groups (日誌群組) 中，選擇 AWSIoTLogsV2。

OTA 更新日誌可包含以下屬性：

accountId

產生日誌的帳戶 AWS ID。

actionType

產生日誌的動作。這可以設為下列其中一個值：

- `CreateOTAUpdate`：已建立 OTA 更新。
- `DeleteOTAUpdate`：已刪除 OTA 更新。
- `StartCodeSigning`：已啟動程式碼簽署任務。
- `CreateAWSJob`：已建立 AWS IoT 任務。
- `CreateStream`：已建立串流。

- `GetStream`：串流的請求已傳送至 AWS IoT MQTT 型檔案交付功能。
- `DescribeStream`：有關串流資訊的請求已傳送至 AWS IoT MQTT 型檔案交付功能。

awsJobId

產生日誌 AWS IoT 的任務 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。

詳細資訊

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。有效的 值如下：

- 成功
- 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 任務時產生的日誌範例：

```
{
    "timestamp": "2018-07-23 22:59:45.363",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "actionType": "CreateAWSJob",
    "otaUpdateId": "08957b03-eea3-448a-87fe-743e6891ca3a",
    "awsJobId": "08957b03-eea3-448a-87fe-743e6891ca3a",
    "details": "Create AWS Job The request status is SUCCESS."
}
```

以下是建立 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 MQTT 型檔案交付功能請求串流時產生的日誌範例：

```
{
    "timestamp": "2018-07-25 22:09:02.678",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "actionType": "GetStream",
    "protocol": "MQTT",
    "clientId": "b9d2e49c-94fe-4ed1-9b07-286afed7e4c8",
    "topicName": "$aws/things/b9d2e49c-94fe-4ed1-9b07-286afed7e4c8/
streams/1e51e9a8-9a4c-4c50-b005-d38452a956af/get/json",
    "streamId": "1e51e9a8-9a4c-4c50-b005-d38452a956af",
    "details": "The request status is SUCCESS."
}
```

以下是裝置呼叫 `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 IoT 記錄 OTA API 呼叫 AWS CloudTrail

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

FreeRTOS 已與 CloudTrail 整合，這項服務會擷取 AWS IoT OTA API 呼叫，並將日誌檔案交付至您指定的 Amazon S3 儲存貯體。CloudTrail 會擷取從程式碼到 OTA API APIs AWS IoT 呼叫。您可以使用 CloudTrail 所收集的資訊，判斷向 OTA AWS IoT 提出的請求、提出請求的來源 IP 地址、提出請求的人員、提出請求的時間等。

如需 CloudTrail 的相關詳細資訊，包括如何設定與啟用，請參閱AWS CloudTrail 使用者指南。

CloudTrail 中的 FreeRTOS 資訊

當 AWS 您的帳戶中啟用 CloudTrail 記錄時，對 AWS IoT OTA 動作發出的 API 呼叫會在 CloudTrail 日誌檔案中進行追蹤，而這些日誌檔案會與其他 AWS 服務記錄一起寫入。CloudTrail 會根據期間與檔案大小，決定何時建立與寫入新檔案。

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**
>
> AWS IoT CloudTrail 不會記錄 OTA 資料平面動作 （裝置端）。使用 CloudWatch 監控這些項目。

每個日誌項目都會包含產生要求之人員的資訊。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 使用者憑證提出。
- 提出該請求時，是否使用了特定角色或聯合身分使用者的暫時安全憑證。
- 請求是否由其他 AWS 服務提出。

如需詳細資訊，請參閱 AWS IoT OTA API 參考中的 CloudTrail userIdentity Element.OTA 動作。AWS IoT

您可以視需要將日誌檔案存放在 Amazon S3 儲存貯體中，但您也可以定義 Amazon S3 生命週期規則來自動封存或刪除日誌檔案。您的日誌檔案預設使用 Amazon S3 伺服器端加密 (SSE) 加密。

如果您想要在日誌檔案交付時收到通知，您可以設定 CloudTrail 來發佈 Amazon SNS 通知。如需詳細資訊，請參閱設定 CloudTrail 的 Amazon SNS 通知。

您也可以將多個 AWS 區域和多個 AWS 帳戶的 AWS IoT OTA 日誌檔案彙總到單一 Amazon S3 儲存貯體。

如需詳細資訊，請參閱從多個區域接收 CloudTrail 日誌檔案和從多個帳戶接收 CloudTrail 日誌檔案。

了解 FreeRTOS 日誌檔案項目

CloudTrail 日誌檔案可以包含一或多個日誌項目。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。

下列範例顯示 CloudTrail 日誌項目，示範從呼叫 `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_CMH"
        ],
        "roleArn": "arn:aws:iam::your_aws_account:role/Role_FreeRTOSJob",
        "files": [
            {
                "fileName": "/sys/mcuflashimg.bin",
                "fileSource": {
                    "fileId": 0,
                    "streamId": "your_stream_id"
                },
                "codeSigning": {
                    "awsSignerJobId": "your_signer_job_id"
                }
            }
        ],
        "targetSelection": "SNAPSHOT",
        "otaUpdateId": "FreeRTOSJob_CMH-23-1535045232806-92"
```

```
    },
    "responseElements": {
        "otaUpdateArn": "arn:aws:iot:your_aws_region:your_aws_account:otaupdate/
FreeRTOSJob_CMH-23-1535045232806-92",
        "otaUpdateStatus": "CREATE_PENDING",
        "otaUpdateId": "FreeRTOSJob_CMH-23-1535045232806-92"
    },
    "requestID": "c9649630-a6f9-11e8-8f9c-e1cf2d0c9d8e",
    "eventID": "ce9bf4d9-5770-4cee-acf4-0e5649b845c0",
    "eventType": "AwsApiCall",
    "recipientAccountId": "recipient_aws_account"
}
```

使用 取得 CreateOTAUpdate 失敗詳細資訊 AWS CLI

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

如果建立 OTA 更新任務的程序失敗，您可能會採取一些動作來修正問題。當您建立 OTA 更新任務時，OTA 管理員服務會建立 IoT 任務，並為目標裝置排程任務，此程序也會在您的帳戶中建立或使用其他類型的 AWS 資源 （程式碼簽署任務、 AWS IoT 串流、Amazon S3 物件）。遇到的任何錯誤都可能導致程序失敗，而無需建立 AWS IoT 任務。在此故障診斷區段中，我們會提供如何擷取失敗詳細資訊的指示。

1. 安裝及設定 AWS CLI。
2. 執行 aws configure並輸入下列資訊。

   ```
   $ aws configure
   AWS Access Key ID [None]: AccessID
   AWS Secret Access Key [None]: AccessKey
   Default region name [None]: Region
   Default output format [None]: json
   ```

   如需詳細資訊，請參閱使用 進行快速組態aws configure。
3. 執行：

```
aws iot get-ota-update --ota-update-id ota_update_job_001
```

其中 *ota_update_job_001* 是您建立 OTA 更新時提供的 ID。

4. 輸出看起來像這樣：

```
{
    "otaUpdateInfo": {
        "otaUpdateId": "ota_update_job_001",
        "otaUpdateArn":
 "arn:aws:iot:region:account_id:otaupdate/ota_update_job_001",
        "creationDate": 1584646864.534,
        "lastModifiedDate": 1584646865.913,
        "targets": [
            "arn:aws:iot:region:account_id:thing/thing_001"
        ],
        "protocols": [
            "MQTT"
        ],
        "awsJobExecutionsRolloutConfig": {},
        "awsJobPresignedUrlConfig": {},
        "targetSelection": "SNAPSHOT",
        "otaUpdateFiles": [
            {
               "fileName": "/12ds",
                "fileLocation": {
                    "s3Location": {
                        "bucket": "bucket_name",
                        "key": "demo.bin",
                        "version": "Z7X.TWSAS7JSi4rybc02nMdcE41W1tV3"
                    }
                },
                "codeSigning": {
                    "startSigningJobParameter": {
                        "signingProfileParameter": {},
                        "signingProfileName": "signing_profile_name",
                        "destination": {
                            "s3Destination": {
                                "bucket": "bucket_name",
                                "prefix": "SignedImages/"
                            }
                        }
```

```
                    },
                    "customCodeSigning": {}
                }
            }
        ],
        "otaUpdateStatus": "CREATE_FAILED",
        "errorInfo": {
            "code": "AccessDeniedException",
            "message": "S3 object demo.bin not accessible. Please check
 your permissions (Service: AWSSigner; Status Code: 403; Error Code:
 AccessDeniedException; Request ID: 01d8e7a1-8c7c-4d85-9fd7-dcde975fdd2d)"
        }
    }
}
```

如果建立失敗，命令輸出中的 `otaUpdateStatus` 欄位將包含 ，`CREATE_FAILED`而 `errorInfo` 欄位將包含失敗的詳細資訊。

使用 取得 OTA 失敗代碼 AWS CLI

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

1. 安裝及設定 AWS CLI。
2. 執行 aws configure並輸入下列資訊。

```
$ aws configure
AWS Access Key ID [None]: AccessID
AWS Secret Access Key [None]: AccessKey
Default region name [None]: Region
Default output format [None]: json
```

如需詳細資訊，請參閱使用 的快速組態aws configure。

3. 執行：

```
aws iot describe-job-execution --job-id JobID --thing-name ThingName
```

其中 *JobID* 是我們想要取得狀態之任務的完整任務 ID 字串（它在建立時與 OTA 更新任務相關聯），而 *ThingName* 是裝置在 中註冊為 的 AWS IoT 物件名稱 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。一般錯誤碼列在 https://docs.aws.amazon.com/freertos/latest/lib-ref/html1/aws__ota__agent_8h.html。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

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

若要在使用相同韌體映像的多個裝置 (物件) 上執行 OTA，請實作函數 (例如 getThingName()) 從非揮發性記憶體擷取 clientcredentialIOT_THING_NAME。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

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

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

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

FreeRTOS 程式庫為 FreeRTOS 核心及其內部程式庫提供額外的功能。您可以使用 FreeRTOS 程式庫，在內嵌應用程式中建立聯網和安全性。FreeRTOS 程式庫也可讓您的應用程式與服務互動 AWS IoT 。FreeRTOS 包含程式庫，可讓您：

- 使用 MQTT 和裝置影子，將裝置安全地連線至 AWS IoT 雲端。
- 探索並連接至 AWS IoT Greengrass 核心。
- 管理 Wi-Fi 連線。
- 聆聽和處理 FreeRTOS Over-the-Air更新。

`libraries` 目錄包含 FreeRTOS 程式庫的原始程式碼。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

下列移植程式庫包含在 FreeRTOS 組態中，可在 FreeRTOS 主控台下載。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。如需將這些程式庫移植到裝置的詳細資訊，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，您的微控制器可以透過閘道裝置與 AWS IoT MQTT 代理程式通訊。如需詳細資訊，請參閱低功耗藍牙程式庫。

程式庫	API 參考	描述
無線更新	OTA 代理程式 API 參考	FreeRTOS AWS IoT Over-the-Air (OTA) 代理程式程式程式庫會將您的 FreeRTOS 裝置連線至 OTA AWS IoT 代理程式。 如需詳細資訊，請參閱OTA 代理程式程式庫。
FreeRTOS+POSIX	FreeRTOS+POSIX API 參考	您可以使用 FreeRTOS+POSIX 程式庫，將與 POSIX 相容的應用程式移植到 FreeRTOS 生態系統。 如需詳細資訊，請參閱 FreeRTOS+POSIX。
Secure Sockets	Secure Sockets API 參考	如需詳細資訊，請參閱Secure Sockets 程式庫。
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。 如需詳細資訊，請參閱 FreeRTOS+TCP。
Wi-Fi	Wi-Fi API 參考	FreeRTOS Wi-Fi 程式庫可讓您與微型控制器的較低層級無線堆疊進行界面。 如需詳細資訊，請參閱Wi-Fi 程式庫。

程式庫	API 參考	描述
corePKCS11		corePKCS11 程式庫是公有金鑰密碼編譯標準 #11 的參考實作，以支援佈建和 TLS 用戶端身分驗證。 如需詳細資訊，請參閱corePKCS11 程式庫。
TLS		如需詳細資訊，請參閱Transport Layer Security。
通用 I/O	通用 I/O API 參考	如需詳細資訊，請參閱通用 I/O。

FreeRTOS 應用程式程式庫

您可以選擇在 FreeRTOS 組態中包含下列獨立應用程式程式庫，以與雲端上的 AWS IoT 服務互動。

> **Note**
>
> 某些應用程式程式庫與適用於 Embedded C 的 Device SDK AWS IoT 中的程式庫具有相同的 APIs。 如需這些程式庫，請參閱 AWS IoT 裝置 SDK C API 參考。如需適用於 Embedded C 的 AWS IoT Device SDK 的詳細資訊，請參閱 AWS IoT 適用於 Embedded C 的裝置 SDK。

FreeRTOS 應用程式程式庫

程式庫	API 參考	描述
AWS IoT Device Defender	Device Defender C SDK API 參考	FreeRTOS AWS IoT Device Defender 程式庫會將您的 FreeRTOS 裝置連接到 AWS IoT Device Defender。 如需詳細資訊，請參閱AWS IoT Device Defender 程式庫。

程式庫	API 參考	描述
AWS IoT Greengrass	Greengrass API 參考	FreeRTOS AWS IoT Greengrass 程式庫會將您的 FreeRTOS 裝置連接到 AWS IoT Greengrass。 如需詳細資訊，請參閱AWS IoT Greengrass 探索程式庫。
MQTT	MQTT (v1.x.x) 程式庫 API 參考 MQTT (v1) 代理程式 API 參考 MQTT (v2.x.x) C SDK API 參考	coreMQTT 程式庫為您的 FreeRTOS 裝置提供用戶端，以發佈和訂閱 MQTT 主題。MQTT 是裝置用來互動的通訊協定 AWS IoT。 如需 coreMQTT 程式庫 3.0.0 版的詳細資訊，請參閱 coreMQTT 程式庫。
AWS IoT 裝置影子	Device Shadow C SDK API 參考	AWS IoT Device Shadow 程式庫可讓您的 FreeRTOS 裝置與 AWS IoT 裝置影子互動。 如需詳細資訊，請參閱AWS IoT 裝置影子程式庫。

FreeRTOS 通用程式庫

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 移植和應用程式程式庫的相依性。

> **Note**
>
> Device AWS IoT SDK for Embedded C 包含具有與這些程式庫相同 APIs和功能的常見程式庫。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 通用程式庫

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。
Linear Containers	Linear Containers C SDK API 參考	如需詳細資訊，請參閱Linear Containers 程式庫。
日誌	Logging C SDKAPI 參考	如需詳細資訊，請參閱記錄程式庫。
Static Memory	Static Memory C SDK API 參考	如需詳細資訊，請參閱Static Memory 程式庫。
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。

設定 FreeRTOS 程式庫

FreeRTOS 和 AWS IoT 適用於內嵌 C 的裝置 SDK 的組態設定定義為 C 前置處理器常數。您可以透過全域組態檔案或使用編譯器選項 (例如 gcc 中的 -D) 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，便必須重新建立程式庫。

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 檔案中，使用 #define 指令來設定 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

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

FreeRTOS 包含一些常見的程式庫，這些程式庫可延伸核心功能，以及用於內嵌應用程式開發的額外資料結構和函數。這些程式庫通常是其他 FreeRTOS 程式庫的相依性。

主題

- 不可部分完成操作
- Linear Containers 程式庫
- 記錄程式庫
- Static Memory 程式庫
- 任務集區程式庫

不可部分完成操作

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

概觀

原子操作可確保並行程式設計中的非封鎖同步。您可以使用原子操作來解決由對共用記憶體採取動作的非同步操作所造成的效能問題。FreeRTOS 支援原子操作，如`iot_atomic.h`標頭檔案中實作。

`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)。

初始化

在使用 FreeRTOS Atomic Operations 之前，您需要選擇要使用的原子操作實作。

1. 開啟 `FreeRTOSConfig.h` 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

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

FreeRTOS 線性容器程式庫定義了線性資料結構，包括清單和佇列，供您在開發內嵌應用程式時使用。

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

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

FreeRTOS 記錄程式庫可讓 FreeRTOS 程式庫將訊息列印到日誌以進行偵錯。

API 參考

如需完整的 API 參考，請參閱記錄 C SDKAPI 參考。

Static Memory 程式庫

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

FreeRTOS 靜態記憶體程式庫會定義一些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

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

概觀

FreeRTOS 支援使用 FreeRTOS 任務集區程式庫進行任務管理。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 API，您可以設定您應用程式的任務，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`)

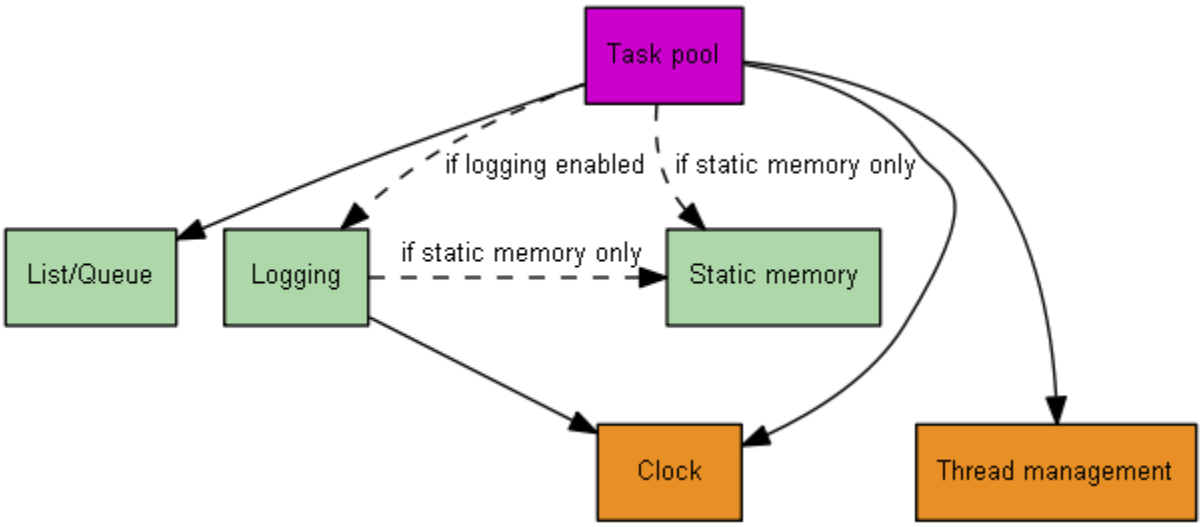
任務集區工作可做為背景工作或定時的背景工作來執行。背景工作是以先進先出的順序開始進行的，且沒有時間限制。定時工作是根據計時器來排定以在背景執行。

> **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。



功能

使用任務集區程式庫 API，您可以執行下列動作：

- 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。

故障診斷

任務集區程式庫函式會以 IotTaskPoolError_t 列舉值傳回錯誤代碼。如需每個錯誤代碼的詳細資訊，請參閱任務集區 C SDK API 參考中的IotTaskPoolError_t列舉資料類型參考文件。 https://docs.aws.amazon.com/freertos/latest/lib-ref/c-sdk/taskpool/index.html

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。

初始化

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_TASKP00L 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，而且您決定使用計時器來排程集合，以呼叫 MQTT 連線、訂閱和發佈 APIs。下列程式碼定義一個回呼函數，以接受 MQTT 中的 AWS IoT Device Defender 指標，並具有中斷連線回呼，該回呼會中斷與 MQTT 連線的連線。

```
/* 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 );
}
```

backoffAlgorithm 程式庫

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

簡介

backoffAlgorithm 程式庫是公用程式程式庫，用於隔開相同資料區塊的重複重新傳輸，以避免網路擁塞。此程式庫會使用指數退避與抖動演算法，計算重試網路操作的退避期（例如與伺服器的網路連線失敗）。

具有抖動的指數退避通常在重試失敗的連線或網路請求到伺服器時，由伺服器的網路擁塞或高負載造成。它用於分散多個裝置同時嘗試網路連線所建立重試請求的時間。在連線能力不佳的環境中，用戶端可以隨時中斷連線；因此退避策略也有助於用戶端在不太可能成功時不重複嘗試重新連線，以節省電池。

程式庫是以 C 撰寫，且設計符合 ISO C90 和 MISRA C：2012。程式庫對標準 C 程式庫以外的任何其他程式庫沒有相依性，也沒有堆積配置，使其適合 IoT 微控制器，但也可完全可攜式至其他平台。

此程式庫可以自由使用，並根據 MIT 開放原始碼授權進行分發。

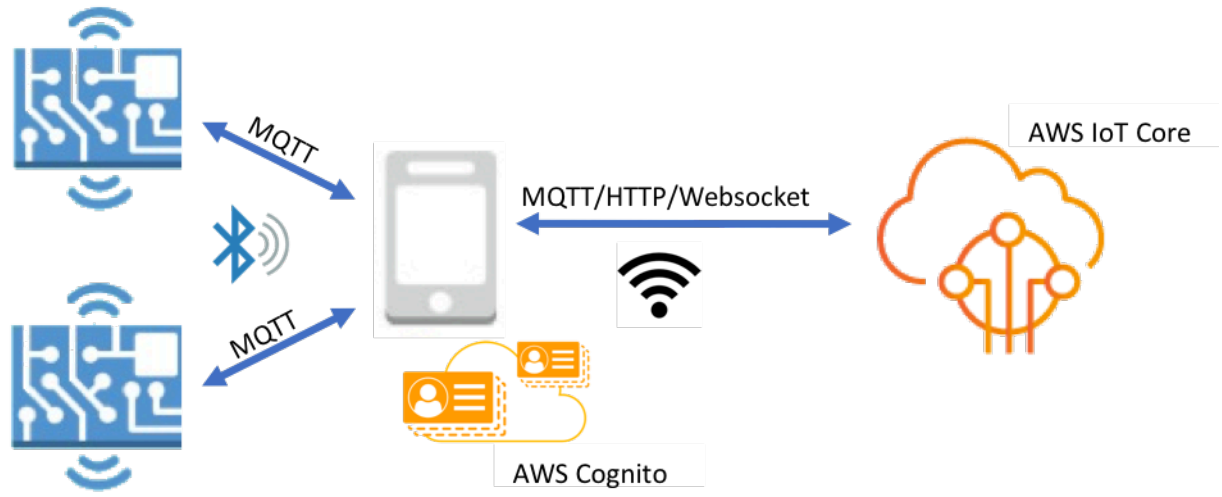
backoffAlgorithm 的程式碼大小（使用 GCC for ARM Cortex-M 產生的範例）		
檔案	使用 -O1 最佳化	使用 -Os 最佳化
backoff_algorithm.c	0.1K	0.1K
預估總數	0.1K	0.1K

低功耗藍牙程式庫

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

概觀

FreeRTOS 支援透過藍牙低功耗代理裝置發佈和訂閱 MQTT 主題，例如行動電話。透過 FreeRTOS 低功耗藍牙程式庫，您的微型控制器可以安全地與 AWS IoT MQTT 代理程式通訊。

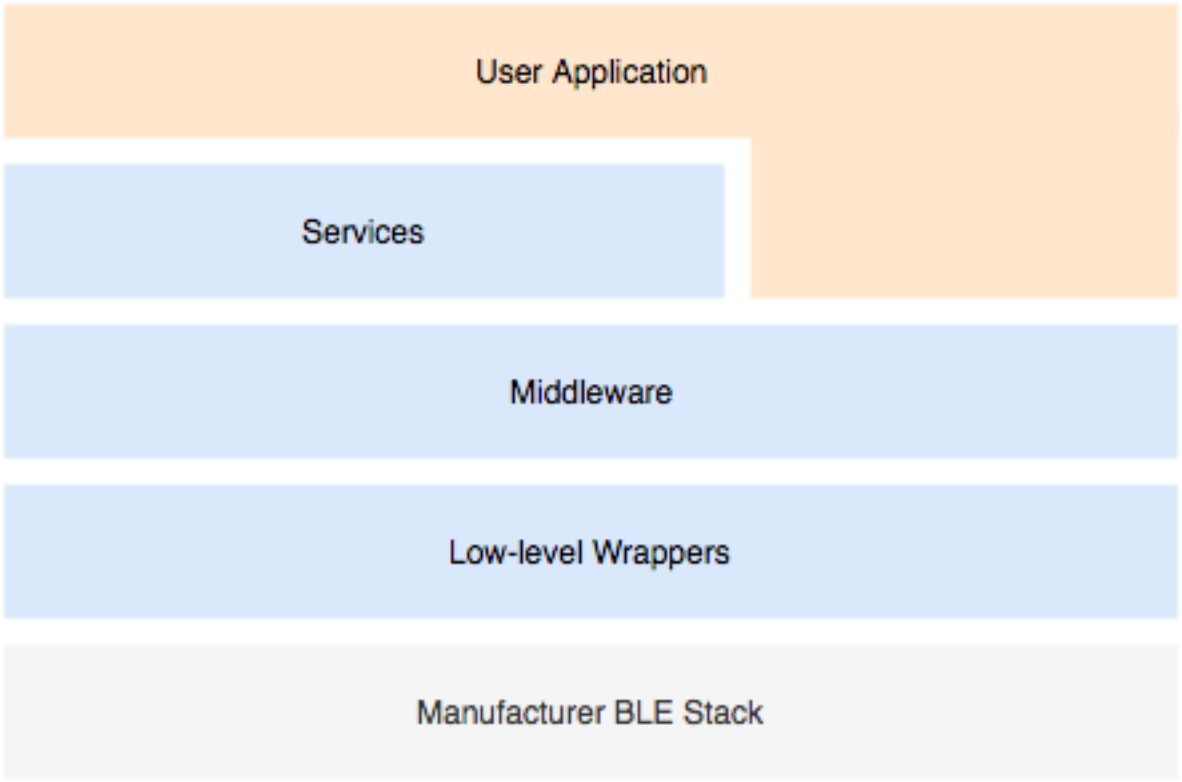


使用適用於 FreeRTOS 藍牙裝置的 Mobile SDKs，您可以撰寫原生行動應用程式，透過低功耗藍牙與微控制器上的內嵌應用程式通訊。如需行動軟體開發套件的詳細資訊，請參閱 適用於 FreeRTOS 藍牙裝置的 Mobile SDKs 。

FreeRTOS 低功耗藍牙程式庫包含設定 Wi-Fi 網路、傳輸大量資料，以及透過低功耗藍牙提供網路摘要的服務。FreeRTOS 低功耗藍牙程式庫也包含中介軟體和低階 APIs可讓您更直接控制低功耗藍牙堆疊。

架構

FreeRTOS 低功耗藍牙程式庫由三層組成：服務、中介軟體和低階包裝函式。



服務

FreeRTOS 低功耗藍牙服務層包含四個通用屬性 (GATT) 服務，其利用中介軟體 APIs：裝置資訊、Wi-Fi 佈建、網路抽象和大型物件傳輸。

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。
- 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。

中介軟體

FreeRTOS 低功耗藍牙中介軟體是低階 APIs抽象。中介軟體 API 對低功耗藍牙堆疊組成了使用者更易用的界面。

使用中介軟體 API，您可以跨多層向單一事件註冊數個回呼。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

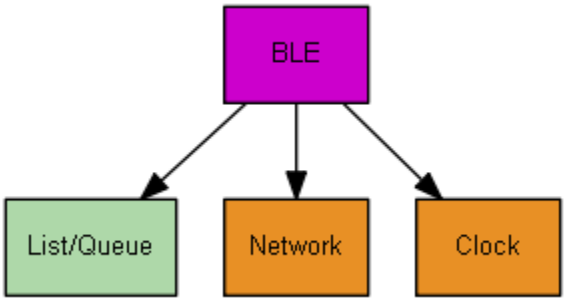
低階 FreeRTOS 低功耗藍牙包裝函式是製造商低功耗藍牙堆疊的抽象。低階包裝函式提供一組常用的 API，以直接控制硬體。低階 API 會最佳化 RAM 使用量，但限於功能方面。

使用低功耗藍牙服務 API 以和低功耗藍牙服務進行互動。服務 API 需要的資源比低階 API 更多。

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
- 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 佈建服務具有 FreeRTOS 程式庫相依性：

GATT 服務	相依性
Wi-Fi 佈建	Wi-Fi 程式庫

若要與 AWS IoT MQTT 代理程式通訊，您必須擁有 AWS 帳戶，而且必須將裝置註冊為 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 低功耗藍牙使用 Amazon Cognito 在您的行動裝置上進行使用者身分驗證。若要使用 MQTT 代理服務，您必須建立 Amazon Cognito 身分和使用者集區。每個 Amazon Cognito Identity 都必須連接適當的政策。如需詳細資訊，請參閱 Amazon Cognito 開發人員指南。

程式庫組態檔案

使用 FreeRTOS MQTT over Bluetooth Low Energy 服務的應用程式必須提供定義組態參數的`iot_ble_config.h`標頭檔案。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。

最佳化

當最佳化主機板的效能時，請考慮以下情況：

- 低階 API 使用更少的 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，這會讓裝置進入僅安全連線模式。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。

初始化

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。

中介軟體

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，則不需要此初始化步驟。示範初始化是由 *freertos*`/demos/network_manager` 中的網路管理員進行處理。

低階 API

如果您不想使用 FreeRTOS 低功耗藍牙 GATT 服務，您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 API

1. 在呼叫 API 前初始化任何低功耗藍牙硬體驅動程式。驅動程式初始化並不屬於低功耗藍牙低階 API。

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 API 之前，您必須啟用低功耗藍牙。

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

3. 藍牙管理員包含低功耗藍牙和藍牙傳統常用的 API。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。

廣告

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。

3. 開啟 vendors/*vendor*/boards/*board*/aws_demos/config_files/iot_ble_config.h，並將 IOT_BLE_SET_CUSTOM_ADVERTISEMENT_MSG 設為 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。

1. 為服務的特性和描述項建立 UUID：

```
#define xServiceUUID_TYPE \
{\
    .uu.uu128 = gattDemoSVC_UUID, \
    .ucType   = eBTuuidType128 \
}
#define xCharCounterUUID_TYPE \
{\
    .uu.uu128 = gattDemoCHAR_COUNTER_UUID,\
    .ucType   = eBTuuidType128\
}
#define xCharControlUUID_TYPE \
{\
    .uu.uu128 = gattDemoCHAR_CONTROL_UUID,\
    .ucType   = eBTuuidType128\
}
#define xClientCharCfgUUID_TYPE \
{\
    .uu.uu16 = gattDemoCLIENT_CHAR_CFG_UUID,\
    .ucType  = eBTuuidType16\
}
```

2. 建立緩衝以註冊特性和描述項的處理常式：

```
static uint16_t usHandlesBuffer[egattDemoNbAttributes];
```

3. 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。

   a. 在 vendors/*vendor*/boards/*board*/aws_demos/config_files/iot_ble_config.h 中將 IOT_BLE_ADD_CUSTOM_SERVICES 設為 1。

   b. 在應用程式中建立 IotBle_AddCustomServicesCb：

```
void IotBle_AddCustomServicesCb(void)
{
    BTStatus_t xStatus;
    /* Select the handle buffer. */
    xStatus = IotBle_CreateService( (BTService_t *)&xGattDemoService,
 (IotBleAttributeEventCallback_t *)pxCallBackArray );
}
```

移植

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 移植到新目標，您必須實作 Wi-Fi 佈建服務和低功耗藍牙功能的一些 APIs。

低功耗藍牙 API

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 與 GAP for Bluetooth Low Energy 之間常用的 API

- `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`

適用於低功耗藍牙的 GAP 特定 API

- `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平台的詳細資訊，請參閱 FreeRTOS 移植指南中的移植低功耗藍牙程式庫。

適用於 FreeRTOS 藍牙裝置的 Mobile SDKs

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

您可以使用適用於 FreeRTOS 的 Mobile SDKs 藍牙裝置，建立透過低功耗藍牙與微型控制器互動的行動應用程式。Mobile SDKs 也可以使用 Amazon Cognito 進行使用者身分驗證 AWS ，與服務通訊。

適用於 FreeRTOS 藍牙裝置的 Android 開發套件

使用適用於 FreeRTOS 藍牙裝置的 Android 開發套件，建置可透過低功耗藍牙與您的微型控制器互動的 Android 行動應用程式。您也可在 GitHub 取得軟體開發套件。

若要安裝適用於 FreeRTOS 藍牙裝置的 Android 開發套件，請遵循專案 README.md 檔案中「設定開發套件」的指示。

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，請參閱先決條件和FreeRTOS 低功耗藍牙行動 SDK 示範應用程式。

適用於 FreeRTOS 藍牙裝置的 iOS 開發套件

使用適用於 FreeRTOS 藍牙裝置的 iOS 開發套件來建置 iOS 行動應用程式，透過低功耗藍牙與您的微型控制器互動。您也可在 GitHub 取得軟體開發套件。

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 SDK (將其新增至您的 podfile 中)：

```
$ 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，請參閱先決條件和FreeRTOS 低功耗藍牙行動 SDK 示範應用程式。

通用 I/O

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

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 API，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 通用輸入/輸出可做為此硬體抽象層。它提供了一組標準 API，用於在支援的參考電路板上存取通用序列裝置。這些通用 API 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 – 在傳輸請求的資料量之前，函數不會傳回。
- 非同步讀取/寫入 – 函數會立即傳回，且資料傳輸會以非同步方式進行。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 – 將多個操作合併為一個交易。用來在一個交易上執行先寫入後讀取動作。
- SPI – 在主要和次要之間傳輸資料，這表示寫入和讀取會同時發生。

移植

如需詳細資訊，請參閱 FreeRTOS 移植指南。

AWS IoT Device Defender 程式庫

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

簡介

您可以使用 AWS IoT Device Defender 程式庫，將安全指標從 IoT 裝置傳送至 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，以找出與您定義之

每個裝置適當行為的偏差。如果看起來不正確， AWS IoT Device Defender 會傳送提醒，以便您可以採取行動來修正問題。與 的互動 AWS IoT Device Defender 使用 MQTT，這是一種輕量型發佈訂閱通訊協定。此程式庫提供 API 來編寫和辨識 所使用的 MQTT 主題字串 AWS IoT Device Defender。

如需詳細資訊，請參閱《AWS IoT 開發人員指南》中的 AWS IoT Device Defender。

程式庫是以 C 撰寫，且設計為符合 ISO C90 和 MISRA C：2012。程式庫對標準 C 程式庫以外的任何其他程式庫沒有相依性。它也沒有任何平台相依性，例如執行緒或同步。它可與任何 MQTT 程式庫和任何 JSON 或 CBOR 程式庫搭配使用。程式庫有證據顯示安全記憶體使用和無堆積配置，使其適合 IoT 微控制器，但也可完全可攜式至其他平台。

您可以自由使用程式 AWS IoT Device Defender 庫，並根據 MIT 開放原始碼授權進行分發。

AWS IoT Device Defender 的程式碼大小（使用 GCC for ARM Cortex-M 產生的範例）		
檔案	使用 -O1 最佳化	使用 -Os 最佳化
defender.c	1.1K	0.6K
預估總數	1.1K	0.6K

AWS IoT Greengrass 探索程式庫

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

概觀

AWS IoT Greengrass Discovery 程式庫是由您的微型控制器裝置用來探索您網路上的 Greengrass 核心。使用 AWS IoT Greengrass Discovery API，您的裝置可在找到核心的端點後將訊息傳送到 Greengrass 核心。

相依性和要求

若要使用 Greengrass 探索程式庫，您必須在 中建立物件 AWS IoT，包括憑證和政策。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 入門 AWS IoT Greengrass。

雖然 Greengrass 連線不需要 coreMQTT 程式庫，但我們強烈建議您安裝它。在搜索到之後，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 裝置會透過向包含 Greengrass 核心連線參數的 AWS IoT JSON 檔案請求 來啟動探索程序。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)，您必須呼叫以下 API：

`GGD_JSONRequestStart`

向 發出 HTTP GET 請求 AWS IoT ，以啟動探索請求，以探索 Greengrass 核心。`GD_SecureConnect_Send` 用於將請求傳送至 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` 會使用 `GGD_SecureConnect_Read` 從通訊端取得 JSON 資料。必須呼叫 `GGD_JSONRequestStart`、`GGD_SecureConnect_Send`、`GGD_JSONRequestGetSize` 來從 AWS IoT接收 JSON 資料。

`GGD_GetIPandCertificateFromJSON`

從 JSON 資料擷取 IP 地址和 Greengrass 核心憑證。您可以透過將 `xAutoSelectFlag` 設為 `True` 來開啟自動選取。自動選取會尋找您 FreeRTOS 裝置能連線到的第一個核心裝置。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

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

適用於小型 IoT 裝置 (MCU 或小型 MPU) 的 HTTP C 用戶端程式庫

簡介

coreHTTP 程式庫是 HTTP/1.1 標準子集的用戶端實作。HTTP 標準提供在 TCP/IP 上執行的無狀態通訊協定，通常用於分散式、協作的超文字資訊系統。

coreHTTP 程式庫會實作 HTTP/1.1 通訊協定標準的子集。此程式庫已針對低記憶體佔用量進行最佳化。程式庫提供完全同步的 API，讓應用程式可以完全管理其並行。它只會使用固定緩衝區，讓應用程式完全控制其記憶體配置策略。

程式庫是以 C 撰寫，且設計符合 ISO C90 和 MISRA C：2012。程式庫的唯一相依性是 Node.js 中 http-parser 的標準 C 程式庫和 LTS 版本 (v12.19.1)。程式庫有證據顯示安全記憶體使用和無堆積配置，使其適合 IoT 微控制器，但也可完全可攜式至其他平台。

在 IoT 應用程式中使用 HTTP 連線時，我們建議您使用安全傳輸介面，例如使用如 coreHTTP 示範（共同身分驗證）所示的 TLS 通訊協定。

此程式庫可以自由使用，並根據 MIT 開放原始碼授權進行分發。

coreHTTP 的程式碼大小（使用 GCC for ARM Cortex-M 產生的範例）		
檔案	使用 -O1 最佳化	使用 -Os 最佳化
core_http_client.c	3.2K	2.6K
http_parser.c (http-parser)	15.7K	13.0K
預估總數	18.9K	15.6K

coreJSON 程式庫

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

簡介

JSON (JavaScript 物件表示法）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，例如使用 AWS IoT Device Shadow 服務，並且是許多 APIs的一部分，例如 GitHub REST API。JSON 由 Ecma International 維護為標準。

coreJSON 程式庫提供支援索引鍵查詢的剖析器，同時嚴格強制執行 ECMA-404 標準 JSON Data Interchange 語法。程式庫以 C 撰寫，旨在符合 ISO C90 和 MISRA C：2012。它具有顯示安全記憶體使用和無堆積配置的證明，使其適合 IoT 微控制器，但也可完全可攜式到其他平台。

記憶體使用

coreJSON 程式庫使用內部堆疊來追蹤 JSON 文件中的巢狀結構。堆疊在單一函數呼叫期間存在；不會保留。堆疊大小可以透過定義巨集 來指定`JSON_MAX_DEPTH`，其預設為 32 個層級。每個層級都會使用單一位元組。

coreJSON 的程式碼大小（使用 GCC for ARM Cortex-M 產生的範例）		
檔案	使用 -O1 最佳化	使用 -Os 最佳化
core_json.c	2.9K	2.4K
預估總數	2.9K	2.4K

coreMQTT 程式庫

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

簡介

coreMQTT 程式庫是 MQTT（訊息佇列遙測傳輸）標準的用戶端實作。MQTT 標準提供輕量型發佈/訂閱（或 PubSub) 訊息通訊協定，在 TCP/IP 上執行，通常用於機器對機器 (M2M) 和物聯網 (IoT) 使用案例。

coreMQTT 程式庫符合 MQTT 3.1.1 通訊協定標準。此程式庫已針對低記憶體佔用量進行最佳化。此程式庫的設計涵蓋不同的使用案例，從僅使用 QoS 0 MQTT PUBLISH 訊息的資源限制平台，到使用 QoS QoS 2 MQTT PUBLISH over TLS (Transport Layer Security) 連線的資源豐富平台。程式庫提供可組合函數的功能表，可加以選擇和組合，以精確符合特定使用案例的需求。

程式庫是以 C 撰寫，且設計為符合 ISO C90 和 MISRA C：2012。此 MQTT 程式庫對任何其他程式庫沒有相依性，但下列除外：

- 標準 C 程式庫
- 客戶實作的網路傳輸介面
- （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，程式庫會與基礎網路驅動程式分離。應用程式寫入器可以選取現有的傳輸介面，或針對其應用程式實作自己的 。

程式庫提供高階 API 來連線至 MQTT 代理程式、訂閱/取消訂閱主題、發佈訊息至主題，以及接收傳入的訊息。此 API 會將上述傳輸界面做為 參數，並使用 來傳送和接收來自 MQTT 代理程式的訊息。

程式庫也會公開低階序列化器/還原序列化器 API。此 API 可用來建置簡單的 IoT 應用程式，僅包含必要的 MQTT 功能子集，無需任何其他額外負荷。序列化器/還原序列化器 API 可與任何可用的傳輸層 API 搭配使用，例如通訊端，以傳送和接收來自代理程式的訊息。

在 IoT 應用程式中使用 MQTT 連線時，我們建議您使用安全傳輸介面，例如使用 TLS 通訊協定的介面。

此 MQTT 程式庫沒有平台相依性，例如執行緒或同步。此程式庫確實有證明安全記憶體使用和無堆積分配的證據，這使其適合 IoT 微控制器，但也可完全可攜式至其他平台。它可以自由使用，並根據 MIT 開放原始碼授權進行分發。

coreMQTT 的程式碼大小 （使用 GCC for ARM Cortex-M 產生的範例）		
檔案	使用 -O1 最佳化	使用 -Os 最佳化
core_mqtt.c	3.0K	2.6K
core_mqtt_state.c	1.4K	1.1K
core_mqtt_serializer.c	2.5K	2.0K
預估總數	6.9K	5.7K

OTA 代理程式程式庫

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

概觀

OTA 代理程式程式庫 可讓您使用 HTTP 或 MQTT 做為通訊協定，管理 FreeRTOS 裝置的韌體更新的通知、下載和驗證。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 OTA 更新的詳細資訊，請參閱FreeRTOS Over-the-Air更新。

功能

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，請參閱 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. 連線至 AWS IoT MQTT 代理程式。如需詳細資訊，請參閱coreMQTT 程式庫。
2. 透過呼叫 `OTA_AgentInit` 初始化 OTA 代理程式。您的應用程式可定義自訂 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. 當 OTA 更新完成時，FreeRTOS 會使用下列其中一個事件來呼叫任務完成回呼：`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。

移植

如需將 OTA 功能移植到平台的詳細資訊，請參閱 FreeRTOS 移植指南中的移植 OTA 程式庫。

記憶體使用

OTA AWS IoT 的程式碼大小 （使用 GCC for ARM Cortex-M 產生的範例）		
檔案	使用 -O1 最佳化	使用 -Os 最佳化
ota.c	8.0K	7.2K
ota_interface.c	0.1K	0.1K
ota_base64.c	0.6K	0.6K
ota_mqtt.c	2.3K	2.2K

OTA AWS IoT 的程式碼大小（使用 GCC for ARM Cortex-M 產生的範例）		
檔案	使用 -O1 最佳化	使用 -Os 最佳化
ota_cbor.c	0.8K	0.6K
ota_http.c	0.3K	0.3K
預估總數	12.1K	11.0K

corePKCS11 程式庫

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

概觀

公有金鑰密碼編譯標準 #11 定義了與平台無關的 API，以管理和使用密碼編譯權杖。PKCS #11 是指標準所定義的 API 和標準本身。PKCS #11 密碼編譯 API 會抽象金鑰儲存、取得/設定密碼編譯物件的屬性，以及工作階段語意。它廣泛用於操縱常見的密碼編譯物件，而且很重要，因為它指定的函數允許應用程式軟體使用、建立、修改和刪除密碼編譯物件，而從未將這些物件暴露到應用程式的記憶體。例如，FreeRTOS AWS 參考整合使用 PKCS #11 API 的一小部分來存取秘密（私有）金鑰，以建立由 Transport Layer Security (TLS) 通訊協定驗證和保護的網路連線，而應用程式從未「查看」金鑰。

corePKCS11 程式庫包含以軟體為基礎的模擬實作，其使用 Mbed TLS 提供的密碼編譯功能 PKCS #11 介面 (API)。使用軟體模擬可實現快速開發和彈性，但預期您將以生產裝置中使用的安全金鑰儲存體專用的實作取代模擬。一般而言，安全加密處理器的廠商，例如受信任平台模組 (TPM)、硬體安全模組 (HSM)、安全元素或任何其他類型的安全硬體環境，會隨硬體分發 PKCS #11 實作。因此，corePKCS11 軟體僅限模擬程式庫的目的是提供非硬體特定 PKCS #11 實作，在切換到生產裝置中的加密處理器特定 PKCS #11 實作之前，允許快速原型設計和開發。

只會實作一部分 PKCS #11 標準，並著重於涉及非對稱金鑰、隨機數字產生和雜湊的操作。目標使用案例包括 TLS 身分驗證的憑證和金鑰管理，以及小型內嵌裝置上的程式碼簽署簽章驗證。請參閱 FreeRTOS 原始程式碼儲存庫中的 檔案 `pkcs11.h`（從標準內文 OASIS 取得）。在 FreeRTOS 參

考實作中，PKCS #11 API 呼叫是由 TLS 協助程式介面進行，以便在 期間執行 TLS 用戶端身分驗證SOCKETS_Connect。PKCS #11 API 呼叫也會由我們的一次性開發人員佈建工作流程進行，以匯入 TLS 用戶端憑證和私有金鑰，以向 MQTT AWS IoT 代理程式進行身分驗證。這兩個使用案例：佈建和 TLS 用戶端身分驗證，只需要實作 PKCS #11 介面標準的一小部分。

功能

使用下列 PKCS #11 子集。此清單的順序大約是支援佈建、TLS 用戶端身分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 參考實作使用 PKCS #11 2048 位元 RSA （僅限簽署） 和 ECDSA 搭配 NIST P-256 曲線。下列指示說明如何根據 P-256 用戶端憑證建立 AWS IoT 物件。

請確定您使用的是下列 （或更新的） 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
```

下列程序假設您使用 aws configure命令來設定 AWS CLI。如需詳細資訊，請參閱AWS Command Line Interface 《 使用者指南》中的使用 進行快速組態aws configure。

根據 P-256 用戶端憑證建立 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 genpkey -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至 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/
86e41339a6d1bbc67abf31faf455092cdebf8f21ffbc67c4d238d1326c7de729"
```

6. 建立政策 (此政策過於寬鬆。 它應該僅用於開發目的。)

```
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。如果您不想為 Greengrass 連線和探索執行 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": "*"
    }
  ]
}
```

7. 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。

移植

如需有關將 corePKCS11 程式庫移植到平台的資訊，請參閱 FreeRTOS 移植指南中的移植 corePKCS11 程式庫。

記憶體使用

corePKCS11 的程式碼大小（使用 GCC for ARM Cortex-M 產生的範例）		
檔案	使用 -O1 最佳化	使用 -Os 最佳化
core_pkcs11.c	0.8K	0.8K
core_pki_utils.c	0.5K	0.3K
core_pkcs11_mbedtls.c	8.7K	7.3K
預估總數	10.0K	8.4K

Secure Sockets 程式庫

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

概觀

您可以使用 FreeRTOS Secure Sockets 程式庫來建立可安全通訊的內嵌應用程式。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 通訊端介面為基礎，並具有 TLS 通訊協定的其他安全通訊選項。如需 FreeRTOS Secure Sockets 程式庫與 Berkeley 通訊端介面之間的差異資訊，請參閱 Secure Sockets API 參考`SOCKETS_SetSockOpt`中的 。

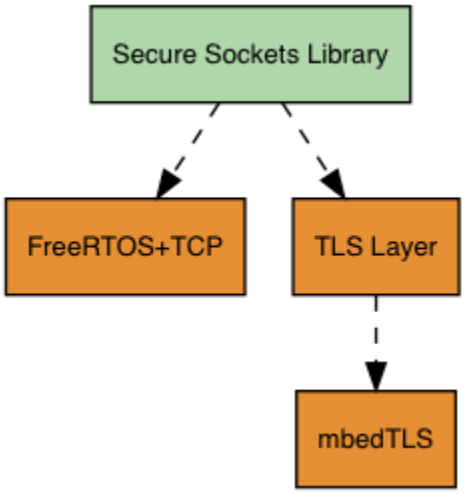
> **Note**
>
> 目前，FreeRTOS Secure Sockets 僅支援用戶端 APIs，以及伺服器端 `Bind` API 的輕量型 IP (lwIP) 實作。

相依性和要求

FreeRTOS Secure Sockets 程式庫取決於 TCP/IP 堆疊和 TLS 實作。FreeRTOS 連接埠透過下列三種方式之一來滿足這些相依性：

- 同時自訂實作 TCP/IP 及 TLS
- TCP/IP 的自訂實作，以及具有 mbedTLS 的 FreeRTOS TLS 層
- FreeRTOS+TCP 和 FreeRTOS TLS layer with mbedTLS

以下相依性圖表顯示 FreeRTOS Secure Sockets 程式庫隨附的參考實作。此參考實作支援透過乙太網路和 Wi-Fi 的 TLS 和 TCP/IP，搭配做為相依性的 FreeRTOS+TCP 和 mbedTLS。如需 FreeRTOS TLS layer 的詳細資訊，請參閱Transport Layer Security。



功能

FreeRTOS Secure Sockets 程式庫功能包括：

- 標準 Berkeley 通訊端型界面
- 用於傳送和接收資料的執行緒安全 API
- 方便啟用 TLS

故障診斷

錯誤代碼

FreeRTOS Secure Sockets 程式庫傳回的錯誤代碼是負值。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，coreMQTT 程式庫則取決於 FreeRTOS Secure Sockets 程式庫的 會傳回錯誤碼 `AWS_IOT_MQTT_SEND_ERROR`。

開發人員支援

FreeRTOS Secure Sockets 程式庫包含兩個用於處理 IP 地址的協助程式巨集：

`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。

FreeRTOS Secure Sockets 程式庫不支援伺服器 APIs，但伺服器端 `Bind` API 的輕量型 IP (lwIP) 實作除外。支援用戶端 APIs。

初始化

若要使用 FreeRTOS Secure Sockets 程式庫，您需要初始化程式庫及其相依性。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。例如，如果 FreeRTOS+TCP 是相依性，則您也需要在應用程式中呼叫 `FreeRTOS_IPInit`。

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.\
Change this to the certificate of your choice. */
static const char cTlsECHO_SERVER_CERTIFICATE_PEM[] =
"-----BEGIN CERTIFICATE-----\n"
"MIIBtjCCAVugAwIBAgITBmyf1XSXNmY/Owua2eiedgPySjAKBggqhkjOPQQDAjA5\n"
"MQswCQYDVQQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwFwYDVQQDExBBbWF6b24g\n"
"Um9vdCBDQSAzMB4XDTE1MDUyNjAwMDAwMFoXDTQwMDUyNjAwMDAwMFowOTELMAkG\n"
"A1UEBhMCVVMxDzANBgNVBAoTBkFtYXpvbjEZMBcGA1UEAxMQQW1hem9uIFJvb3Qg\n"
"Q0EgMzBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABCmXp8ZBf8ANm+gBG1bG8lKl\n"
"ui2yEujSLtf6ycXYqm0fc4E7O5hrOXwzpcVOho6AF2hiRVd9RFgdszflZwjrZt6j\n"
"QjBAMA8GA1UdEwEB/wQFMAMBAf8wDgYDVR0PAQH/BAQDAgGGMB0GA1UdDgQWBBSr\n"
"ttvXBp43rDCGB5Fwx5zEGbF4wDAKBggqhkjOPQQDAgNJADBGAiEA4IWSoxe3jfkr\n"
"BqWTrBqYaGFy+uGh0PsceGCmQ5nFuMQCIQCcAu/xlJyzlvnrxir4tiz+OpAUFteM\n"
"YyRIHN8wfdVoOw==\n"
"-----END CERTIFICATE-----\n";

static const uint32_t ulTlsECHO_SERVER_CERTIFICATE_LENGTH =
 sizeof( cTlsECHO_SERVER_CERTIFICATE_PEM );

void vConnectToServerWithSecureSocket( void )
{
    Socket_t xSocket;
    SocketsSockaddr_t xEchoServerAddress;
    BaseType_t xTransmitted, lStringLength;

    xEchoServerAddress.usPort = SOCKETS_htons( configCLIENT_PORT );
    xEchoServerAddress.ulAddress = SOCKETS_inet_addr_quick( configSERVER_ADDR0,
                                                            configSERVER_ADDR1,
                                                            configSERVER_ADDR2,
                                                            configSERVER_ADDR3 );

    /* Create a TCP socket. */
    xSocket = SOCKETS_Socket( SOCKETS_AF_INET, SOCKETS_SOCK_STREAM,
 SOCKETS_IPPROTO_TCP );
    configASSERT( xSocket != SOCKETS_INVALID_SOCKET );
```

```
    /* Set a timeout so a missing reply does not cause the task to block indefinitely.
 */
    SOCKETS_SetSockOpt( xSocket, 0, SOCKETS_SO_RCVTIMEO, &xReceiveTimeOut,
 sizeof( xReceiveTimeOut ) );
    SOCKETS_SetSockOpt( xSocket, 0, SOCKETS_SO_SNDTIMEO, &xSendTimeOut,
 sizeof( xSendTimeOut ) );

    /* Set the socket to use TLS. */
    SOCKETS_SetSockOpt( xSocket, 0, SOCKETS_SO_REQUIRE_TLS, NULL, ( size_t ) 0 );
    SOCKETS_SetSockOpt( xSocket, 0, SOCKETS_SO_TRUSTED_SERVER_CERTIFICATE,
 cTlsECHO_SERVER_CERTIFICATE_PEM, ulTlsECHO_SERVER_CERTIFICATE_LENGTH );

    if( SOCKETS_Connect( xSocket, &xEchoServerAddress, sizeof( xEchoServerAddress ) )
 == 0 )
    {
        /* Send the string to the socket. */
        xTransmitted = SOCKETS_Send( xSocket,                         /* The socket
 receiving. */
                                     ( void * )"some message",        /* The data being
 sent. */
                                     12,                              /* The length of
 the data being sent. */
                                     0 );                             /* No flags. */

        if( xTransmitted < 0 )
        {
            /* Error while sending data */
            return;
        }

        SOCKETS_Shutdown( xSocket, SOCKETS_SHUT_RDWR );
    }
    else
    {
        //failed to connect to server
    }

    SOCKETS_Close( xSocket );
}
```

如需完整範例，請參閱 Secure Sockets echo 用戶端示範。

移植

FreeRTOS Secure Sockets 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：

- FreeRTOS+TCP TCP/IP 堆疊
- corePKCS11 程式庫
- Transport Layer Security

如需移植的詳細資訊，請參閱 FreeRTOS 移植指南中的移植 Secure Sockets Library。

AWS IoT 裝置影子程式庫

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

簡介

您可以使用 AWS IoT Device Shadow 程式庫來存放和擷取每個已註冊裝置的目前狀態 (影子)。裝置影子是裝置的持久性虛擬表示，即使裝置離線，您也可以在 Web 應用程式中與之互動。裝置狀態會在 JSON 文件中擷取為其影子。您可以透過 MQTT 或 HTTP 將命令傳送至 AWS IoT Device Shadow 服務，以查詢最新的已知裝置狀態，或變更狀態。每個裝置的影子都是由對應物件的名稱、 AWS 雲端上特定裝置或邏輯實體的代表所唯一識別。如需詳細資訊，請參閱使用 管理裝置 AWS IoT。如需陰影的詳細資訊，請參閱 AWS IoT 文件。

AWS IoT Device Shadow 程式庫對標準 C 程式庫以外的其他程式庫沒有相依性。它也沒有任何平台相依性，例如執行緒或同步。它可與任何 MQTT 程式庫和任何 JSON 程式庫搭配使用。

此程式庫可以自由使用，並根據 MIT 開放原始碼授權進行分發。

AWS IoT Device Shadow 的程式碼大小（使用 GCC for ARM Cortex-M 產生的範例）		
檔案	使用 -O1 最佳化	使用 -Os 最佳化
shadow.c	1.2K	0.9K

AWS IoT Device Shadow 的程式碼大小（使用 GCC for ARM Cortex-M 產生的範例）		
檔案	使用 -O1 最佳化	使用 -Os 最佳化
預估總數	1.2K	0.9K

AWS IoT 任務程式庫

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

簡介

AWS IoT 任務是一項服務，可通知一或多個連線裝置待定任務。您可以使用任務來管理裝置機群、更新裝置上的韌體和安全性憑證，或執行管理任務，例如重新啟動裝置和執行診斷。如需詳細資訊，請參閱《 AWS IoT 開發人員指南》中的任務。與 AWS IoT Jobs 服務的互動使用 MQTT，這是一種輕量型發佈訂閱通訊協定。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。

AWS IoT Jobs 程式庫以 C 撰寫，且設計符合 ISO C90 和 MISRA C：2012。程式庫對標準 C 程式庫以外的任何其他程式庫沒有相依性。它可與任何 MQTT 程式庫和任何 JSON 程式庫搭配使用。程式庫有證據顯示安全記憶體使用和無堆積配置，使其適合 IoT 微控制器，但也可完全可攜式至其他平台。

此程式庫可以自由使用，並根據 MIT 開放原始碼授權進行分發。

AWS IoT 任務的程式碼大小（使用 GCC for ARM Cortex-M 產生的範例）		
檔案	使用 -O1 最佳化	使用 -Os 最佳化
job.c	1.8K	1.5K
預估總數	1.8K	1.5K

Transport Layer Security

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

FreeRTOS Transport Layer Security (TLS) 界面是一種精簡、選用的包裝函式，用於在通訊協定堆疊中從其上方的 Secure Sockets Layer (SSL) 界面抽象密碼編譯實作詳細資訊。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 原始程式碼儲存庫`iot_tls.h`中的 。

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 介面的詳細資訊，請參閱 FreeRTOS 移植指南中的移植 TLS 程式庫。

Wi-Fi 程式庫

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

概觀

FreeRTOS Wi-Fi 程式庫會將連接埠特定的 Wi-Fi 實作抽象化為通用 API，可簡化具備 Wi-Fi 功能的所有 FreeRTOS 合格主機板的應用程式開發和移植。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

FreeRTOS Wi-Fi 程式庫需要 FreeRTOS+TCP 核心。

功能

Wi-Fi 程式庫包含下列功能：

- 支援 WEP、WPA、WPA2 和 WPA3 身分驗證
- 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 裝置可以處於三種模式之一：站台、存取點或 P2P。您可以透過呼叫 `WIFI_GetMode` 來取得 Wi-Fi 裝置的目前模式。您可以透過呼叫 `WIFI_SetMode` 來設定裝置的 Wi-Fi 模式。呼叫 `WIFI_SetMode` 切換模式會中斷連線裝置 (若該裝置已連線到網路)。

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。若要設定 AP 模式，請呼叫 `WIFI_ConfigureAP`。若要將您的裝置置入 AP 模式中，請呼叫 `WIFI_StartAP`。若要關閉 AP 模式，請呼叫 `WIFI_StopAP`。

> **Note**
>
> FreeRTOS 程式庫不提供 AP 模式中的 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。

安全

Wi-Fi API 支援 WEP、WPA、WPA2 和 WPA3 安全類型。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。您可以透過呼叫 `WIFI_Disconnect` 來從網路中斷 Wi-Fi 裝置的連線。如需掃描與連接的詳細資訊，請參閱範例使用方式和 API 參考。

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。這可讓您儲存網路設定，並在裝置重新連線到 Wi-Fi 網路時擷取，而無須在先前曾連線到網路之後重新佈建裝置。`WIFI_NetworkAdd` 會新增網路描述檔。`WIFI_NetworkGet` 會擷取網路描述檔。`WIFI_NetworkDel` 則會刪除網路描述檔。您可以儲存的描述檔數量需視平台而定。

組態

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。

> **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。

初始化

除了 FreeRTOS 元件外，您還需要將一些主機板特定的元件初始化，才能使用 Wi-Fi 程式庫。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和 API 參考。

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
}
```

掃描附近的 AP

```
WIFINetworkParams_t xNetworkParams;
WIFIReturnCode_t xWifiStatus;

configPRINT( ("Turning on wifi...\n") );
xWifiStatus = WIFI_On();

configPRINT( ("Checking status...\n") );
if( xWifiStatus == eWiFiSuccess )
{
    configPRINT( ("WiFi module initialized.\n") );
}
else
{
    configPRINTF( ("WiFi module failed to initialize.\n" ) );
    // Handle module init failure
}

WIFI_SetMode(eWiFiModeStation);

/* Some boards might require additional initialization steps to use the Wi-Fi library.
 */

while (1)
{
    configPRINT( ("Starting scan\n") );
    const uint8_t ucNumNetworks = 12; //Get 12 scan results
    WIFIScanResult_t xScanResults[ ucNumNetworks ];
    xWifiStatus = WIFI_Scan( xScanResults, ucNumNetworks ); // Initiate scan

    configPRINT( ("Scan started\n") );
```

```
    // For each scan result, print out the SSID and RSSI
    if ( xWifiStatus == eWiFiSuccess )
    {
        configPRINT( ("Scan success\n") );
        for ( uint8_t i=0; i<ucNumNetworks; i++ )
        {
            configPRINTF( ("%s : %d \n", xScanResults[i].cSSID,
 xScanResults[i].cRSSI) );
        }
    } else {
        configPRINTF( ("Scan failed, status code: %d\n", (int)xWifiStatus) );
    }

    vTaskDelay(200);
}
```

移植

`iot_wifi.c` 實作需要實作 `iot_wifi.h` 中定義的函數。至少，實作需要針對任何非必要或不支援的函數傳回 `eWiFiNotSupported`。

如需有關移植 Wi-Fi 程式庫的詳細資訊，請參閱 FreeRTOS 移植指南中的移植 Wi-Fi 程式庫。

FreeRTOS 示範

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

FreeRTOS 在主要 FreeRTOS 目錄下的 demos 資料夾中包含一些示範應用程式。可由 FreeRTOS 執行的所有範例都會顯示在 common 資料夾的 下demos。資料夾下也有每個符合 FreeRTOS 資格的平台的demos資料夾。

嘗試使用示範應用程式前，我們建議您先完成 FreeRTOS 入門中的教學。它說明如何設定和執行 coreMQTT 相互身分驗證示範。

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
- Microchip Curiosity PIC32MZEF 的示範開機載入器
- AWS IoT Device Defender 示範
- AWS IoT Greengrass 探索示範應用程式
- coreHTTP 示範
- AWS IoT 任務程式庫示範
- coreMQTT 示範
- 無線更新示範應用程式
- Secure Sockets echo 用戶端示範
- AWS IoT Device Shadow 示範應用程式

DEMO_RUNNER_RunDemos 函數位於 *freertos*/demos/demo_runner/iot_demo_runner.c 檔案中，可初始化單一示範應用程式執行所在的分離執行緒。根據預設， DEMO_RUNNER_RunDemos 只會呼叫 並啟動 coreMQTT 相互身分驗證示範。根據您在下載 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預設可能會啟動。若要啟用示範應用程

式，請開啟 *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h 檔案，並定義您要執行的示範。

> **Note**
>
> 並非所有範例組合都一起運作。視組合而定，由於記憶體限制，軟體可能無法在所選目標上執行。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

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

概觀

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 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 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 示範

  此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，請遵循中的步驟FreeRTOS 入門。

先決條件

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 適用於 FreeRTOS 藍牙裝置的 iOS 開發套件或 適用於 FreeRTOS 藍牙裝置的 Android 開發套件。

設定 AWS IoT 和 Amazon Cognito for FreeRTOS 低功耗藍牙

若要 AWS IoT 跨 MQTT 將裝置連線至 ，您需要設定 AWS IoT 和 Amazon Cognito。

設定 AWS IoT

1. 在 https://aws.amazon.com/ 上設定 AWS 帳戶。
2. 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)。
3. 選擇 Create (建立)，然後選擇 Create a single thing (建立單一實物)。
4. 輸入您裝置的名稱，然後選擇 Next (下一步)。
5. 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 Mobile SDKs 使用 Amazon Cognito 進行裝置身分驗證，因此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。
6. 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 入門 AWS IoT。

建立 Amazon Cognito 使用者集區

1. 開啟 Amazon Cognito 主控台，然後選擇管理使用者集區。
2. 選擇 Create a user pool (建立使用者集區)。
3. 為使用者集區提供一個名稱，然後選擇 Review defaults (檢閱預設值)。

4. 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)。
5. 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)。
6. 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。
7. 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 政策並將其連接至已驗證的身分

1. 開啟 IAM 主控台，然後從導覽窗格中選擇角色。
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)。

保留您的 AWS IoT 和 Amazon Cognito 資訊。您需要端點和 IDs才能透過 AWS 雲端驗證行動應用程式。

為低功耗藍牙設定 FreeRTOS 環境

若要設定您的環境，您需要在微控制器低功耗藍牙程式庫上使用 下載 FreeRTOS，並在行動裝置上下載並設定適用於 FreeRTOS 藍牙裝置的 Mobile SDK。

使用 FreeRTOS 低功耗藍牙設定微控制器的環境

1. 從 GitHub 下載或複製 FreeRTOS。如需說明，請參閱 README.md 檔案。
2. 在微控制器上設定 FreeRTOS。

   如需有關在符合 FreeRTOS 資格的微控制器上開始使用 FreeRTOS 的資訊，請參閱 FreeRTOS 入門中的主機板指南。

   > **Note**
   >
   > 您可以在任何啟用低功耗藍牙的微控制器上執行示範，其中包含 FreeRTOS 和移植的 FreeRTOS 低功耗藍牙程式庫。目前，FreeRTOS 透過低功耗藍牙執行的 MQTT示範專案已完全移植到下列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元件：

- Network Manager
- 低功耗藍牙 Mobile SDK 示範應用程式

Network Manager

網路管理員可管理您的微型控制器的網路連線。其位於您的 FreeRTOS 目錄中的 `demos/network_manager/aws_iot_network_manager.c`。如果您已同時為 Wi-Fi 和低功耗藍牙啟用網路管理員，系統依預設會從低功耗藍牙開始執行示範。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啟用網路連線類型，請將網路連線類型新增至 `vendors/vendor/boards/board/aws_demos/config_files/aws_iot_network_config.h` 中的 `configENABLED_NETWORKS` 參數 (其中 *vendor* 是廠商名稱，而 *board* 是您用來執行示範的主機板名稱)。

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 低功耗藍牙行動 SDK 示範應用程式

FreeRTOS 低功耗藍牙行動 SDK 示範應用程式位於 GitHub at Android SDK for FreeRTOS 藍牙裝置 下`amazon-freertos-ble-android-sdk/app`，以及 iOS SDK for FreeRTOS 藍牙裝置 下`amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo`。在本範例中，我們會使用 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 適用於 FreeRTOS 藍牙裝置的 iOS 開發套件。
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. 在區域中建立 AWS IoT 政策（如果您尚未建立）。

> **Note**
>
> 此政策與為 Amazon Cognito 驗證身分建立的 IAM 政策不同。

   a. 開啟 AWS IoT 主控台。
   b. 在導覽窗格中，選擇安全，選擇政策，然後選擇建立。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。將 *aws-region* 和 *aws-account* 取代為您的 AWS 區域和帳戶 ID。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:Connect",
            "Resource":"arn:aws:iot:region:account-id:*"
        },
        {
            "Effect": "Allow",
```

```
            "Action": "iot:Publish",
            "Resource": "arn:aws:iot:region:account-id:*"
        },
        {
             "Effect": "Allow",
             "Action": "iot:Subscribe",
             "Resource": "arn:aws:iot:region:account-id:*"
        },
        {
             "Effect": "Allow",
             "Action": "iot:Receive",
             "Resource": "arn:aws:iot:region:account-id:*"
        }
    ]
}
```

c. 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：您想要發佈至其中的 MQTT 主題。

6. 打開 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：您的應用程式用戶端 ID。
   - AppClientSecret：您的應用程式用戶端密碼。
   - 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 適用於 FreeRTOS 藍牙裝置的 Android 開發套件。
2. 在區域中建立 AWS IoT 政策 （ 如果您尚未建立 ）。

> **Note**
>
> 此政策與為 Amazon Cognito 驗證身分建立的 IAM 政策不同。

   a. 開啟 AWS IoT 主控台。
   b. 在導覽窗格中，選擇安全，選擇政策，然後選擇建立。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。將 *aws-region* 和 *aws-account* 取代為您的 AWS 區域和帳戶 ID。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:Connect",
            "Resource":"arn:aws:iot:region:account-id:*"
        },
        {
            "Effect": "Allow",
            "Action": "iot:Publish",
            "Resource": "arn:aws:iot:region:account-id:*"
        },
        {
             "Effect": "Allow",
             "Action": "iot:Subscribe",
             "Resource": "arn:aws:iot:region:account-id:*"
        },
        {
             "Effect": "Allow",
             "Action": "iot:Receive",
             "Resource": "arn:aws:iot:region:account-id:*"
        }
    ]
```

```
}
```

c. 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`：您的應用程式用戶端 ID。
   - `AppClientSecret`：您的應用程式用戶端密碼。
   - `Region`：您的 AWS 區域。

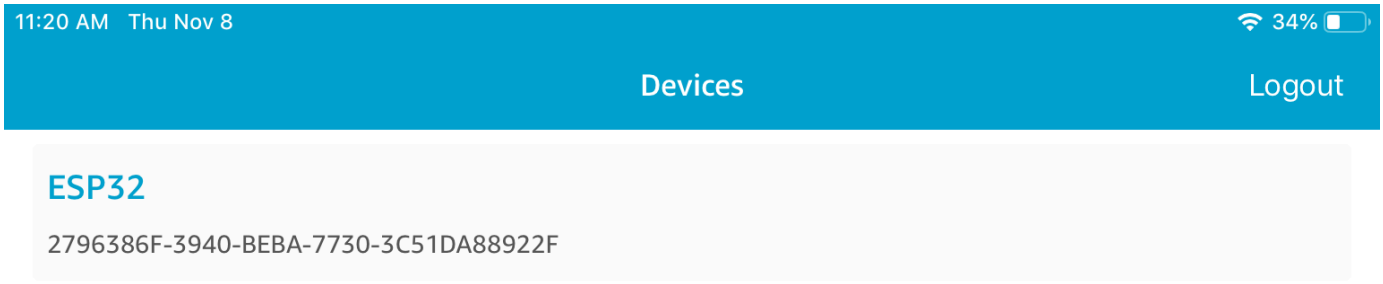
透過低功耗藍牙探索並建立微型控制器的安全連線

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模擬器 (例如 TeraTerm)。依照安裝終端機模擬器中的指示，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：

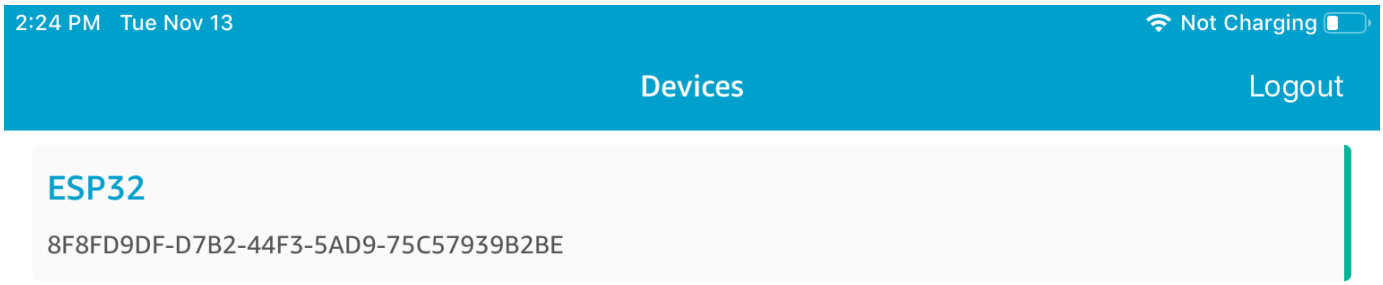
   ```
   $ ./gradlew installDebug
   ```

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。

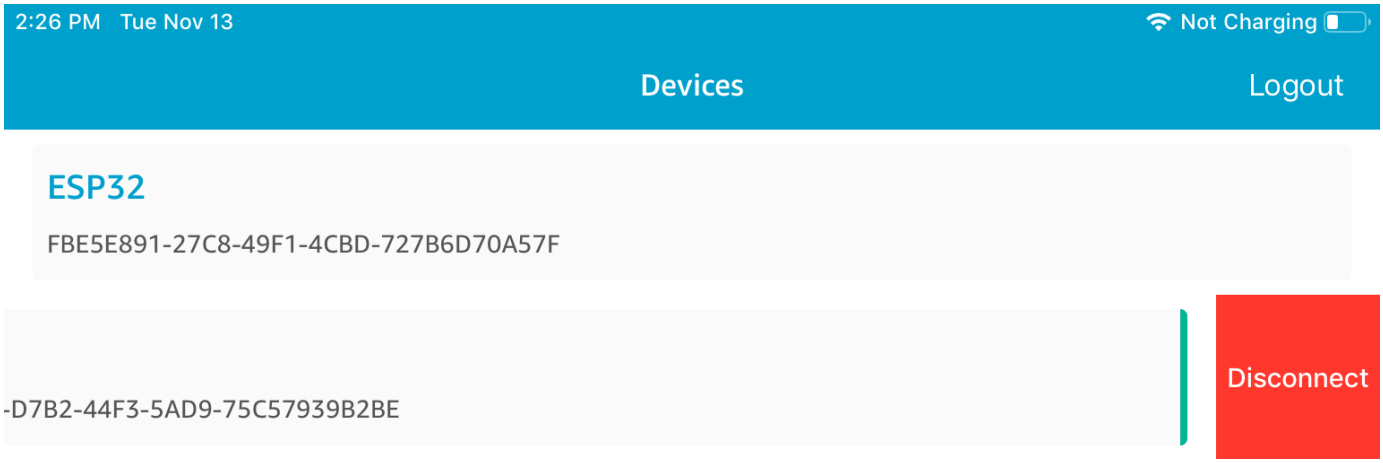


> **Note**
>
> 所有具有 FreeRTOS 的裝置，以及位於範圍內的裝置資訊服務 (*freertos*/.../device_information) 都會顯示在清單中。

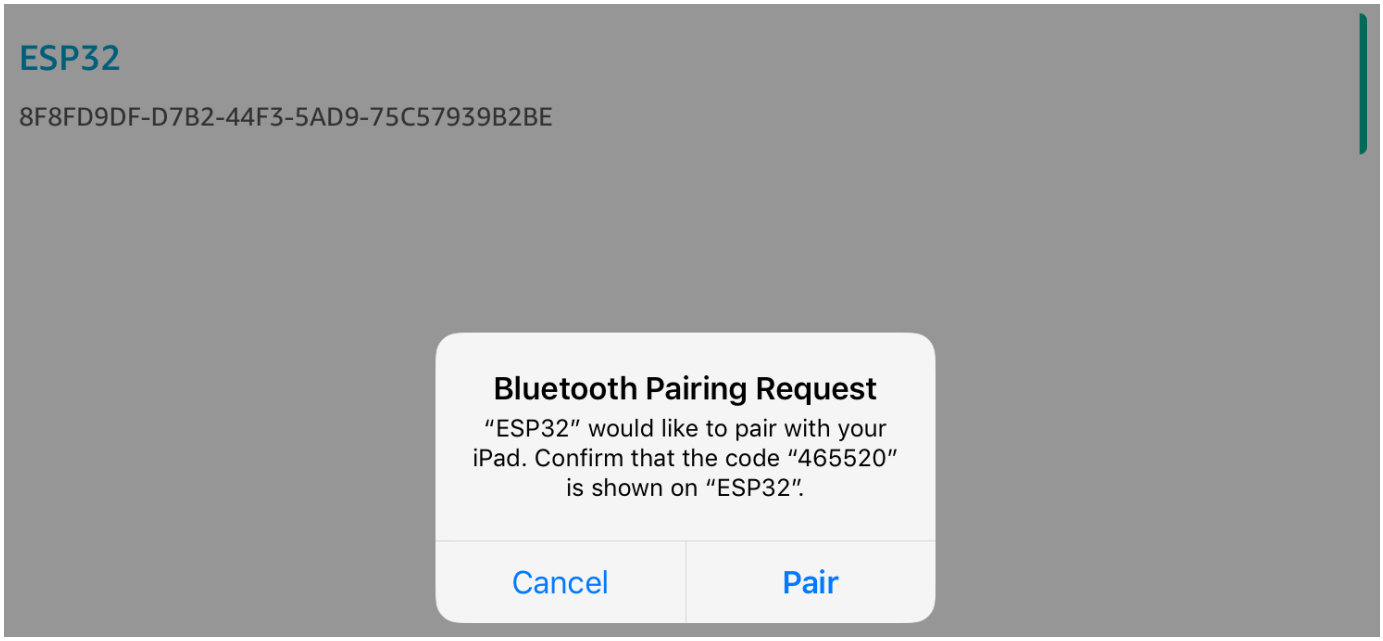
5. 從裝置清單選擇您的微型控制器。應用程式會與主機板建立連線，而所連線的裝置旁會出現綠色線條。



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**
>
> 低功耗藍牙行動開發套件示範應用程式使用 Amazon Cognito 進行使用者身分驗證。請確定您已設定 Amazon Cognito 使用者和身分集區，而且您已將 IAM 政策連接至已驗證的身分。

透過低功耗藍牙執行的 MQTT

在 MQTT over Bluetooth Low Energy 示範中，您的微控制器會透過 MQTT 代理將訊息發佈至 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。

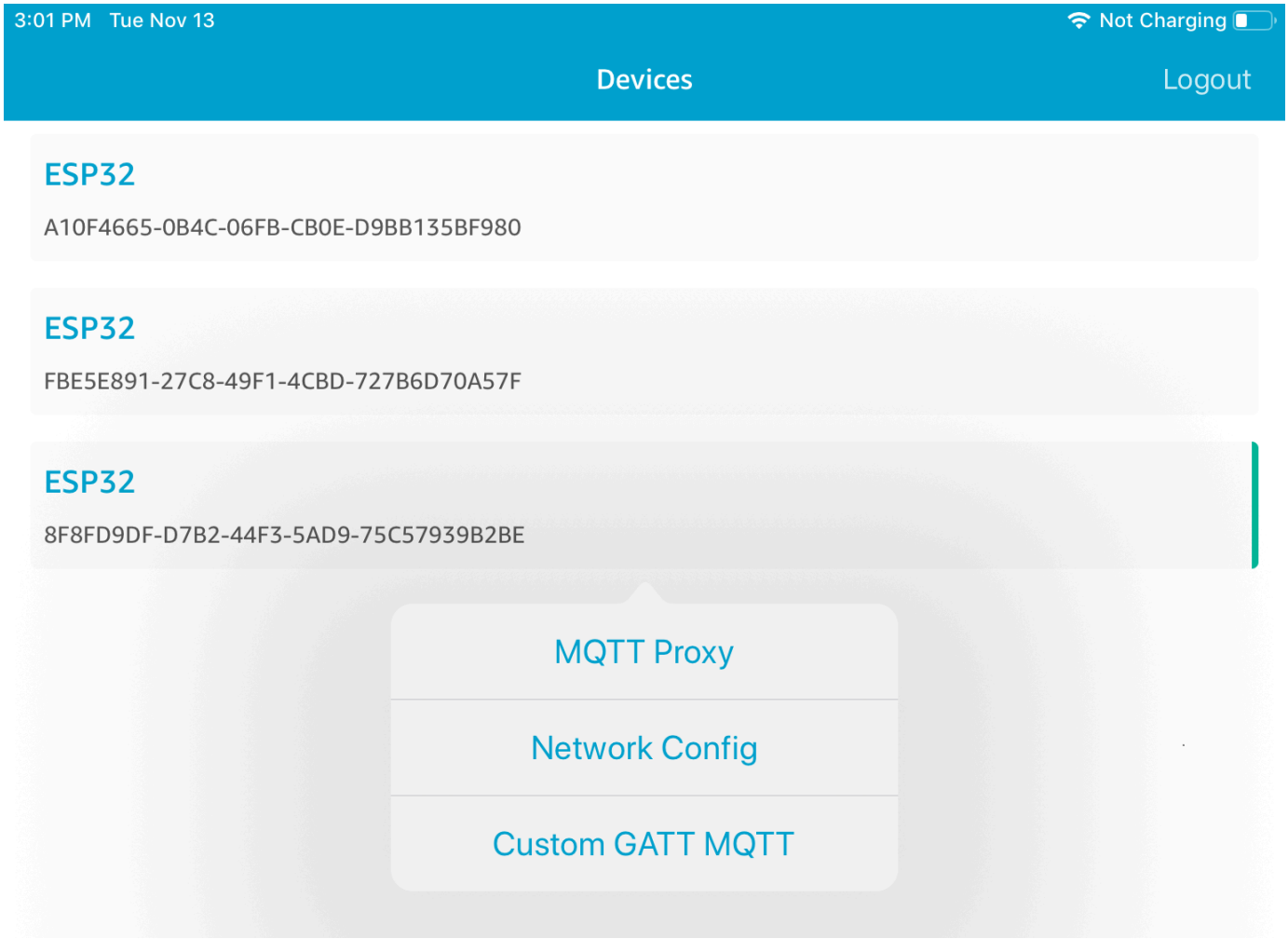
如果您是使用低功耗藍牙來配對微型控制器與行動裝置，則系統會在行動裝置上透過低功耗藍牙 Mobile SDK 示範應用程式路由 MQTT 訊息。

透過低功耗藍牙啟用示範

- 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 低功耗藍牙行動 SDK 示範應用程式 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，可讓您透過低功耗藍牙，安全地將 Wi-Fi 網路憑證從行動裝置傳送至微型控制器。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

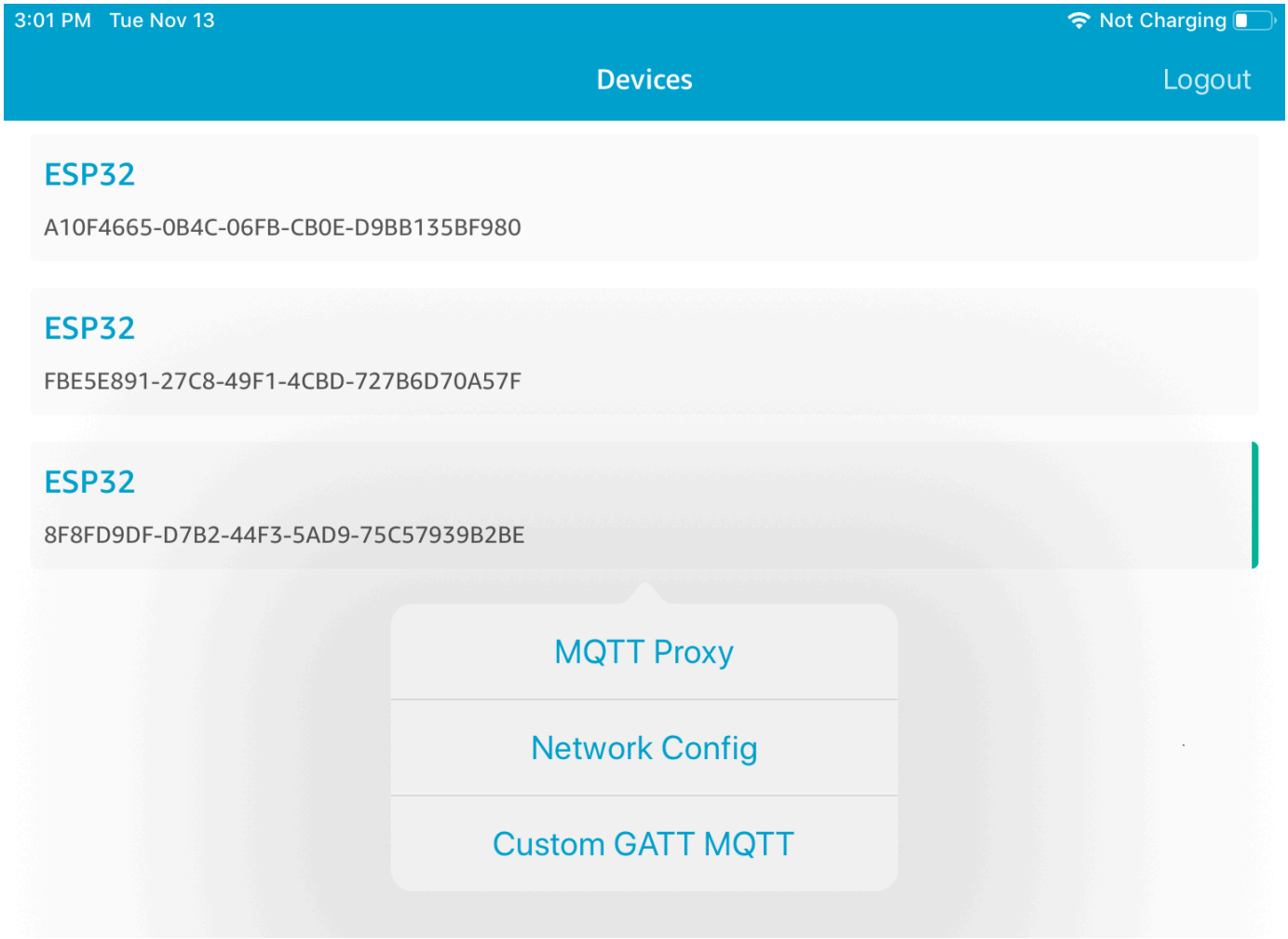
1. 啟用 Wi-Fi 佈建服務。開啟 `vendors/vendor/boards/board/aws_demos/config_files/iot_ble_config.h`，並將 `#define IOT_BLE_ENABLE_WIFI_PROVISIONING` 設定為 `1` (其中 *vendor* 是廠商名稱，而 *board* 是您用來執行示範的主機板名稱)。

   > **Note**
   >
   > 在預設情況下，Wi-Fi 佈建服務會處於停用狀態。

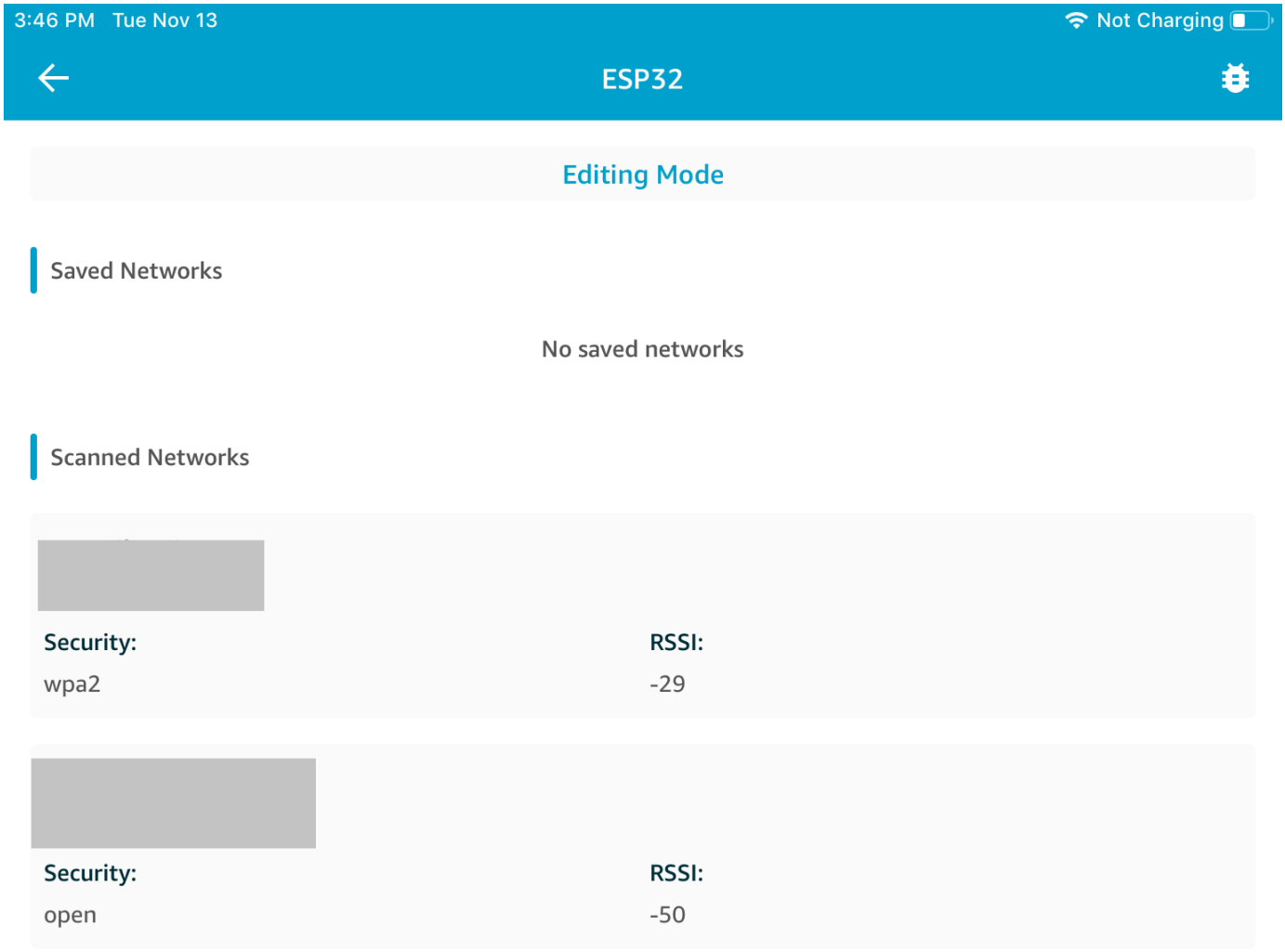
2. 設定 Network Manager，即可同時啟用低功耗藍牙和 Wi-Fi。

執行示範

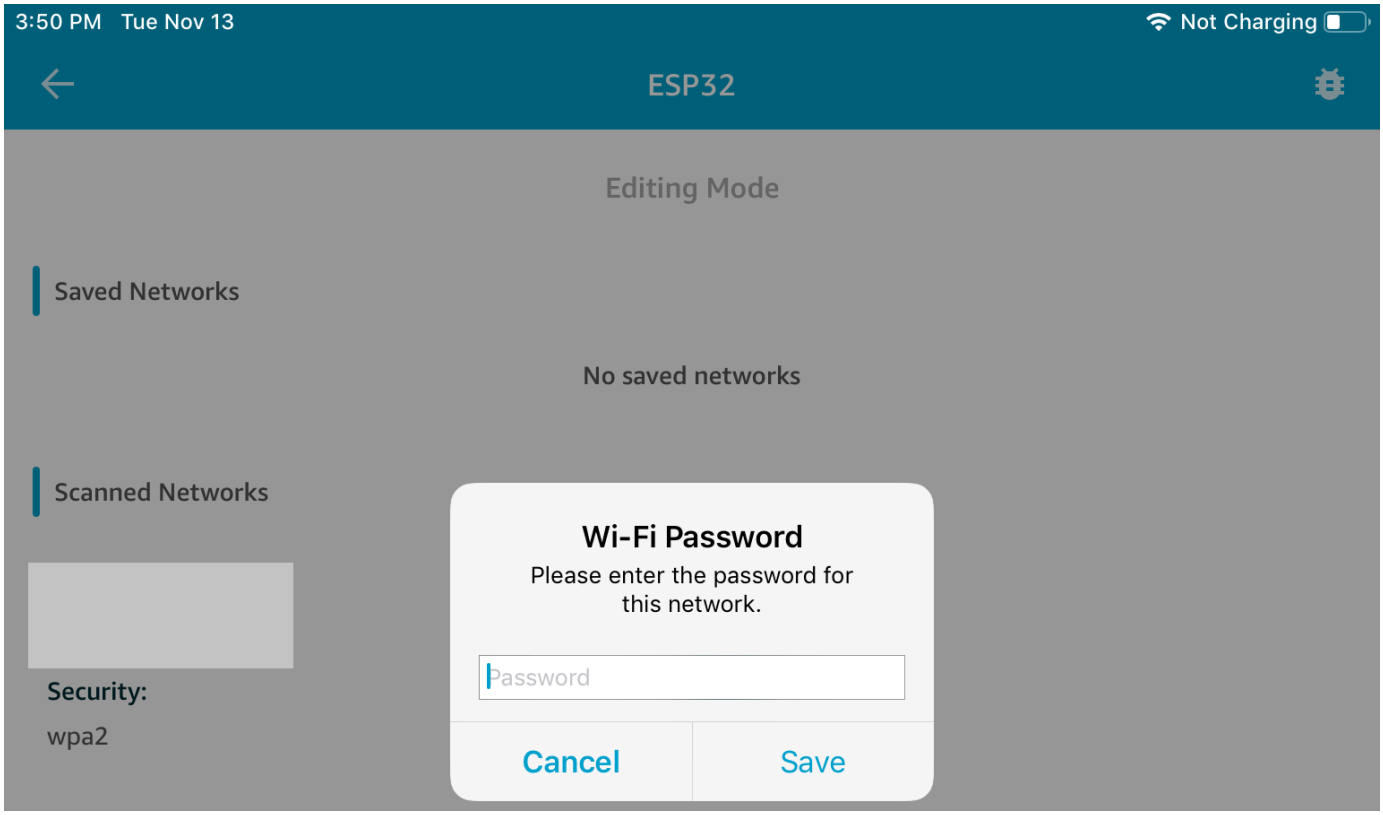
1. 在微型控制器上建置和執行示範專案。
2. 確保已使用 FreeRTOS 低功耗藍牙行動 SDK 示範應用程式 將您的微型控制器與行動裝置配對。
3. 從示範行動應用程式的 Devices (裝置) 清單中選擇您的微型控制器，然後選擇 Network Config (網路組態) 以開啟網路組態設定。



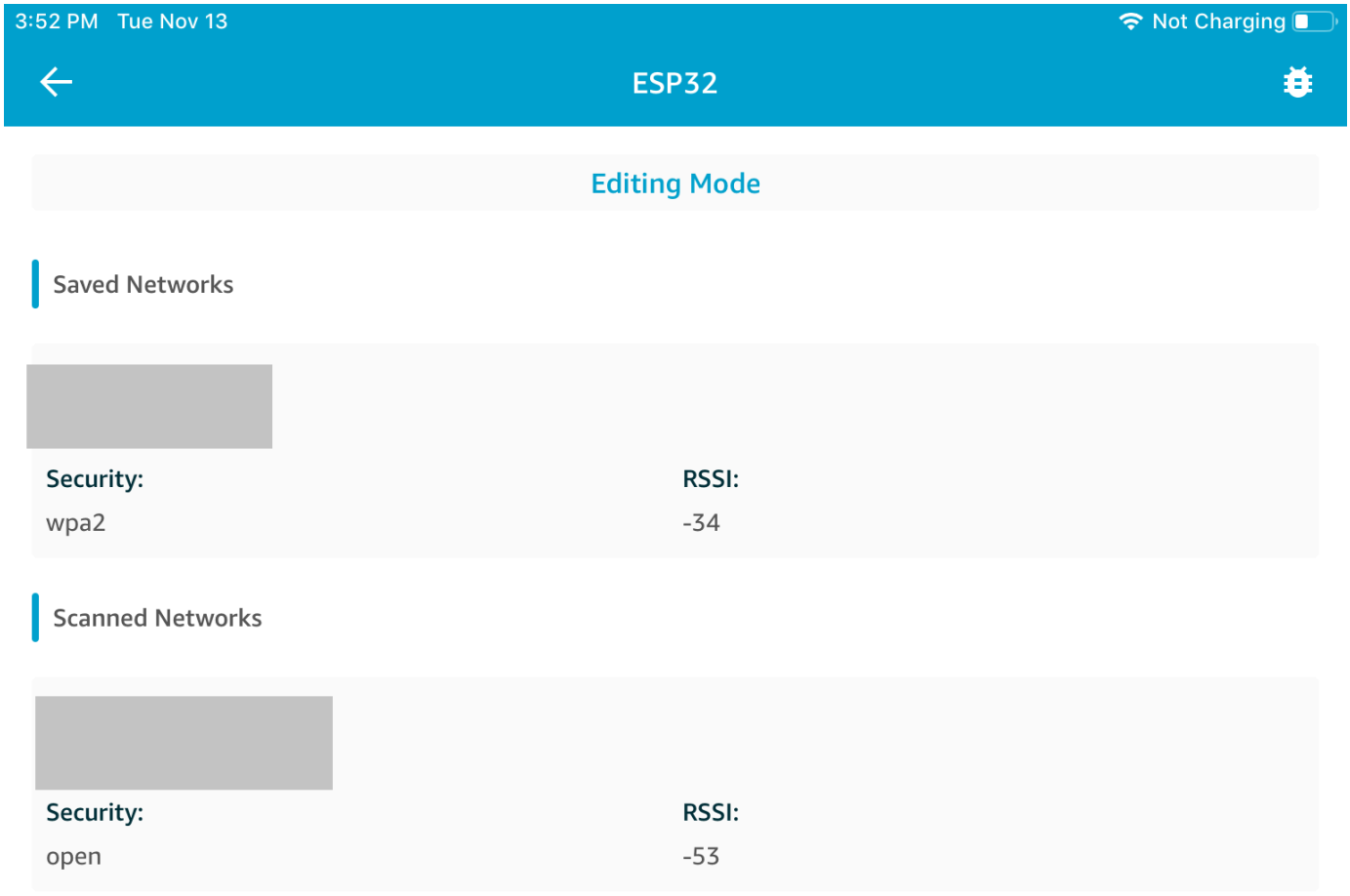
4. 選擇主機板的 Network Config (網路組態) 後，微型控制器會傳送鄰近的網路清單至行動裝置。可用的 Wi-Fi 網路會出現在 Scanned Networks (掃描的網路) 下的清單中。



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)。

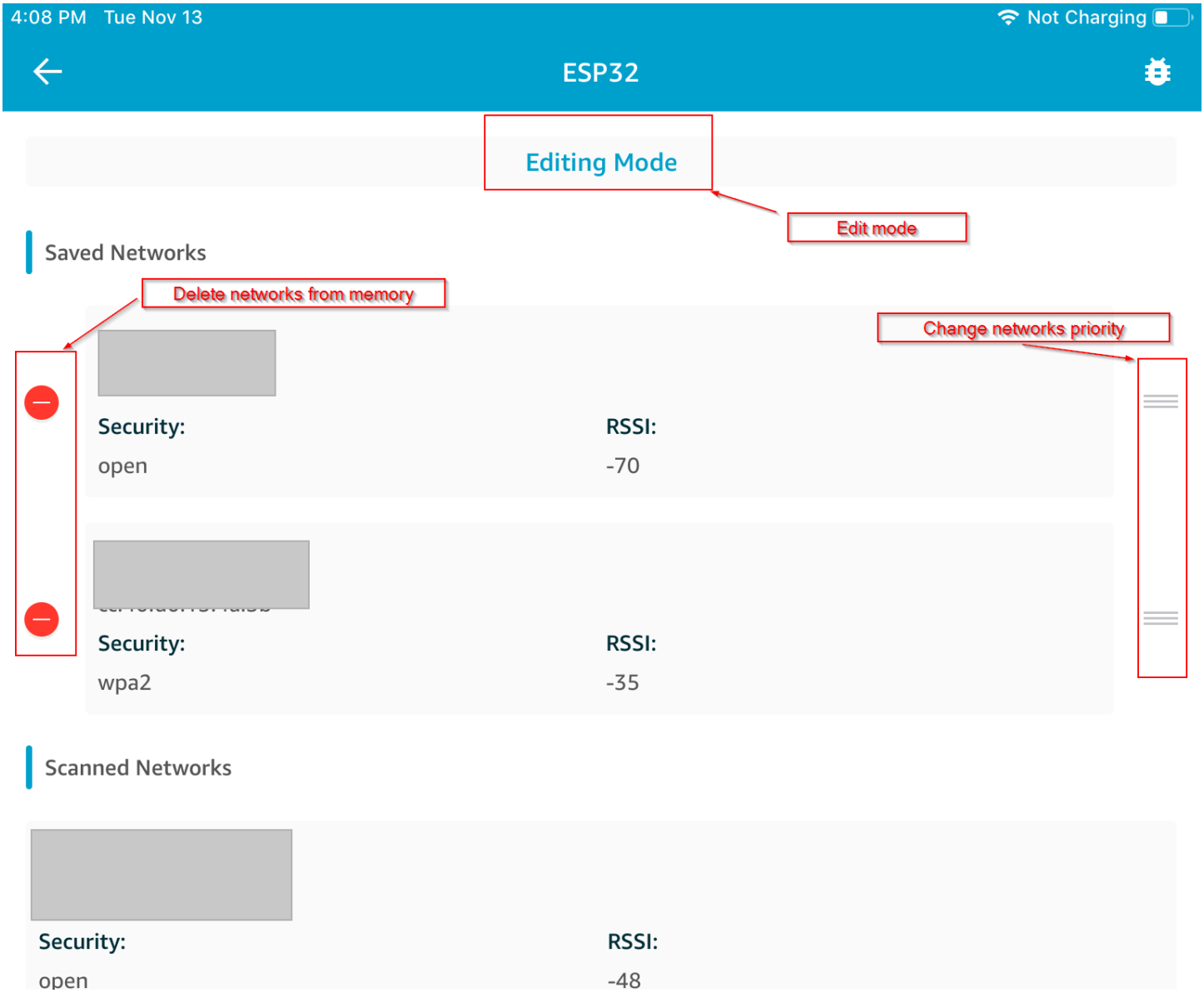


微型控制器會連接至並儲存網路。網路會出現在 Saved Networks (儲存的網路) 下。



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 低功耗藍牙行動 SDK 示範應用程式。

透過低功耗藍牙 Mobile SDK，您可以為連接至微型控制器上 GATT 伺服器的行動裝置建立自己的 GATT 用戶端，並同時執行示範行動應用程式。

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` (其中 *vendor* 是廠商名稱，而 *board* 是您用來執行示範

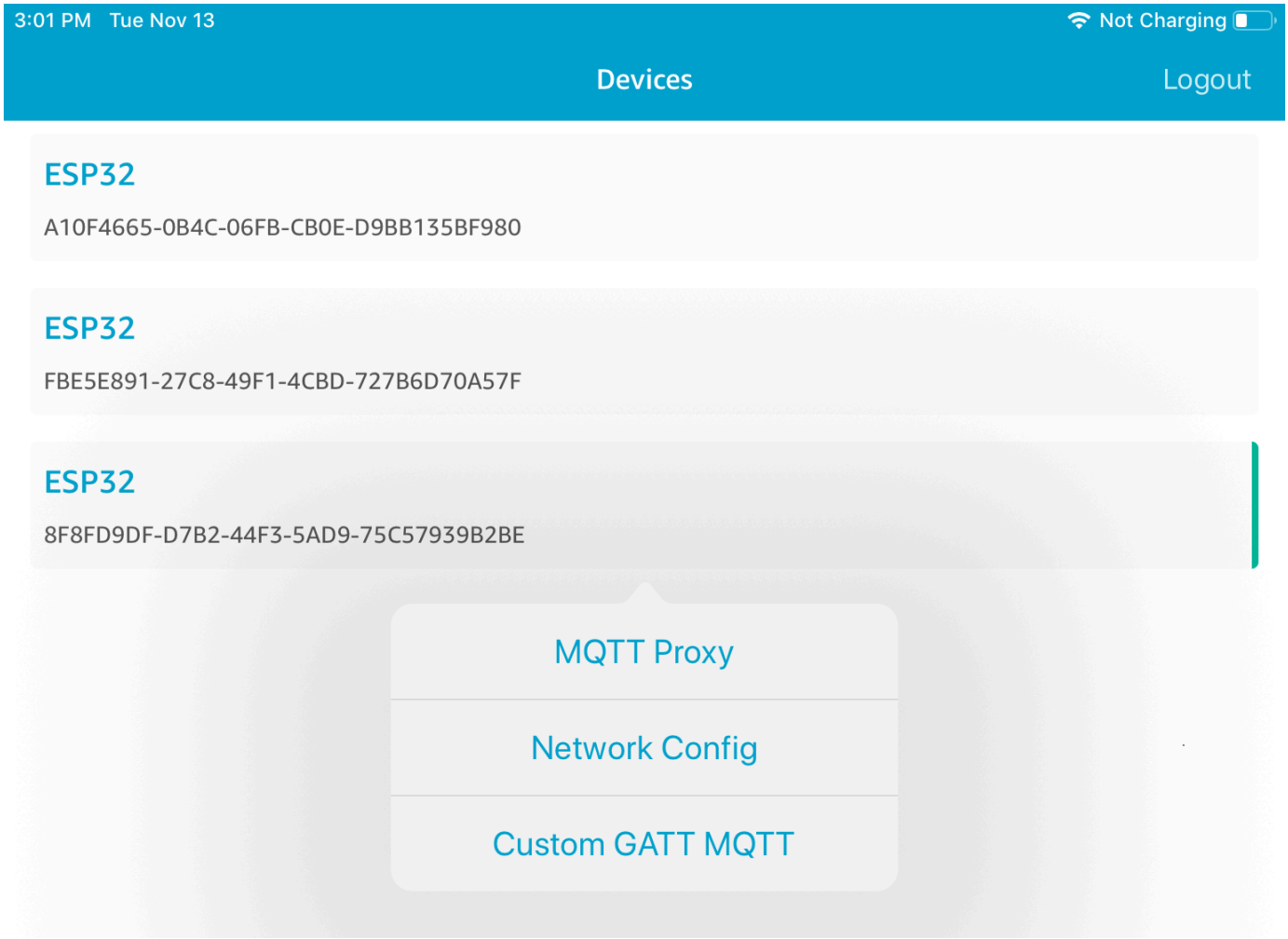
的主機板名稱) 中，將 `#define IOT_BLE_ADD_CUSTOM_SERVICES ( 1 )` 新增至 define 陳述式的清單。

> **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 低功耗藍牙行動 SDK 示範應用程式 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 ***your-thing-name*/example/topic** MQTT 主題。

   啟用 MQTT Proxy 之後，Hello World 和遞增的計數器訊息會顯示在 *your-thing-name*/example/topic 主題上。

Microchip Curiosity PIC32MZEF 的示範開機載入器

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

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 over-the-air(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。

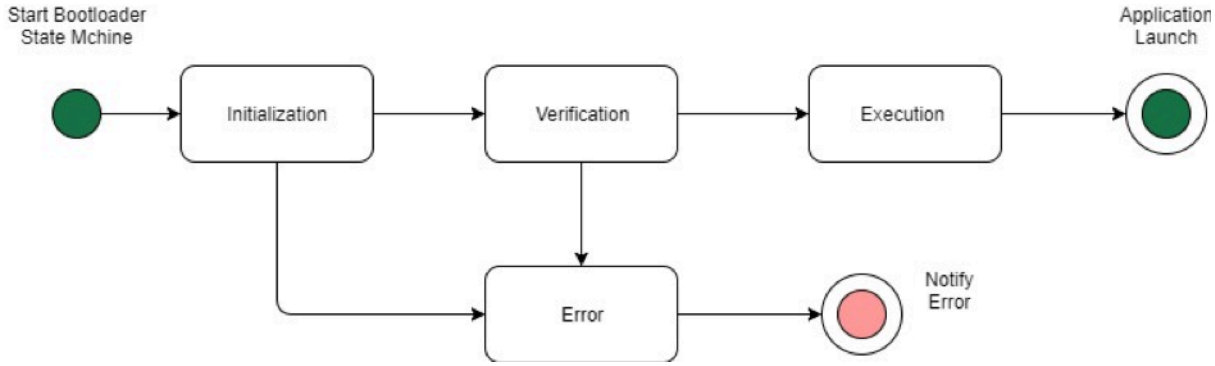
開機載入器支援下列 OTA 所需的功能：

- 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，請遵循中的步驟FreeRTOS 入門。

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。

開機載入器狀態	描述
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

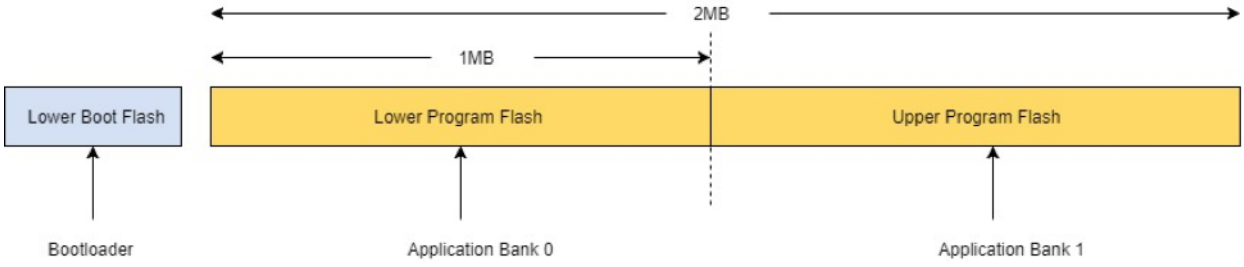
如果啟用了啟動預設映像的組態選項，則開機載入器將從預設執行地址啟動應用程式。除了偵錯外，此選項必須停用。

開機載入器錯誤狀態

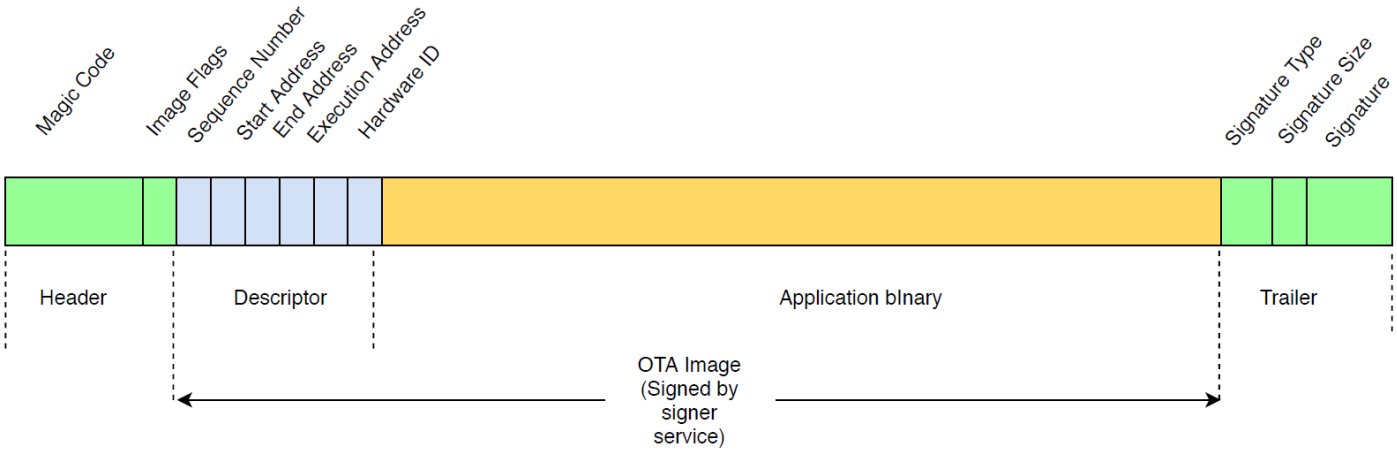
開機載入器處於錯誤狀態，且裝置上不存在有效映像。開機載入器必須通知使用者。預設實作會向主控台傳送日誌訊息，並無限期快速閃爍電路板上的 LED。

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。開機載入器在啟動映像之前，會檢查是否存在有效的神奇代碼。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	Value	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。有效值介於 1 到 4294967295 之間。

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

示範開機載入器包含在 FreeRTOS 原始程式碼儲存庫中 的 aws_demos 專案*freertos*/vendors/microchip/boards/curiosity_pic32mzef/aws_demos/mplab/中，做為可載入的專案。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

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

簡介

此示範說明如何使用 AWS IoT Device Shadow 程式庫連線至 AWS IoT Device Defender。示範使用 coreMQTT 程式庫與 MQTT 代理程式和 coreJSON 程式庫建立 TLS（相互身分驗證）的 AWS IoT

MQTT 連線，以驗證和剖析從中收到的回應 AWS IoT Device Defender。示範示範如何使用從裝置收集的指標建立 JSON 格式的報告，以及如何將報告提交給 AWS IoT Device Defender。示範也會示範如何向 coreMQTT 程式庫註冊回呼函數，以處理 AWS IoT Device Defender 傳送的回應，以確認報告是否被接受或拒絕。

> **Note**
>
> 若要設定和執行 FreeRTOS 示範，請遵循中的步驟FreeRTOS 入門。

功能

此示範會建立單一應用程式任務，示範如何收集指標、以 JSON 格式建構裝置防禦者報告，並透過安全 MQTT AWS IoT Device Defender 連線至 AWS IoT MQTT 代理程式將其提交至 。

我們收集指標的方式取決於使用的 TCP/IP 堆疊。對於 FreeRTOS+TCP 和支援的 lwIP 組態，我們提供指標收集實作，從裝置收集真實指標並在 AWS IoT Device Defender 報告中提交它們。您可以在 GitHub 上找到 FreeRTOS+TCP 和 lwIP 的隱含。

對於使用任何其他 TCP/IP 堆疊的電路板，會提供指標集合函數的 stub 定義，這些函數會為所有指標傳回零。若要使用此存根實作在主機板上傳送實際指標，請在 中*freertos*/demos/device_defender_for_aws/metrics_collector/stub/metrics_collector.c為您的網路堆疊實作 函數。檔案也可在 GitHub 網站上取得。

對於 ESP32，預設的 lwIP 組態不使用核心鎖定，因此示範將使用截斷的指標。如果您想要使用參考 lwIP 指標集合實作，請在 中定義下列巨集lwiopts.h：

```
#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/js
on/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/js
on/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,"versio
n": "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":
24 3346 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:892] PUBLISH sent for topic $aws/things/MyThingName/defender/met
rics/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=MQTTSucc
ess.
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/met
rics/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/met
rics/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-------
```

如果您的電路板未使用 FreeRTOS+TCP 或支援的 lwIP 組態，輸出會如下所示。

```
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/js
on/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/js
on/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! P
lease 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 GetEstablishedCon
nections! 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,"versio
n": "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"
28 2848 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:892] PUBLISH sent for topic $aws/things/MyThingName/defender/met
rics/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=MQTTSucc
ess.
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/met
rics/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/met
rics/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-------
```

示範的原始程式碼位於您在 *freertos*/demos/device_defender_for_aws/目錄或 GitHub 網站上的下載中。

訂閱 AWS IoT Device Defender 主題

subscribeToDefenderTopics 函數會訂閱 MQTT 主題，其中將接收對已發佈 Device Defender 報告的回應。它使用 巨集`DEFENDER_API_JSON_ACCEPTED`來建構主題字串，以接收可接受裝置防禦者報告的回應。它使用 巨集`DEFENDER_API_JSON_REJECTED`來建構主題字串，以接收被拒絕的裝置防禦者報告的回應。

收集裝置指標

collectDeviceMetrics 函數會使用 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和封包數量、開啟的 TCP 連接埠、開啟的 UDP 連接埠，以及已建立的 TCP 連線。

產生 AWS IoT Device Defender 報告

generateDeviceMetricsReport 函數會使用 中定義的函數產生裝置防禦者報告`report_builder.h`。該函數採用聯網指標和緩衝區，以 預期的格式建立 JSON 文件，並將其 AWS IoT Device Defender 寫入提供的緩衝區。AWS IoT 開發人員指南中的裝置端指標中 AWS IoT Device Defender 會指定 預期的 JSON 文件格式。

發佈 AWS IoT Device Defender 報告

AWS IoT Device Defender 報告會發佈在用於發佈 JSON AWS IoT Device Defender 報告的 MQTT 主題上。報告使用巨集 建構`DEFENDER_API_JSON_PUBLISH`，如 GitHub 網站上的此程式碼片段所示。

處理回應的回呼

publishCallback 函數會處理傳入的 MQTT 發佈訊息。它使用來自程式 AWS IoT Device Defender 庫的`Defender_MatchTopic` 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，報告 ID 會經過驗證，與報告中傳送的報告 ID 相同。

AWS IoT Greengrass 探索示範應用程式

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

執行 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，請遵循 中的指示設定 AWS 您的帳戶和許可。若要設定 AWS IoT Greengrass，您需要建立 Greengrass 群組，然後新增 Greengrass 核心。如需設定的詳細資訊 AWS IoT Greengrass，請參閱 入門 AWS IoT Greengrass。

設定 AWS 和 之後 AWS IoT Greengrass，您需要為 設定一些額外的許可 AWS IoT Greengrass。

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 (AmazonS3FullAccess) 和 AWSIoTFullAccess (AWSIoTFullAccess)，然後選擇 Attach 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)。

將電路板連接至 AWS IoT 並設定 FreeRTOS 示範。

1. 向 註冊您的 MCU 電路板 AWS IoT

   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。

   建立新的 AWS IoT Greengrass 政策

   1. 瀏覽至 AWS IoT 主控台。
   2. 在導覽窗格中，選擇安全，選擇政策，然後選擇建立。
   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": "*"
}
```

此政策會授予所有 資源的 AWS IoT Greengrass 許可。

5. 選擇 Create (建立)。

將 AWS IoT Greengrass 政策連接至裝置的憑證

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

> **Note**
>
> 如果您要從 FreeRTOS 主控台下載 FreeRTOS，請選擇連線至 AWS IoT Greengrass**##**，而非連線至 AWS IoT**##**。

3. 設定 FreeRTOS 示範.

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`。

設定 AWS IoT 和 之後 AWS IoT Greengrass，以及下載和設定 FreeRTOS 之後，您可以在裝置上建置、刷新和執行 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中的指示操作。

Greengrass 示範會將一系列訊息發佈到 Greengrass 核心和 AWS IoT MQTT 用戶端。若要檢視 AWS IoT MQTT 用戶端中的訊息，請開啟AWS IoT 主控台，選擇測試，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

1. 尋找與您的 Amazon EC2 執行個體相關聯的公有 DNS (IPv4)：前往 Amazon EC2 主控台，然後在左側導覽面板中，選擇執行個體。選擇您的 Amazon EC2 執行個體，然後選擇描述面板。尋找 Public DNS (IPv4) (公有 DNS (IPv4)) 的項目，並記下該項目。
2. 尋找安全群組的項目，然後選擇連接至 Amazon EC2 執行個體的安全群組。
3. 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	-

4. 在 AWS IoT 主控台中選擇 Greengrass，然後選擇群組，然後選擇您先前建立的 Greengrass 群組。選擇設定。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)。
5. 在導覽窗格中，選擇 Cores (核心)，然後選取群組核心。
6. 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)。
7. 將您建立的 FreeRTOS IoT 實物新增到 GG 群組。

   a. 選擇返回箭頭以返回 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)。

   b. 選擇 Select an IoT Thing (選取 IoT 實物)。選擇您的裝置，然後選擇 Finish (完成)。
8. 在 Greengrass 群組頁面中新增必要的訂閱，選擇訂閱，然後選擇新增訂閱，然後輸入資訊，如下所示。

   訂閱

來源	目標	主題
TIGG1	IoT Cloud (IoT 雲端)	freertos/demos/ggd

   其中「來源」是您在此處範例註冊電路板時，在 AWS IoT 主控台中建立之 AWS IoT 物件的名稱 - "TIGG1"。
9. 啟動 AWS IoT Greengrass 群組的部署，並確保部署成功。您現在應該能夠成功執行 AWS IoT Greengrass 探索示範。

coreHTTP 示範

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

這些示範可協助您了解如何使用 coreHTTP 程式庫。

主題

- coreHTTP 相互身分驗證示範
- coreHTTP 基本 Amazon S3 上傳示範
- coreHTTP 基本 S3 下載示範
- coreHTTP 基本多執行緒示範

coreHTTP 相互身分驗證示範

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

簡介

coreHTTP （相互身分驗證） 示範專案說明如何使用 TLS 建立與 HTTP 伺服器的連線，並在用戶端與伺服器之間進行相互身分驗證。此示範使用 mbedTLS 型傳輸介面實作來建立伺服器和用戶端驗證的 TLS 連線，並在 HTTP 中示範請求回應工作流程。

> **Note**
>
> 若要設定和執行 FreeRTOS 示範，請遵循中的步驟FreeRTOS 入門。

功能

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：

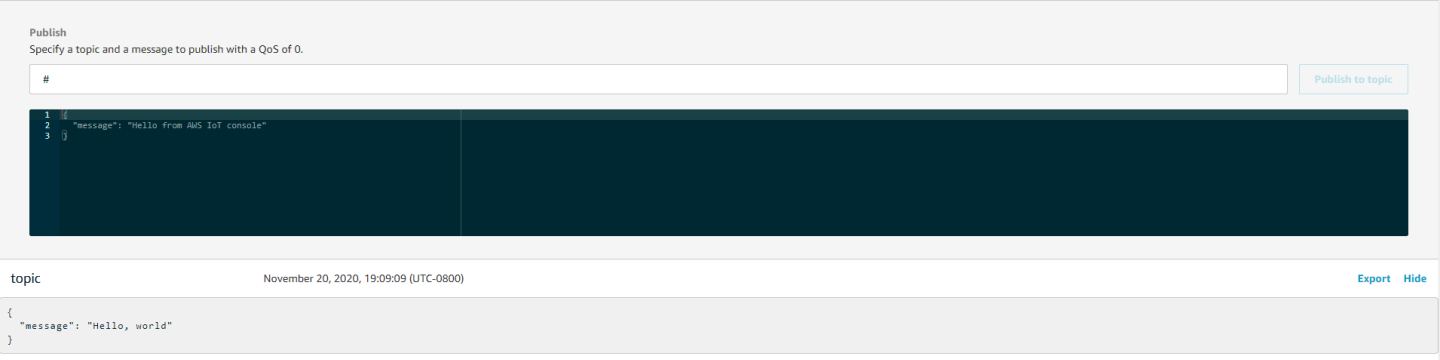
- 連線至 AWS IoT 端點上的 HTTP 伺服器。
- 傳送 POST 請求。
- 接收回應。
- 中斷與伺服器的連線。

完成這些步驟後，示範會產生類似下列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[0x68F5]: The answer to 'a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com' (35.166.102.88) will be stored
16 1622 [iot_thread] DNS[0x68F5]: The answer to 'a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com' (35.166.2.12) will be stored
17 1622 [iot_thread] DNS[0x68F5]: The answer to 'a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com' (44.239.154.164) will be stored
18 1622 [iot_thread] DNS[0x68F5]: The answer to 'a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com' (44.239.97.33) will be stored
19 1622 [iot_thread] DNS[0x68F5]: The answer to 'a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com' (44.238.43.70) will be stored
20 1622 [iot_thread] DNS[0x68F5]: The answer to 'a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com' (52.32.144.134) will be stored
21 2042 [iot_thread] [INFO] [HTTPDemo] [http_demo_mutual_auth.c:393] 22 2042 [iot_thread] Sending HTTP POST request to a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com/topics/topic?qos=1...23 2042 [iot_thread]
24 2082 [iot_thread] [INFO] [HTTPDemo] [http_demo_mutual_auth.c:418] 25 2082 [iot_thread] Received HTTP response from a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com/topics/topic?qos=1...
26 2082 [iot_thread]
27 2082 [iot_thread] [INFO] [HTTPDemo] [http_demo_mutual_auth.c:283] 28 2082 [iot_thread] Demo completed successfully.29 2082 [iot_thread]
30 2082 [iot_thread] [INFO ][DEMO][2082] memory_metrics::freertos_heap::before::bytes::2088152
31 2082 [iot_thread] [INFO ][DEMO][2082] memory_metrics::freertos_heap::after::bytes::1990104
32 2082 [iot_thread] [INFO ][DEMO][2082] memory_metrics::demo_task_stack::before::bytes::1908
33 2082 [iot_thread] [INFO ][DEMO][2082] memory_metrics::demo_task_stack::after::bytes::1908
34 3082 [iot_thread] [INFO ][DEMO][3082] Demo completed successfully.
35 3084 [iot_thread] [INFO ][INIT][3084] SDK cleanup done.
36 3084 [iot_thread] [INFO ][DEMO][3084] -------DEMO FINISHED-------
```

AWS IoT 主控台會產生類似下列螢幕擷取畫面的輸出。



原始碼組織

示範來源檔案已命名http_demo_mutual_auth.c，可在 *freertos*/demos/coreHTTP/目錄和 GitHub 網站上找到。

連線至 AWS IoT HTTP 伺服器

connectToServerWithBackoffRetries 函數會嘗試與 AWS IoT HTTP 伺服器建立相互驗證的 TLS 連線。如果連線失敗，則會在逾時後重試。逾時值會呈指數增加，直到達到最大嘗試次數或達到最大逾時值為止。RetryUtils_BackoffAndSleep 函數提供指數增加的逾時值，並在達到最大嘗試次數RetryUtilsRetriesExhausted時傳回。如果在設定的嘗試次數後無法建立與代理程式的 TLS 連線，connectToServerWithBackoffRetries則函數會傳回失敗狀態。

傳送 HTTP 請求並接收回應

prvSendHttpRequest 函數示範如何將 POST 請求傳送至 AWS IoT HTTP 伺服器。如需向 中的 REST API 提出請求的詳細資訊 AWS IoT，請參閱裝置通訊協定 - HTTPS。系統會使用相同的 coreHTTP API 呼叫 接收回應HTTPClient_Send。

coreHTTP 基本 Amazon S3 上傳示範

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

簡介

此範例示範如何將 PUT 請求傳送至 Amazon Simple Storage Service (Amazon S3) HTTP 伺服器並上傳小型檔案。它也會執行 GET 請求，以在上傳後驗證檔案大小。此範例使用使用 mbedTLS 的網路傳輸介面，在執行 coreHTTP 的 IoT 裝置用戶端與 Amazon S3 HTTP 伺服器之間建立相互驗證的連線。

> **Note**
>
> 若要設定和執行 FreeRTOS 示範，請遵循中的步驟FreeRTOS 入門。

單一螺紋與多螺紋

有兩種 coreHTTP 使用模式：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，但實際上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改為在代理程式（或協助程式）任務的背景中傳送 HTTP 請求。

原始碼組織

示範來源檔案已命名`http_demo_s3_upload.c`，可在 *`freertos`*`/demos/coreHTTP/`目錄和 GitHub 網站上找到。

設定 Amazon S3 HTTP 伺服器連線

此示範使用預先簽章的 URL 來連線至 Amazon S3 HTTP 伺服器，並授權物件的存取以進行下載。Amazon S3 HTTP 伺服器的 TLS 連線只會使用伺服器身分驗證。在應用程式層級，會使用預先簽章 URL 查詢中的參數來驗證物件的存取。請依照下列步驟來設定您的連線 AWS。

1. 設定 AWS 帳戶：
    a. 如果您尚未建立帳戶，請建立 AWS 帳戶。

b. 使用 AWS Identity and Access Management (IAM) 設定帳戶和許可。您可以使用 IAM 來管理帳戶中每個使用者的許可。根據預設，使用者在根擁有者授予 之前沒有許可。

   i. 若要將 IAM 使用者新增至 AWS 您的帳戶，請參閱 IAM 使用者指南。

   ii. 將許可授予 AWS 您的帳戶以存取 FreeRTOS，並 AWS IoT 新增此政策：

   - AmazonS3FullAccess

2. 遵循 Amazon S3 Simple Storage Service 使用者指南中的如何建立 S3 儲存貯體？中的步驟，在 Amazon S3 中建立儲存貯體。
3. 遵循如何將檔案和資料夾上傳至 S3 儲存貯體中的步驟，將檔案上傳至 Amazon S3？。 S3
4. 使用位於 `FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/presigned_urls_gen.py` 檔案的指令碼產生預先簽章的 URL。

   如需使用說明，請參閱 `FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generato/README.md` 檔案。

功能

示範會先使用 TLS 伺服器身分驗證連線至 Amazon S3 HTTP 伺服器。然後，它會建立 HTTP 請求來上傳 中指定的資料democonfigDEMO_HTTP_UPLOAD_DATA。上傳檔案後，它會請求檔案大小，以檢查檔案是否已成功上傳。您可以在 GitHub 網站上找到示範的原始程式碼。

連線至 Amazon S3 HTTP 伺服器

connectToServerWithBackoffRetries 函數會嘗試與 HTTP 伺服器建立 TCP 連線。如果連線失敗，則會在逾時後重試。逾時值會呈指數增加，直到達到最大嘗試次數或達到最大逾時值為止。如果在設定的嘗試次數之後無法建立與伺服器的 TCP 連線，`connectToServerWithBackoffRetries`則函數會傳回失敗狀態。

`prvConnectToServer` 函數示範如何僅使用伺服器身分驗證來建立 Amazon S3 HTTP 伺服器的連線。它使用在 `FreeRTOS-Plus/Source/Application-Protocols/network_transport/freertos_plus_tcp/using_mbedtls/using_mbedtls.c` 檔案中實作的 mbedTLS 型傳輸界面。您可以在 GitHub 網站上`prvConnectToServer`找到 的定義。

上傳資料

`prvUploadS3ObjectFile` 函數示範如何建立 PUT 請求，並指定要上傳的檔案。上傳檔案的 Amazon S3 儲存貯體，以及要上傳的檔案名稱，會在預先簽章的 URL 中指定。為了節省記憶體，請

求標頭和 都會使用相同的緩衝區來接收回應。使用 HTTPClient_Send API 函數同步接收回應。回應200 OK狀態碼預期來自 Amazon S3 HTTP 伺服器。任何其他狀態碼都是錯誤。

您可以在 GitHub 網站上prvUploadS3ObjectFile()找到 的原始碼。

驗證上傳

prvVerifyS3ObjectFileSize 函數會呼叫 prvGetS3ObjectFileSize來擷取 S3 儲存貯體中物件的大小。Amazon S3 HTTP 伺服器目前不支援使用預先簽章 URL 的 HEAD 請求，因此請求第 0 個位元組。檔案的大小包含在回應的Content-Range標頭欄位中。預期來自伺服器的206 Partial Content回應。任何其他回應狀態碼都是錯誤。

您可以在 GitHub 網站上prvGetS3ObjectFileSize()找到 的原始碼。

coreHTTP 基本 S3 下載示範

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

簡介

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 HTTPClient_AddRangeHeader建立 HTTP 請求時，coreHTTP API 原生支援範圍請求。對於微型控制器環境，強烈建議提出範圍請求。透過下載不同範圍的大型檔案，而不是單一請求，可以在不封鎖網路通訊端的情況下處理檔案的每個區段。範圍請求可降低捨棄封包的風險，這需要在 TCP 連線上重新傳輸，因此可改善裝置的耗電量。

此範例使用使用 mbedTLS 的網路傳輸界面，在執行 coreHTTP 的 IoT 裝置用戶端與 Amazon S3 HTTP 伺服器之間建立相互驗證的連線。

> **Note**
>
> 若要設定和執行 FreeRTOS 示範，請遵循中的步驟FreeRTOS 入門。

單一螺紋與多螺紋

有兩種 coreHTTP 使用模式：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，但實際上示範如何在單一執行緒環境中使用 coreHTTP （ 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改為在代理程式 （ 或協助程式 ） 任務的背景中傳送 HTTP 請求。

原始碼組織

示範專案已命名`http_demo_s3_download.c`，可在 *`freertos`*`/demos/coreHTTP/`目錄和 GitHub 網站上找到。

設定 Amazon S3 HTTP 伺服器連線

此示範使用預先簽章的 URL 來連線至 Amazon S3 HTTP 伺服器，並授權對物件的存取進行下載。Amazon S3 HTTP 伺服器的 TLS 連線只會使用伺服器身分驗證。在應用程式層級，會使用預先簽章 URL 查詢中的參數來驗證對物件的存取。請依照下列步驟來設定您的連線 AWS。

1. 設定 AWS 帳戶：
   a. 如果您尚未建立和啟用帳戶，請建立和啟用 AWS 帳戶。
   b. 使用 AWS Identity and Access Management (IAM) 設定帳戶和許可。IAM 可讓您管理帳戶中每個使用者的許可。在根擁有者授予 之前，使用者預設沒有許可。
      i. 若要將 IAM 使用者新增至 AWS 您的帳戶，請參閱 IAM 使用者指南。
      ii. 將許可授予 AWS 您的帳戶以存取 FreeRTOS AWS IoT ，並新增這些政策：
         - AmazonS3FullAccess
2. 遵循 Amazon Simple Storage Service 主控台使用者指南中的如何建立 S3 儲存貯體？中的步驟，在 S3 中建立儲存貯體。 S3
3. 遵循如何將檔案和資料夾上傳至 S3 儲存貯體中的步驟，將檔案上傳至 S3？。 S3
4. 使用位於 的指令碼產生預先簽章的 URL`FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/presigned_urls_gen.py`。如需使用說明，請參閱 `FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generato/README.md`。

功能

示範會先擷取檔案大小。然後，它會在迴圈中依序請求每個位元組範圍，範圍大小為 democonfigRANGE_REQUEST_LENGTH。

您可以在 GitHub 網站上找到示範的原始程式碼。

連線至 Amazon S3 HTTP 伺服器

函數 connectToServerWithBackoffRetries() 會嘗試與 HTTP 伺服器建立 TCP 連線。如果連線失敗，則會在逾時後重試。逾時值會呈指數增加，直到達到最大嘗試次數或達到最大逾時值為止。如果在設定的嘗試次數後無法建立與伺服器的 TCP 連線，則 會connectToServerWithBackoffRetries()傳回失敗狀態。

函數prvConnectToServer()示範如何使用伺服器身分驗證建立 Amazon S3 HTTP 伺服器的連線。它使用在檔案 FreeRTOS-Plus/Source/Application-Protocols/network_transport/freertos_plus_tcp/using_mbedtls/using_mbedtls.c 中實作的 mbedTLS 型傳輸界面。

您可以在 GitHub 上prvConnectToServer()找到 的原始碼。

建立範圍請求

API 函數HTTPClient_AddRangeHeader()支援將位元組範圍序列化為 HTTP 請求標頭，以形成範圍請求。此示範會使用範圍請求來擷取檔案大小，以及請求檔案的每個區段。

函數會prvGetS3ObjectFileSize()擷取 S3 儲存貯體中檔案的大小。Connection: keep-alive 標頭會新增至 Amazon S3 的第一次請求中，以在傳送回應後保持連線開啟。S3 HTTP 伺服器目前不支援使用預先簽署 URL 的 HEAD 請求，因此請求第 0 個位元組。檔案的大小包含在回應的Content-Range標頭欄位中。206 Partial Content 預期回應來自伺服器；收到的任何其他回應狀態碼都是錯誤。

您可以在 GitHub 上prvGetS3ObjectFileSize()找到 的原始碼。

擷取檔案大小後，此示範會為要下載檔案的每個位元組範圍建立新的範圍請求。它HTTPClient_AddRangeHeader()用於 檔案的每個區段。

傳送範圍請求和接收回應

函數會在迴圈中prvDownloadS3ObjectFile()傳送範圍請求，直到下載整個檔案為止。API 函數HTTPClient_Send()會傳送請求並同步接收回應。當函數傳回時，回應會在 中收到xResponse。

然後，狀態碼會驗證為 206 Partial Content，而且到目前為止下載的位元組數會依Content-Length標頭值增加。

您可以在 GitHub 上prvDownloadS3ObjectFile()找到 的原始碼。

coreHTTP 基本多執行緒示範

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

簡介

此示範使用 FreeRTOS 的執行緒安全佇列來保留等待處理的請求和回應。在此示範中，有三個任務需要注意。

- 主要任務會等待請求出現在請求佇列中。它會透過網路傳送這些請求，然後將回應放入回應佇列。
- 請求任務會建立 HTTP 程式庫請求物件以傳送至伺服器，並將它們放入請求佇列。每個請求物件都會指定應用程式已設定用於下載的 S3 檔案的位元組範圍。
- 回應任務會等待回應出現在回應佇列中。它會記錄收到的每個回應。

此基本多執行緒示範設定為僅使用 TLS 連線進行伺服器身分驗證，這是 Amazon S3 HTTP 伺服器的必要項目。應用程式層身分驗證是使用預先簽章 URL 查詢中的 Signature 第 4 版參數來完成。

原始碼組織

示範專案已命名http_demo_s3_download_multithreaded.c，可在 *freertos*/demos/coreHTTP/目錄和 GitHub 網站中找到。

建置示範專案

示範專案使用 Visual Studio 的免費社群版本。若要建置示範：

1. 從 mqtt_multitask_demo.sln Visual Studio IDE 內開啟 Visual Studio 解決方案檔案。
2. 從 IDE 的建置功能表中選取建置解決方案。

> **Note**
>
> 如果您使用的是 Microsoft Visual Studio 2017 或更早版本，則必須選取與您的版本相容的平台工具集：Project -> RTOSDemos Properties -> Platform Toolset。

設定示範專案

示範使用 FreeRTOS+TCP TCP/IP 堆疊，因此請依照 TCP/IP 入門專案提供的指示進行：

1. 安裝必要元件 （例如 WinPCap)。
2. 選擇性地設定靜態或動態 IP 地址、閘道地址和網路遮罩。
3. 選擇性地設定 MAC 地址。
4. 選取主機機器上的乙太網路介面。
5. 在嘗試執行 HTTP 示範之前，請務必測試您的網路連線。

設定 Amazon S3 HTTP 伺服器連線

遵循 coreHTTP 基本下載示範設定 Amazon S3 HTTP 伺服器連線中的指示。

功能

示範總共會建立三個任務：

- 透過網路傳送請求和接收回應的 。
- 建立要傳送請求的 。
- 處理收到的回應。

在此示範中，主要任務為：

1. 建立請求和回應佇列。
2. 建立與伺服器的連線。
3. 建立請求和回應任務。
4. 等待請求佇列透過網路傳送請求。
5. 將透過網路收到的回應放入回應佇列。

請求任務：

1. 建立每個範圍請求。

回應任務：

1. 處理收到的每一個回應。

Typedefs

示範會定義下列結構以支援多執行緒。

請求項目

下列結構會定義要放入請求佇列的請求項目。請求任務建立 HTTP 請求後，請求項目會複製到佇列中。

```
/**
 * @brief Data type for the request queue.
 *
 * Contains the request header struct and its corresponding buffer, to be
 * populated and enqueued by the request task, and read by the main task. The
 * buffer is included to avoid pointer inaccuracy during queue copy operations.
 */
typedef struct RequestItem
{
    HTTPRequestHeaders_t xRequestHeaders;
    uint8_t ucHeaderBuffer[ democonfigUSER_BUFFER_LENGTH ];
} RequestItem_t;
```

回應項目

下列結構定義要放入回應佇列的回應項目。主要 HTTP 任務透過網路收到回應後，回應項目會複製到佇列。

```
/**
 * @brief Data type for the response queue.
 *
 * Contains the response data type and its corresponding buffer, to be enqueued
 * by the main task, and interpreted by the response task. The buffer is
```

```
 * included to avoid pointer inaccuracy during queue copy operations.
 */
typedef struct ResponseItem
{
    HTTPResponse_t xResponse;
    uint8_t ucResponseBuffer[ democonfigUSER_BUFFER_LENGTH ];
} ResponseItem_t;
```

主要 HTTP 傳送任務

主要應用程式任務：

1. 剖析主機地址的預先簽章 URL，以建立與 Amazon S3 HTTP 伺服器的連線。
2. 剖析 S3 儲存貯體中物件路徑的預先簽章 URL。
3. 使用具有伺服器身分驗證的 TLS 連線至 Amazon S3 HTTP 伺服器。
4. 建立請求和回應佇列。
5. 建立請求和回應任務。

函數會`prvHTTPDemoTask()`執行此設定，並提供示範狀態。您可以在 Github 上找到此函數的原始碼。

在函數 中`prvDownloadLoop()`，主要任務會封鎖請求佇列並等待請求。當它收到請求時，會使用 API 函數 傳送請求`HTTPClient_Send()`。如果 API 函數成功，則會將回應放入回應佇列。

您可以在 Github 上`prvDownloadLoop()`找到 的原始碼。 https://github.com/FreeRTOS/FreeRTOS/blob/202012.00/FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/HTTP_S3_Download_Multithreaded/DemoTasks/S3DownloadMultithreadedHTTPExample.c#L1071-L1174

HTTP 請求任務

請求任務是在 函數 中指定`prvRequestTask`。您可以在 Github 上找到此函數的原始碼。

請求任務會擷取 Amazon S3 儲存貯體中檔案大小。這是在 函數 中完成的`prvGetS3ObjectFileSize`。「連線：保持連線」標頭會新增至 Amazon S3 的此請求，以在傳送回應後保持連線開啟。Amazon S3 HTTP 伺服器目前不支援使用預先簽章 URL 的 HEAD 請求，因此請求第 0 個位元組。檔案的大小包含在回應的`Content-Range`標頭欄位中。預期從伺服器`206 Partial Content`收到回應；收到的任何其他回應狀態碼都是錯誤。

您可以在 Github 上`prvGetS3ObjectFileSize`找到 的原始碼。

擷取檔案大小後，請求任務會繼續請求檔案的每個範圍。每個範圍請求都會放入要傳送的主要任務的請求佇列中。檔案範圍是由示範使用者在巨集 中設定democonfigRANGE_REQUEST_LENGTH。HTTP 用戶端程式庫 API 使用 函數原生支援範圍請求HTTPClient_AddRangeHeader。函數prvRequestS3ObjectRange示範如何使用 HTTPClient_AddRangeHeader()。

您可以在 Github 上prvRequestS3ObjectRange找到 函數的原始碼。

HTTP 回應任務

回應任務會在回應佇列中等待透過網路收到的回應。主要任務會在成功收到 HTTP 回應時填入回應佇列。此任務會透過記錄狀態碼、標頭和內文來處理回應。例如，真實世界應用程式可以透過將回應內文寫入快閃記憶體來處理回應。如果回應狀態碼不是 206 partial content，則任務會通知主要任務示範應失敗。回應任務是在 函數 中指定prvResponseTask。您可以在 Github 上找到此函數的原始碼。

AWS IoT 任務程式庫示範

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

簡介

AWS IoT Jobs 程式庫示範說明如何透過 MQTT 連線連線至 AWS IoT Jobs 服務、從中擷取任務 AWS IoT，以及在裝置上處理任務。 AWS IoT 任務示範專案使用 FreeRTOS Windows 連接埠，因此可以使用 Windows 上的 Visual Studio 社群版本建置和評估。不需要微控制器硬體。示範會建立使用 TLS 與 AWS IoT MQTT 相互身分驗證示範相同的 MQTT 代理程式安全連線。

> **Note**
>
> 若要設定和執行 FreeRTOS 示範，請遵循中的步驟FreeRTOS 入門。

原始碼組織

示範程式碼位於 jobs_demo.c 檔案中，可在 GitHub 網站或 *freertos*/demos/jobs_for_aws/目錄中找到。

設定 AWS IoT MQTT 代理程式連線

在此示範中，您會使用 MQTT 連線至 AWS IoT MQTT 代理程式。此連線的設定方式與 MQTT 相互身分驗證示範中的相同。

功能

示範顯示用來從 接收任務 AWS IoT 並在裝置上處理任務的工作流程。示範是互動式的，要求您使用 AWS IoT 主控台或 AWS Command Line Interface () 來建立任務AWS CLI。如需建立任務的詳細資訊，請參閱 AWS CLI 命令參考中的 create-job。示範需要任務文件將`action`金鑰設定為 `print`，才能將訊息列印到 主控台。

請參閱此任務文件的下列格式。

```
{
    "action": "print",
    "message": "ADD_MESSAGE_HERE"
}
```

您可以使用 AWS CLI 來建立任務，如下列範例命令所示。

```
aws iot create-job \
    --job-id t12 \
    --targets arn:aws:iot:region:123456789012:thing/device1 \
    --document '{"action":"print","message":"hello world!"}'
```

此示範也會使用 `action`金鑰設定為 的任務文件`publish`，將訊息重新發佈至主題。請參閱此任務文件的下列格式。

```
{
    "action": "publish",
    "message": "ADD_MESSAGE_HERE",
    "topic": "topic/name/here"
}
```

示範會循環，直到收到`action`金鑰設為 的任務文件`exit`，以結束示範。任務文件的格式如下所示。

```
{
```

```
    "action: "exit"
}
```

任務示範的進入點

您可以在 GitHub 上找到 Jobs 示範進入點函數的原始碼。此函數會執行下列操作：

1. 使用 中的協助程式函數建立 MQTT 連線`mqtt_demo_helpers.c`。
2. 使用 中的協助程式函數，訂閱 `NextJobExecutionChanged` API 的 MQTT 主題`mqtt_demo_helpers.c`。主題字串會較舊使用 AWS IoT Jobs 程式庫定義的巨集來組合。
3. 使用 中的協助程式函數，發佈至 `StartNextPendingJobExecution` API 的 MQTT 主題`mqtt_demo_helpers.c`。主題字串會較舊使用 AWS IoT Jobs 程式庫定義的巨集來組合。
4. 重複呼叫 `MQTT_ProcessLoop` 以接收傳遞給 `prvEventCallback`進行處理的傳入訊息。
5. 示範收到結束動作後，請使用 `mqtt_demo_helpers.c` 檔案中的協助程式函數，取消訂閱 MQTT 主題並中斷連線。

已接收 MQTT 訊息的回呼

prvEventCallback 函數`Jobs_MatchTopic`會從 AWS IoT 任務程式庫呼叫 來分類傳入的 MQTT 訊息。如果訊息類型對應至新任務，則 `prvNextJobHandler()` 會呼叫 。

prvNextJobHandler 函數及其呼叫的函數，從 JSON 格式的訊息剖析任務文件，並執行任務指定的動作。特別感興趣的是 `prvSendUpdateForJob`函數。

傳送執行中任務的更新

函數 prvSendUpdateForJob() `Jobs_Update()` 從任務程式庫呼叫 ，以填入 MQTT 發佈操作中使用的主題字串，緊接著是。

coreMQTT 示範

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

這些示範可協助您了解如何使用 coreMQTT 程式庫。

主題

- coreMQTT 相互身分驗證示範
- coreMQTT 連線共用示範

coreMQTT 相互身分驗證示範

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

簡介

coreMQTT 相互身分驗證示範專案說明如何使用 TLS 建立與 MQTT 代理程式的連線，並在用戶端和伺服器之間進行相互身分驗證。此示範使用以 mbedTLS 為基礎的傳輸介面實作來建立伺服器和用戶端驗證的 TLS 連線，並示範 MQTT 在 QoS 1 層級的訂閱發佈工作流程。它會訂閱主題篩選條件，然後發佈至符合篩選條件的主題，並等待從 QoS 1 層級的伺服器接收這些訊息。此發佈至代理程式並從代理程式接收相同訊息的週期會無限期重複。此示範中的訊息會在 QoS 1 傳送，根據 MQTT 規格保證至少一個交付。

> **Note**
>
> 若要設定和執行 FreeRTOS 示範，請遵循中的步驟FreeRTOS 入門。

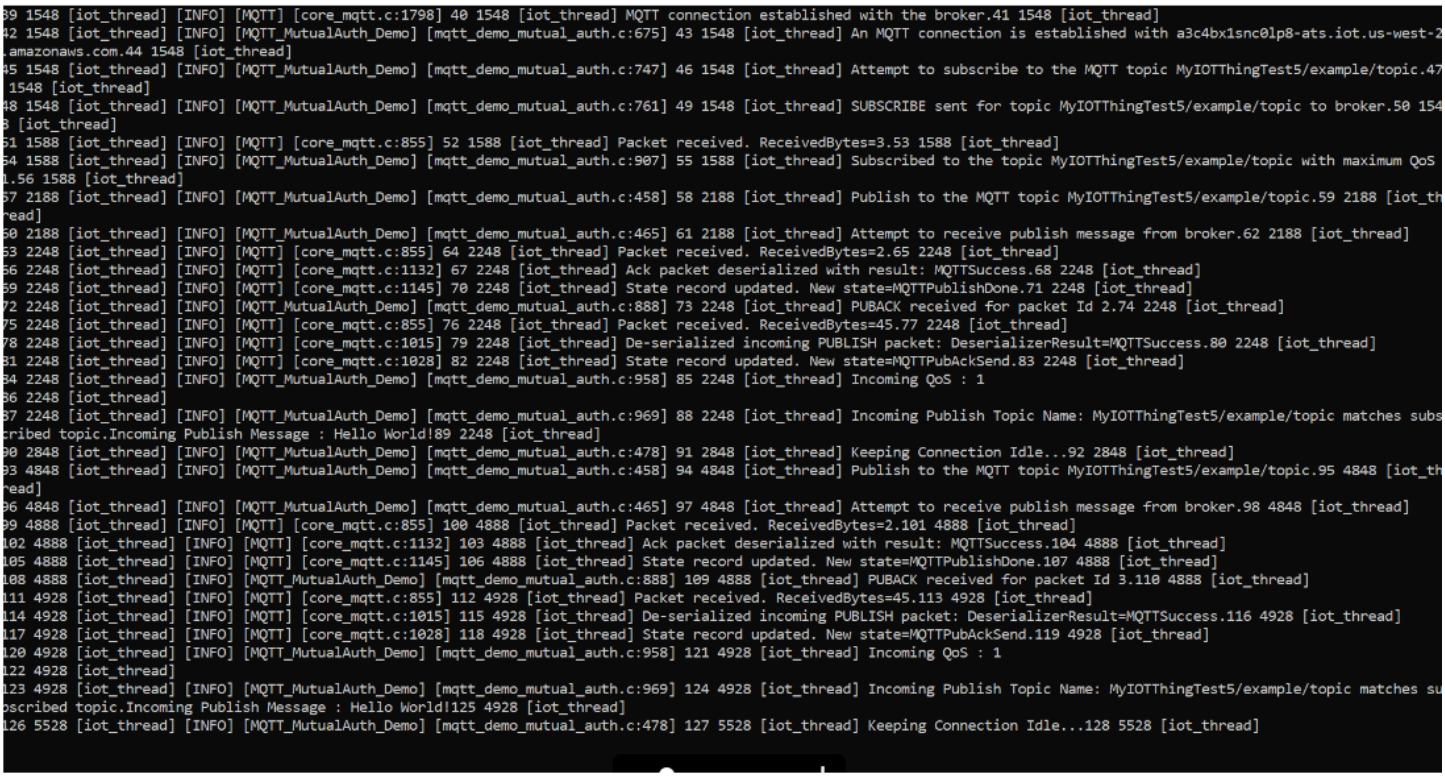
來源碼

示範來源檔案已命名`mqtt_demo_mutual_auth.c`，可在 *`freertos`*`/demos/coreMQTT/`目錄和 GitHub 網站中找到。

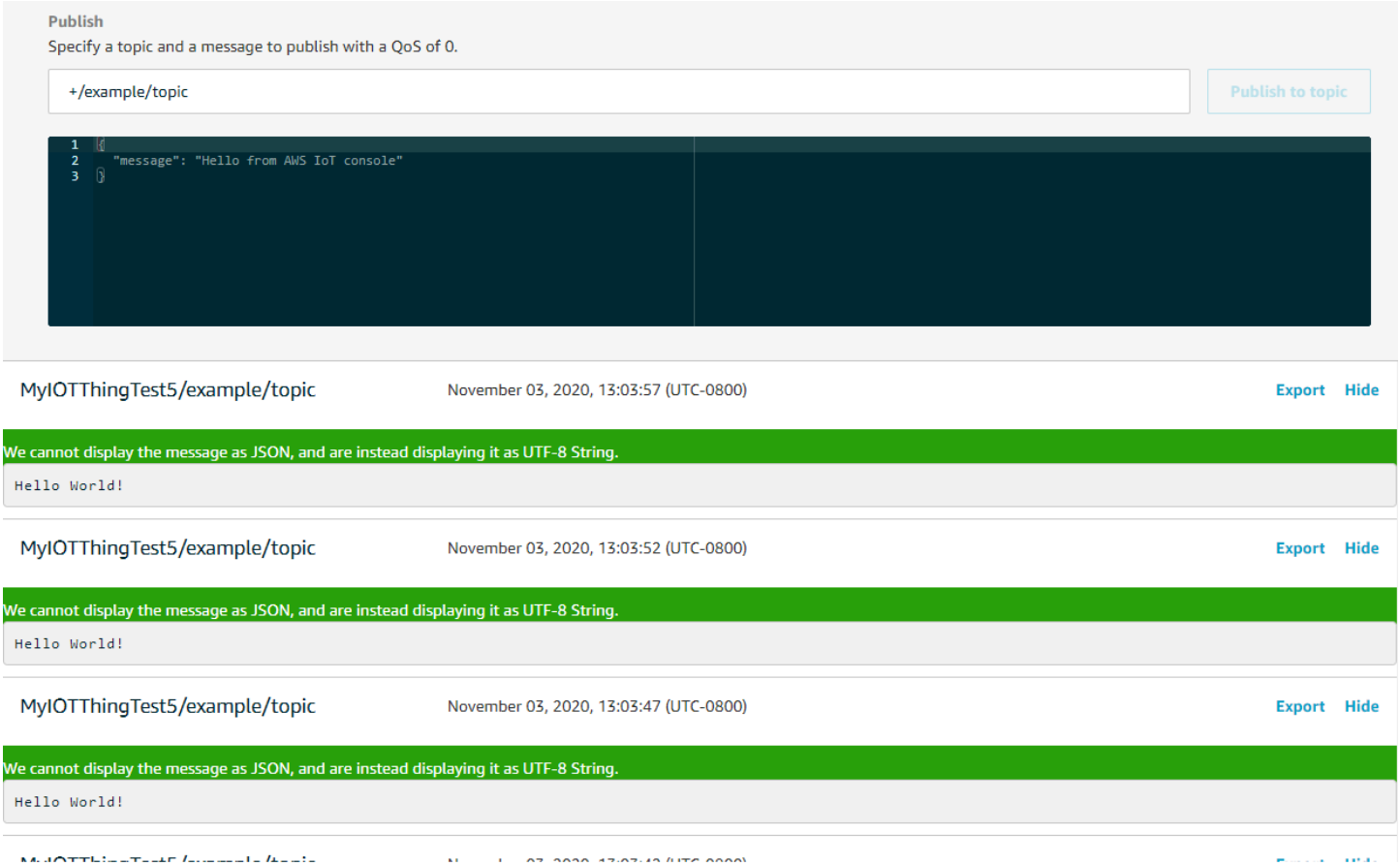
功能

示範會建立單一應用程式任務，透過一組範例循環，示範如何連線至代理程式、訂閱代理程式主題、發佈至代理程式主題，最後中斷與代理程式的連線。示範應用程式同時訂閱和發佈至相同的主題。每次示範發佈訊息到 MQTT 代理程式時，代理程式都會將相同的訊息傳回示範應用程式。

成功完成示範後，會產生類似下列影像的輸出。



AWS IoT 主控台將產生類似下列映像的輸出。



使用指數退避和抖動重試邏輯

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 連線或 MQTT 訂閱請求。函數會計算下一次重試嘗試的退避期間，如果重試嘗試尚未用盡，則會執行退避延遲。由於延遲期間的計算需要產生隨機數字，函數會使用 PKCS11 模組來產生隨機數字。如果廠商平台支援，則使用 PKCS11 模組允許存取真隨機數字產生器 (TRNG)。我們建議您使用裝置特定的熵來源植入隨機數字產生器，以降低連線重試期間裝置發生碰撞的機率。

連線至 MQTT 代理程式

prvConnectToServerWithBackoffRetries 函數會嘗試與 MQTT 代理程式建立相互驗證的 TLS 連線。如果連線失敗，則會在退避期間之後重試。退避期間會呈指數增加，直到達到最大嘗試次數或達到最大退避期間為止。BackoffAlgorithm_GetNextBackoff 函數提供指數增加的退避值，並在達到最大嘗試次數RetryUtilsRetriesExhausted時傳回。如果在設定的嘗試次數之後無法建立與代理程式的 TLS 連線，prvConnectToServerWithBackoffRetries則函數會傳回失敗狀態。

prvCreateMQTTConnectionWithBroker 函數示範如何使用乾淨的工作階段建立 MQTT 與 MQTT 代理程式的連線。它使用 TLS 傳輸界面，該界面在 FreeRTOS-Plus/Source/Application-

Protocols/platform/freertos/transport/src/tls_freertos.c 檔案中實作。請記住，我們正在 中設定代理程式的保持連線秒數xConnectInfo。

下一個函數顯示如何使用 函數在 MQTT 內容中設定 TLS 傳輸界面和時間MQTT_Init函數。它也會顯示如何設定事件回呼函數指標 (prvEventCallback)。此回呼用於報告傳入訊息。

訂閱 MQTT 主題

prvMQTTSubscribeWithBackoffRetries 函數示範如何在 MQTT 代理程式上訂閱主題篩選條件。此範例示範如何訂閱一個主題篩選條件，但可以在相同的訂閱 API 呼叫中傳遞主題篩選條件清單，以訂閱多個主題篩選條件。此外，如果 MQTT 代理程式拒絕 訂閱請求，訂閱將會以指數退避的方式重試RETRY_MAX_ATTEMPTS。

發佈到主題

prvMQTTPublishToTopic 函數示範如何發佈至 MQTT 代理程式上的主題。

接收傳入訊息

應用程式會在連線到代理程式之前註冊事件回呼函數，如前所述。prvMQTTDemoTask 函數會呼叫 MQTT_ProcessLoop函數來接收傳入的訊息。收到傳入 MQTT 訊息時，它會呼叫應用程式註冊的事件回呼函數。 prvEventCallback 函數是這類事件回呼函數的範例。會prvEventCallback檢查傳入封包類型，並呼叫適當的處理常式。在下面的範例中，函數會呼叫prvMQTTProcessIncomingPublish()來處理傳入發佈訊息或prvMQTTProcessResponse()處理確認 (ACK)。

處理傳入的 MQTT 發佈封包

prvMQTTProcessIncomingPublish 函數示範如何處理來自 MQTT 代理程式的發佈封包。

從主題取消訂閱

工作流程中的最後一個步驟是取消訂閱主題，讓代理程式不會從 傳送任何已發佈的訊息mqttexampleTOPIC。以下是函數 prvMQTTUnsubscribeFromTopic 的定義。

變更示範中使用的根 CA

根據預設，FreeRTOS 示範會使用 Amazon Root CA 1 憑證 (RSA 2048 位元金鑰） 來驗證 AWS IoT Core 伺服器。可以使用其他 CA 憑證進行伺服器身分驗證，包括 Amazon Root CA 3 憑證 (ECC 256 位元金鑰）。若要變更 coreMQTT 相互身分驗證示範的根 CA：

1. 在文字編輯器中，開啟 *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/mqtt_demo_mutual_auth_config.h 檔案。
2. 在 檔案中，找到以下行。

   ```
   * #define democonfigROOT_CA_PEM    "...insert here..."
   ```

   取消註解此行，並視需要將其移至註解區塊結束 */。
3. 複製您要使用的 CA 憑證，然後貼到"...insert here..."文字中。結果應如下列範例所示。

   ```
   #define democonfigROOT_CA_PEM   "-----BEGIN CERTIFICATE-----\n"\
   "MIIBtjCCAVugAwIBAgITBmyf1XSXNmY/Owua2eiedgPySjAKBggqhkjOPQQDAjA5\n"\
   "MQswCQYDVQQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwFwYDVQQDExBBbWF6b24g\n"\
   "Um9vdCBDQSAzMB4XDTE1MDUyNjAwMDAwMFoXDTQwMDUyNjAwMDAwMFowOTELMAkG\n"\
   "A1UEBhMCVVMxDzANBgNVBAoTBkFtYXpvbjEZMBcGA1UEAxMQQW1hem9uIFJvb3Qg\n"\
   "Q0EgMzBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABCmXp8ZBf8ANm+gBG1bG8lKl\n"\
   "ui2yEujSLtf6ycXYqm0fc4E7O5hrOXwzpcVOho6AF2hiRVd9RFgdszflZwjrZt6j\n"\
   "QjBAMA8GA1UdEwEB/wQFMAMBAf8wDgYDVR0PAQH/BAQDAgGGMB0GA1UdDgQWBBSr\n"\
   "ttvXBp43rDCGB5Fwx5zEGbF4wDAKBggqhkjOPQQDAgNJADBGAiEA4IWSoxe3jfkr\n"\
   "BqWTrBqYaGFy+uGh0PsceGCmQ5nFuMQCIQCcAu/xlJyzlvnrxir4tiz+OpAUFteM\n"\
   "YyRIHN8wfdVoOw==\n"\
   "-----END CERTIFICATE-----\n"
   ```
4. （選用）您可以變更其他示範的根 CA。針對每個*freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/*demo-name*_config.h檔案重複步驟 1 到 3。

coreMQTT 連線共用示範

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

簡介

coreMQTT 連線共用示範專案說明如何使用多執行緒應用程式，使用 TLS 建立與 AWS MQTT 代理程式的連線，並在用戶端和伺服器之間進行相互身分驗證。此示範使用 mbedTLS 型傳輸介面實作來建立伺服器和用戶端驗證的 TLS 連線，並示範 MQTT 在 QoS 1 層級的訂閱發佈工作流程。示範會訂閱

主題篩選條件，發佈至符合篩選條件的主題，然後等待從 QoS 1 層級的伺服器接收這些訊息。此發佈至代理程式並從代理程式接收相同訊息的週期會無限期重複。此示範中的訊息會在 QoS 1 傳送，根據 MQTT 規格保證至少一個交付。

> **Note**
>
> 若要設定和執行 FreeRTOS 示範，請遵循中的步驟FreeRTOS 入門。

此示範使用執行緒安全佇列來保留命令，以與 MQTT API 互動。此示範中需要注意四個任務。

- 命令（主要）任務會從命令佇列取得命令並處理它們。其他任務會將命令放置在此佇列中，以進行處理。此任務會進入迴圈，在此期間處理命令。如果收到終止命令，則此任務會中斷迴圈。
- 同步發佈者任務會建立一系列發佈操作，並將其推送至命令佇列。然後，命令任務會執行這些操作。此任務使用同步發佈，這表示此任務會等待每個發佈操作完成，然後再排程下一個發佈操作。
- 非同步發佈者任務會建立一系列發佈操作，並將其推送至命令佇列。然後，命令任務會執行這些操作。此任務和上一個任務之間的差異在於，在排程下一個任務之前，不會等待發佈操作完成。它會在所有發佈操作新增至佇列後檢查每個發佈操作的狀態。請注意，同步和非同步發佈之間的差異僅在這些任務的行為中。實際發佈命令沒有差異。
- 訂閱者任務會建立與兩個發佈者任務發佈之所有訊息主題相符的主題篩選條件的 MQTT 訂閱。此任務進入迴圈，並等待接收其他任務發佈的訊息。

任務可以有佇列來保留收到的訊息。命令任務會將傳入訊息推送到訂閱傳入主題之任何任務的佇列。

此示範使用具有相互身分驗證的 TLS 連線來連線 AWS。如果網路在示範期間意外中斷連線，則用戶端會嘗試使用指數退避邏輯重新連線。如果用戶端成功重新連線，但代理程式無法繼續先前的工作階段，則用戶端會重新訂閱與先前工作階段相同的主題。

單一執行緒與多執行緒

有兩種 coreMQTT 用量模型：單一執行緒和多執行緒（多任務）。此示範會示範如何建立自己的多執行緒方案。還有另一個多執行緒範例，會在代理程式（或協助程式）任務的背景中執行 MQTT 通訊協定。如需詳細資訊，請參閱使用 coreMQTT 的 MWTT 代理程式和示範。如果您在代理程式任務中執行 MQTT 通訊協定，則不需要明確管理任何 MQTT 狀態或呼叫 `MQTT_ProcessLoop`函數。此外，如果您使用代理程式任務，多個應用程式任務可以共用單一 MQTT 連線，而不需要同步基本概念，例如 Mutex。

來源碼

示範來源檔案已命名mqtt_demo_connection_sharing.c，可在 *freertos*/demos/coreMQTT/目錄和 GitHub 網站中找到。

功能

此示範總共會建立四個任務：三個請求 MQTT API 呼叫，另一個處理這些請求，這是主要任務。在此示範中，主要任務會進入迴圈，以建立三個子任務、呼叫處理迴圈，並在之後清除。主要任務會建立單一的 MQTT 連線，連接到子任務之間共用的代理程式。其中兩個子任務會將訊息發佈至代理程式，而第三個任務會使用 MQTT 訂閱，將訊息傳回至符合所發佈訊息所有主題的主題篩選條件。

Typedefs

示範會定義下列結構、列舉和函數指標。

命令

這些任務使用 Command_t 結構來建立命令，指示主要任務為其呼叫適當的 API 操作，而不是直接進行 MQTT API 呼叫。命令具有下列類型：

- PROCESSLOOP
- PUBLISH
- SUBSCRIBE
- UNSUBSCRIBE
- PING
- DISCONNECT
- RECONNECT
- TERMINATE

TERMINATE 命令沒有對應的 MQTT API 操作。它用於示範，指示主要任務停止處理命令並開始清除操作。由於某些 MQTT 命令需要一些額外資訊，例如發佈或訂閱資訊，例如 MQTT_Publish、MQTT_Subscribe和 MQTT_Unsubscribe，因此我們會使用 CommandContext_t 欄位。這三個命令需要此欄位，其他命令則為選用。

由於這些命令需要此內容，因此在命令放入佇列後，請勿變更此值，直到命令完成為止。當命令完成時，可以叫用選用的回呼。在此示範中，我們使用建立任務通知的回呼，通知呼叫任務命令已完

成。對於需要確認的 MQTT 操作 （ 訂閱、取消訂閱和發佈 QoS 大於 0)，在收到確認後，命令即視為已完成。否則，一旦傳回對應的 MQTT API 呼叫，命令就會完成。

您可以在 `mqtt_demo_connection_sharing.c` 檔案中找到下列定義：

- Command_t 結構
- CommandContext_t 結構
- CommandType_t 列舉
- CommandCallback_t 函數指標

致謝

由於某些 MQTT 操作需要確認，因此它們會使用 AckInfo_t 陣列，其中包含預期確認的封包識別符，以及預期的原始命令，以便叫用其完成回呼。

訂閱

此示範可以追蹤每個任務的訂閱。若要這樣做，請求訂閱的每個任務都必須提供訊息佇列，以便接收已發佈的訊息 (SubscriptionElement_t)。多個任務可以訂閱相同的主題篩選條件，因為它們需要使用單獨的回應佇列。

收到已發佈的訊息

由於任務與主要任務平行執行，因此主要任務必須等待每個訂閱的任務讀取收到的已發佈訊息，會很困難且耗時。因此，收到的每則訊息都會複製到訂閱已發佈訊息主題 (PublishElement_t) 之任何任務的回應佇列。由於從 MQTT 用戶端收到的發佈封包包含用戶端網路緩衝區的指標，傳入訊息的承載和主題名稱會複製到單獨的緩衝區，再插入回應佇列。如此一來，訂閱的任務仍然可以在 MQTT 用戶端清除其網路緩衝區後讀取收到的資訊。

主要任務

主要應用程式任務 RunCoreMQTTConnectionSharingDemo 會建立持久性 MQTT 工作階段、建立三個子任務，並執行處理迴圈，直到收到終止命令為止。使用持久性工作階段，因此如果網路意外中斷連線，示範會重新連線至背景的代理程式，而不會遺失訂閱或任何來自代理程式的已發佈訊息。若要為每次執行建立新的持久性工作階段，示範會使用 `clean session`旗標集連接到代理程式，然後中斷連線，並與 旗標取消設定重新連線。處理迴圈終止後，它會中斷與代理程式的連線，並再次從進行網路重新連線的時間點開始迴圈。

成功完成示範後，會產生類似下列影像的輸出。

```
128 11020 [AsyncPublisher] [INFO] [MQTTDemo] [prvAsyncPublishTask:1960] 129 11020 [AsyncPublisher] Publish operation queued. Sleeping for 100 ms.
130 11020 [AsyncPublisher]
131 11036 [MQTTDemo] [DEBUG] [MQTTDemo] [prvCommandLoop:1788] 132 11036 [MQTTDemo] Processed 2 operations.133 11036 [MQTTDemo]
134 11036 [MQTTDemo] [DEBUG] [MQTTDemo] [prvProcessCommand:1433] 135 11036 [MQTTDemo] Publishing message to filter/sync/1.136 11036 [MQTTDemo]
137 11056 [MQTTDemo] [DEBUG] [MQTTDemo] [prvCommandLoop:1788] 138 11056 [MQTTDemo] Processed 3 operations.139 11056 [MQTTDemo]
140 11056 [MQTTDemo] [DEBUG] [MQTTDemo] [prvProcessCommand:1433] 141 11056 [MQTTDemo] Publishing message to filter/async/1.142 11056 [MQTTDemo]
143 11056 [MQTTDemo] [DEBUG] [MQTTDemo] [prvHandleIncomingPublish:1571] 144 11056 [MQTTDemo] Adding publish to response queue for filter/+/+
145 11056 [MQTTDemo]
146 11056 [Subscriber] [INFO] [MQTTDemo] [prvSubscribeTask:2059] 147 11056 [Subscriber] Received publish on topic filter/sync/1
Message payload: Hello World! Sync: 1
148 11056 [Subscriber]
149 11056 [SyncPublisher] [INFO] [MQTTDemo] [prvSyncPublishTask:1888] 150 11057 [SyncPublisher] Publish operation complete. Sleeping for 100 ms.
151 11057 [SyncPublisher]
152 11122 [AsyncPublisher] [INFO] [MQTTDemo] [prvAsyncPublishTask:1954] 153 11122 [AsyncPublisher] Adding publish operation for message Hello World! Async: 4
on topic filter/async/4154 11122 [AsyncPublisher]
155 11122 [AsyncPublisher] [INFO] [MQTTDemo] [prvAsyncPublishTask:1960] 156 11122 [AsyncPublisher] Publish operation queued. Sleeping for 100 ms.
157 11122 [AsyncPublisher]
158 11157 [SyncPublisher] [INFO] [MQTTDemo] [prvSyncPublishTask:1873] 159 11157 [SyncPublisher] Adding publish operation for message Hello World! Sync: 2
on topic filter/sync/2160 11157 [SyncPublisher]
161 11157 [SyncPublisher] [INFO] [MQTTDemo] [prvSyncPublishTask:1878] 162 11157 [SyncPublisher] Waiting for publish 2 to complete.163 11157 [SyncPublisher]
164 11196 [MQTTDemo] [DEBUG] [MQTTDemo] [prvCommandLoop:1788] 165 11196 [MQTTDemo] Processed 4 operations.166 11196 [MQTTDemo]
167 11196 [MQTTDemo] [DEBUG] [MQTTDemo] [prvProcessCommand:1433] 168 11196 [MQTTDemo] Publishing message to filter/async/2.169 11196 [MQTTDemo]
170 11196 [MQTTDemo] [DEBUG] [MQTTDemo] [prvHandleIncomingPublish:1571] 171 11196 [MQTTDemo] Adding publish to response queue for filter/+/+
172 11196 [MQTTDemo]
173 11196 [Subscriber] [INFO] [MQTTDemo] [prvSubscribeTask:2059] 174 11196 [Subscriber] Received publish on topic filter/async/1
Message payload: Hello World! Async: 1
175 11196 [Subscriber]
176 11222 [AsyncPublisher] [INFO] [MQTTDemo] [prvAsyncPublishTask:1954] 177 11222 [AsyncPublisher] Adding publish operation for message Hello World! Async: 5
on topic filter/async/5178 11222 [AsyncPublisher]
179 11222 [AsyncPublisher] [INFO] [MQTTDemo] [prvAsyncPublishTask:1960] 180 11222 [AsyncPublisher] Publish operation queued. Sleeping for 100 ms.
```

命令迴圈

命令迴圈 prvCommandLoop 會等待命令放置在命令佇列中，然後呼叫適當的 MQTT API。除了DISCONNECT和 TERMINATE之外，所有命令也會MQTT_ProcessLoop被呼叫。此示範會設定通訊端喚醒回呼，以在通訊端上提供資料時，將PROCESSLOOP命令新增至佇列。不過，在那時，佇列中可能會有多個命令。為了確保我們在處理其他命令時不會忽略傳入資料， MQTT_ProcessLoop 會在每個命令之後呼叫 進行單一反覆運算。

處理命令

請參閱 prvProcessCommand 函數。

同步發佈者任務

同步發佈者任務 prvSyncPublishTask 會同步建立PUBLISH操作，並等待每個操作完成，然後再排程下一個操作。此示範使用 QoS 1 來發佈訊息，這表示在收到發佈確認封包之前，這些操作不會被視為已完成。

非同步發佈者任務

非同步發佈者任務 prvAsyncPublishTask 不會等待發佈完成，再將下一個發佈放入佇列。這表示任務不一定需要等待 MQTT 操作完成，才能繼續。由於每個發佈命令都需要自己的內容結構，因此此任務無法像同步發佈者任務那樣重複使用單一內容結構，因為先前的命令可能仍然需要它。因此，它會為每個內容結構配置記憶體，然後等待所有要發佈的訊息都放入佇列後，釋放所有配置的記憶體。

訂閱者任務

訂閱者任務 prvSubscribeTask 會訂閱主題篩選條件，以符合從同步和非同步任務發佈的訊息的所有主題。然後，它會等待接收所有已發佈的訊息，再取消訂閱。此任務也負責建立 TERMINATE操作，以發出結束命令迴圈的主要任務訊號。

無線更新示範應用程式

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

FreeRTOS 包含示範應用程式，可示範over-the-air(OTA) 程式庫的功能。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. 使用 建立任務以執行 OTA 程式庫vRunOTAUpdateDemo()。
3. 使用 _establishMqttConnection() 建立 MQTT 用戶端。
4. 使用 連線至 AWS IoT MQTT 代理程式，IotMqtt_Connect()並註冊 MQTT 中斷連線回呼：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 重複使用 MQTT 連線 xOTAConnectionCtx.pvControlClient = _mqttConnection;
7. 如果 MQTT 中斷連線，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線，然後繼續 OTA 代理程式。

您必須先完成 FreeRTOS Over-the-Air更新中的所有必要條件，才能才能使用 OTA 更新。

完成 OTA 更新設定後，請在支援 OTA 功能的平台上下載、建置、刷新和執行 FreeRTOS OTA 示範。下列 FreeRTOS 合格裝置可使用裝置特定的示範說明：

- Texas Instruments CC3220SF-LAUNCHXL
- Microchip Curiosity PIC32MZEF
- Espressif ESP32

- 在 Renesas RX65N 上下載、建置、刷新和執行 FreeRTOS OTA 示範

在裝置上建置、刷新和執行 OTA 示範應用程式後，您可以使用 AWS IoT 主控台或 AWS CLI 來建立 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-LaunchPad/streams/327

61 3456 [OTA Task] [OTA] Accepted final image. Commit.
62 3578 [OTA Task] Sending command to MQTT task.
63 3578 [MQTT] Received message 70000 from queue.
64 3779 [MQTT] MQTT Publish was successful.
65 3780 [MQTT] Notifying task.
66 3780 [OTA Task] Command sent to MQTT task passed.
67 3781 [OTA Task] [OTA] Published 'SUCCEEDED' status to $aws/things/TI-LaunchPad/jobs/
fe18c7ec_8c31_4438_b0b9_ad55acd95610/upd68 3781 [OTA Task] [OTA] Returned buffer to
 MQTT Client.
69 4251 [OTA] [OTA] Queued: 1   Processed: 1   Dropped: 0
70 4381 [OTA Task] [OTA] Missing job parameter: execution
71 4382 [OTA Task] [OTA] Missing job parameter: jobId
72 4382 [OTA Task] [OTA] Missing job parameter: jobDocument
73 4382 [OTA Task] [OTA] Missing job parameter: ts_ota
74 4382 [OTA Task] [OTA] Missing job parameter: files
75 4382 [OTA Task] [OTA] Missing job parameter: streamname
76 4382 [OTA Task] [OTA] Missing job parameter: certfile
77 4382 [OTA Task] [OTA] Missing job parameter: filepath
```

```
78 4383 [OTA Task] [OTA] Missing job parameter: filesize
79 4383 [OTA Task] [OTA] Missing job parameter: sig-sha1-rsa
80 4383 [OTA Task] [OTA] Missing job parameter: fileid
81 4383 [OTA Task] [OTA] Missing job parameter: attr
82 4383 [OTA Task] [OTA] Returned buffer to MQTT Client.
83 5251 [OTA] [OTA] Queued: 2   Processed: 2   Dropped: 0
```

Over-the-air示範組態

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

OTA 示範組態是 中提供的示範特定組態選項`aws_iot_ota_update_demo.c`。這些組態與 OTA 程式庫組態檔案中提供的 OTA 程式庫組態不同。

OTA_DEMO_KEEP_ALIVE_SECONDS

對於 MQTT 用戶端，此組態是完成一個控制封包傳輸到開始傳送下一個控制封包之間可能經過的最大時間間隔。如果沒有控制封包，則會傳送 PINGREQ。代理程式必須中斷用戶端的連線，該用戶端在此保持運作間隔的一倍半內不會傳送訊息或 PINGREQ 封包。此組態應根據應用程式的需求進行調整。

OTA_DEMO_CONN_RETRY_BASE_INTERVAL_SECONDS

重試網路連線之前的基本間隔，以秒為單位。OTA 示範將嘗試在此基本時間間隔之後重新連線。每次嘗試失敗後，間隔會加倍。隨機延遲也會新增至間隔，最多可達此基本延遲的上限。

OTA_DEMO_CONN_RETRY_MAX_INTERVAL_SECONDS

重試網路連線之前的最大間隔，以秒為單位。重新連線延遲會在每次失敗嘗試時加倍，但只能達到此最大值，加上相同間隔的抖動。

在 Texas Instruments CC3220SF-LAUNCHXL 上下載、建置、刷新和執行 FreeRTOS OTA 示範

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

下載 FreeRTOS 和 OTA 示範程式碼

- 您可以在 GitHub 網站 https：//https://github.com/FreeRTOS/FreeRTOS 下載原始程式碼。

建置示範應用程式

1. 請遵循 中的指示，將aws_demos專案FreeRTOS 入門匯入 Code Composer Studio、設定您的 AWS IoT 端點、Wi-Fi SSID 和密碼，以及電路板的私有金鑰和憑證。
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_OTA_UPDATE_DEMO_ENABLED。
3. 建置解決方案並確認過程中沒有發生任何錯誤。
4. 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
5. 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 上下載、建置、刷新和執行 FreeRTOS OTA 示範

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

下載 FreeRTOS OTA 示範程式碼

- 您可以在 GitHub 網站 https：//https://github.com/FreeRTOS/FreeRTOS 下載原始程式碼。

建置 OTA 更新示範應用程式

1. 請遵循 中的指示，將aws_demos專案FreeRTOS 入門匯入 MPLAB X IDE、設定您的 AWS IoT 端點、Wi-Fi SSID 和密碼，以及電路板的私有金鑰和憑證。
2. 打開 `demos/include/aws_ota_codesigner_certificate.h`.
3. 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"
   "MIIBXTCCAQOgAwIBAgIJAM4DeybZcTwKMAoGCCqGSM49BAMCMCExHzAdBgNVBAMM\n"
   "FnRlc3Rf62lnbmVyQGFtYXpvbi5jb20wHhcNMTcxMTAzMTkxODM1WhcNMTgxMTAz\n"
   "MTkxODM2WjAhMR8wHQYDVQBBZZZ0ZXN0X3NpZ25lckBhbWF6b24uY29tMFkwEwYH\n"
   "KoZIzj0CAQYIKoZIzj0DAQcDQgAERavZfvwL1X+E4dIF7dbkVMUn4IrJ1CAsFkc8\n"
   "gZxPzn683H40XMKltDZPEwr9ng78w9+QYQg7ygnr2stz8yhh06MkMCIwCwYDVR0P\n"
   "BAQDAgeAMBMGA1UdJQQMMAoGCCsGAQUFBwMDMAoGCCqGSM49BAMCA0gAMEUCIF0R\n"
   "r5cb7rEUNtWOvGd05MacrgOABfSoVYvBOK9fP63WAqt5h3BaS123coKSGg84twlq\n"
   "TkO/pV/xEmyZmZdV+HxV/OM=\n"
   "-----END CERTIFICATE-----\n";
   ```

4. 安裝 Python 3 或更新版本。
5. 透過執行 `pip install pyopenssl` 來安裝 pyOpenSSL。

6. 複製 demos/ota/bootloader/utility/codesigner_cert_utility/ 路徑中格式為 .pem 的程式碼簽署憑證。將該憑證檔案重新命名為 aws_ota_codesigner_certificate.pem。
7. 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。
8. 建置解決方案並確認過程中沒有發生任何錯誤。
9. 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
10. 從電路板中拔除偵錯器，並在電路板上執行專案，以確認它可以連線至 Wi-Fi 和 AWS IoT MQTT 訊息代理程式。

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] vDHCPProcess: offer ac140a0eip
                                                  1 36297 [IP-task] vDHCPProcess: offer
 ac140a0eip
                2 36297 [IP-task]

IP Address: 172.20.10.14
```

```
3 36297 [IP-task] Subnet Mask: 255.255.255.240
4 36297 [IP-task] Gateway Address: 172.20.10.1
5 36297 [IP-task] DNS Server Address: 172.20.10.1


6 36299 [OTA] OTA demo version 0.9.2
7 36299 [OTA] Creating MQTT Client...
8 36299 [OTA] Connecting to broker...
9 38673 [OTA] Connected to broker.
10 38793 [OTA Task] [prvSubscribeToJobNotificationTopics] OK: $aws/things/devthingota/
jobs/$next/get/accepted
11 38863 [OTA Task] [prvSubscribeToJobNotificationTopics] OK: $aws/things/devthingota/
jobs/notify-next
12 38863 [OTA Task] [OTA_CheckForUpdate] Request #0
13 38964 [OTA] [OTA_AgentInit] Ready.
14 38973 [OTA Task] [prvParseJSONbyModel] Extracted parameter [ clientToken:
 0:devthingota ]
15 38973 [OTA Task] [prvParseJSONbyModel] parameter not present: execution
16 38973 [OTA Task] [prvParseJSONbyModel] parameter not present: jobId
17 38973 [OTA Task] [prvParseJSONbyModel] parameter not present: jobDocument
18 38973 [OTA Task] [prvParseJSONbyModel] parameter not present: streamname
19 38973 [OTA Task] [prvParseJSONbyModel] parameter not present: files
20 38975 [OTA Task] [prvParseJSONbyModel] parameter not present: filepath
21 38975 [OTA Task] [prvParseJSONbyModel] parameter not present: filesize
22 38975 [OTA Task] [prvParseJSONbyModel] parameter not present: fileid
23 38975 [OTA Task] [prvParseJSONbyModel] parameter not present: certfile
24 38975 [OTA Task] [prvParseJSONbyModel] parameter not present: sig-sha256-ecdsa
25 38975 [OTA Task] [prvParseJobDoc] Ignoring job without ID.
26 38975 [OTA Task] [prvOTA_Close] Context->0x8003b620
27 38975 [OTA Task] [prvPAL_Abort] Abort - OK
28 39964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
29 40964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
30 41964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
31 42964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
32 43964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
33 44964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
34 45964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
35 46964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
36 47964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
```

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] vDHCPProcess: offer c0a804beip
3 7230 [IP-task] vDHCPProcess: offer c0a804beip
4 7230 [IP-task]

IP Address: 192.168.4.190
5 7230 [IP-task] Subnet Mask: 255.255.240.0
6 7230 [IP-task] Gateway Address: 192.168.0.1
7 7230 [IP-task] DNS Server Address: 208.67.222.222


8 7232 [OTA] OTA demo version 0.9.0
9 7232 [OTA] Creating MQTT Client...
10 7232 [OTA] Connecting to broker...
11 7232 [OTA] Sending command to MQTT task.
12 7232 [MQTT] Received message 10000 from queue.
13 8501 [IP-task] Socket sending wakeup to MQTT task.
```

```
14 10207 [MQTT] Received message 0 from queue.
15 10256 [IP-task] Socket sending wakeup to MQTT task.
16 10256 [MQTT] Received message 0 from queue.
17 10256 [MQTT] MQTT Connect was accepted. Connection established.
18 10256 [MQTT] Notifying task.
19 10257 [OTA] Command sent to MQTT task passed.
20 10257 [OTA] Connected to broker.
21 10258 [OTA Task] Sending command to MQTT task.
22 10258 [MQTT] Received message 20000 from queue.
23 10306 [IP-task] Socket sending wakeup to MQTT task.
24 10306 [MQTT] Received message 0 from queue.
25 10306 [MQTT] MQTT Subscribe was accepted. Subscribed.
26 10306 [MQTT] Notifying task.
27 10307 [OTA Task] Command sent to MQTT task passed.
28 10307 [OTA Task] [OTA] Subscribed to topic: $aws/things/Microchip/jobs/$next/get/
accepted

29 10307 [OTA Task] Sending command to MQTT task.
30 10307 [MQTT] Received message 30000 from queue.
31 10336 [IP-task] Socket sending wakeup to MQTT task.
32 10336 [MQTT] Received message 0 from queue.
33 10336 [MQTT] MQTT Subscribe was accepted. Subscribed.
34 10336 [MQTT] Notifying task.
35 10336 [OTA Task] Command sent to MQTT task passed.
36 10336 [OTA Task] [OTA] Subscribed to topic: $aws/things/Microchip/jobs/notify-next

37 10336 [OTA Task] [OTA] Check For Update #0
38 10336 [OTA Task] Sending command to MQTT task.
39 10336 [MQTT] Received message 40000 from queue.
40 10366 [IP-task] Socket sending wakeup to MQTT task.
41 10366 [MQTT] Received message 0 from queue.
42 10366 [MQTT] MQTT Publish was successful.
43 10366 [MQTT] Notifying task.
44 10366 [OTA Task] Command sent to MQTT task passed.
45 10376 [IP-task] Socket sending wakeup to MQTT task.
46 10376 [MQTT] Received message 0 from queue.
47 10376 [OTA Task] [OTA] Set job doc parameter [ clientToken: 0:Microchip ]
48 10376 [OTA Task] [OTA] Missing job parameter: execution
49 10376 [OTA Task] [OTA] Missing job parameter: jobId
50 10376 [OTA Task] [OTA] Missing job parameter: jobDocument
51 10378 [OTA Task] [OTA] Missing job parameter: ts_ota
52 10378 [OTA Task] [OTA] Missing job parameter: files
53 10378 [OTA Task] [OTA] Missing job parameter: streamname
54 10378 [OTA Task] [OTA] Missing job parameter: certfile
```

```
55 10378 [OTA Task] [OTA] Missing job parameter: filepath
56 10378 [OTA Task] [OTA] Missing job parameter: filesize
57 10378 [OTA Task] [OTA] Missing job parameter: sig-sha256-ecdsa
58 10378 [OTA Task] [OTA] Missing job parameter: fileid
59 10378 [OTA Task] [OTA] Missing job parameter: attr
60 10378 [OTA Task] [OTA] Returned buffer to MQTT Client.
61 11367 [OTA] [OTA] Queued: 1   Processed: 1   Dropped: 0
62 12367 [OTA] [OTA] Queued: 1   Processed: 1   Dropped: 0
63 13367 [OTA] [OTA] Queued: 1   Processed: 1   Dropped: 0
64 14367 [OTA] [OTA] Queued: 1   Processed: 1   Dropped: 0
65 15367 [OTA] [OTA] Queued: 1   Processed: 1   Dropped: 0
66 16367 [OTA] [OTA] Queued: 1   Processed: 1   Dropped: 0
```

此輸出顯示 Microchip Curiosity PIC32MZEF 可以連線至 AWS IoT 並訂閱 OTA 更新所需的 MQTT 主題。`Missing job parameter` 訊息是在預期中的內容，因為沒有任何擱置中的 OTA 更新任務。

在 Espressif ESP32 上下載、建置、刷新和執行 FreeRTOS OTA 示範

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

1. 從 GitHub 下載 FreeRTOS 來源。如需說明，請參閱 README.md 檔案。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`。

在 Renesas RX65N 上下載、建置、刷新和執行 FreeRTOS OTA 示範

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

本章說明如何在 Renesas RX65N 上下載、建置、刷新和執行 FreeRTOS OTA 示範應用程式。

主題

- 設定您的操作環境
- 設定您的 AWS 資源
- 匯入、設定標頭檔案並建置 aws_demos 和 boot_loader

設定您的操作環境

本節中的程序使用以下環境：

- IDE：e^2 Studio 7.8.0、e^2 Studio 2020-07
- 工具鏈：CCRX Compiler v3.0.1
- 目標裝置：RSKRX65N-2MB
- 偵錯工具：E^2、E^2 Lite 模擬器
- 軟體：Renesas Flash Programmer、Renesas Secure Flash Programmer.exe、Tera Term

設定您的硬體

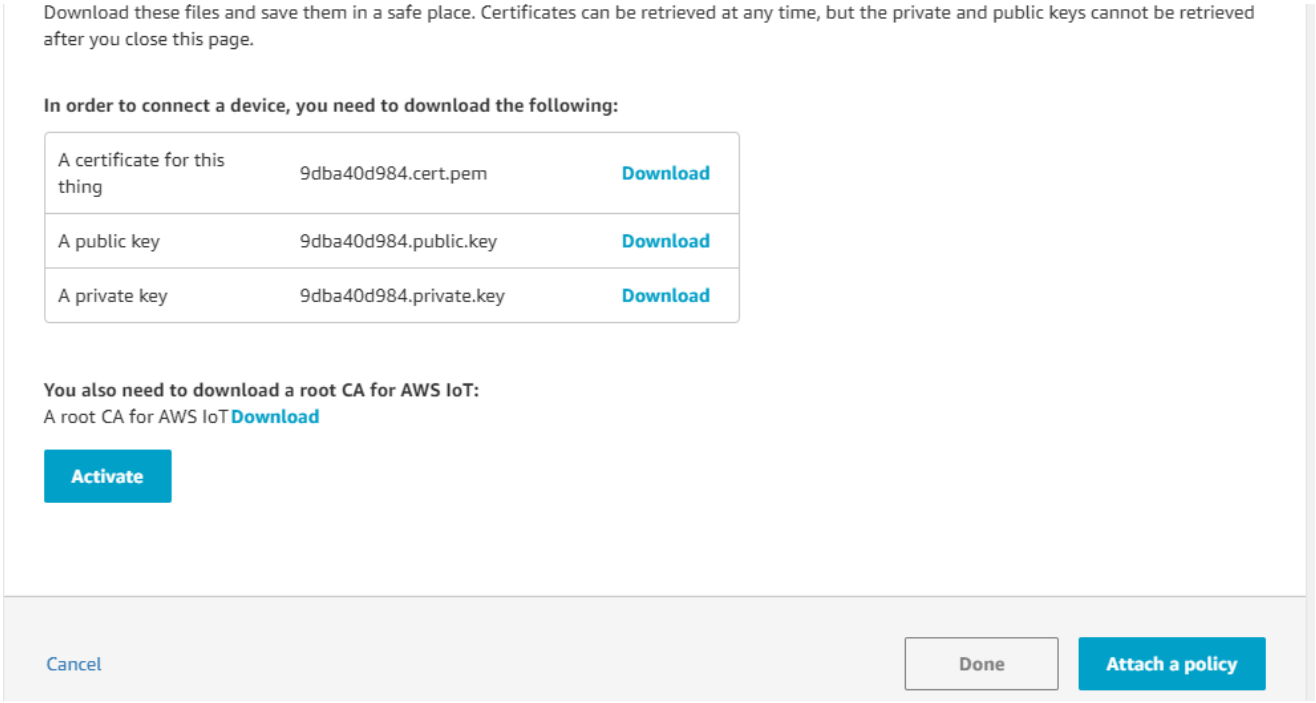
1. 將 E^2 Lite 模擬器和 USB 序列埠連接到 RX65N 主機板和 PC。
2. 將電源連接至 RX65N。

設定您的 AWS 資源

1. 若要執行 FreeRTOS 示範，您必須擁有具有存取 AWS IoT 服務許可的 AWS Identity and Access Management (IAM) 使用者 AWS 帳戶。如果您尚未完成，請遵循中的步驟設定 AWS 您的帳戶和許可。
2. 若要設定 OTA 更新，請遵循中的步驟OTA 更新先決條件。特別是，請遵循 中的步驟使用 MQTT 進行 OTA 更新的先決條件。
3. 開啟 AWS IoT 主控台。
4. 在左側導覽窗格中，選擇管理，然後選擇要建立物件的物件。

   物件是其中裝置或邏輯實體的表示法 AWS IoT。它可以是實體裝置或感應器 (例如燈泡或牆上的開關)。它也可以是邏輯實體，例如未連線的應用程式執行個體或實體實體 AWS IoT，但與這樣做的裝置相關 （ 例如具有引擎感應器或控制面板的汽車 ）。 AWS IoT 提供物件登錄檔，協助您管理物件。

   a. 選擇建立，然後選擇建立單一物件。
   b. 輸入物件的名稱，然後選擇下一步。
   c. 選擇 Create certificate (建立憑證)。
   d. 下載建立的三個檔案，然後選擇啟用。
   e. 選擇 Attach a policy (連接政策)。



   f. 選取您在 中建立的政策裝置政策。

使用 MQTT 接收 OTA 更新的每個裝置都必須在 中註冊為物件， AWS IoT 而且該物件必須具有如所列之附加政策。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。

備註

- iot:Connect 許可允許您的裝置 AWS IoT 透過 MQTT 連線至 。
- 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。
- OTA 串流 (.../streams/*) AWS IoT 主題的 iot:Subscribe和 iot:Publish許可允許連接的裝置從中擷取 OTA 更新資料 AWS IoT。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 在這些主題上發佈訊息到連線的裝置。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. 建立程式碼簽署設定檔並註冊程式碼簽署憑證 AWS。

   a. 若要建立金鑰和認證，請參閱 Renesas MCU 韌體更新設計政策中的第 7.3 節「使用 OpenSSL 產生 ECDSA-SHA256 金鑰對」。

   b. 開啟 AWS IoT 主控台。在左側導覽窗格中，選取管理，然後選取任務。選取建立任務，然後選取建立 OTA 更新任務。

   c. 在選取要更新的裝置下，選擇選取，然後選擇您先前建立的物件。選取下一步。

   d. 在建立 FreeRTOS OTA 更新任務頁面上，執行下列動作：

      i. 針對選取韌體映像傳輸的通訊協定，選擇 MQTT。

      ii. 針對選取並簽署您的韌體映像，選擇為我簽署新的韌體映像。

      iii. 針對程式碼簽署設定檔，選擇建立。

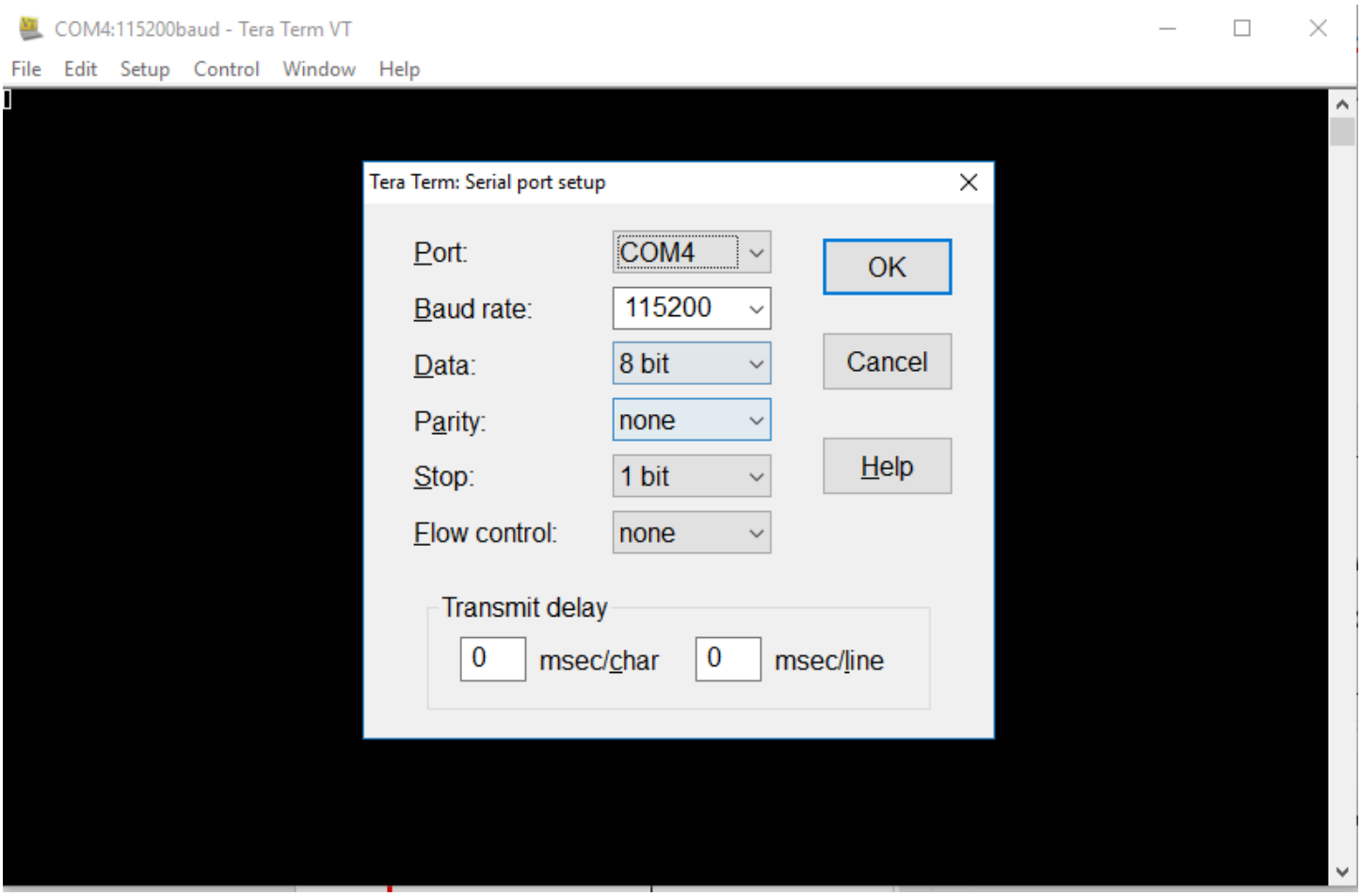
      iv. 在建立程式碼簽署設定檔視窗中，輸入設定檔名稱。針對裝置硬體平台，選取 Windows Simulator。針對程式碼簽署憑證，選擇匯入。

      v. 瀏覽以選取憑證 (secp256r1.crt)、憑證私有金鑰 (secp256r1.key) 和憑證鏈 (ca.crt)。

      vi. 在裝置上輸入程式碼簽署憑證的路徑名稱。然後選擇 Create (建立)。

6. 若要授予 程式碼簽署的存取權 AWS IoT，請遵循中的步驟授予 程式碼簽署的存取權 AWS IoT。

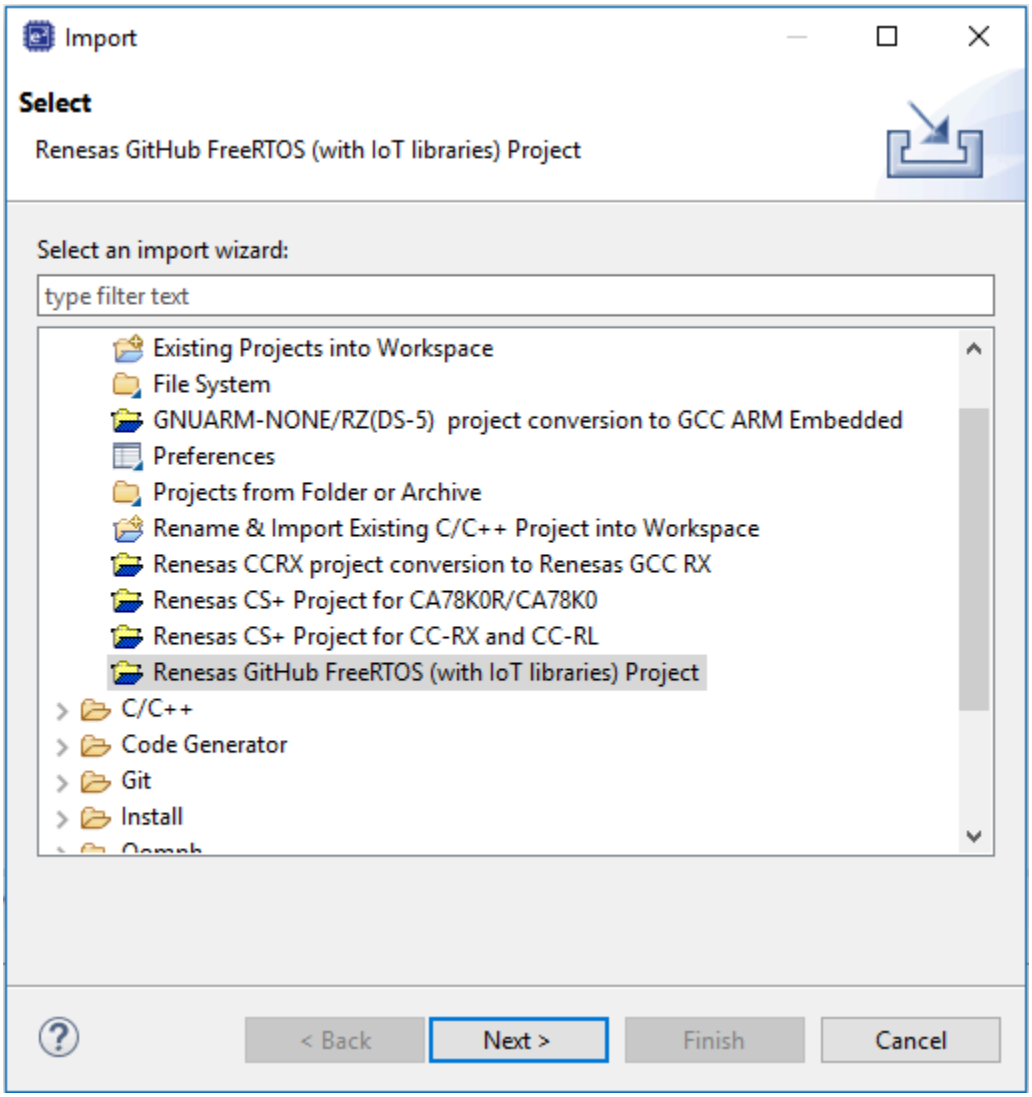
如果您的 PC 上未安裝 Tera Term，您可以從 https://ttssh2.osdn.jp/index.html.en：// 下載，並如此處所示進行設定。請確定您將 USB 序列連接埠從裝置插入 PC。



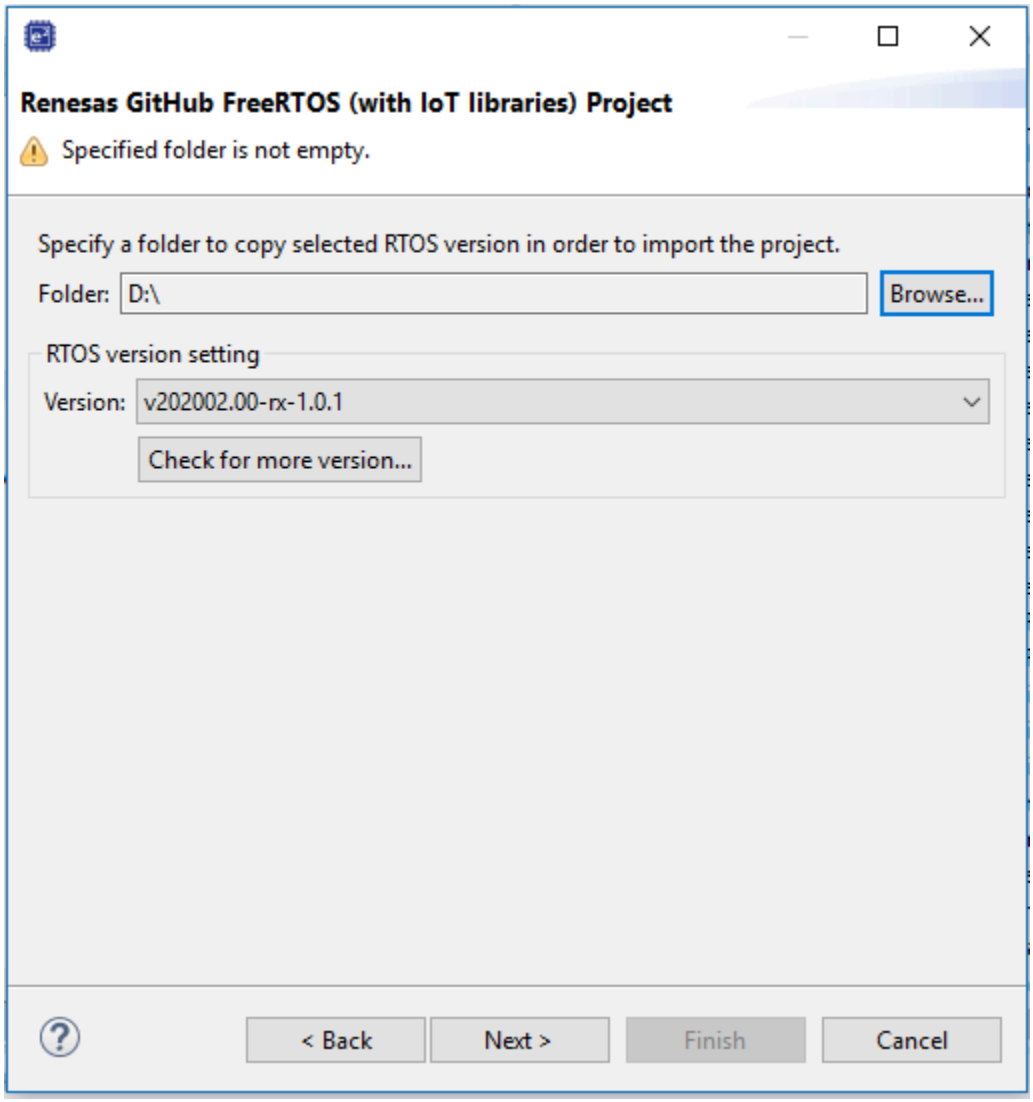
匯入、設定標頭檔案並建置 aws_demos 和 boot_loader

若要開始，請選取最新版本的 FreeRTOS 套件，這會從 GitHub 下載並自動匯入專案。如此一來，您就可以專注於設定 FreeRTOS 和撰寫應用程式程式碼。

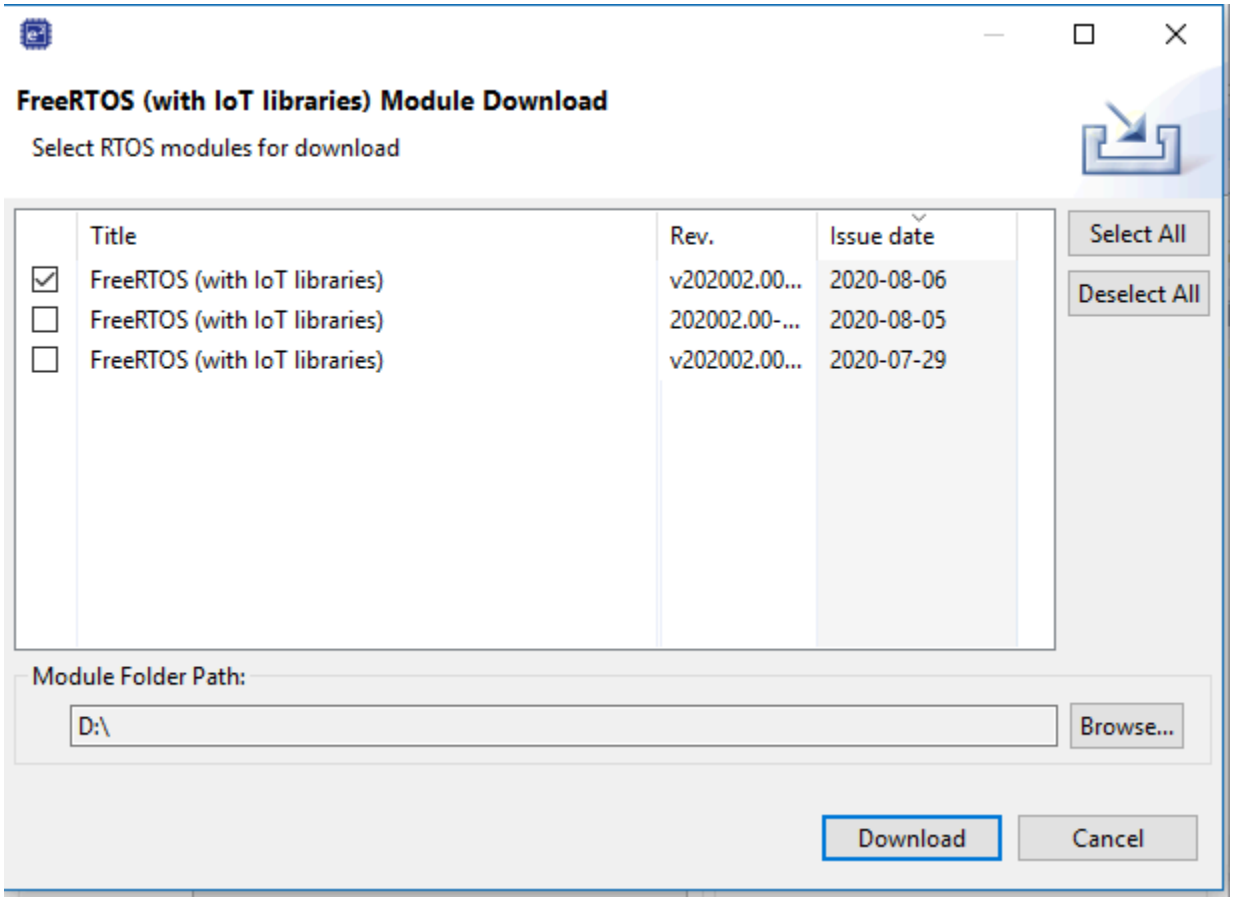
1. 啟動 e^2 Studio。
2. 選擇檔案，然後選擇匯入...。
3. 選取 Renesas GitHub FreeRTOS（含 IoT 程式庫）專案。



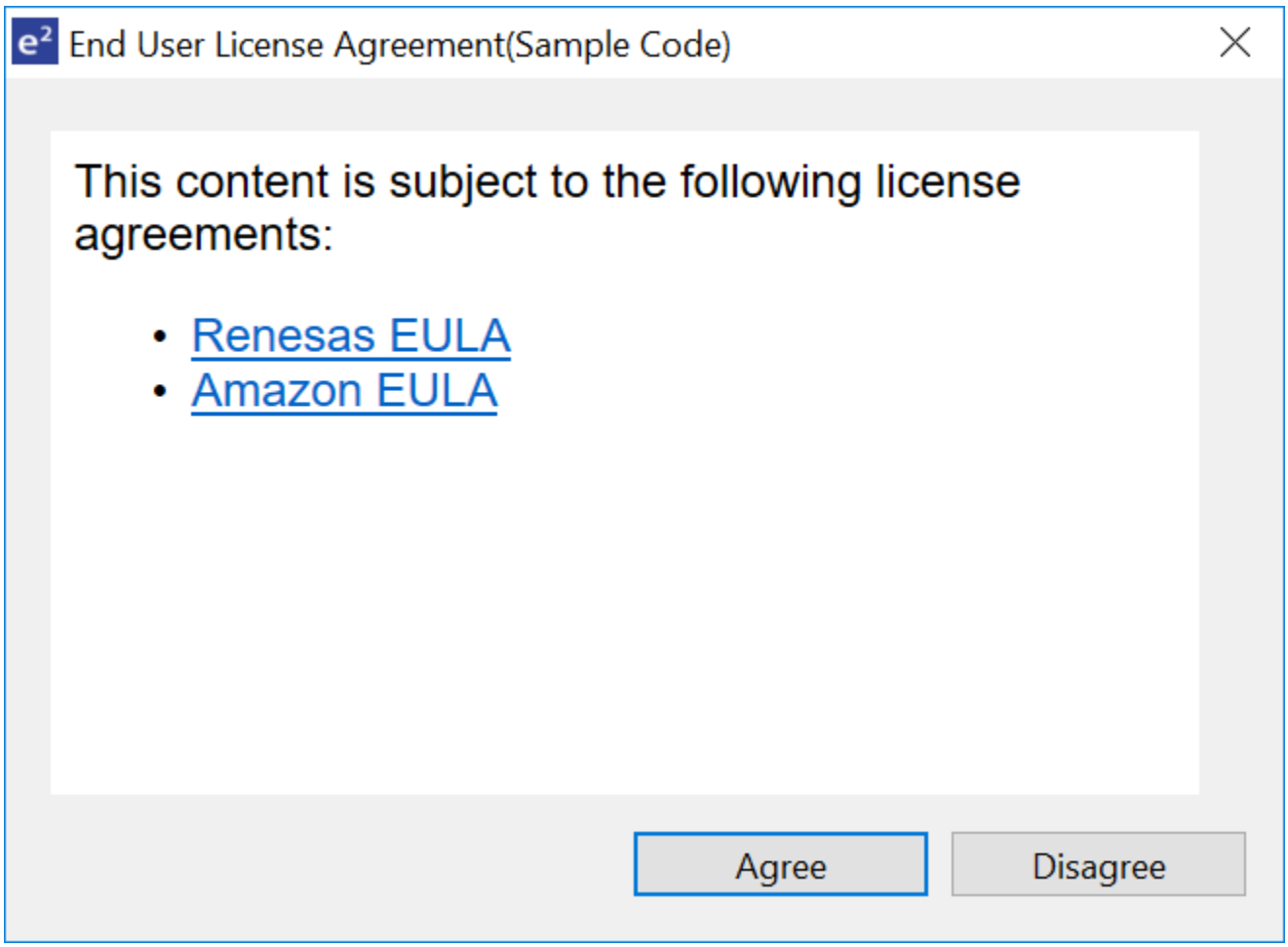
4. 選擇檢查更多版本... 以顯示下載對話方塊。


Renesas GitHub FreeRTOS (with IoT libraries) Project
Specified folder is not empty.
Specify a folder to copy selected RTOS version in order to import the project.
Folder: D:\
Browse...
RTOS version setting
Version: v202002.00-rx-1.0.1
Check for more version...
< Back
Next >
Finish
Cancel

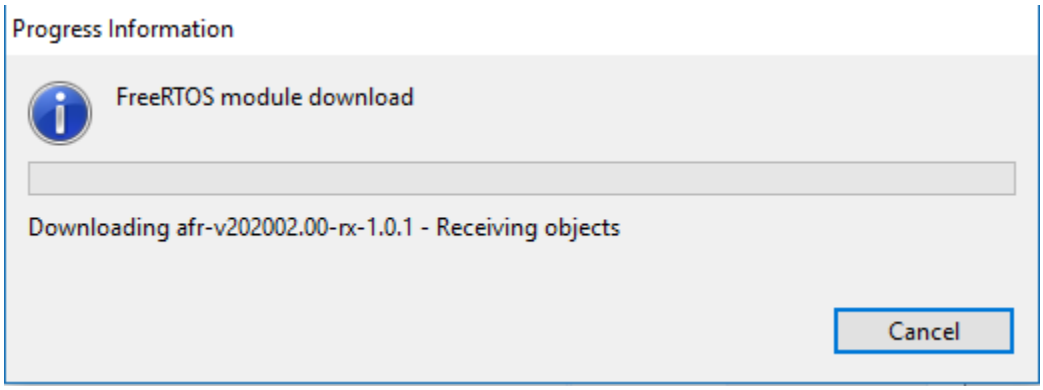
5. 選取最後一個套件。



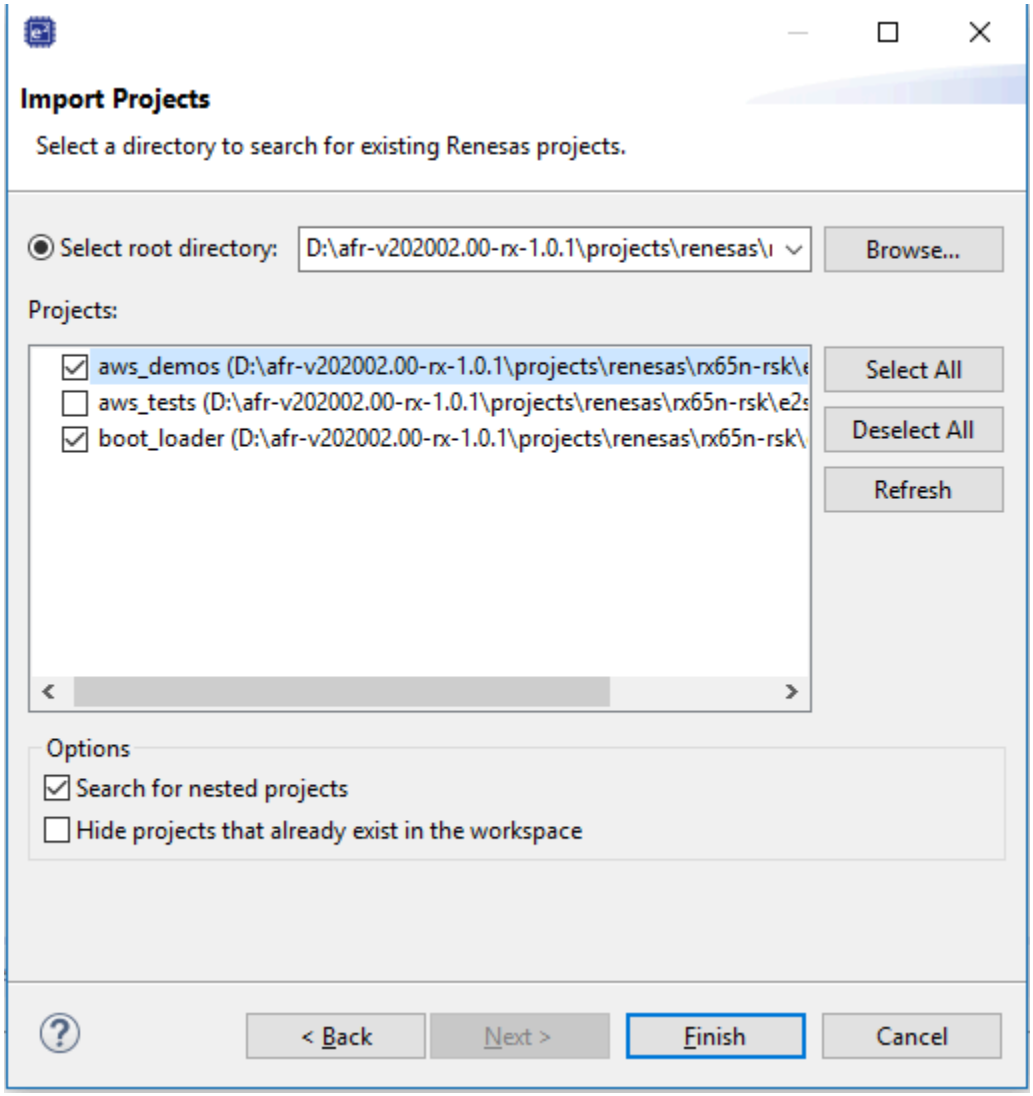
6. 選擇同意接受最終使用者授權合約。



7. 等待下載完成。



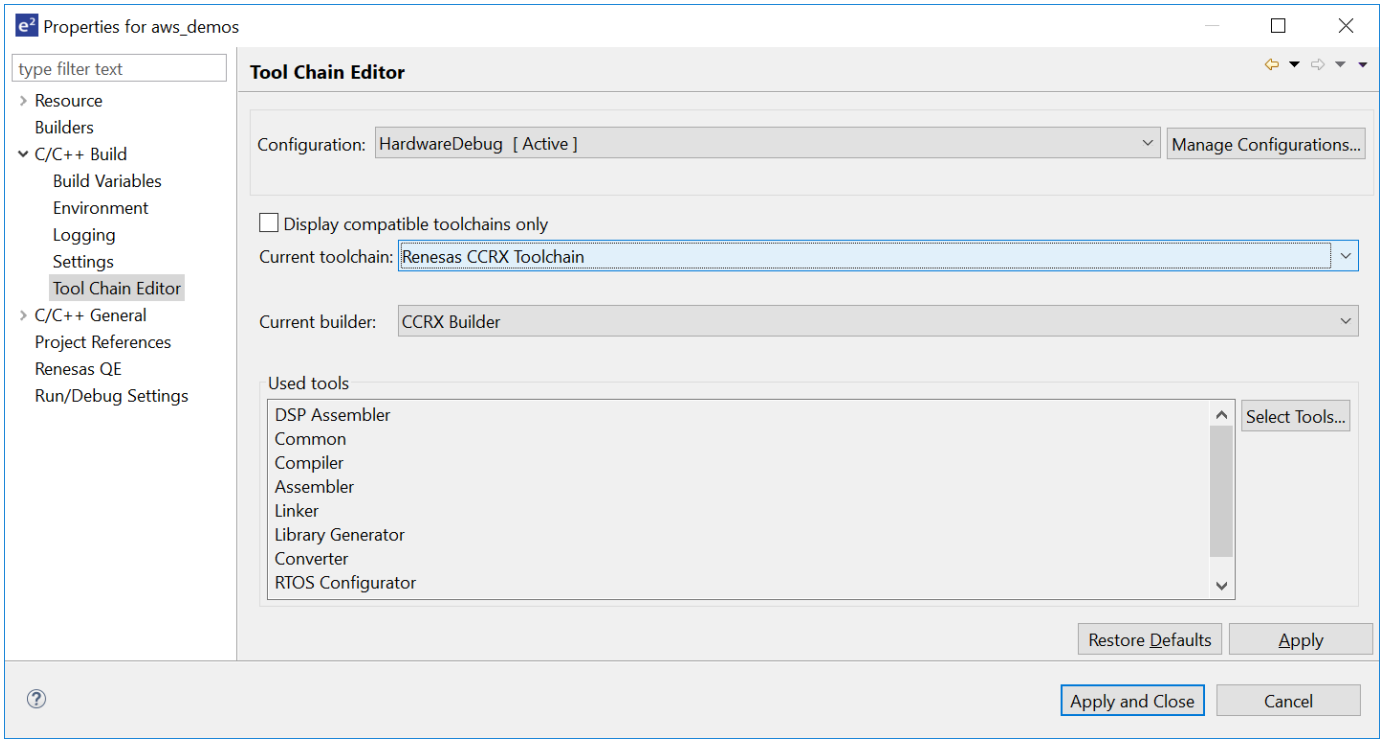
8. 選取 aws_demos 和 boot_loader 專案，然後選擇完成以匯入專案。



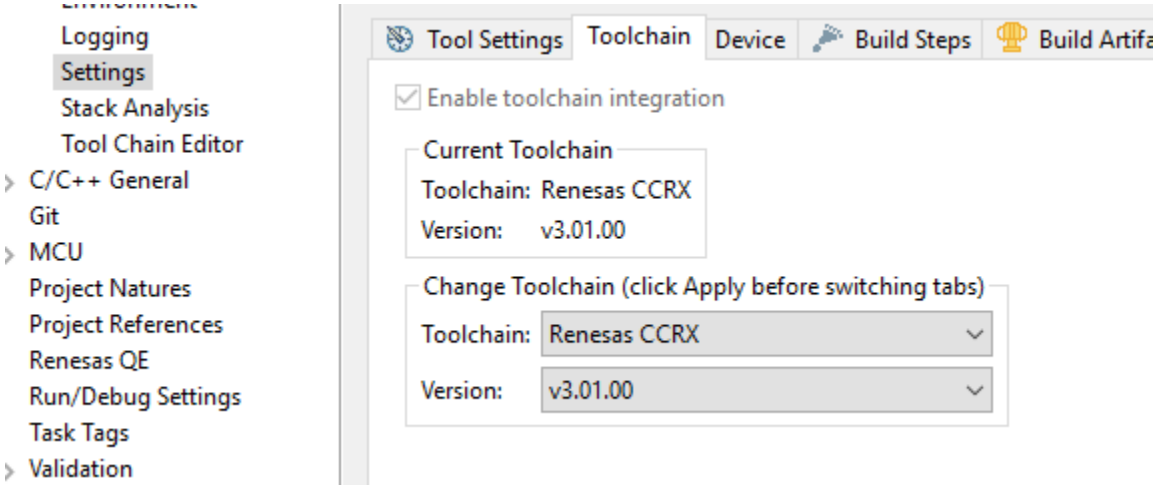
9. 對於這兩個專案，開啟專案屬性。在導覽窗格中，選擇工具鏈編輯器。

    a. 選擇目前的工具鏈。

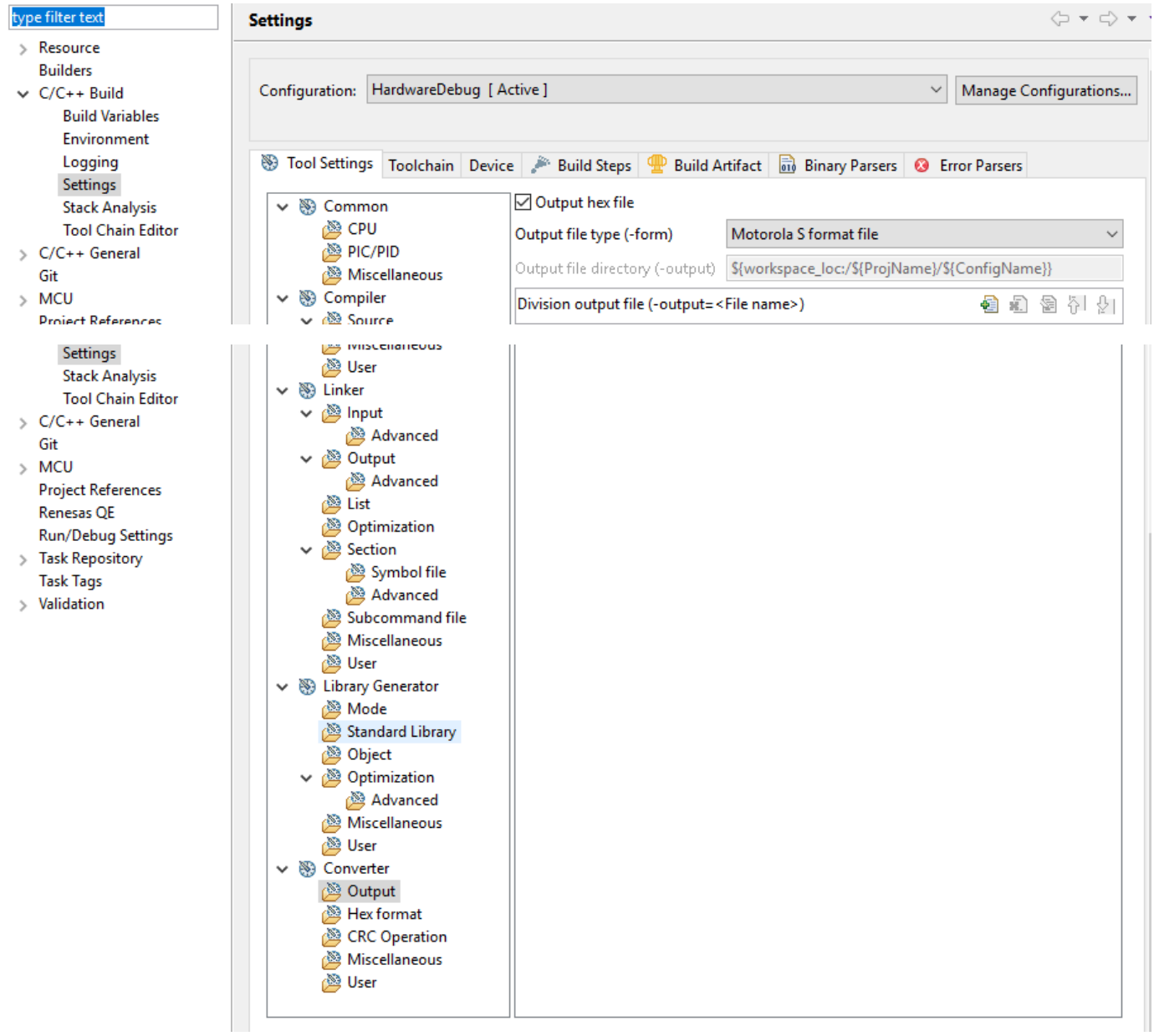
    b. 選擇目前的建置器。



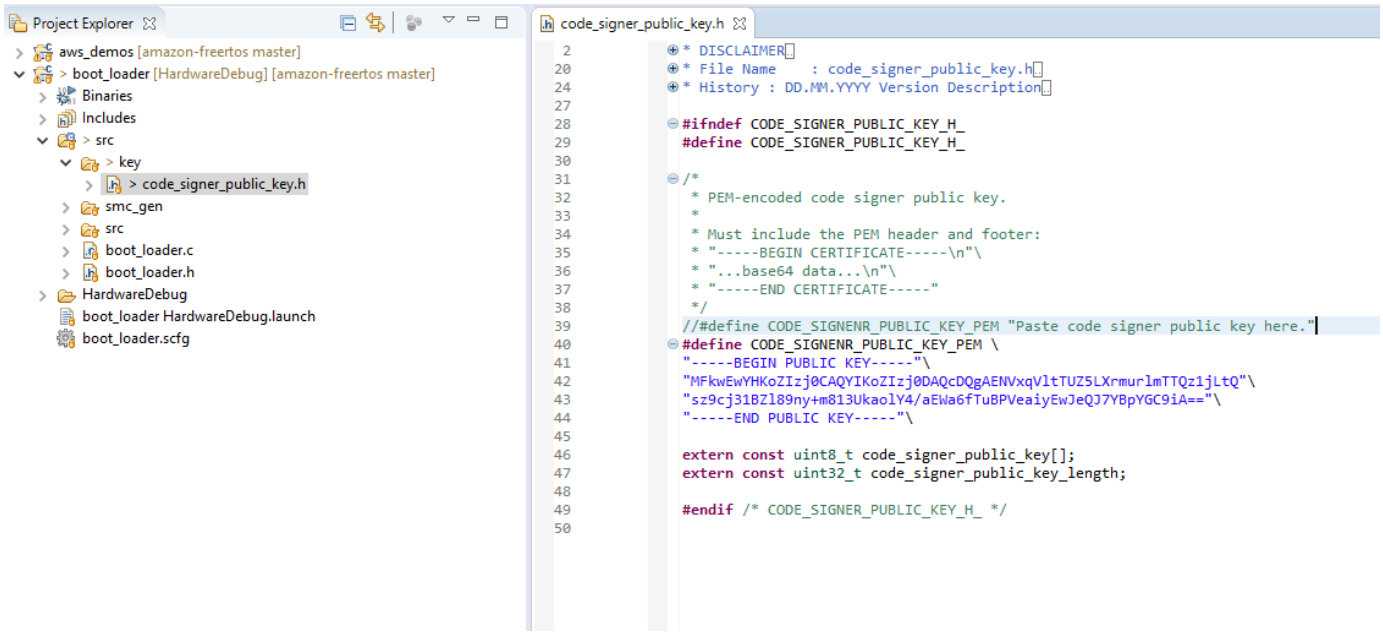
10. 在導覽窗格中，選擇設定。選擇工具鏈索引標籤，然後選擇工具鏈版本。



選擇工具設定索引標籤，展開轉換器，然後選擇輸出。在主視窗中，確定已選取輸出十六進位檔案，然後選擇輸出檔案類型。



11. 在開機載入器專案中，開啟`projects\renesas\rx65n-rsk\e2studio\boot_loader\src\key\code_signer_public_key.h`並輸入公有金鑰。如需如何建立公有金鑰的資訊，請參閱 Renesas MCU 韌體更新設計政策中的如何在 RX65N 上使用 Amazon Web Services 實作 FreeRTOS OTA，以及第 7.3 節「使用 OpenSSL 產生 ECDSA-SHA256 金鑰對」。



然後建置專案以建立 boot_loader.mot。

12. 開啟aws_demos專案。

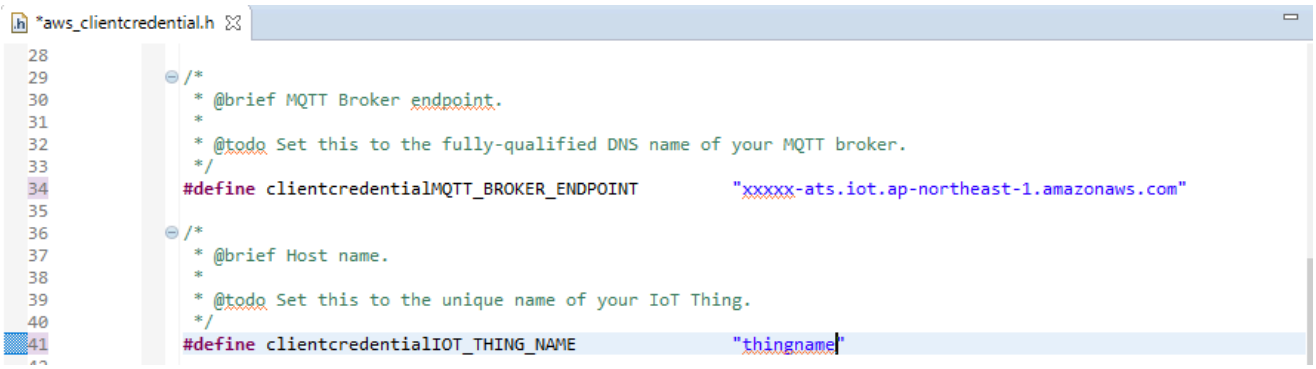
    a. 開啟 AWS IoT 主控台。

    b. 在左側的導覽窗格中，選擇設定。記下您的自訂端點。

    c. 選擇管理，然後選擇物件。記下電路板的 AWS IoT 物件名稱。

    d. 在aws_demos專案中，開啟demos/include/aws_clientcredential.h並指定下列值。

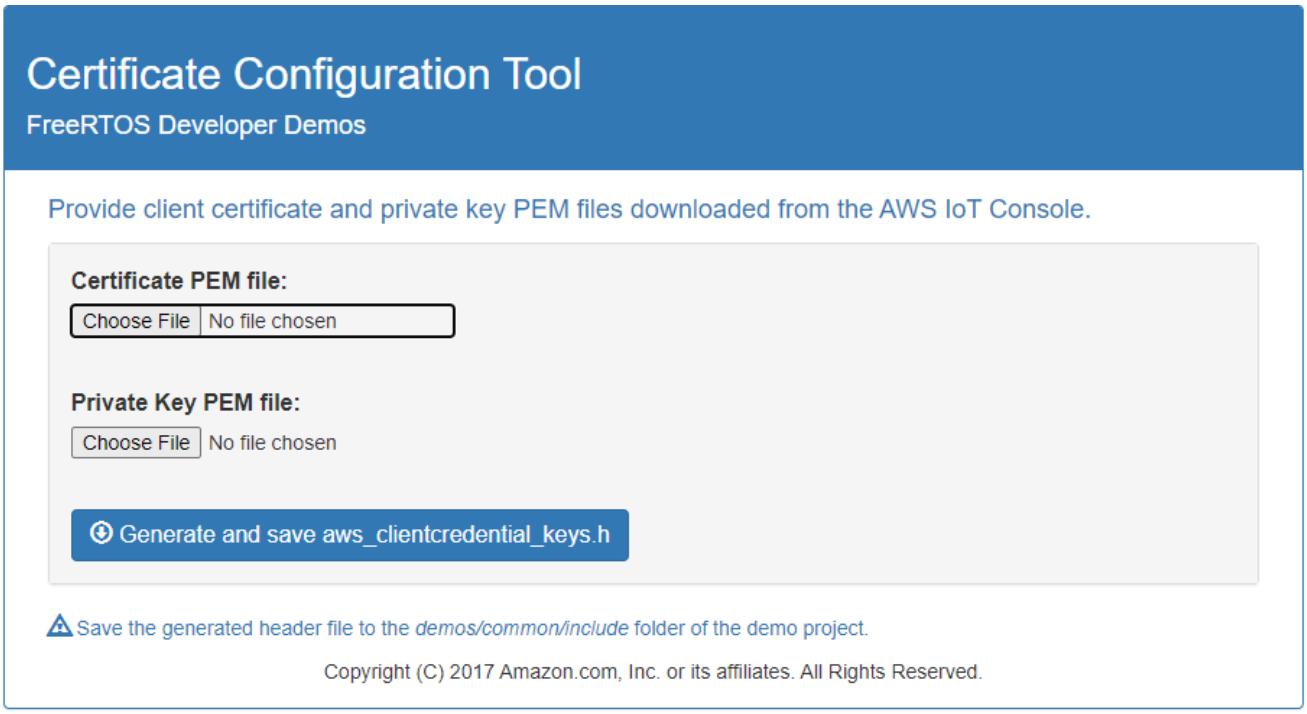
```
#define clientcredentialMQTT_BROKER_ENDPOINT[] = "Your AWS IoT endpoint";
#define clientcredentialIOT_THING_NAME "The AWS IoT thing name of your board"
```



    e. 開啟 tools/certificate_configuration/CertificateConfigurator.html 檔案。

    f. 匯入您先前下載的憑證 PEM 檔案和私密金鑰 PEM 檔案。

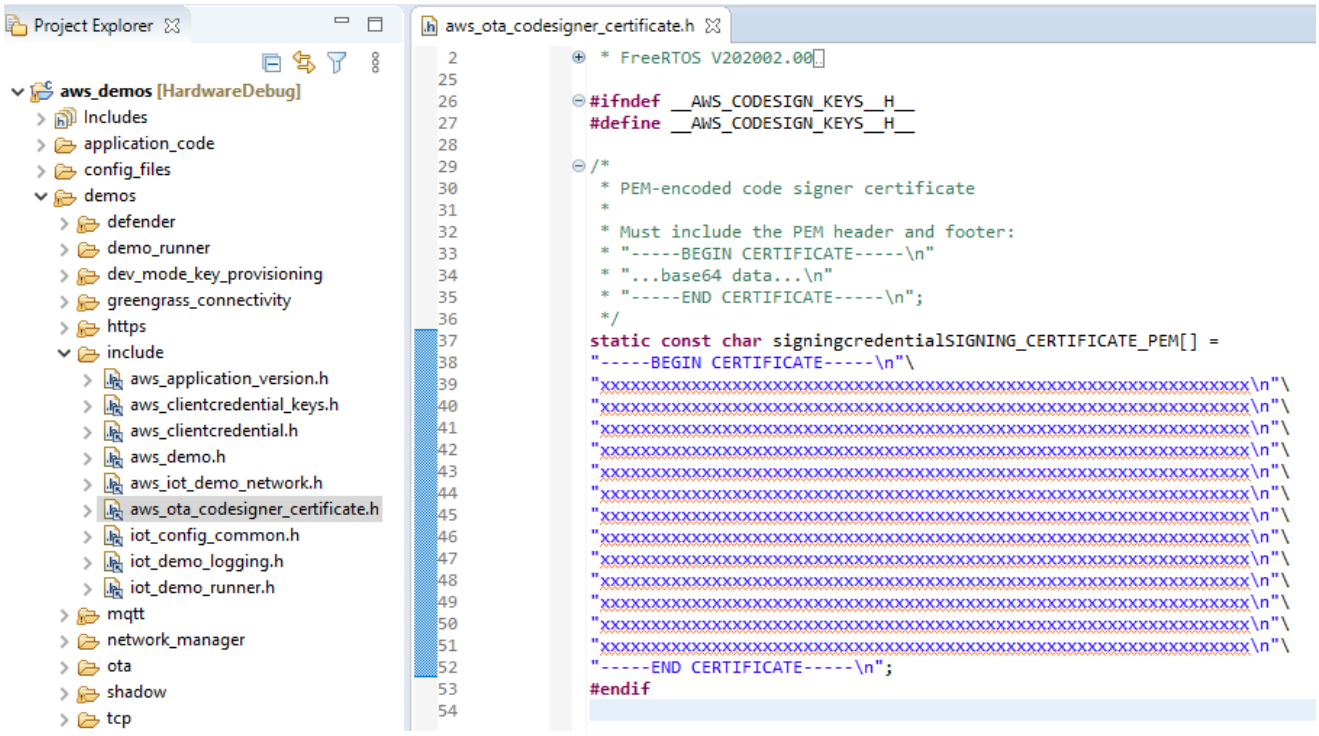
g. 選擇產生並儲存 aws_clientcredential_keys.h，並在 demos/include/目錄中取代此檔案。



h. 開啟 demos/include/aws_ota_codesigner_certificate.h 檔案，並指定這些值。

```
#define signingcredentialSIGNING_CERTIFICATE_PEM [] = "your-certificate-key";
```

其中 *your-certificate-key* 是 檔案 的值secp256r1.crt。請記得在認證中的每一行後面新增 "\"。如需建立secp256r1.crt檔案的詳細資訊，請參閱 Renesas MCU 韌體更新設計政策中的如何在 RX65N 上使用 Amazon Web Services 實作 FreeRTOS OTA，以及第 7.3 節「使用 OpenSSL 產生 ECDSA-SHA256 金鑰對」。



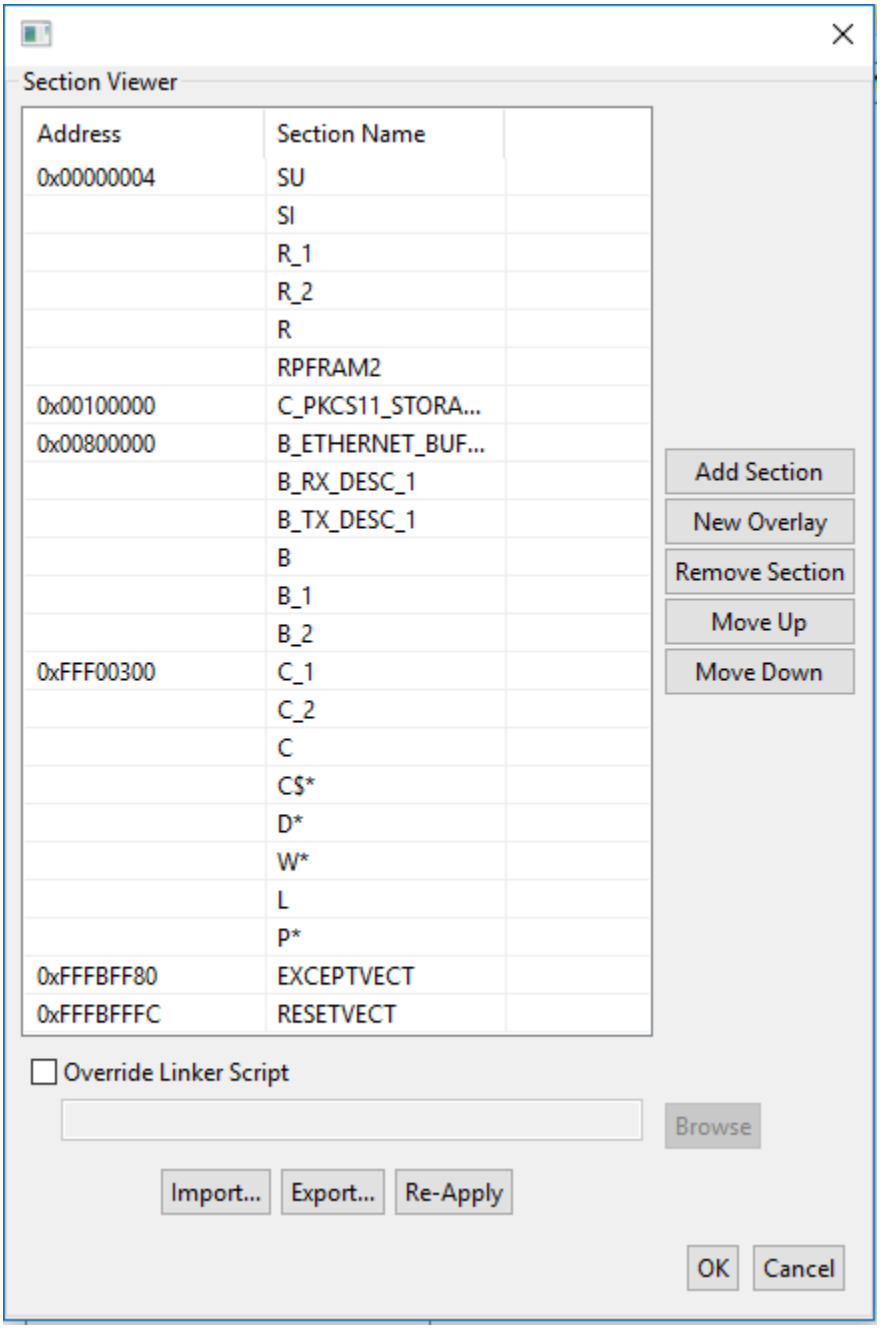
13. 任務 A：安裝韌體的初始版本

    a. 開啟 vendors/renesas/boards/board/aws_demos/config_files/ aws_demo_config.h 檔案、註解 #define CONFIG_MQTT_DEMO_ENABLED並定義 CONFIG_OTA_UPDATE_DEMO_ENABLED。

    b. 開啟 demos/include/ aws_application_version.h 檔案，並將韌體的初始版本設定為 0.9.2。

```
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
```

    c. 在區段檢視器中變更下列設定。



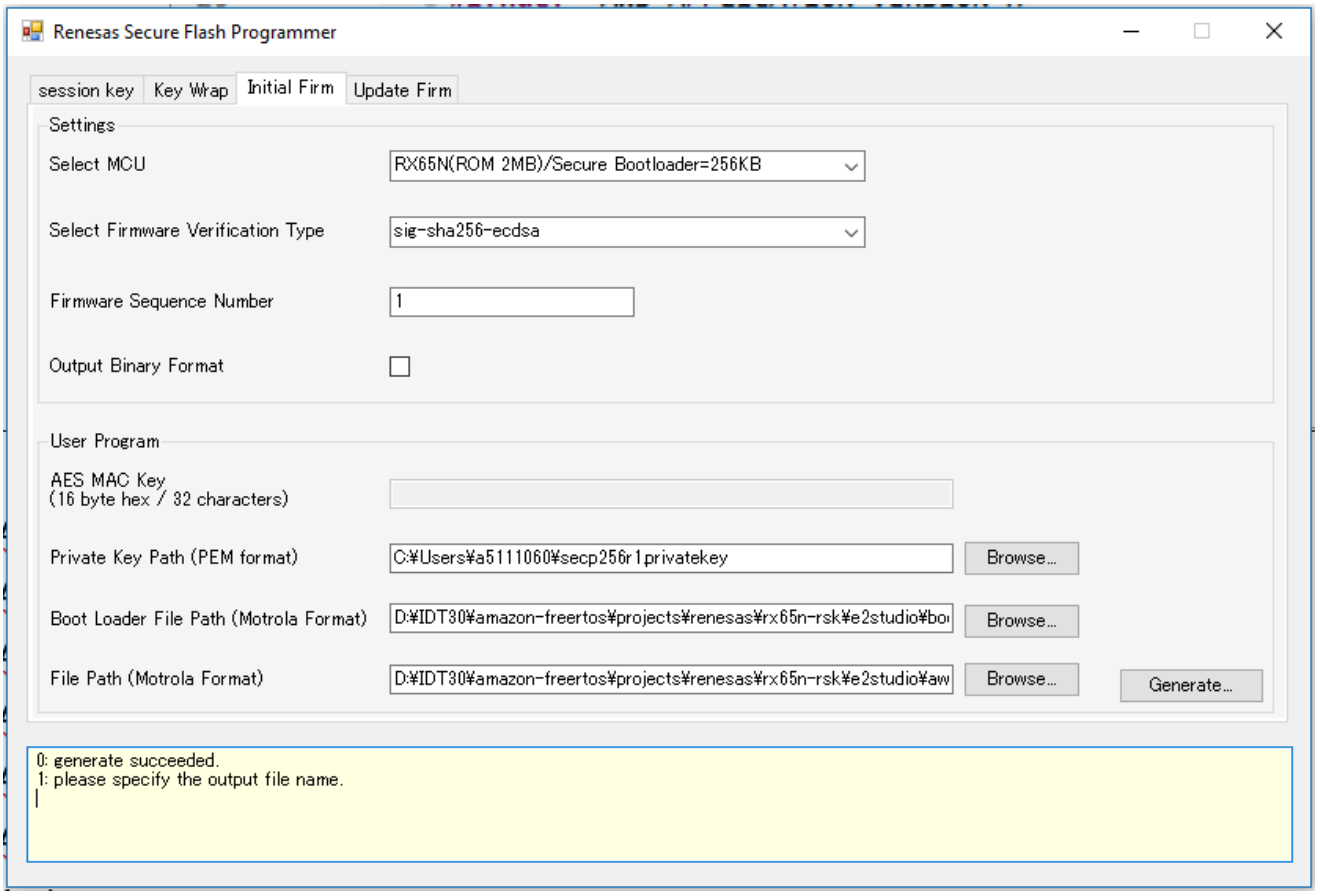
d. 選擇建置以建立aws_demos.mot檔案。

14. userprog.mot 使用 Renesas Secure Flash Programmer 建立 檔案。 userprog.mot是aws_demos.mot和 的組合boot_loader.mot。您可以將此檔案刷新至 RX65N-RSK 以安裝初始韌體。

a. 下載 https://github.com/renesas/Amazon-FreeRTOS-Tools 並開啟 Renesas Secure Flash Programmer.exe。

b. 選擇初始公司索引標籤，然後設定下列參數：

- 私密金鑰路徑 – 的位置`secp256r1.privatekey`。
- Boot Loader 檔案路徑 – `boot_loader.mot()` 的位置`projects\renesas\rx65n-rsk\e2studio\boot_loader\HardwareDebug`。
- 檔案路徑 – `aws_demos.mot()` 的位置`projects\renesas\rx65n-rsk\e2studio\aws_demos\HardwareDebug`。



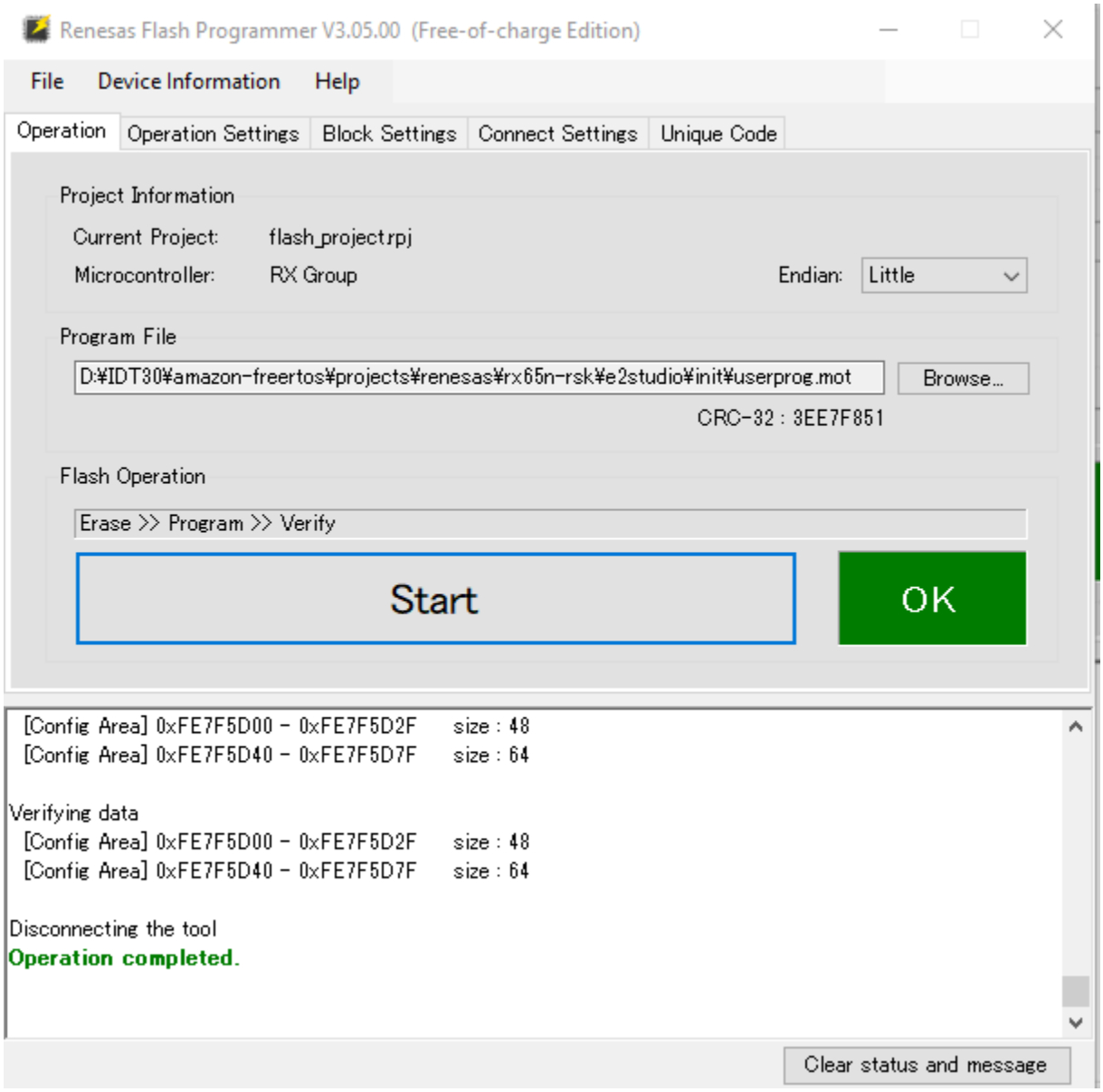
c. 建立名為 的目錄`init_firmware`、產生 `userprog.mot`，並將其儲存至 `init_firmware` 目錄。確認產生成功。

15. 快閃 RX65N-RSK 上的初始韌體。

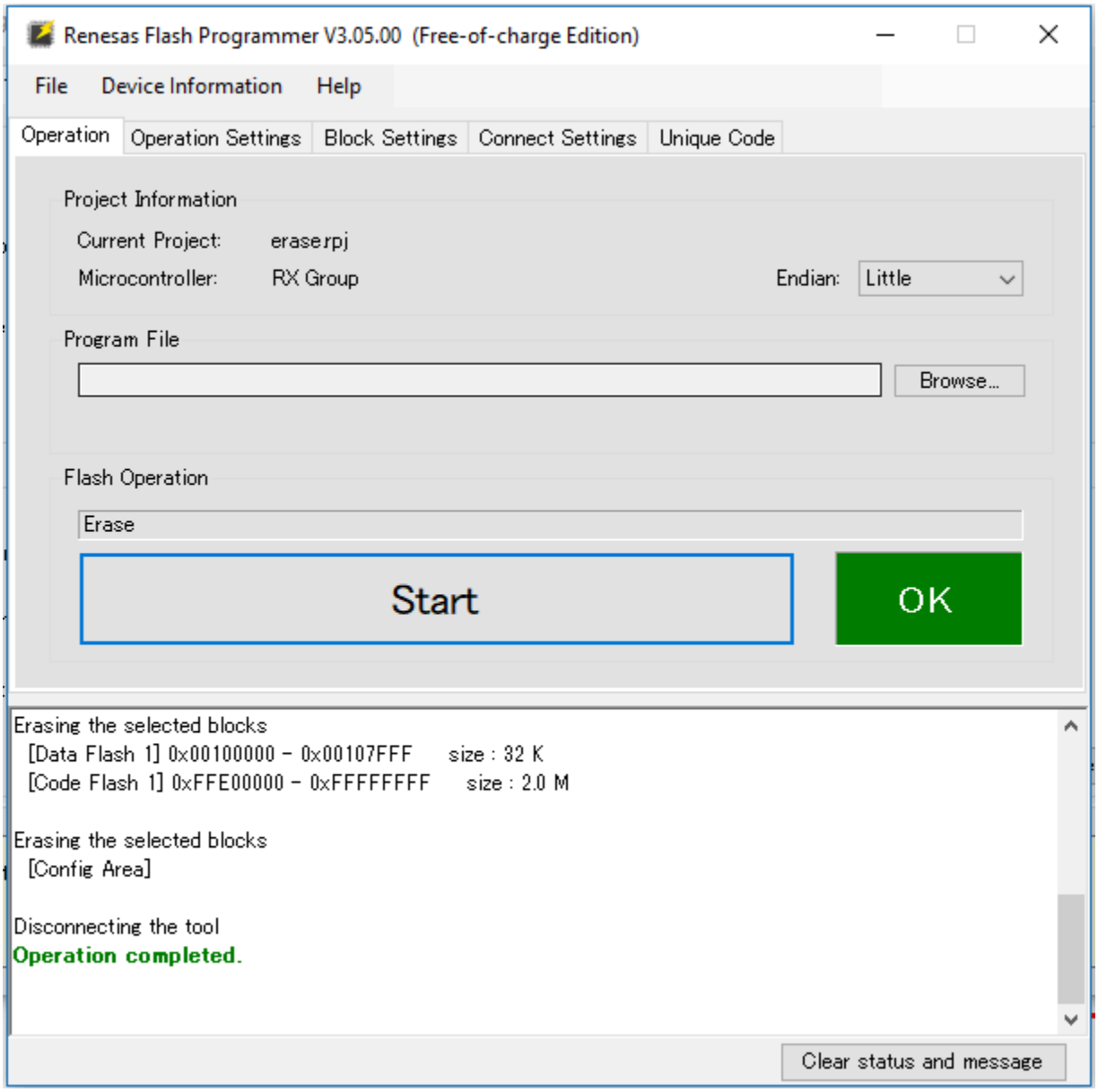
a. 從 https://www.renesas.com/tw/en/products/software-tools/tools/programmer/renesas-flash-programmer-programming-gui.html：// 下載最新版本的 Renesas Flash Programmer（程式設計 GUI)。

b. 開啟 `vendors\renesas\rx_mcu_boards\boards\rx65n-rsk\aws_demos\flash_project\erase_from_bank\ erase.rpj` 檔案以清除銀行上的資料。

c. 選擇開始以清除銀行。



d. 若要刷新 `userprog.mot`，請選擇瀏覽...，然後導覽至 `init_firmware` 目錄，選取 `userprog.mot` 檔案，然後選擇開始。



16. 0.9.2 版（初始版本）的韌體已安裝到您的 RX65N-RSK。RX65N-RSK 主機板現在正在接聽 OTA 更新。如果您在 PC 上開啟了 Tera Term，則在初始韌體執行時，您會看到類似以下內容的內容。

```
------------------------------------------------
RX65N secure boot program
------------------------------------------------
Checking flash ROM status.
bank 0 status = 0xff [LIFECYCLE_STATE_BLANK]
bank 1 status = 0xfc [LIFECYCLE_STATE_INSTALLING]
bank info = 1. (start bank = 0)
start installing user program.
```

```
copy secure boot (part1) from bank0 to bank1...OK
copy secure boot (part2) from bank0 to bank1...OK
update LIFECYCLE_STATE from [LIFECYCLE_STATE_INSTALLING] to [LIFECYCLE_STATE_VALID]
bank1(temporary area) block0 erase (to update LIFECYCLE_STATE)...OK
bank1(temporary area) block0 write (to update LIFECYCLE_STATE)...OK
swap bank...
-----------------------------------------------
RX65N secure boot program
-----------------------------------------------
Checking flash ROM status.
bank 0 status = 0xf8 [LIFECYCLE_STATE_VALID]
bank 1 status = 0xff [LIFECYCLE_STATE_BLANK]
bank info = 0. (start bank = 1)
integrity check scheme = sig-sha256-ecdsa
bank0(execute area) on code flash integrity check...OK
jump to user program
#0 1 [ETHER_RECEI] Deferred Interrupt Handler Task started
1 1 [ETHER_RECEI] Network buffers: 3 lowest 3
2 1 [ETHER_RECEI] Heap: current 234192 lowest 234192
3 1 [ETHER_RECEI] Queue space: lowest 8
4 1 [IP-task] InitializeNetwork returns OK
5 1 [IP-task] xNetworkInterfaceInitialise returns 0
6 101 [ETHER_RECEI] Heap: current 234592 lowest 233392
7 2102 [ETHER_RECEI] prvEMACHandlerTask: PHY LS now 1
8 3001 [IP-task] xNetworkInterfaceInitialise returns 1
9 3092 [ETHER_RECEI] Network buffers: 2 lowest 2
10 3092 [ETHER_RECEI] Queue space: lowest 7
11 3092 [ETHER_RECEI] Heap: current 233320 lowest 233320
12 3193 [ETHER_RECEI] Heap: current 233816 lowest 233120
13 3593 [IP-task] vDHCPProcess: offer c0a80a09ip
14 3597 [ETHER_RECEI] Heap: current 233200 lowest 233000
15 3597 [IP-task] vDHCPProcess: offer c0a80a09ip
16 3597 [IP-task] IP Address: 192.168.10.9
17 3597 [IP-task] Subnet Mask: 255.255.255.0
18 3597 [IP-task] Gateway Address: 192.168.10.1
19 3597 [IP-task] DNS Server Address: 192.168.10.1
20 3600 [Tmr Svc] The network is up and running
21 3622 [Tmr Svc] Write certificate...
22 3697 [ETHER_RECEI] Heap: current 232320 lowest 230904
23 4497 [ETHER_RECEI] Heap: current 226344 lowest 225944
24 5317 [iot_thread] [INFO ][DEMO][5317] ---------STARTING DEMO---------

25 5317 [iot_thread] [INFO ][INIT][5317] SDK successfully initialized.
```

```
26 5317 [iot_thread] [INFO ][DEMO][5317] Successfully initialized the demo. Network
 type for the demo: 4
27 5317 [iot_thread] [INFO ][MQTT][5317] MQTT library successfully initialized.
28 5317 [iot_thread] [INFO ][DEMO][5317] OTA demo version 0.9.2

29 5317 [iot_thread] [INFO ][DEMO][5317] Connecting to broker...

30 5317 [iot_thread] [INFO ][DEMO][5317] MQTT demo client identifier is rx65n-gr-
rose (length 13).
31 5325 [ETHER_RECEI] Heap: current 206944 lowest 206504
32 5325 [ETHER_RECEI] Heap: current 206440 lowest 206440
33 5325 [ETHER_RECEI] Heap: current 206240 lowest 206240
38 5334 [ETHER_RECEI] Heap: current 190288 lowest 190288
39 5334 [ETHER_RECEI] Heap: current 190088 lowest 190088
40 5361 [ETHER_RECEI] Heap: current 158512 lowest 158168
41 5363 [ETHER_RECEI] Heap: current 158032 lowest 158032
42 5364 [ETHER_RECEI] Network buffers: 1 lowest 1
43 5364 [ETHER_RECEI] Heap: current 156856 lowest 156856
44 5364 [ETHER_RECEI] Heap: current 156656 lowest 156656
46 5374 [ETHER_RECEI] Heap: current 153016 lowest 152040
47 5492 [ETHER_RECEI] Heap: current 141464 lowest 139016
48 5751 [ETHER_RECEI] Heap: current 140160 lowest 138680
49 5917 [ETHER_RECEI] Heap: current 138280 lowest 138168
59 7361 [iot_thread] [INFO ][MQTT][7361] Establishing new MQTT connection.
62 7428 [iot_thread] [INFO ][MQTT][7428] (MQTT connection 81cfc8, CONNECT operation
 81d0e8) Wait complete with result SUCCESS.
63 7428 [iot_thread] [INFO ][MQTT][7428] New MQTT connection 4e8c established.
64 7430 [iot_thread] [OTA_AgentInit_internal] OTA Task is Ready.
65 7430 [OTA Agent T] [prvOTAAgentTask] Called handler. Current State [Ready] Event
 [Start] New state [RequestingJob]
66 7431 [OTA Agent T] [INFO ][MQTT][7431] (MQTT connection 81cfc8) SUBSCRIBE
 operation scheduled.
67 7431 [OTA Agent T] [INFO ][MQTT][7431] (MQTT connection 81cfc8, SUBSCRIBE
 operation 818c48) Waiting for operation completion.
68 7436 [ETHER_RECEI] Heap: current 128248 lowest 127992
69 7480 [OTA Agent T] [INFO ][MQTT][7480] (MQTT connection 81cfc8, SUBSCRIBE
 operation 818c48) Wait complete with result SUCCESS.
70 7480 [OTA Agent T] [prvSubscribeToJobNotificationTopics] OK: $aws/things/rx65n-
gr-rose/jobs/$next/get/accepted
71 7481 [OTA Agent T] [INFO ][MQTT][7481] (MQTT connection 81cfc8) SUBSCRIBE
 operation scheduled.
72 7481 [OTA Agent T] [INFO ][MQTT][7481] (MQTT connection 81cfc8, SUBSCRIBE
 operation 818c48) Waiting for operation completion.
```

```
73 7530 [OTA Agent T] [INFO ][MQTT][7530] (MQTT connection 81cfc8, SUBSCRIBE
 operation 818c48) Wait complete with result SUCCESS.
74 7530 [OTA Agent T] [prvSubscribeToJobNotificationTopics] OK: $aws/things/rx65n-
gr-rose/jobs/notify-next
75 7530 [OTA Agent T] [prvRequestJob_Mqtt] Request #0
76 7532 [OTA Agent T] [INFO ][MQTT][7532] (MQTT connection 81cfc8) MQTT PUBLISH
 operation queued.
77 7532 [OTA Agent T] [INFO ][MQTT][7532] (MQTT connection 81cfc8, PUBLISH
 operation 818b80) Waiting for operation completion.
78 7552 [OTA Agent T] [INFO ][MQTT][7552] (MQTT connection 81cfc8, PUBLISH
 operation 818b80) Wait complete with result SUCCESS.
79 7552 [OTA Agent T] [prvOTAAgentTask] Called handler. Current State
 [RequestingJob] Event [RequestJobDocument] New state [WaitingForJob]
80 7552 [OTA Agent T] [prvParseJSONbyModel] Extracted parameter [ clientToken:
 0:rx65n-gr-rose ]
81 7552 [OTA Agent T] [prvParseJSONbyModel] parameter not present: execution
82 7552 [OTA Agent T] [prvParseJSONbyModel] parameter not present: jobId
83 7552 [OTA Agent T] [prvParseJSONbyModel] parameter not present: jobDocument
84 7552 [OTA Agent T] [prvParseJSONbyModel] parameter not present: afr_ota
85 7552 [OTA Agent T] [prvParseJSONbyModel] parameter not present: protocols
86 7552 [OTA Agent T] [prvParseJSONbyModel] parameter not present: files
87 7552 [OTA Agent T] [prvParseJSONbyModel] parameter not present: filepath
99 7651 [ETHER_RECEI] Heap: current 129720 lowest 127304
100 8430 [iot_thread] [INFO ][DEMO][8430] State: Ready  Received: 1   Queued: 0
 Processed: 0   Dropped: 0
101 9430 [iot_thread] [INFO ][DEMO][9430] State: WaitingForJob  Received: 1
 Queued: 0   Processed: 0   Dropped: 0
102 10430 [iot_thread] [INFO ][DEMO][10430] State: WaitingForJob  Received: 1
 Queued: 0   Processed: 0   Dropped: 0
103 11430 [iot_thread] [INFO ][DEMO][11430] State: WaitingForJob  Received: 1
 Queued: 0   Processed: 0   Dropped: 0
104 12430 [iot_thread] [INFO ][DEMO][12430] State: WaitingForJob  Received: 1
 Queued: 0   Processed: 0   Dropped: 0
105 13430 [iot_thread] [INFO ][DEMO][13430] State: WaitingForJob  Received: 1
 Queued: 0   Processed: 0   Dropped: 0
106 14430 [iot_thread] [INFO ][DEMO][14430] State: WaitingForJob  Received: 1
 Queued: 0   Processed: 0   Dropped: 0
107 15430 [iot_thread] [INFO ][DEMO][15430] State: WaitingForJob  Received: 1
 Queued: 0   Processed: 0   Dropped: 0
```

17. 任務 B：更新韌體版本

    a. 開啟 demos/include/aws_application_version.h檔案並將APP_VERSION_BUILD字符值增加到 0.9.3。

b. 重新建置專案。

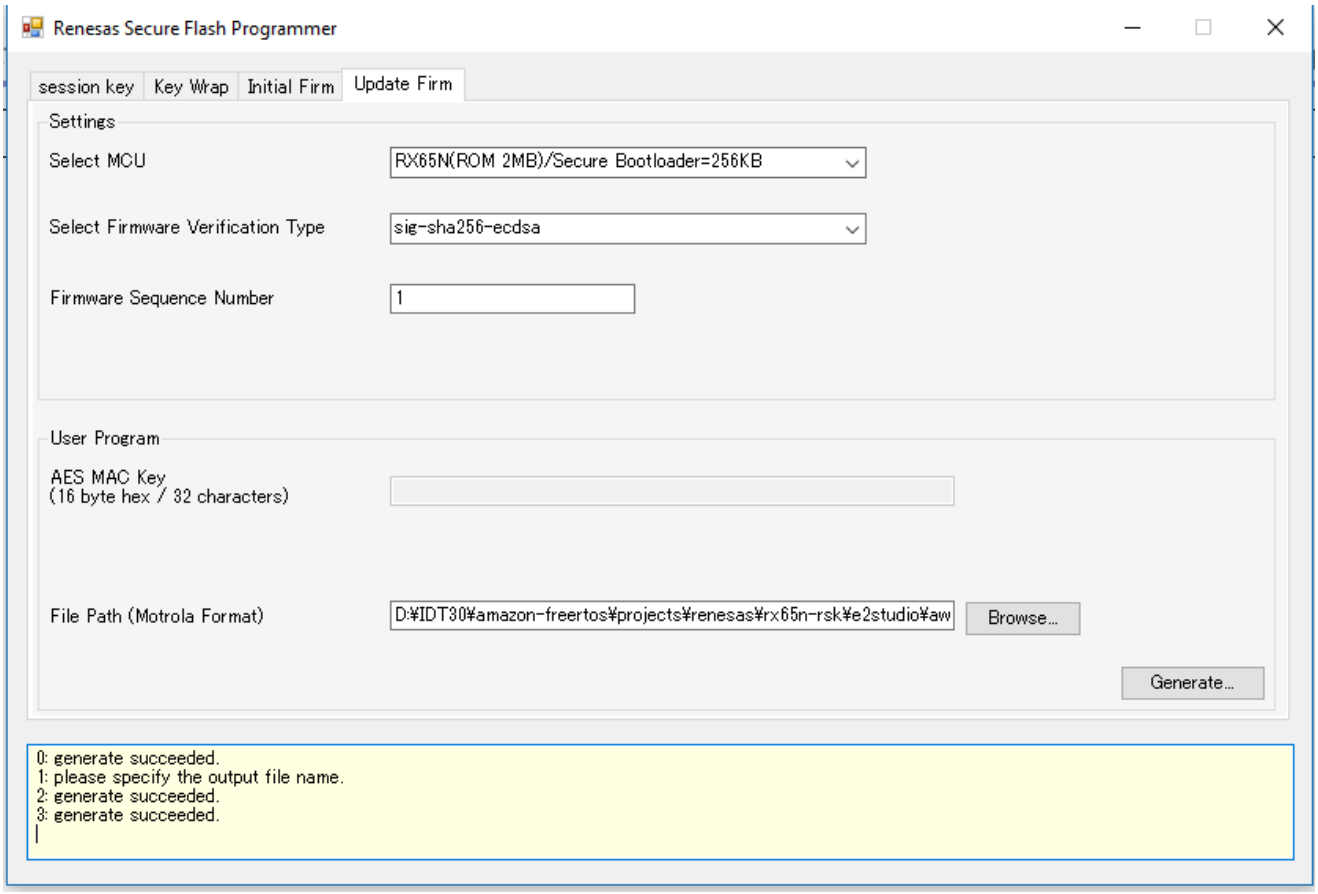
18. 使用 Renesas Secure Flash Programmer 建立 `userprog.rsu` 檔案，以更新韌體版本。

a. 開啟 `Amazon-FreeRTOS-Tools\Renesas Secure Flash Programmer.exe` 檔案。

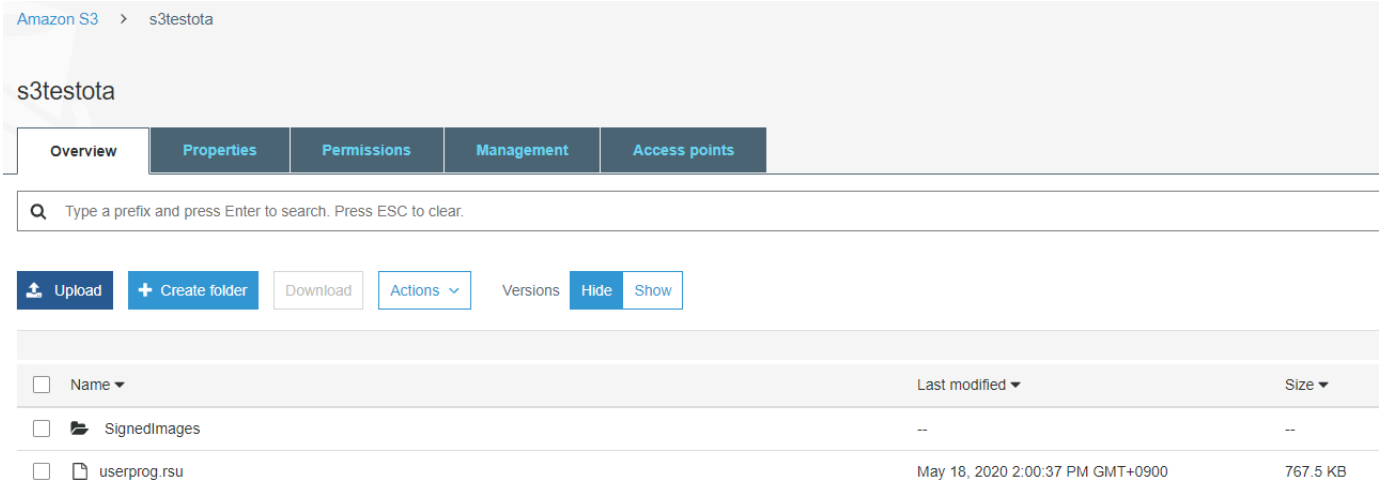
b. 選擇更新公司索引標籤，並設定下列參數：

- 檔案路徑 – `aws_demos.mot` 檔案的位置 (`projects\renesas\rx65n-rsk\e2studio\aws_demos\HardwareDebug`)。

c. 建立名為 `update _firmware` 的目錄。產生`userprog.rsu`並儲存到 `update_firmware` 目錄。確認產生成功。



19. 將韌體更新 上傳至 Amazon S3 儲存貯體`userproj.rsu`，如 中所述[建立 Amazon S3 儲存貯體以存放您的更新]。



20. 建立任務以更新 RX65N-RSK 上的韌體。

    AWS IoT 任務是一項服務，可通知一或多個連線裝置有待定的任務。任務可用來管理裝置機群、更新裝置上的韌體和安全憑證，或執行重新啟動裝置和執行診斷等管理任務。

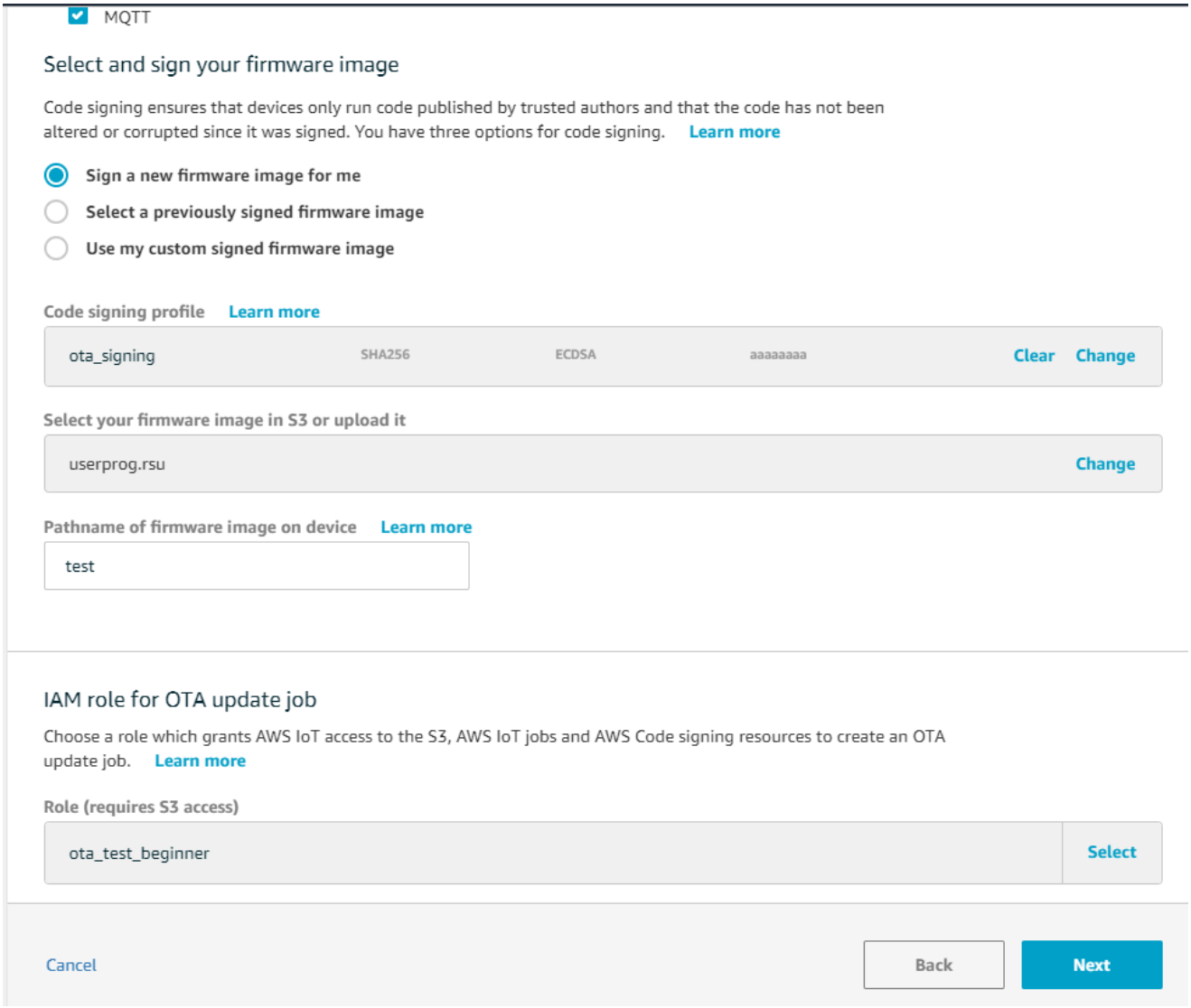
    a. 登入 AWS IoT 主控台。在導覽窗格中，選擇管理，然後選擇任務。

    b. 選擇建立任務，然後選擇建立 OTA 更新任務。選取物件，然後選擇下一步。

    c. 建立 FreeRTOS OTA 更新任務，如下所示：

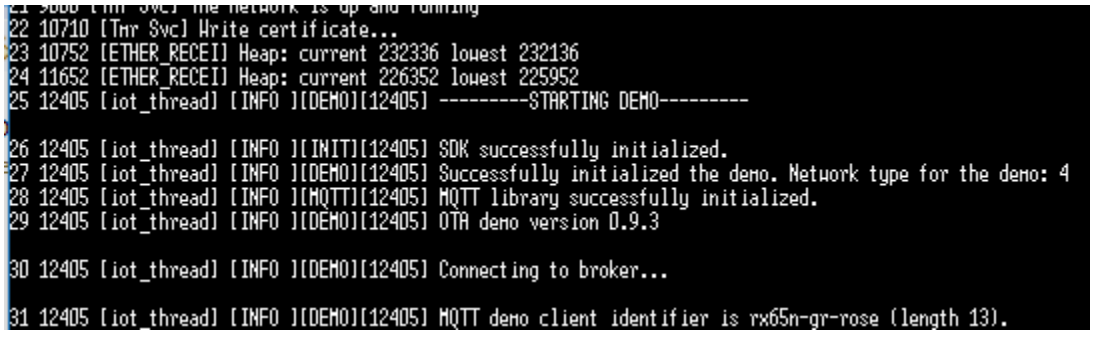
    - 選擇 MQTT。
    - 選取您在上一節中建立的程式碼簽署設定檔。
    - 選取您上傳至 Amazon S3 儲存貯體的韌體映像。
    - 針對裝置上的韌體映像路徑名稱，輸入 **`test`**。
    - 選擇您在上一節中建立的 IAM 角色。

    d. 選擇 Next (下一步)。

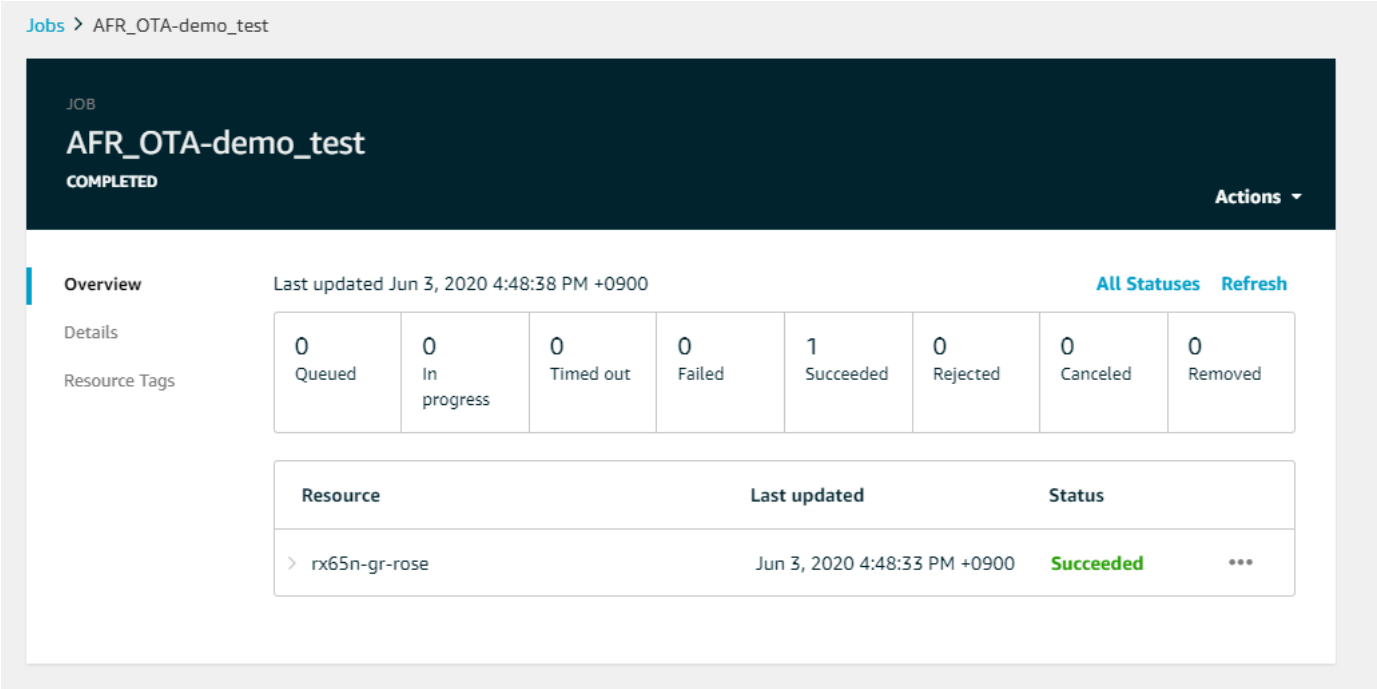


e. 輸入 ID，然後選擇建立。

21. 重新開啟 Tera Term 以確認韌體已成功更新至 OTA 示範版本 0.9.3。



22. 在 AWS IoT 主控台上，確認任務狀態是否成功。



AWS IoT Device Shadow 示範應用程式

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

簡介

此示範示範如何使用 AWS IoT Device Shadow 程式庫連線至 AWS Device Shadow 服務。它使用 coreMQTT 程式庫 建立與 MQTT 代理程式和 coreJSON 程式庫剖析器的 TLS （相互身分驗證） 的 AWS IoT MQTT 連線，以剖析從 AWS Shadow 服務收到的陰影文件。示範顯示基本陰影操作，例如如何更新陰影文件，以及如何刪除陰影文件。此示範也會示範如何向 coreMQTT 程式庫註冊回呼函數，以處理陰影等訊息，`/update`以及從 AWS IoT Device Shadow 服務傳送`/update/delta`的訊息。

此示範僅供學習練習使用，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。在逼真的生產案例中，外部應用程式會請求遠端更新裝置的狀態，即使裝置目前未連線。裝置會在連線時確認更新請求。

> **Note**
>
> 若要設定和執行 FreeRTOS 示範，請遵循中的步驟FreeRTOS 入門。

功能

示範會建立單一應用程式任務，逐一查看一組範例，示範陰影/update和回/update/delta呼，以模擬切換遠端裝置的狀態。它會傳送具有新desired狀態的陰影更新，並等待裝置變更其reported狀態以回應新desired狀態。此外，陰影回/update呼會用來列印變更的陰影狀態。此示範也會使用安全 MQTT 連線至 AWS IoT MQTT 代理程式，並假設裝置影子中有一個power0n狀態。

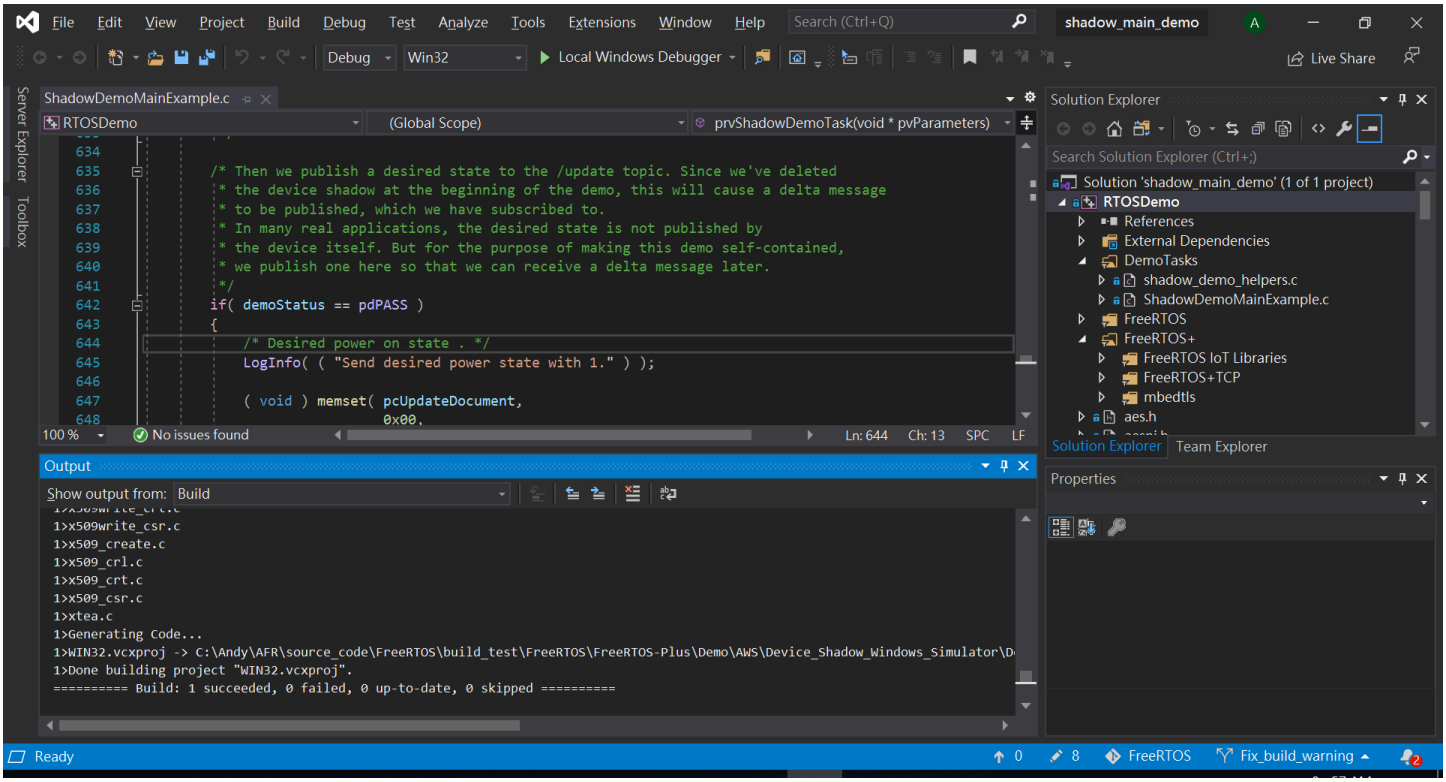
示範會執行下列操作：

1. 使用 中的協助程式函數建立 MQTT 連線shadow_demo_helpers.c。
2. 使用 AWS IoT Device Shadow 程式庫定義的巨集，為裝置影子操作組合 MQTT 主題字串。
3. 發佈至用於刪除裝置影子的 MQTT 主題，以刪除任何現有的裝置影子。
4. 訂閱 的 MQTT 主題/update/delta，/update/accepted並在 /update/rejected中使用協助程式函數shadow_demo_helpers.c。
5. 在 中使用power0n協助程式函數發佈所需的狀態shadow_demo_helpers.c。這會導致/update/delta訊息傳送到裝置。
6. 在 中處理傳入 MQTT 訊息prvEventCallback，並使用 Device Shadow 程式庫 () 定義的函數，判斷訊息是否與 AWS IoT 裝置影子相關Shadow_MatchTopic。如果訊息是裝置影子/update/delta訊息，則主要示範函數會發佈第二則訊息，將報告狀態更新為 power0n。如果收到/update/accepted訊息，請確認訊息與先前在更新訊息中發佈的訊息clientToken相同。這將標記示範的結尾。

```
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:$aws/things/testClient16:34:41/shadow/update/delta.87 9296 [Shad
owDemo]
88 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateDeltaHandler:256] 89 9296 [ShadowDemo] /update/delta json payload:{"version":1,"timestamp":1602751002,"state":{"powerOn":1}
,"metadata":{"powerOn":{"timestamp":1602751002}},"clientToken":"009136"}.90 9296 [ShadowDemo]
91 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateDeltaHandler:298] 92 9296 [ShadowDemo] version: 193 9296 [ShadowDemo]
94 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateDeltaHandler:308] 95 9296 [ShadowDemo] version:1, ulCurrentVersion:0
96 9296 [ShadowDemo]
97 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateDeltaHandler:342] 98 9296 [ShadowDemo] The new power on state newState:1, ulCurrentPowerOnState:0
99 9296 [ShadowDemo]
100 9296 [ShadowDemo] [INFO] [SHADOW] [prvEventCallback:482] 101 9296 [ShadowDemo] pPublishInfo->pTopicName:$aws/things/testClient16:34:41/shadow/update/accepted.102 9296
 [ShadowDemo]
103 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:376] 104 9296 [ShadowDemo] /update/accepted json payload:{"state":{"desired":{"powerOn":1}},"metadata":{"d
esired":{"powerOn":{"timestamp":1602751002}}},"version":1,"timestamp":1602751002,"clientToken":"009136"}.105 9296 [ShadowDemo]
106 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:424] 107 9296 [ShadowDemo] clientToken: 009136108 9296 [ShadowDemo]
109 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:429] 110 9296 [ShadowDemo] receivedToken:9136, clientToken:0
111 9296 [ShadowDemo]
112 9296 [ShadowDemo] [WARN] [SHADOW] [prvUpdateAcceptedHandler:442] 113 9296 [ShadowDemo] The received clientToken=9136 is not identical with the one=0 we sent 114 9296
[ShadowDemo]
115 9696 [ShadowDemo] [INFO] [SHADOW] [prvShadowDemoTask:670] 116 9696 [ShadowDemo] Report to the state change: 1117 9696 [ShadowDemo]
118 9856 [ShadowDemo] [INFO] [SHADOW] [prvEventCallback:482] 119 9856 [ShadowDemo] pPublishInfo->pTopicName:$aws/things/testClient16:34:41/shadow/update/accepted.120 9856
 [ShadowDemo]
121 9856 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:376] 122 9856 [ShadowDemo] /update/accepted json payload:{"state":{"reported":{"powerOn":1}},"metadata":{"
reported":{"powerOn":{"timestamp":1602751003}}},"version":2,"timestamp":1602751003,"clientToken":"009696"}.123 9856 [ShadowDemo]
124 9856 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:424] 125 9856 [ShadowDemo] clientToken: 009696126 9856 [ShadowDemo]
127 9856 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:429] 128 9856 [ShadowDemo] receivedToken:9696, clientToken:9696
129 9856 [ShadowDemo]
130 9856 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:437] 131 9856 [ShadowDemo] Received response from the device shadow. Previously published update with clie
ntToken=9696 has been accepted. 132 9856 [ShadowDemo]
133 10256 [ShadowDemo] [INFO] [SHADOW] [prvShadowDemoTask:698] 134 10256 [ShadowDemo] Start to unsubscribe shadow topics and disconnect from MQTT.
135 10256 [ShadowDemo]
136 12036 [ShadowDemo] [INFO] [SHADOW] [prvShadowDemoTask:747] 137 12036 [ShadowDemo] Demo completed successfully.138 12036 [ShadowDemo]
139 12036 [ShadowDemo] [INFO] [SHADOW] [prvShadowDemoTask:756] 140 12036 [ShadowDemo] Deleting shadow demo task.141 12036 [ShadowDemo]
```

您可以在 檔案*freertos*/demos/device_shadow_for_aws/shadow_demo_main.c或 GitHub 中找到示範。

下列螢幕擷取畫面顯示示範成功時的預期輸出。



連線至 AWS IoT MQTT 代理程式

若要連線至 AWS IoT MQTT 代理程式，我們使用與 MQTT_Connect() 相同的方法coreMQTT 相互身分驗證示範。

刪除影子文件

若要刪除影子文件，請使用 AWS IoT Device Shadow 程式庫定義的巨集，xPublishToTopic以空白訊息呼叫 。這會使用 MQTT_Publish來發佈至 /delete 主題。下列程式碼區段顯示如何在函數 中完成此操作prvShadowDemoTask。

```
/* 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 );
```

訂閱陰影主題

訂閱 Device Shadow 主題，以接收代理 AWS IoT 程式有關陰影變更的通知。Device Shadow 主題是由 Device Shadow 程式庫中定義的巨集組合而成。下列程式碼區段顯示如何在 prvShadowDemoTask函數中完成此操作。

```
/* Then try to subscribe shadow topics. */
if( returnStatus == EXIT_SUCCESS )
{
    returnStatus = SubscribeToTopic(
                     SHADOW_TOPIC_STRING_UPDATE_DELTA( THING_NAME ),
                     SHADOW_TOPIC_LENGTH_UPDATE_DELTA( THING_NAME_LENGTH ) );
}

if( returnStatus == EXIT_SUCCESS )
{
    returnStatus = SubscribeToTopic(
                     SHADOW_TOPIC_STRING_UPDATE_ACCEPTED( THING_NAME ),
                     SHADOW_TOPIC_LENGTH_UPDATE_ACCEPTED( THING_NAME_LENGTH ) );
}
```

```
if( returnStatus == EXIT_SUCCESS )
{
    returnStatus = SubscribeToTopic(
                     SHADOW_TOPIC_STRING_UPDATE_REJECTED( THING_NAME ),
                     SHADOW_TOPIC_LENGTH_UPDATE_REJECTED( THING_NAME_LENGTH ) );
}
```

傳送陰影更新

若要傳送影子更新，示範會使用 Device Shadow 程式庫定義的巨集，xPublishToTopic呼叫 JSON 格式的訊息。這會使用 MQTT_Publish來發佈至 /delete 主題。下列程式碼區段顯示如何在 prvShadowDemoTask函數中完成此操作。

```
#define SHADOW_REPORTED_JSON    \
    "{"                         \
    "\"state\":{"               \
    "\"reported\":{"            \
    "\"powerOn\":%01d"          \
    "}"                         \
    "},"                        \
    "\"clientToken\":\"%06lu\"" \
    "}"
snprintf( pcUpdateDocument,
          SHADOW_REPORTED_JSON_LENGTH + 1,
          SHADOW_REPORTED_JSON,
           ( int ) ulCurrentPowerOnState,
           ( long unsigned ) ulClientToken );

xPublishToTopic( SHADOW_TOPIC_STRING_UPDATE( THING_NAME ),
                 SHADOW_TOPIC_LENGTH_UPDATE( THING_NAME_LENGTH ),
                 pcUpdateDocument,
                 ( SHADOW_DESIRED_JSON_LENGTH + 1 ) );
```

處理陰影差異訊息和陰影更新訊息

使用 函數向 coreMQTT Client Library 註冊的使用者回呼MQTT_Init函數，將通知我們傳入的封包事件。請參閱 GitHub 上的回呼函數 prvEventCallback。

回呼函數會確認傳入封包的類型為 MQTT_PACKET_TYPE_PUBLISH，並使用 Device Shadow Library API Shadow_MatchTopic來確認傳入訊息是陰影訊息。

如果傳入訊息是類型為 的陰影訊息ShadowMessageTypeUpdateDelta，則我們呼叫 prvUpdateDeltaHandler 來處理此訊息。處理常式prvUpdateDeltaHandler使用 coreJSON 程式庫剖析訊息以取得 powerOn 狀態的差異值，並將其與本機維護的目前裝置狀態進行比較。如果這些不同，則會更新本機裝置狀態，以反映陰影文件中powerOn狀態的新值。

如果傳入訊息是類型為 的陰影訊息ShadowMessageTypeUpdateAccepted，則我們呼叫 prvUpdateAcceptedHandler 來處理此訊息。處理常式會使用 coreJSON prvUpdateAcceptedHandler 程式庫剖析訊息，clientToken從訊息中取得 。此處理常式函數會檢查來自 JSON 訊息的用戶端字符是否符合應用程式所使用的用戶端字符。如果不相符，函數會記錄警告訊息。

Secure Sockets echo 用戶端示範

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，適用於 FreeRTOS 版本 202012.00。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

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 下載至微控制器，並建置和執行 FreeRTOS 示範專案。您可以從 GitHub 複製或下載 FreeRTOS。如需說明，請參閱 README.md 檔案。

執行示範

> **Note**
>
> 若要設定和執行 FreeRTOS 示範，請遵循中的步驟FreeRTOS 入門。
> 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。

1. 請遵循 FreeRTOS 移植指南中設定 TLS Echo Server 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)

2. 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。
3. 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

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

您可以使用 FreeRTOS 的 AWS IoT Device Tester (IDT) 來驗證 FreeRTOS 作業系統是否可在您的裝置本機運作，並可與 AWS IoT 雲端通訊。具體而言，它會驗證 FreeRTOS 程式庫的移植層界面是否正確實作。它也會使用 end-to-end測試 AWS IoT Core。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 執行的測試會在 FreeRTOS GitHub 儲存庫中定義。

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括 FreeRTOS、半導體廠商移植的 FreeRTOS 介面，以及主機板裝置驅動程式。測試的目的是驗證移植的 FreeRTOS 介面在裝置驅動程式上是否正常運作。

IDT for FreeRTOS 會產生測試報告，您可以提交至 AWS IoT ，將硬體新增至 AWS 合作夥伴裝置目錄。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主機板的主機電腦 (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。

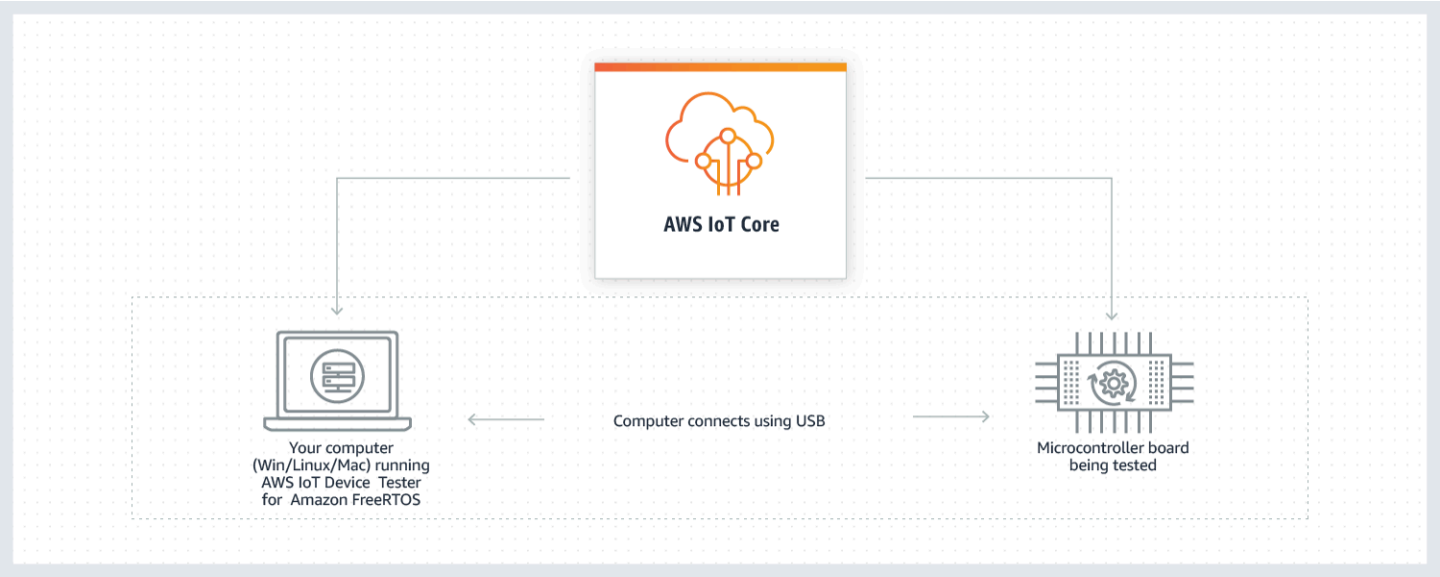
除了測試裝置之外，IDT for FreeRTOS 還會建立資源 （ 例如物件、 AWS IoT FreeRTOS 群組、Lambda 函數等 ），以促進資格程序。

為了建立這些資源，IDT for FreeRTOS 會使用 中設定的 AWS 登入`config.json`資料來代表您進行 API 呼叫。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，它會執行下列步驟：

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。



若要執行 IDT for FreeRTOS，您可以使用測試資源。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。

FreeRTOS AWS IoT Device Tester 支援的 版本 FreeRTOS

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

本主題列出支援的 IDT for FreeRTOS 版本。最佳實務是建議您使用支援 FreeRTOS 目標版本的最新版本 IDT for FreeRTOS。每個版本的 IDT for FreeRTOS 都有一或多個對應的 FreeRTOS 版本。FreeRTOS 的新版本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。

FreeRTOS AWS IoT Device Tester 的最新版本

使用以下連結下載最新版本的 IDT for FreeRTOS。

IDT v4.0.3 和測試套件 1.5.0 for FreeRTOS 202012.00（使用 FreeRTOS 202012.00 LTS 程式庫）

- IDT v4.0.3 搭配 Linux 版測試套件 FRQ_1.5.0
- 適用於 macOS 的測試套件 FRQ_1.5.0 的 IDT v4.0.3
- IDT v4.0.3 搭配測試套件 FRQ_1.5.0 for 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。此實務可能會導致當機或資料損毀。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

- 支援在硬體安全模組上使用鎖定憑證的裝置資格。
- 少量错误修复和改进。

測試套件版本

- FRQ_1.5.0
  - 發行2021.04.16。

FreeRTOS 的早期 IDT 版本

也支援下列舊版的 IDT for FreeRTOS。

IDT v4.0.1 和測試套件 1.4.1 for FreeRTOS 202012.00（使用 FreeRTOS 202012.00 LTS 程式庫）

- IDT v4.0.1 搭配 Linux 版測試套件 FRQ_1.4.1
- 適用於 macOS 的測試套件 FRQ_1.4.1 的 IDT v4.0.1
- IDT v4.0.1 搭配適用於 Windows 的測試套件 FRQ_1.4.1

版本備註

- 支援使用 FreeRTOS LTS 程式庫的 FreeRTOS 202012.00。如需 FreeRTOS 202012.00 版本中包含內容的詳細資訊，請參閱 GitHub 中的 CHANGELOG.md 檔案。
- 推出其他 OTA Over-the-air) E2E end-to-end測試案例。
- 支援執行 FreeRTOS 202012.00 且使用 FreeRTOS LTS 程式庫的開發板資格。
- 新增對使用行動連線的 FreeRTOS 開發板資格的支援。
- 修正 echo 伺服器組態中的錯誤。
- 可讓您使用 AWS IoT Device Tester for FreeRTOS 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。
- 提供程式碼簽署的 IDT 應用程式，因此您不需要在 Windows 或 macOS 下執行時授予許可。
- 精簡 BLE 測試結果剖析邏輯。

測試套件版本

- FRQ_1.4.1
  - 發行2021.01.19。

IDT v3.4.0 和測試套件版本 1.3.0 for FreeRTOS 202011.01

- IDT v3.4.0 搭配 Linux 版測試套件 FRQ_1.3.0
- IDT v3.4.0 搭配適用於 macOS 的測試套件 FRQ_1.3.0
- IDT v3.4.0 搭配測試套件 FRQ_1.3.0 for Windows

版本備註

- 支援 FreeRTOS 202011.01。如需 FreeRTOS 202011.01 版本中包含內容的詳細資訊，請參閱 GitHub 中的 CHANGELOG.md 檔案。
- 已修正 'RSA' 不是有效 PKCS11 組態選項的錯誤。
- 修正 OTA 測試後未正確清除 Amazon S3 儲存貯體的錯誤。
- 更新以支援 FullMQTT 測試群組內的新測試案例。

測試套件版本

- FRQ_1.3.0
  - 發行2020.11.05。

IDT v3.3.0 和 Test Suite 1.2.0 for FreeRTOS 202007.00

- IDT v3.3.0 搭配 Linux 版測試套件 FRQ_1.2.0
- 適用於 macOS 的測試套件 FRQ_1.2.0 的 IDT v3.3.0
- IDT v3.3.0 搭配測試套件 FRQ_1.2.0 for Windows

版本備註

- 支援 FreeRTOS 202007.00。如需 FreeRTOS 202007.00 版本中包含內容的詳細資訊，請參閱 GitHub 中的 CHANGELOG.md 檔案。
- 新的端對端測試，以驗證無線 (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。
- 已將 `--update-managed-policy` 參數新增至 `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。
- 內部改善和錯誤修正，包括：
  - 對於自動測試套件更新，改善組態檔案升級。

測試套件版本

- FRQ_1.2.0
  - 發行2020.09.17。
  - 內部改善和錯誤修正，包括：
    - 對於 OTA，如果已設定 MQTT，中斷連線並繼續測試現在會正確使用 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，請參閱適用於 FreeRTOS AWS IoT Device Tester 的 支援政策。

FreeRTOS 不支援的 IDT 版本

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

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，請參閱適用於 FreeRTOS AWS IoT Device Tester 的 支援政策。

不再支援下列版本的 IDT-FreeRTOS。

IDT v3.0.2 for FreeRTOS 202002.00

版本備註

- 支援 FreeRTOS 202002.00。如需 FreeRTOS 202002.00 版本中包含內容的詳細資訊，請參閱 GitHub 中的 CHANGELOG.md 檔案。
- 新增 IDT 內的測試套件自動更新。IDT 現在可下載 FreeRTOS 版本可用的最新測試套件。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`* 可設為 'y' 以一律下載或設為 'n' 以使用現有版本。

    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`選項列出已安裝的 IDT 版本支援的 FreeRTOS 和測試套件版本。
  - 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 – 列出已安裝 IDT 版本支援的 FreeRTOS 和測試套件版本。
- 新增`list-test-cases`命令 – 列出測試群組中可用的測試案例。
- 新增 `run-suite`命令`test-id`的選項 – 使用此選項在測試群組中執行個別測試案例。

測試套件版本

- FRQ_1.0.1

IDT v1.7.1 for FreeRTOS 202002.00

版本備註

- 支援 FreeRTOS 202002.00。如需 FreeRTOS 202002.00 版本中包含內容的詳細資訊，請參閱 GitHub 中的 CHANGELOG.md 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`AWSIoTDeviceTesterForFreeRTOSFullAccess`了具有執行所需許可的 AWS 受管政策 AWS IoT Device Tester。如果新版本需要額外許可，我們會將它們新增至此受管政策，讓您不必手動更新 IAM 許可。
- 結果目錄命名為 `AFQ_Report.xml` 的檔案現在是 `FRQ_Report.xml`。

IDT v1.6.2 for FreeRTOS 201912.00

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 開發板的資格。
- 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

版本備註

- 支援具有安全元素 （ 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

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
- 更新用於執行 IDT 的 IAM 政策。

IDT v1.3.2 for FreeRTOS 201906.00 Major

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

先決條件

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

本節說明使用 測試微控制器的先決條件 AWS IoT Device Tester。

下載 FreeRTOS

您可以使用下列命令從 GitHub 下載 FreeRTOS 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> 是對應於 中所列 IDT 版本的 FreeRTOS 版本 （ 例如 202007.00)FreeRTOS AWS IoT Device Tester 支援的 版本 FreeRTOS。這可確保您擁有完整的原始程式碼，包括子模組，並針對 FreeRTOS 版本使用正確的 IDT 版本，反之亦然。

Windows 的路徑長度限制為 260 個字元。FreeRTOS 的路徑結構非常深入，因此如果您使用的是 Windows，請將檔案路徑保持在 260 個字元的限制以下。例如，將 FreeRTOS 複製到 `C:\FreeRTOS`而非 `C:\Users\username\programs\projects\myproj\FreeRTOS\`。

LTS 資格 （ 使用 LTS 程式庫的 FreeRTOS 資格 ）

- 為了讓您的微型控制器在 AWS Partner Device Catalog 中指定為支援長期支援 (LTS) 型 FreeRTOS 版本，您必須提供資訊清單檔案。如需詳細資訊，請參閱 FreeRTOS 資格指南中的 FreeRTOS 資格檢查清單。 FreeRTOS

- 為了驗證您的微型控制器是否支援以 LTS 為基礎的 FreeRTOS 版本，並有資格提交至 AWS Partner Device Catalog，您必須使用 AWS IoT Device Tester (IDT) 搭配 FreeRTOS Qualification (FRQ) 測試套件版本 v1.4.x。
- FreeRTOS 的 LTS 型版本支援僅限於 FreeRTOS 的 202012.xx 版本。

下載 IDT for FreeRTOS

每個 FreeRTOS 版本都有對應的 IDT for FreeRTOS 版本，以執行資格測試。從 下載適當的 IDT for FreeRTOS 版本FreeRTOS AWS IoT Device Tester 支援的 版本 FreeRTOS。

將 IDT for FreeRTOS 解壓縮到檔案系統上具有讀取和寫入許可的位置。由於 Microsoft Windows 具有路徑長度的字元限制，請將 IDT for FreeRTOS 擷取至根目錄，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。

註冊 AWS 帳戶

如果您沒有 AWS 帳戶，請完成下列步驟來建立一個。

註冊 AWS 帳戶

1. 開啟 https://portal.aws.amazon.com/billing/signup。
2. 請遵循線上指示進行。

   註冊程序的一部分包括接聽電話或文字訊息，並在電話鍵盤上輸入驗證碼。

   當您註冊 時 AWS 帳戶，AWS 帳戶根使用者會建立 。根使用者有權存取該帳戶中的所有 AWS 服務 和資源。作為安全最佳實務，請將管理存取權指派給使用者，並且僅使用根使用者來執行需要根使用者存取權的任務。

AWS 會在註冊程序完成後傳送確認電子郵件給您。您可以隨時登錄 https://aws.amazon.com/ 並選擇我的帳戶，以檢視您目前的帳戶活動並管理帳戶。

建立具有管理存取權的使用者

註冊 後 AWS 帳戶，請保護 AWS 帳戶根使用者、啟用 AWS IAM Identity Center和建立管理使用者，以免將根使用者用於日常任務。

保護您的 AWS 帳戶根使用者

1. 選擇根使用者並輸入 AWS 帳戶 您的電子郵件地址，以帳戶擁有者AWS Management Console身分登入 。在下一頁中，輸入您的密碼。

   如需使用根使用者登入的說明，請參閱 AWS 登入 使用者指南中的以根使用者身分登入。

2. 若要在您的根使用者帳戶上啟用多重要素驗證 (MFA)。

   如需說明，請參閱《IAM 使用者指南》中的為您的 AWS 帳戶 根使用者 （主控台） 啟用虛擬 MFA 裝置。

建立具有管理存取權的使用者

1. 啟用 IAM Identity Center。

   如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的啟用 AWS IAM Identity Center。

2. 在 IAM Identity Center 中，將管理存取權授予使用者。

   如需使用 IAM Identity Center 目錄 做為身分來源的教學課程，請參閱AWS IAM Identity Center《 使用者指南》中的使用預設值設定使用者存取 IAM Identity Center 目錄。

以具有管理存取權的使用者身分登入

- 若要使用您的 IAM Identity Center 使用者簽署，請使用建立 IAM Identity Center 使用者時傳送至您電子郵件地址的簽署 URL。

  如需使用 IAM Identity Center 使用者登入的說明，請參閱AWS 登入 《 使用者指南》中的登入 AWS 存取入口網站。

指派存取權給其他使用者

1. 在 IAM Identity Center 中，建立一個許可集來遵循套用最低權限的最佳實務。

   如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的建立許可集。

2. 將使用者指派至群組，然後對該群組指派單一登入存取權。

   如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的新增群組。

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`

  准許取得 IDT 支援的 FreeRTOS 版本和測試套件版本清單，以便從 取得 AWS CLI。
- `iot-device-tester:LatestIdt`

  准許取得可供下載的 AWS IoT Device Tester 最新版本。
- `iot-device-tester:CheckVersion`

  准許檢查產品、測試套件和 AWS IoT Device Tester 版本的組合是否相容。
- `iot-device-tester:DownloadTestSuite`

  准許 AWS IoT Device Tester 下載測試套件。
- `iot-device-tester:SendMetrics`

  准許發佈 AWS IoT Device Tester 用量指標資料。

（選用）安裝 AWS Command Line Interface

您可能偏好使用 AWS CLI 來執行一些操作。如果您沒有 AWS CLI 安裝 ，請遵循安裝 AWS CLI 中的指示。

aws configure 從命令列執行， AWS CLI 為您要使用的 AWS 區域設定 。如需支援 IDT for FreeRTOS AWS 的區域相關資訊，請參閱AWS 區域和端點。如需詳細資訊，aws configure請參閱使用 進行快速組態aws configure。

首次準備測試微型控制器主機板

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

您可以在移植 FreeRTOS 介面時，使用 IDT for FreeRTOS 進行測試。在您為電路板的裝置驅動程式移植 FreeRTOS 介面之後，您可以使用 在微型控制器電路板上 AWS IoT Device Tester 執行資格測試。

新增程式庫移植層

若要為您的裝置移植 FreeRTOS，請遵循 FreeRTOS 移植指南中的指示。

設定您的 AWS 登入資料

您需要設定 的 AWS 登入資料 AWS IoT Device Tester ，才能與 AWS 雲端通訊。如需詳細資訊，請參閱設定 AWS 用於開發的登入資料和區域。必須在*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集區中的多個裝置。為了加速資格程序，IDT for FreeRTOS 可以平行測試具有相同規格的裝置。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。

若要為不同的測試群組啟用來源碼的平行建置，IDT for FreeRTOS 會將來源碼複製到 IDT for FreeRTOS 解壓縮資料夾內的結果資料夾。您必須使用 `testdata.sourcePath`或 `sdkPath`變數來參考建置或 Flash 命令中的原始程式碼路徑。IDT for FreeRTOS 會將此變數取代為複製來源碼的暫時路徑。如需詳細資訊，請參閱 IDT for FreeRTOS 變數。

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",
              "preProvisioned"           : "Yes | No"
            },

 ********************************************************************************************
            "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 Partner Device Catalog 中列出您的電路板，您在此處指定的 SKU 必須符合您在列出程序中使用的 SKU。

`features`

包含裝置支援功能的陣列。 AWS IoT Device Tester 會使用此資訊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。No 如果 設定為 ，則必須`Cellular`設定為 `Yes`。

`Cellular`

指出您的電路板是否具有行動功能。No 如果 設定為 ，則必須`WIFI`設定為 `Yes`。當此功能設為時`Yes`，將使用 AWS t2.micro EC2 執行個體執行 FullSecureSockets 測試，這可能會對您的帳戶產生額外費用。如需詳細資訊，請參閱 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`。需有主要佈建才能符合資格。

- 如果主機板可允許匯入私有金鑰，請使用 `Import`。IDT 將建立私有金鑰，並將其建置至 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。

範例格式：

```
3059 3013 0607 2a86 48ce 3d02 0106 082a
8648 ce3d 0301 0703 4200 04cd 6569 ceb8
1bb9 1e72 339f e8cf 60ef 0f9f b473 33ac
6f19 1813 6999 3fa0 c293 5fae 08f1 1ad0
41b7 345c e746 1046 228e 5a5f d787 d571
dcb2 4e8d 75b3 2586 e2cc 0c
```

如果您的公有金鑰是 .der 格式，您可以直接對公有金鑰進行十六進位編碼，以產生十六進位檔案。

.der 公有金鑰產生十六進位檔案的範例命令：

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

如果您的公有金鑰是 .pem 格式，您可以擷取 base64 編碼的部分，將其解碼為二進位格式，然後進行十六進位編碼以產生十六進位檔案。

例如，使用這些命令來產生 .pem 公有金鑰的十六進位檔案：

1. 取出金鑰的 base64 編碼部分（略過標頭和頁尾），並將其存放在檔案中，例如將其命名為 `base64key`，請執行此命令將其轉換為 .der 格式：

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

2. 執行 `xxd`命令，將其轉換為十六進位格式。

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

`devices.secureElementConfig.SecureElementSerialNumber`

(選用) 安全元素的序號。當您執行 FreeRTOS 示範/測試專案時，請在序號與裝置公有金鑰一起印出時提供此欄位。

`devices.secureElementConfig.preProvisioned`

（選用）如果裝置具有具有鎖定登入資料的預先佈建安全元素，且無法匯入、建立或銷毀物件，請設定為「是」。此組態只有在 `features` `KeyProvisioning`設為 "Onboard" 且 `PKCS11` 設為 "ECC" 時才會生效。

`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

若要讓 IDT for FreeRTOS 自動在您的主機板上建置和刷新測試，您必須設定 IDT 來執行硬體的建置和刷新命令。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。我們透過在 中載入用戶端和伺服器憑證和金鑰來支援 Echo Server 組態`customPath`。如需詳細資訊，請參閱 FreeRTOS 移植指南中的設定 echo 伺服器。下列 JSON 範例示範如何設定 IDT for FreeRTOS 來測試多個裝置：

```
{
```

```
   "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",
           "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:resourcetype:resource:qualifier",
            // ***********Use signCommand if you choose custom for
 signingMethod***************
            "signCommand": [
                "/absolute-path-to/sign.sh {{inputImageFilePath}}
 {{outputSignatureFilePath}}"
            ]
        }
    },
    // ***********Remove the section below if you're not configuring
 CMake***************
    "cmakeConfiguration": {
        "boardName": "board-name",
        "vendorName": "vendor-name",
        "compilerName": "compiler-name",
        "frToolchainPath": "/path/to/freertos/toolchain",
        "cmakeToolchainPath": "/path/to/cmake/toolchain"
    },
    "freertosFileConfiguration": {
        "required": [
            {
                "configName": "pkcs11Config",
                "filePath": "{{testData.sourcePath}}/vendors/vendor-name/boards/board-
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 原始程式碼根目錄的路徑。若要使用 SDK 進行平行測試，sourcePath可以使用{{userData.sdkConfiguration.path}}預留位置設定 。例如：

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

vendorPath

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" }
```

只有在在沒有 SDK 的情況下執行 時，才需要 vendorPath變數，否則可以將其移除。

> **Note**
>
> 在沒有 SDK 的情況下平行執行測試時，必須在 vendorPath、buildTool、flashTool欄位中使用{{testData.sourcePath}}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。使用 SDK 執行 時，必須在 sourcePath、 buildTool和 flashTool命令中使用{{sdkPath}}預留位置。

sdkConfiguration

如果您正在限定 FreeRTOS，且對檔案和資料夾結構所做的任何修改超過移植所需的修改，則需要在此區塊中設定 SDK 資訊。如果您不符合 SDK 內移植 FreeRTOS 的資格，則應完全省略此區塊。

sdkConfiguration.name

您搭配 FreeRTOS 使用的 SDK 名稱。如果您不是使用 SDK，則應該省略整個sdkConfiguration區塊。

sdkConfiguration.version

您搭配 FreeRTOS 使用的 SDK 版本。如果您不是使用 SDK，則應該省略整個sdkConfiguration區塊。

sdkConfiguration.path

包含 FreeRTOS 程式碼之 SDK 目錄的絕對路徑。如果您不是使用 SDK，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。組建命令中對原始程式碼路徑的所有參考都必須取代為 AWS IoT Device Tester 變數，{{testdata.sourcePath}}而 SDK 路徑的參考應該取代為 {{sdkPath}}。

buildImageInfo

testsImageName

從 *freertos-source*/tests 資料夾建置測試時，建置命令產生的檔案名稱。

demosImageName

從 *freertos-source*/demos 資料夾建置測試時，建置命令產生的檔案名稱。

flashTool

包含裝置刷新命令的刷新指令碼 (.sh 或 .bat) 完整路徑。所有對 flash 命令中原始程式碼路徑的參考都必須取代為 IDT for FreeRTOS 變數{{testdata.sourcePath}}，而所有對 SDK 路徑的參考都必須取代為 IDT for FreeRTOS 變數 {{sdkPath}}。

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。(兩個存取點可以是相同的。) 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。請確定兩個存取點與執行 IDT 的主機電腦位於相同的子網路上。

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。(兩個存取點可以是相同的。) 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。請確定兩個存取點與執行 IDT 的主機電腦位於相同的子網路上。

`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 伺服器憑證產生預留位置。此欄位為必填。

certificateGenerationMethod

指定伺服器憑證是自動產生還是手動提供。

customPath

如果 certificateGenerationMethod是「自訂」，certificatePathprivateKeyPath則為必要項目。

certificatePath

指定伺服器憑證的 filepath。

privateKeyPath

指定私有金鑰的檔案路徑。

eccCurveFormat

指定主機板支援的曲線格式。在 中PKCS11將 設為 "ecc" 時為必要device.json。有效值為「P224」、「P256」、「P384」或「P521」。

echoServerConfiguration

適用於 WiFi 和安全通訊端測試的可設定 echo 伺服器連接埠。此欄位為選用欄位。

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`

aws_demo_config.h 的完整路徑，位於 *afr-source*/vendors/vendor/boards/board/aws_demos/config_files/ 內。這些檔案包含在 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**
>
> 對於北京和寧夏區域，請使用 Custom。這些區域不支援AWS程式碼簽署。

`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 程式碼簽署方法，請針對上傳至 的信任憑證使用 Amazon Resource Name (ARN) AWS Certificate Manager。

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。

`signerCertificateFileName`

在裝置上的程式碼簽署憑證位置。

`compileSignerCertificate`

`true` 如果未佈建或刷新程式碼簽署者簽章驗證憑證，則設定為 ，因此必須將其編譯至專案。 會 AWS IoT Device Tester 擷取信任的憑證並將其編譯至 `aws_codesigner_certifiate.h`。

`untrustedSignerCertificateArn`

上傳至 ACM 之程式碼簽署憑證的 ARN。

`signerPlatform`

AWS Code Signer 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 SHA1 和 RSA 則選擇 `AmazonFreeRTOS-TI-CC3220SF`。
- 如果是 SHA256 和 ECDSA 則選擇 `AmazonFreeRTOS-Default`。

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 `signingMethod` 選擇 `Custom`，請設定 `signCommand`。

`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 `{{inputImageFilePath}}` 和 `{{outputSignatureFilePath}}` 兩個預留位置。`{{inputImageFilePath}}` 是由 IDT 建立要簽署之影像的檔案路徑。`{{outputSignatureFilePath}}` 是將由指令碼產生簽章的檔案路徑。

otaDemoConfigFilePath

aws_demo_config.h 的完整路徑，位於 *afr-source*/vendors/vendor/boards/board/aws_demos/config_files/ 內。這些檔案包含在 FreeRTOS 提供的移植程式碼範本中。

cmakeConfiguration

CMake 組態 [選用]

> **Note**
>
> 您需要提供主機板名稱、廠商名稱，以及 frToolchainPath 或 compilerName，才能執行 CMake 測試案例。cmakeToolchainPath 如果您有 CMake 工具鏈的自訂路徑，您也可以提供 。

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

CMake 工具鏈的完整路徑。此為選用欄位。

freertosFileConfiguration

IDT 在執行測試之前修改的 FreeRTOS 檔案組態。

required

本節指定您已移動其組態檔案的必要測試，例如 PKCS11、TLS 等。

`configName`

正在設定的測試名稱。

`filePath`

`freertos` 儲存庫中組態檔案的絕對路徑。使用 `{{testData.sourcePath}}`變數來定義路徑。

`optional`

本節指定您已移動其組態檔案的選用測試，例如 OTA、WiFi 等。

`configName`

正在設定的測試名稱。

`filePath`

`freertos` 儲存庫中組態檔案的絕對路徑。使用 `{{testData.sourcePath}}`變數來定義路徑。

> **Note**
>
> 您需要提供主機板名稱、廠商名稱，以及 `afrToolchainPath` 或 `compilerName`，才能執行 CMake 測試案例。如果您有 CMake 工具鏈的自訂路徑，也可以提供 `cmakeToolchainPath`。

IDT for FreeRTOS 變數

建置程式碼和刷新裝置的命令可能需要連線或裝置的其他資訊，才能成功執行。 AWS IoT Device Tester 可讓您參考刷新中的裝置資訊，並使用 JsonPath 建置命令。透過使用簡單的 JsonPath 運算式，您可以擷取 `device.json` 檔案中指定的必要資訊。

路徑變數

IDT for FreeRTOS 定義下列路徑變數，可用於命令列和組態檔案：

`{{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}}`

在組建和快閃記憶體命令中使用`userData.sdkConfiguration.path`時，會擴展到 中的值。

`{{device.connectivity.serialPort}}`

展開至序列埠。

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

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

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

本節說明如何使用 AWS IoT Device Tester for FreeRTOS 設定和執行藍牙測試。核心資格不需要進行藍牙測試。如果您不想使用 FreeRTOS 藍牙支援測試裝置，可以略過此設定，請務必將 device.json 中的 BLE 功能設為 No。

先決條件

- 請遵循中的說明進行首次準備測試微型控制器主機板
- Raspberry Pi 4B 或 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

若要測試待測裝置的 BLE 功能 (DUT)，您必須擁有 Raspberry Pi Model 4B 或 3B+。

設定您的 Raspberry Pi 以執行 BLE 測試

1. 下載其中一個自訂 Yocto 映像，其中包含執行測試所需的軟體。

   - Raspberry Pi 4B 的影像
   - Raspberry Pi 3B+ 的影像

   > **Note**
   >
   > Yocto 映像只能用於使用 AWS IoT Device Tester for FreeRTOS 進行測試，不能用於任何其他用途。

2. 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-sd.img` 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。從電腦退出 SD 卡，並將 microSD 卡插入 Raspberry Pi。

3. 設定您的 Raspberry Pi。

   a. 第一次啟動時，建議您將 Raspberry Pi 連接到螢幕、鍵盤和滑鼠。

   b. 將您的 Raspberry Pi 連接到 micro USB 電源。

   c. 使用預設的登入資料進行登入。使用者 ID 請輸入 **`root`**。密碼請輸入 **`idtafr`**。

   d. 使用乙太網路或 Wi-Fi 連線將 Raspberry Pi 連接到您的網路。

      i. 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"
```

```
        }
```

ii. 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。

e. 若為乙太網路連線，請執行 `ifconfig eth0`。若為 Wi-Fi 連線，請執行 `ifconfig wlan0`。請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。

f. (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。

i. 使用 OpenSSL `ssh-keygen` 命令建立 SSH 金鑰。如果您的主機電腦上已有 SSK 金鑰對，最佳實務是建立新的金鑰對，以允許 AWS IoT Device Tester FreeRTOS 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. 若要將 SSH 金鑰新增至 Raspberry Pi AWS IoT Device Tester ，以便 FreeRTOS 可以登入裝置，請從主機電腦使用 `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. 提示您輸入密碼時，請輸入 **`idtafr`**。此為 yocto 映像預設的密碼。

> **Note**
>
> `ssh-copy-id` 命令會假設公有金鑰名稱為 `id_rsa.pub`。在 macOS 和 Linux，預設位置是 `~/.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，則必須在 `ssh-copy-id` 中使用 `-i` 選項，以指定 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. 在您解壓縮 AWS IoT Device Tester 的目錄中，建立名為 的檔案`resource.json`。

ii. 新增有關您 Raspberry Pi 的以下資訊至檔案，再以 Raspberry Pi 的 IP 地址替換 *rasp-pi-ip-address*。

```
[
    {
        "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. 如果您未選擇對 SSH 使用公有金鑰身分驗證，請將以下內容新增至 `resource.json` 檔案的 `connectivity`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 功能新增到 `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 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

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

您可以使用 AWS IoT Device Tester for FreeRTOS 可執行檔與 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**
>
> 如果您在 Windows 上執行 IDT for FreeRTOS，請使用正斜線 (/) 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_1.0.1  \
    --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，就可以省略此選項。

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。

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

（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。如果此參數設定為 Y，則如果 IDT 偵測到有較新的版本可用，則會停止測試執行。如果此參數設定為 N，IDT 將繼續測試執行。

update-managed-policy

（選用）如果未使用此參數，且 IDT 偵測到您的受管政策不是up-to-date，系統會提示您更新受管政策。如果此參數設定為 Y，則如果 IDT 偵測到您的受管政策不是up-to-date，則會停止測試執行。如果此參數設定為 N，IDT 將繼續測試執行。

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**
>
> 如果您在 Windows 上執行 IDT for FreeRTOS，請使用正斜線 (/) 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。

> **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。

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

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

隨著新版本的 IDT for FreeRTOS 資格測試發佈，或隨著您更新電路板特定的套件或裝置驅動程式，您可以使用 IDT for FreeRTOS 來測試微控制器電路板。如需後續資格，請確定您擁有最新版本的 FreeRTOS 和 IDT for FreeRTOS，並再次執行資格測試。

AWS IoT Device Tester for FreeRTOS 測試套件版本

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

IDT for FreeRTOS 會將測試組織成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個別測試，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，請參閱適用於 FreeRTOS AWS IoT Device Tester 的 支援政策。

您可以使用 `upgrade-test-suite` 命令下載測試套件。或者，您可以在啟動 IDT 時使用選用參數 `-upgrade-test-suite` *`flag`*，其中 *`flag`* 可設為 'y' 以一律下載最新版本，或設為 'n' 以使用現有的版本。

您也可以執行 `list-supported-versions`命令來列出目前 IDT 版本支援的 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

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

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。這兩份報告都會從資格測試套件執行擷取結果。

awsiotdevicetester_report.xml 是您提交 AWS 至 以在 AWS Partner Device Catalog 中列出裝置的資格測試報告。該報告包含下列元素：

- IDT for FreeRTOS 版本。
- 已測試的 FreeRTOS 版本。
- 根據通過的測試，裝置支援的 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 (如 FullWiFi、FullMQTT 等) 進行測試的程式庫測試案例結果明細。
- FreeRTOS 的此資格是否適用於使用 LTS 程式庫的 202012.00 版。

FRQ_Report.xml 報告採用標準的 JUnit XML 格式。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目。

disabled

此屬性未使用，可忽略。

如果沒有測試案例失敗或錯誤，您的裝置符合執行 FreeRTOS 的技術需求，並且可以與 AWS IoT 服務互通。如果您選擇在 AWS 合作夥伴裝置目錄中列出您的裝置，您可以使用此報告做為資格證據。

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 SDK 執行 IDT，此區塊會包含 SDK 的名稱和版本。如果您未使用 SDK 執行 IDT，則此區塊包含：

```
<sdk>
    <name>N/A</vame>
    <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。下列程式碼片段顯示此項目在 awsiotdevicetester_report.xml 檔案中的顯示方式。

```
<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>
```

如果必要功能沒有測試失敗或錯誤，您的裝置會符合執行 FreeRTOS 的技術需求，並且可以與 AWS IoT 服務互通。如果您想要在AWS 合作夥伴裝置目錄中列出您的裝置，您可以使用此報告做為資格證據。

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">
```

格式類似於 <testsuites>標籤，但具有未使用的skipped屬性，可以忽略。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，則為 true，否則為 false。

<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，請參閱故障診斷。

檢視 日誌

您可以在 中找到 IDT for FreeRTOS 從測試執行產生的日誌*devicetester-extract-location*/results/*execution-id*/logs。該工具會產生兩組日誌：

`test_manager.log`

包含從 IDT for FreeRTOS 產生的日誌（例如，日誌相關的組態和報告產生）。

`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

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

從 IDT v4.0.0 開始，IDT for FreeRTOS 結合了標準化的組態設定和結果格式，以及測試套件環境，可讓您為裝置和裝置軟體開發自訂測試套件。您可以為自己的內部驗證新增自訂測試，或將其提供給客戶進行裝置驗證。

使用 IDT 來開發和執行自訂測試套件，如下所示：

開發自訂測試套件

- 為您要測試的裝置建立具有自訂測試邏輯的測試套件。
- 提供 IDT 您的自訂測試套件來測試執行器。包含測試套件特定設定組態的相關資訊。

執行自訂測試套件

- 設定您要測試的裝置。
- 實作您想要使用的測試套件所需的設定組態。
- 使用 IDT 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日誌。

下載最新版本的 AWS IoT Device Tester for FreeRTOS

下載最新版本的 IDT，並將軟體解壓縮到檔案系統上具有讀取和寫入許可的位置。

> **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

測試套件由三種類型的檔案組成：

- 為 IDT 提供如何執行測試套件相關資訊的 JSON 組態檔案。
- 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。

完成下列基本步驟以建立自訂 IDT 測試：

1. 為您的測試套件建立 JSON 組態檔案。
2. 建立包含測試套件測試邏輯的測試案例可執行檔。
3. 驗證並記錄測試執行器執行測試套件所需的組態資訊。
4. 確認 IDT 可以執行您的測試套件，並如預期產生測試結果。

若要快速建置範例自訂套件並進行執行，請遵循 中的指示教學課程：建置並執行範例 IDT 測試套件。

若要開始在 Python 中建立自訂測試套件，請參閱 教學課程：開發簡單的 IDT 測試套件。

教學課程：建置並執行範例 IDT 測試套件

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

AWS IoT Device Tester 下載包含範例測試套件的原始程式碼。您可以完成此教學課程來建置和執行範例測試套件，以了解如何使用 AWS IoT Device Tester FreeRTOS 來執行自訂測試套件。雖然本教學課程使用 SSH，但了解如何 AWS IoT Device Tester 搭配 FreeRTOS 裝置使用 會很有用。

在本教學課程中，您將完成下列步驟：

1. 建置範例測試套件
2. 使用 IDT 執行範例測試套件

先決條件

為了完成本教學，您需要以下項目：

- 主機電腦需求
  - 的最新版本 AWS IoT Device Tester
  - Python 3.7 或更新版本

    若要檢查安裝在電腦上的 Python 版本，請執行下列命令：

    ```
    python3 --version
    ```

    在 Windows 上，如果使用此命令傳回錯誤，請`python --version`改用 。如果傳回的版本編號為 3.7 或更新版本，請在 Powershell 終端機中執行下列命令，將 `python3`設定為`python`命令的別名。

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

    如果未傳回版本資訊或版本編號小於 3.7，請遵循下載 Python 中的指示安裝 Python 3.7+。如需詳細資訊，請參閱 Python 文件。
  - urllib3

    若要驗證 `urllib3` 是否正確安裝，請執行下列命令：

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

    如果`urllib3`未安裝 ，請執行下列命令來安裝它：

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

- 裝置要求
  - 具有 Linux 作業系統和與主機電腦相同網路連線的裝置。

    我們建議您將 Raspberry Pi 與 Raspberry Pi 作業系統搭配使用。請務必在 Raspberry Pi 上設定 SSH 以遠端連線到它。

設定 IDT 的裝置資訊

設定您的裝置資訊，讓 IDT 執行測試。您必須使用下列資訊更新位於 *<device-tester-extract-location>*/configs 資料夾的 device.json 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

您裝置的使用者定義唯一識別符。

connectivity.ip

您裝置的 IP 地址。

connectivity.port

選用。用於 SSH 連線至您裝置的連接埠號碼。

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

用於驗證的燈入資料。

connectivity.auth.credentials.user

用來登入裝置的使用者名稱。

connectivity.auth.credentials.privKeyPath

用於登入您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

用於登入您裝置的密碼。

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

<device-tester-extract-location>/samples/python 資料夾包含範例組態檔案、原始程式碼和 IDT 用戶端 SDK，您可以使用提供的建置指令碼將其合併為測試套件。下列目錄樹狀目錄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 資料夾內的 *<device-tester-extract-location>*/tests 資料夾中建立範例測試套件。檢閱 IDTSampleSuitePython_1.0.0 資料夾中的檔案，以了解範例測試套件的結構，並查看各種測試案例可執行檔和測試組態 JSON 檔案的範例。

下一步：使用 IDT 執行您建立的範例測試套件。

使用 IDT 執行範例測試套件

若要執行範例測試套件，請在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 會執行範例測試套件，並將結果串流至主控台。當測試完成執行時，您會看到下列資訊：

```
========== 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
```

故障診斷

使用以下資訊來協助解決完成教學課程的任何問題。

測試案例未成功執行

如果測試未成功執行，IDT 會將錯誤日誌串流至主控台，以協助您對測試執行進行疑難排解。請確定您符合本教學課程的所有先決條件。

無法連線至待測裝置

請確認下列內容：

- 您的 `device.json` 檔案包含正確的 IP 地址、連接埠和身分驗證資訊。
- 您可以從主機電腦透過 SSH 連線至裝置。

教學課程：開發簡單的 IDT 測試套件

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

測試套件結合了下列項目：

- 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

本教學課程說明如何使用 IDT for FreeRTOS 來開發包含單一測試案例的 Python 測試套件。雖然本教學課程使用 SSH，但了解如何 AWS IoT Device Tester 搭配 FreeRTOS 裝置使用 會很有用。

在本教學課程中，您將完成下列步驟：

1. 建立測試套件目錄
2. 建立 JSON 組態檔案
3. 建立測試案例可執行檔
4. 執行測試套件

先決條件

為了完成本教學，您需要以下項目：

- 主機電腦需求
  - 的最新版本 AWS IoT Device Tester

- Python 3.7 或更新版本

  若要檢查安裝在電腦上的 Python 版本，請執行下列命令：

  ```
  python3 --version
  ```

  在 Windows 上，如果使用此命令傳回錯誤，請python --version改用 。如果傳回的版本編號為 3.7 或更新版本，請在 Powershell 終端機中執行下列命令，將 python3設定為python命令的別名。

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

  如果未傳回版本資訊或版本編號小於 3.7，請遵循下載 Python 中的指示安裝 Python 3.7+。如需詳細資訊，請參閱 Python 文件。

- urllib3

  若要驗證 urllib3 是否正確安裝，請執行下列命令：

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

  如果urllib3未安裝 ，請執行下列命令來安裝它：

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

- 裝置要求
  - 具有 Linux 作業系統和與主機電腦相同網路連線的裝置。

    我們建議您將 Raspberry Pi 與 Raspberry Pi 作業系統搭配使用。請確定您在 Raspberry Pi 上設定 SSH 以遠端連線到它。

建立測試套件目錄

IDT 會在邏輯上將測試案例分成每個測試套件內的測試群組。每個測試案例都必須位於測試群組內。在本教學課程中，請建立名為 的資料夾，MyTestSuite_1.0.0並在此資料夾中建立下列目錄樹狀目錄：

```
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：

必要的 JSON 檔案

`suite.json`

包含測試套件的相關資訊。請參閱 設定 suite.json。

`group.json`

包含測試群組的相關資訊。您必須為測試套件中的每個測試群組建立`group.json`檔案。請參閱 設定 group.json。

`test.json`

包含測試案例的相關資訊。您必須為測試套件中的每個測試案例建立`test.json`檔案。請參閱 設定 test.json。

1. 在 `MyTestSuite_1.0.0/suite`資料夾中，建立具有下列結構`suite.json`的檔案：

```
{
    "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，建立具有下列結構`group.json`的檔案：

```
{
    "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，建立具有下列結構`test.json`的檔案：

```
{
    "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"
            ]
        }
    }
}
```

資料夾的目錄樹狀目錄現在MyTestSuite_1.0.0看起來應該如下所示：

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

取得 IDT 用戶端 SDK

您可以使用 IDT 用戶端 SDK 讓 IDT 與待測裝置互動，並報告測試結果。在本教學課程中，您將使用開發套件的 Python 版本。

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。

若要確認軟體開發套件已成功複製，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。測試套件可以包含多個測試案例可執行檔。在本教學課程中，您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，建立包含下列內容myTestCase.py的檔案：

   ```
   from idt_client import *

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

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

2. 使用用戶端 SDK 函數將下列測試邏輯新增至您的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. 將測試結果傳送至 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

設定您的裝置資訊，讓 IDT 執行測試。您必須使用下列資訊更新位於 *<device-tester-extract-location>*/configs 資料夾的 device.json 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

  您裝置的使用者定義唯一識別符。

connectivity.ip

  您裝置的 IP 地址。

connectivity.port

  選用。用於 SSH 連線至您裝置的連接埠號碼。

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

用於驗證的燈入資料。

connectivity.auth.credentials.user

用來登入裝置的使用者名稱。

connectivity.auth.credentials.privKeyPath

用於登入您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

用於登入您裝置的密碼。

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，您想要確保其如預期般運作。完成下列步驟，以使用您現有的裝置集區執行測試套件。

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。當測試完成執行時，您會看到下列資訊：

```
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
```

故障診斷

使用以下資訊來協助解決完成教學課程的任何問題。

測試案例未成功執行

如果測試未成功執行，IDT 會將錯誤日誌串流至主控台，以協助您對測試執行進行疑難排解。檢查錯誤日誌之前，請確認下列事項：

- IDT 用戶端 SDK 位於正確的資料夾中，如此步驟所述。
- 您符合本教學課程的所有先決條件。

無法連線至待測裝置

請確認下列內容：

- 您的 `device.json` 檔案包含正確的 IP 地址、連接埠和身分驗證資訊。
- 您可以從主機電腦透過 SSH 連線至裝置。

建立 IDT 測試套件組態檔案

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

本節說明您在撰寫自訂測試套件時所包含之建立 JSON 組態檔案的格式。

必要的 JSON 檔案

`suite.json`

包含測試套件的相關資訊。請參閱 設定 suite.json。

`group.json`

包含測試群組的相關資訊。您必須為測試套件中的每個測試群組建立`group.json`檔案。請參閱 設定 group.json。

`test.json`

包含測試案例的相關資訊。您必須為測試套件中的每個測試案例建立`test.json`檔案。請參閱 設定 test.json。

選用 JSON 檔案

`state_machine.json`

定義 IDT 執行測試套件時如何執行測試。請參閱 設定 state_machine.json。

`userdata_schema.json`

定義測試執行器可以包含在其設定組態中的userdata.json檔案結構描述。userdata.json 檔案會用於執行測試所需的任何其他組態資訊，但不會出現在device.json檔案中。請參閱 設定 userdata_schema.json。

JSON 組態檔案會放在您的 中*<custom-test-suite-folder>*，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

suite.json 檔案會設定環境變數，並判斷是否需要使用者資料才能執行測試套件。使用下列範本來設定您的*<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`id`必須符合 `suite.json` 檔案所在的測試套件資料夾名稱。套件名稱和套件版本也必須符合下列要求：

- *`<suite-name>`* 不能包含底線。
- *`<suite-version>`* 表示為 *`x.x.x`*，其中 x是數字。

ID 會顯示在 IDT 產生的測試報告中。

`title`

此測試套件正在測試之產品或功能的使用者定義名稱。名稱會顯示在測試執行器的 IDT CLI 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。如果您將此值設定為 `true`，您還必須在測試套件資料夾中包含 `userdata_schema.json` 檔案。

`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

`group.json` 檔案會定義測試群組為必要或選用。使用下列範本來設定您的*`<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`id`必須符合 `group.json` 檔案所在的測試群組資料夾名稱。ID 用於 IDT 產生的測試報告。

`title`

測試群組的描述性名稱。名稱會顯示在測試執行器的 IDT CLI 中。

`details`

測試群組用途的簡短描述。

`optional`

選用。設定為 `true` 以在 IDT 完成執行所需的測試後，將此測試群組顯示為選用群組。預設值為 `false`。

設定 test.json

`test.json` 檔案會決定測試案例可執行檔和測試案例所使用的環境變數。如需建立測試案例可執行檔的詳細資訊，請參閱 建立 IDT 測試案例可執行檔。

使用下列範本來設定您的*<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`id`必須符合 `test.json` 檔案所在的測試案例資料夾名稱。ID 用於 IDT 產生的測試報告。

`title`

測試案例的描述性名稱。名稱會顯示在測試執行器的 IDT CLI 中。

details

測試案例用途的簡短描述。

requireDUT

選用。true 如果需要裝置執行此測試，請將 設定為 ，否則將 設定為 false。預設值為 true。測試執行器會設定將用來在其 device.json 檔案中執行測試的裝置。

requiredResources

選用。陣列，提供執行此測試所需的資源裝置相關資訊。

requiredResources.name

執行此測試時提供給資源裝置的唯一名稱。

requiredResources.features

使用者定義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相符。

requiredResources.features.version

選用。功能的版本。此值會與 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，則不會檢查該功能。如果功能不需要版本號碼，請將此欄位保留空白。

requiredResources.features.jobSlots

選用。此功能可支援的同時測試數量。預設值為 1。如果您希望 IDT 針對個別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（以毫秒為單位）。如需設定此值的詳細資訊，請參閱 建立 IDT 測試案例可執行檔。

execution.*os*

要根據執行 IDT 之主機電腦的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

您想要在指定作業系統上執行的測試案例可執行檔路徑。此位置必須位於系統路徑中。

`execution.os.args`

選用。用來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` 檔案和 檔案中指定相同的環境變數`suite.json`，則 `test.json` 檔案中的值優先。

設定 state_machine.json

狀態機器是一種控制測試套件執行流程的建構。它決定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，並繼續轉換這些狀態，直到達到結束狀態為止。

如果您的測試套件不包含使用者定義的狀態機器，IDT 會為您產生狀態機器。預設狀態機器會執行下列函數：

- 讓測試執行器能夠選取和執行特定測試群組，而不是整個測試套件。
- 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。
- 產生報告並列印主控台摘要，顯示每個測試群組和測試案例的測試結果。

如需 IDT 狀態機器如何運作的詳細資訊，請參閱 設定 IDT 狀態機器。

設定 userdata_schema.json

`userdata_schema.json` 檔案會決定測試執行器提供使用者資料的結構描述。如果您的測試套件需要`device.json`檔案中不存在的資訊，則需要使用者資料。例如，您的測試可能需要使

用者必須提供的 Wi-Fi 網路登入資料、特定開放連接埠或憑證。此資訊可以做為名為 的輸入參數提供給 IDT，這是使用者在其*<device-tester-extract-location>*/config資料夾中建立userdata.json的檔案userdata值。檔案的格式userdata.json取決於您在測試套件中包含的 userdata_schema.json 檔案。

若要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 中*<custom-test-suite-folder>*，建立 userdata_schema.json 檔案。
3. 編輯 userdata_schema.json 檔案以建立有效的 IETF 草稿 v4 JSON 結構描述。

當 IDT 執行您的測試套件時，它會自動讀取結構描述，並使用它來驗證測試執行器所提供的userdata.json檔案。如果有效，userdata.json檔案的內容可在 IDT 內容和狀態機器內容中使用。

設定 IDT 狀態機器

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

狀態機器是一種控制測試套件執行流程的建構。它決定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，並繼續轉換這些狀態，直到達到結束狀態為止。

如果您的測試套件不包含使用者定義的狀態機器，IDT 會為您產生狀態機器。預設狀態機器會執行下列函數：

- 讓測試執行器能夠選取和執行特定測試群組，而不是整個測試套件。
- 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。
- 產生報告並列印主控台摘要，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：

- 每個狀態對應於 IDT 要採取的動作，例如執行測試群組或製作報告檔案的產品。
- 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。
- 結束狀態必須為 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`StartAt`必須設定為 `States` 物件中列出的其中一個狀態。

`States`

將使用者定義狀態名稱映射至有效 IDT 狀態的物件。每個 States.*state-name* 物件都包含映射至 *state-name* 的有效狀態定義。

States 物件必須包含 Succeed和 Fail 狀態。如需有效狀態的資訊，請參閱 有效狀態和狀態定義。

有效狀態和狀態定義

本節說明可在 IDT 狀態機器中使用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，建議您在測試群組層級設定狀態轉換規則，而非測試案例層級，除非絕對必要。

狀態定義

- RunTask
- Choice
- 平行
- AddProductFeatures
- 報告
- LogMessage
- SelectGroup
- 失敗
- Succeed

RunTask

RunTask 狀態會從測試套件中定義的測試群組執行測試案例。

```
{
    "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`

在目前狀態下執行動作後，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，IDT 會執行測試執行器選取的測試群組。

`TestCases`

選用。來自 中指定群組的測試案例 IDs 陣列`TestGroup`。根據 `TestGroup`和 的值`TestCases`，IDT 會決定測試執行行為，如下所示：

- 同時指定 `TestCases` `TestGroup`和 時，IDT 會從測試群組執行指定的測試案例。
- `TestCases` 指定`TestGroup`但未指定 時，IDT 會執行指定的測試案例。
- `TestGroup` 指定 但未指定 `TestCases` 時，IDT 會執行指定測試群組中的所有測試案例。
- 未指定 `TestCases` `TestGroup`或 時，IDT 會從測試執行器從 IDT CLI 中選取的測試群組中執行所有測試案例。若要啟用測試執行器的群組選擇，您必須在 `statemachine.json` 檔案中同時包含 `RunTask`和 `Choice` 狀態。如需如何運作的範例，請參閱範例狀態機器：執行使用者選取的測試群組。

  如需為測試執行器啟用 IDT CLI 命令的詳細資訊，請參閱 the section called “啟用 IDT CLI 命令”。

`ResultVar`

使用測試執行結果設定的內容變數名稱。如果您未指定 的值，請勿指定此值`TestGroup`。IDT `false`會根據下列項目，將您在 中定義的變數值`ResultVar`設定為 `true`或 ：

- 如果變數名稱的格式為 *`text`*`_`*`text`*`_passed`，則值會設定為是否通過或略過第一個測試群組中的所有測試。
- 在所有其他情況下，該值會設定為是否通過或略過所有測試群組中的所有測試。

一般而言，您將使用 `RunTask` 狀態來指定測試群組 ID，而不指定個別測試案例 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裝置才能執行，而且只有單一裝置可用，則測試案例將改為依序執行。

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，則也會將狀態機器內容中的`hasExecutionError`變數設定為 `true`。

Choice

Choice 狀態可讓您動態設定下一個狀態，以根據使用者定義的條件轉換為 。

```
{
    "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 中定義的任何表達式Choices都無法評估為 ，則要轉換為的預設狀態true。

`FallthroughOnError`

選用。指定狀態在評估表達式時發生錯誤時的行為。true 如果您想要在評估導致錯誤時略過表達式，請將 設定為 。如果沒有符合的表達式，則狀態機器會轉換為 Default 狀態。如果未指定FallthroughOnError值，則預設為 false。

`Choices`

運算式和狀態的陣列，用於判斷在目前狀態下執行動作後要轉換到哪個狀態。

`Choices.Expression`

評估為布林值的表達式字串。如果表達式評估為 true，則狀態機器會轉換為 中定義的狀態Choices.Next。表達式字串會從狀態機器內容擷取值，然後對其執行操作以到達布林值。如需存取狀態機器內容的資訊，請參閱 狀態機器內容。

`Choices.Next`

如果 中定義的表達式Choices.Expression評估為 ，則要轉換為的狀態名稱true。

錯誤處理

在下列情況下， Choice 狀態可能需要錯誤處理：

- 選擇表達式中的某些變數不存在於狀態機器內容中。
- 表達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。如果您想要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，並根據您的使用案例執行下列其中一項操作：

- 如果您正在存取的變數預期在某些情況下不存在，請使用 值Default和其他Choices區塊來指定下一個狀態。
- 如果您存取的變數應一律存在，請將Default狀態設定為 Fail。

平行

Parallel 狀態可讓您彼此平行定義和執行新的狀態機器。

```
{
    "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

在目前狀態下執行動作後，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無法參考其定義以外的狀態。

> **Note**
>
> 由於每個分支狀態機器共用相同的狀態機器內容，因此在一個分支中設定變數，然後從另一個分支讀取這些變數可能會導致意外行為。

Parallel 狀態只有在執行所有分支狀態機器之後才會移至下一個狀態。每個需要裝置的狀態都會等待執行，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。如果沒有足夠的裝置可用，則測試案例將依序執行。由於測試案例在平行執行時以隨機順序執行，因此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

請確定分支狀態機器和父系狀態機器都會轉換為 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系狀態機器，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，請在共用狀態機器內容中使用 hasExecutionErrors值。如需如何運作的範例，請參閱 狀態機器範例：平行執行兩個測試群組。

AddProductFeatures

AddProductFeatures 狀態可讓您將產品功能新增至 IDT 產生的awsiotdevicetester_report.xml檔案。

產品功能是有關裝置可能符合的特定條件的使用者定義資訊。例如，MQTT產品功能可以指定裝置正確發佈 MQTT 訊息。在報告中，產品功能會根據是否通過指定的測試，設定為 supported、 not-supported或自訂值。

> **Note**
>
> AddProductFeatures 狀態本身不會產生報告。此狀態必須轉換為 Report 狀態才能產生報告。

```
{
    "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`

在目前狀態下執行動作後，要轉換為的狀態名稱。

`Features`

要在 `awsiotdevicetester_report.xml` 檔案中顯示的產品功能陣列。

`Feature`

功能的名稱

`FeatureValue`

選用。要在報告中使用的自訂值，而非 `supported`。如果未指定此值，則根據測試結果，功能值會設為 `supported`或 `not-supported`。

如果您使用 的自訂值`FeatureValue`，則可以使用不同的條件測試相同的功能，IDT 會串連支援條件的功能值。例如，以下摘錄顯示具有兩個不同`MyFeature`特徵值的功能：

```
...
{
    "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，才能支援此功能。

`OneOfGroups`

選用。測試群組 IDs的陣列。至少一個指定測試群組內的所有測試都必須通過，才能支援此功能。

`TestCases`

選用。測試案例 IDs的陣列。如果您指定此值，則適用下列條件：

- 所有指定的測試案例都必須通過，才能支援此功能。
- `Groups` 只能包含一個測試群組 ID。
- `OneOfGroups` 不得指定。

`IsRequired`

選用。設定為 `false` 以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 `device.json` 檔案中指定`protocol`值的執行方法陣列。如果指定此值，則測試執行器必須指定`protocol`符合此陣列中其中一個值的值，以在報告中包含 功能。如果未指定此值，則功能一律會包含在報告中。

若要使用 `AddProductFeatures` 狀態，您必須將 `RunTask` 狀態`ResultVar`中的 值設定為下列其中一個值：

- 如果您指定個別測試案例 IDs，請將 `ResultVar`設定為 *`group-id_test-id`*_passed。
- 如果您未指定個別測試案例 IDs，請將 `ResultVar`設定為 *`group-id`*_passed。

`AddProductFeatures` 狀態會以下列方式檢查測試結果：

- 如果您未指定任何測試案例 IDs，則每個測試群組的結果取決於狀態機器內容中的 *`group-id`*_passed變數值。

- 如果您確實指定了測試案例 IDs，則每個測試的結果取決於狀態機器內容中的 *group-id_test-id*_passed變數值。

錯誤處理

如果在此狀態下提供的群組 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，則也會將狀態機器內容中的hasExecutionErrors變數設定為 true。

報告

Report 狀態會產生 *suite-name*_Report.xml和 awsiotdevicetester_report.xml 檔案。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

在目前狀態下執行動作後，要轉換為的狀態名稱。

您應該一律在測試執行流程結束時轉換為 Report 狀態，以便測試執行器可以檢視測試結果。一般而言，此狀態之後的下一個狀態為 Succeed。

錯誤處理

如果此狀態在產生報告時遇到問題，則會發出ReportError執行錯誤。

LogMessage

LogMessage 狀態會產生 test_manager.log 檔案，並將日誌訊息串流至 主控台。

```
{
    "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`

在目前狀態下執行動作後，要轉換為的狀態名稱。

`Level`

建立日誌訊息的錯誤層級。如果您指定的關卡無效，此狀態會產生錯誤訊息並予以捨棄。

`Message`

要記錄的訊息。

SelectGroup

`SelectGroup` 狀態會更新狀態機器內容，以指出已選取哪些群組。此狀態設定的值會由任何後續`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`

在目前狀態下執行動作後，要轉換為的狀態名稱。

`TestGroups`

將標記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`變數會在內容`true`中設定為 。請務必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。

失敗

`Fail` 狀態表示狀態機器未正確執行。這是狀態機器的結束狀態，每個狀態機器定義都必須包含此狀態。

```
{
    "Type": "Fail"
}
```

Succeed

Succeed 狀態表示狀態機器已正確執行。這是狀態機器的結束狀態，每個狀態機器定義都必須包含此狀態。

```
{
    "Type": "Succeed"
}
```

狀態機器內容

狀態機器內容是唯讀 JSON 文件，其中包含可在執行期間提供給狀態機器的資料。狀態機器內容只能從狀態機器存取，並包含決定測試流程的資訊。例如，您可以使用 userdata.json 檔案中測試執行器設定的資訊來判斷是否需要執行特定測試。

狀態機器內容使用下列格式：

```
{
    "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`

為測試執行選取之裝置集區的相關資訊。對於選取的裝置集區，此資訊會從 `device.json`檔案中定義的對應頂層裝置集區陣列元素擷取。

`userData`

`userdata.json` 檔案中的資訊。

`config`

資訊鎖定`config.json`檔案。

`suiteFailed`

狀態機器啟動`false`時，此值會設為 。如果測試群組在 `RunTask` 狀態失敗，則此值會在狀態機器執行的剩餘持續時間內設定為 `true` 。

`specificTestGroups`

如果測試執行器選取要執行的特定測試群組，而不是整個測試套件，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，而不是整個測試套件，則會建立此金鑰並包含特定測試案例 IDs的清單。

`hasExecutionErrors`

狀態機器啟動時不會結束。如果任何狀態遇到執行錯誤，則會建立此變數`true`，並在狀態機器執行的剩餘持續時間內設定為 。

您可以使用 JSONPath 表示法查詢內容。狀態定義中 JSONPath 查詢的語法為 {{$.*query*}}。您可以在某些狀態下使用 JSONPath 查詢做為預留位置字串。IDT 會將預留位置字串取代為內容中評估 JSONPath 查詢的值。您可以針對下列值使用預留位置：

- 狀態`TestCases`的值`RunTask`。
- `Expression` 值`Choice`狀態。

當您從狀態機器內容存取資料時，請確定符合下列條件：

- 您的 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 表示法從內容存取資料的詳細資訊，請參閱 使用 IDT 內容。

執行錯誤

執行錯誤是狀態機器在執行狀態時遇到的狀態機器定義錯誤。IDT 會記錄`test_manager.log`檔案中每個錯誤的相關資訊，並將日誌訊息串流至主控台。

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。
- 檢查狀態機器內容中`hasExecutionErrors`值的值。

擷取

若要使用 `Catch`，請將下列項目新增至您的狀態定義：

```
"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，則狀態機器會轉換為 中指定的狀態`Catch.Next`。如需其產生的錯誤類型的相關資訊，請參閱每個狀態定義。

`Catch.Next`

如果目前狀態遇到符合 中指定其中一個值的執行錯誤，則要轉換為`Catch.ErrorEquals`的下一個狀態。

擷取區塊會依序處理，直到一個相符為止。如果沒有錯誤符合 Catch 區塊中列出的錯誤，則狀態機器會繼續執行。由於執行錯誤是不正確狀態定義的結果，我們建議您在狀態遇到執行錯誤時轉換為失敗狀態。

hasExecutionError

當某些狀態遇到執行錯誤時，除了發出錯誤之外，也會在狀態機器內容true中將hasExecutionError值設定為 。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為 Fail 狀態。

此方法具有下列特性。

- 狀態機器不會以指派給 的任何值開始hasExecutionError，且此值在特定狀態設定之前無法使用。這表示您必須針對存取此值FallthroughOnErrorfalseChoice的狀態，將 明確設定為 ，以防止狀態機器在未發生執行錯誤時停止。
- 一旦設定為 true， hasExecutionError 永遠不會設定為 false 或從內容中移除。這表示此值只有在第一次設定為 時才有用true，而且對於所有後續狀態，它不提供有意義的值。
- 該hasExecutionError值會與 Parallel 狀態的所有分支狀態機器共用，這可能會導致非預期的結果，具體取決於存取它的順序。

由於這些特性，如果您可以改用 Catch 區塊，我們不建議您使用此方法。

狀態機器範例

本節提供一些範例狀態機器組態。

範例

- 狀態機器範例：執行單一測試群組
- 狀態機器範例：執行使用者選取的測試群組
- 狀態機器範例：使用產品功能執行單一測試群組
- 狀態機器範例：平行執行兩個測試群組

狀態機器範例：執行單一測試群組

此狀態機器：

- 執行 ID 為 的測試群組GroupA，該群組必須存在於 group.json 檔案的 套件中。
- 檢查執行錯誤，並在發現任何 Fail 時轉換為 。
- Succeed 如果沒有錯誤，則產生報告並轉換為 ，Fail否則。

```
{
```

```
    "Comment": "Runs a single group and then generates a report.",
    "StartAt": "RunGroupA",
    "States": {
        "RunGroupA": {
            "Type": "RunTask",
            "Next": "Report",
            "TestGroup": "GroupA",
            "Catch": [
                {
                    "ErrorEquals": [
                        "RunTaskError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "Report": {
            "Type": "Report",
            "Next": "Succeed",
            "Catch": [
                {
                    "ErrorEquals": [
                        "ReportError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "Succeed": {
            "Type": "Succeed"
        },
        "Fail": {
            "Type": "Fail"
        }
    }
}
```

狀態機器範例：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在未同時選取測試群組的情況下選取測試案例。

- 如果選取測試群組：
  - 在選取的測試群組中執行測試案例。若要這樣做，狀態機器不會明確指定 RunTask 狀態中的任何測試群組或測試案例。
  - 執行所有測試並結束之後產生報告。
- 如果未選取測試群組：
  - 在測試群組 中執行測試GroupA。
  - 產生報告並結束。

```
{
    "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"
        }
    }
}
```

狀態機器範例：使用產品功能執行單一測試群組

此狀態機器：

- 執行測試群組 GroupA。
- 檢查執行錯誤，並在發現任何 Fail 時轉換為 。
- 將 FeatureThatDependsOnGroupA功能新增至 awsiotdevicetester_report.xml 檔案：
  - 如果GroupA通過，則功能會設定為 supported。
  - 該功能在報告中不會標記為選用。
- Succeed 如果沒有錯誤，則產生報告並轉換為 ，Fail否則

```
{
    "Comment": "Runs GroupA and adds product features based on GroupA",
    "StartAt": "RunGroupA",
    "States": {
        "RunGroupA": {
            "Type": "RunTask",
            "Next": "AddProductFeatures",
            "TestGroup": "GroupA",
            "ResultVar": "GroupA_passed",
            "Catch": [
                {
                    "ErrorEquals": [
                        "RunTaskError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "AddProductFeatures": {
            "Type": "AddProductFeatures",
            "Next": "Report",
            "Features": [
                {
                    "Feature": "FeatureThatDependsOnGroupA",
                    "Groups": [
                        "GroupA"
                    ],
                    "IsRequired": true
                }
            ]
        },
        "Report": {
            "Type": "Report",
            "Next": "Succeed",
            "Catch": [
                {
                    "ErrorEquals": [
                        "ReportError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
```

```
        "Succeed": {
            "Type": "Succeed"
        },
        "Fail": {
            "Type": "Fail"
        }
    }
}
```

狀態機器範例：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。分支狀態機器中RunTask狀態存放在內容中的ResultVar變數可供 AddProductFeatures 狀態使用。
- 檢查執行錯誤，並在發現任何 Fail 時轉換為 。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通過，則功能會設定為 supported。
  - 該功能在報告中不會標記為選用。
- Succeed 如果沒有錯誤，則產生報告並轉換為 ，Fail否則

如果在裝置集區中設定了兩個裝置，則 GroupA和 GroupB 可以同時執行。不過，如果 GroupA或有GroupB多個測試，則兩個裝置可能會配置到這些測試。如果只設定一個裝置，測試群組將依序執行。

```
{
    "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": "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

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

您可以透過下列方式，在測試套件資料夾中建立和放置測試案例可執行檔：

- 對於使用`test.json`檔案中引數或環境變數來決定要執行哪些測試的測試套件，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。
- 對於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建立一個測試案例可執行檔。

身為測試寫入器，您可以判斷哪種方法適合您的使用案例，並據以建構您的測試案例可執行檔。請確定您的 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 會讀取下列檔案：

- 所選測試案例`test.json`的 會決定要啟動的程序，以及要設定的環境變數。
- 測試套件`suite.json`的 決定要設定的環境變數。

IDT 會根據`test.json`檔案中指定的命令和引數啟動必要的測試可排除程序，並將所需的環境變數傳遞給程序。

使用 IDT 用戶端 SDK

IDT 用戶端 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 和待測裝置互動。IDT 目前提供下列 SDKs：

- 適用於 Python 的 IDT 用戶端 SDK
- IDT 用戶端 SDK for Go

這些 SDKs位於 *<device-tester-extract-location>*/sdks 資料夾中。建立新的測試案例可執行檔時，您必須將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，並在程式碼中參考開發套件。本節提供可在測試案例可執行檔中使用的可用 API 命令的簡短描述。

本節內容

- 裝置互動
- IDT 互動
- 主機互動

裝置互動

下列命令可讓您與待測裝置通訊，而不必實作任何額外的裝置互動和連線管理功能。

`ExecuteOnDevice`

允許測試套件在支援 SSH 或 Docker shell 連線的裝置上執行 Shell 命令。

`CopyToDevice`

允許測試套件將本機檔案從執行 IDT 的主機機器複製到支援 SSH 或 Docker shell 連線之裝置上的指定位置。

`ReadFromDevice`

允許測試套件從支援 UART 連線的裝置序列連接埠讀取。

> **Note**
>
> 由於 IDT 不會管理從內容使用裝置存取資訊所建立之裝置的直接連線，我們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，則您可以從 IDT 內容擷取裝置存取資訊，並使用它從測試套件直接連線至裝置。

若要進行直接連線，請分別擷取 中的資訊，device.connectivity以及待測裝置和資源裝置resource.devices.connectivity的欄位。如需使用 IDT 內容的詳細資訊，請參閱 使用 IDT 內容。

IDT 互動

下列命令可讓您的測試套件與 IDT 通訊。

`PollForNotifications`

允許測試套件檢查來自 IDT 的通知。

`GetContextValue` 和 `GetContextString`

允許測試套件從 IDT 內容擷取值。如需詳細資訊，請參閱使用 IDT 內容。

`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中每個測試案例的結尾呼叫。

主機互動

下列命令可讓您的測試套件與主機機器通訊。

`PollForNotifications`

允許測試套件檢查來自 IDT 的通知。

`GetContextValue` 和 `GetContextString`

允許測試套件從 IDT 內容擷取值。如需詳細資訊，請參閱使用 IDT 內容。

`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生命週期。

啟用 IDT CLI 命令

`run-suite` 命令 IDT CLI 提供數個選項，可讓測試執行器自訂測試執行。若要允許測試執行器使用這些選項來執行您的自訂測試套件，請實作對 IDT CLI 的支援。如果您未實作支援，測試執行器仍然可以執行測試，但某些 CLI 選項將無法正常運作。為了提供理想的客戶體驗，建議您在 IDT CLI 中為 `run-suite`命令實作下列引數的支援：

`timeout-multiplier`

指定大於 1.0 的值，將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的測試案例的逾時。當測試執行器在其`run-suite`命令中指定此引數時，IDT 會使用它來計算 IDT_TEST_TIMEOUT 環境變數的值，並在 IDT 內容中設定 `config.timeoutMultiplier` 欄位。若要支援此引數，您必須執行下列動作：

- 讀取 IDT_TEST_TIMEOUT 環境變數以取得正確計算的逾時值，而不是直接使用 `test.json` 檔案的逾時值。
- 從 IDT 內容擷取`config.timeoutMultiplier`值，並將其套用至長時間執行的逾時。

如需因逾時事件而提早結束的詳細資訊，請參閱 指定結束行為。

`stop-on-first-failure`

指定 IDT 應在遇到失敗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 會在遇到失敗時立即停止執行測試。不過，如果測試案例平行執行，則這可能會導致非預期的結果。若要實作支援，請確定如果 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、清除臨時資源，並向 IDT 報告測試結果。如需在故障時提早結束的詳細資訊，請參閱 指定結束行為。

`group-id` 和 `test-id`

指定 IDT 應僅執行選取的測試群組或測試案例。

測試執行器可以將這些引數與其`run-suite`命令搭配使用，以指定下列測試執行行為：

- 在指定的測試群組內執行所有測試。
- 從指定的測試群組內執行一系列測試。

若要支援這些引數，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。如果您不是使用自訂狀態機器，則預設的 IDT 狀態機器會包含所需的狀態，而且您不需要採取其他動作。不過，如果您使用的是自訂狀態機器，請使用 狀態機器範例：執行使用者選取的測試群組做為範例，在狀態機器中新增所需的狀態。

如需 IDT CLI 命令的詳細資訊，請參閱 偵錯並執行自訂測試套件。

寫入事件日誌

測試執行時，您會將資料傳送至 `stderr` `stdout`，並將事件日誌和錯誤訊息寫入主控台。如需主控台訊息格式的資訊，請參閱 主控台訊息格式。

當 IDT 完成執行測試套件時，此資訊也可在 資料夾的 test_manager.log 檔案中取得*<devicetester-extract-location>*/results/*<execution-id>*/logs。

您可以設定每個測試案例，將日誌從其測試執行寫入至位於 *<device-tester-extract-location>*/results/*execution-id*/logs 資料夾中*<group-id>*_*<test-id>*的檔案，包括來自待測裝置的日誌。若要這樣做，請從具有testData.logFilePath查詢的 IDT 內容擷取日誌檔案的路徑，在該路徑建立檔案，然後寫入您想要的內容。IDT 會根據正在執行的測試案例自動更新路徑。如果您選擇不建立測試案例的日誌檔案，則該測試案例不會產生任何檔案。

您也可以設定文字可執行檔，視需要在 *<device-tester-extract-location>*/logs 資料夾中建立其他日誌檔案。建議您為日誌檔案名稱指定唯一的字首，以免您的檔案遭到覆寫。

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>*/兩個報告都會擷取測試套件執行的結果。如需 IDT 用於這些報告之結構描述的詳細資訊，請參閱 檢閱 IDT 測試結果和日誌

若要填入*suite-name*_report.xml檔案的內容，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，則會發出測試案例的錯誤。下列 Python 摘錄顯示將測試結果傳送至 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 IDT 內容中testData.testArtifactsPath存檔的 中。在此資料夾中，IDT 會使用第一個依字母順序排序的 XML 檔案做為測試結果。

如果您的測試邏輯產生 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 將結果提交給 IDT。

如果您使用此方法，請確定您的測試邏輯準確摘要測試結果，並以與 檔案相同的格式格式化結果*suite-name*_report.xml檔案。IDT 不會對您提供的資料執行任何驗證，但以下情況除外：

- IDT 會忽略testsuites標籤的所有屬性。反之，它會從其他報告的測試群組結果計算標籤屬性。
- 必須至少有一個testsuite標籤存在於 中testsuites。

由於 IDT 對所有測試案例使用相同的成品資料夾，而且不會在測試執行之間刪除結果檔案，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。建議您在所有測試案例中使用與產生 XML 結果檔案相同的名稱，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法在測試套件中報告，也就是說，對某些測試案例使用 XML 結果檔案，並透過 API 向其他人提交結果，但不建議使用此方法。

指定結束行為

將您的文字可執行檔設定為一律以結束碼 0 結束，即使測試案例報告失敗或錯誤結果亦然。僅使用非零結束代碼來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 收到非零的結束碼時，表示測試案例發生錯誤，導致無法執行。

IDT 可能會請求或預期測試案例在下列事件完成之前停止執行。使用此資訊來設定您的測試案例可執行檔，以從測試案例偵測下列每個事件：

Timeout (逾時)

當測試案例執行的時間超過`test.json`檔案中指定的逾時值時，便會發生。如果測試執行器使用`timeout-multiplier`引數指定逾時乘數，則 IDT 會使用乘數計算逾時值。

若要偵測此事件，請使用 IDT_TEST_TIMEOUT 環境變數。當測試執行器啟動測試時，IDT 會將 IDT_TEST_TIMEOUT 環境變數的值設定為計算的逾時值 （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 Ctrl+C。

由於終端機會將訊號傳播到所有子程序，因此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，以檢查 `PollForNotifications` API `CancellationRequested` 回應中的布林值。當 IDT `CancellationRequested` 收到中斷訊號時，它會將布林值設定為 `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`引數指定 IDT 應在遇到任何失敗時停止時，便會發生。

若要偵測此事件，您可以定期輪詢 API，以檢查 `PollForNotifications` API `CancellationRequested` 回應中的布林值。當 IDT 遇到失敗並設定為在第一次失敗時停止時，它會將`CancellationRequested`布林值設定為 `true`。

當這些事件發生時，IDT 會等待 5 分鐘，讓任何目前正在執行的測試案例完成執行。如果所有執行中的測試案例未在 5 分鐘內結束，IDT 會強制其每個程序停止。如果 IDT 在程序結束之前尚未收到測試結果，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. 停止執行正常測試邏輯。
2. 清除任何臨時資源，例如受測裝置上的測試成品。
3. 向 IDT 報告測試結果，例如測試失敗或錯誤。
4. 結束。

使用 IDT 內容

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

當 IDT 執行測試套件時，測試套件可以存取一組資料，用於判斷每個測試的執行方式。此資料稱為 IDT 內容。例如，`userdata.json`在 IDT 內容中，測試執行器提供的使用者資料組態可用於測試套件。

IDT 內容可視為唯讀 JSON 文件。測試套件可以使用物件、陣列、數字等標準 JSON 資料類型，從 擷取資料並將資料寫入內容。

內容結構描述

IDT 內容使用下列格式：

```
{
    "config": {
        <config-json-content>
        "timeoutMultiplier": timeout-multiplier
    },
    "device": {
        <device-json-device-element>
    },
    "devicePool": {
```

```
        <device-json-pool-element>
    },
    "resource": {
        "devices": [
            {
                <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。config 欄位也包含下列其他欄位：

`config.timeoutMultiplier`

測試套件使用之任何逾時值的乘數。此值由 IDT CLI 的測試執行器指定。預設值為 1。

`device`

為測試執行選取之裝置的相關資訊。此資訊等同於所選裝置的 device.json 檔案中的devices陣列元素。

`devicePool`

為測試執行選取之裝置集區的相關資訊。此資訊等同於所選裝置集區device.json檔案中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 `resource.json` 檔案中定義的`devices`陣列。每個`devices`元素都包含下列額外欄位：

`resource.device.name`

資源裝置的名稱。此值設定為 `test.json` 檔案中的 `requiredResource.name`值。

`testData.awsCredentials`

測試用來連線至 AWS 雲端的 AWS 登入資料。此資訊是從 `config.json` 檔案取得。

`testData.logFilePath`

測試案例寫入日誌訊息的日誌檔案路徑。如果檔案不存在，測試套件會建立此檔案。

`userData`

測試執行器在 `userdata.json` 檔案中提供的資訊。

存取內容中的資料

您可以使用 JSON 檔案的 JSONPath 標記法，以及使用 `GetContextValue`和 `GetContextString` APIs 的文字可執行檔來查詢內容。用於存取 IDT 內容的 JSONPath 字串語法會有所不同，如下所示：

- 在 `suite.json`和 中`test.json`，您可以使用 {{*query*}}。也就是說，請勿使用根元素`$.`來啟動表達式。
- 在 中`statemachine.json`，您可以使用 {{`$.`*query*}}。
- 在 API 命令中，您可以使用 *query*或 {{`$.`*query*}}，視命令而定。如需詳細資訊，請參閱 SDKs 中的內嵌文件。

下表說明典型foobar JSONPath 表達式中的運算子：

運算子	描述
`$`	根元素。由於 IDT 的最上層內容值是 物件，因此您通常會使用 `$.`來啟動查詢。
`.childName`	`childName` 使用 物件的名稱存取子元素。如果套用至陣列， 會產生新的陣列，並將此運算子套用至每個元素。元素名稱區分大小寫。例

運算子	描述
	如，存取 config 物件中awsRegion 值的查詢是 $.config.awsRegion 。
[start:end]	從陣列篩選元素，擷取從start索引開始到end索引的項目，兩者皆包含在內。
[index1, index2, ... , indexN]	從陣列篩選元素，僅從指定的索引擷取項目。
[?(expr)]	使用 expr表達式從陣列篩選元素。此表達式必須評估為布林值。

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）

    如果表達式中的 JSONPath 或 值是陣列、布林值或物件值，則這是您可以唯一支援的二進位運算子。
  - >= （大於或等於）
  - > （大於）
  - =~ （規則表達式比對）。若要在篩選條件表達式中使用此運算子，表達式左側的 JSONPath 或值必須評估為字串，右側必須是遵循 RE2 語法的模式值。

您可以使用格式為 {{*query*}} 的 JSONPath 查詢，做為test.json檔案中 args和 environmentVariables 欄位內，以及suite.json檔案中 environmentVariables 欄位內的預留位置字串。IDT 會執行內容查詢，並將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，下列考量適用於您的查詢：

- 在所有小寫中，您必須在查詢中每次出現 `devicePool`金鑰。也就是說，請`devicepool`改用 。
- 對於陣列，您只能使用字串陣列。此外，陣列使用非標準`item1, item2,...,itemN`格式。如果陣列只包含一個元素，則會將其序列化為 `item`，使其與字串欄位無法區分。
- 您無法使用預留位置從內容擷取物件。

由於這些考量，我們建議您盡可能使用 API 來存取測試邏輯中的內容，而不是 `test.json`和 `suite.json` 檔案中的預留位置字串。不過，在某些情況下，使用 JSONPath 預留位置擷取單一字串以設定為環境變數可能會更方便。

設定測試執行器的設定

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

若要執行自訂測試套件，測試執行器必須根據他們想要執行的測試套件來設定其設定。根據位於 *<device-tester-extract-location>*/configs/ 資料夾中的 JSON 組態檔案範本來指定設定。如有需要，測試執行器也必須設定 IDT 用來連線至 AWS 雲端的 AWS 登入資料。

身為測試寫入器，您需要設定這些檔案來偵錯您的測試套件。您必須提供測試執行器的說明，以便他們可以視需要設定下列設定來執行您的測試套件。

設定 device.json

`device.json` 檔案包含執行測試之裝置的相關資訊（例如 IP 地址、登入資訊、作業系統和 CPU 架構）。

測試執行器可以使用位於 *<device-tester-extract-location>*/configs/ 資料夾中的下列範本`device.json`檔案提供此資訊。

```
[
    {
        "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 用於追蹤合格裝置。

> **Note**
>
> 如果您想要在 AWS Partner Device Catalog 中列出您的電路板，您在此處指定的 SKU 必須符合您在列出程序中使用的 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有關要包含在 `device.json` 檔案中的功能名稱和值的資訊。例如，如果您想要測試可做為其他裝置的 MQTT 伺服器運作的裝置，則可以設定測試邏輯來驗證名為 之功能的特定支援層級`MQTT_QOS`。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。您可以從具有`devicePool.features`查詢的 IDT 內容，或從具有`pool.features`查詢的狀態機器內容中擷取提供的資訊。

`features.name`

功能的名稱。

`features.value`

支援的功能值。

`features.configs`

如有需要，此功能的組態設定。

`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，uart唯一支援的值為實體裝置的 ssh和 ，以及 Docker 容器docker的 和 。

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 連線的連接埠號碼。

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`

用於驗證的燈入資料。

`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。裝置所連接的序列連接埠。

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。容器內使用者的名稱。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線，您可以從`pool.Devices[0].Connectivity.Protocol`狀態機器內容擷取，並將其與`Choice`狀態中的預期值進行比較。如果使用不正確的通訊協定，請使用 `LogMessage` 狀態列印訊息並轉換為 `Fail` 狀態。

或者，您可以使用錯誤處理程式碼來報告不正確裝置類型的測試失敗。

（選用）設定 userdata.json

userdata.json 檔案包含測試套件所需的任何其他資訊，但未在 device.json 檔案中指定。此檔案的格式取決於測試套件中定義的userdata_scheme.json檔案。如果您是測試寫入器，請務必將此資訊提供給將執行您所寫入測試套件的使用者。

（選用）設定 resource.json

resource.json 檔案包含將用作資源裝置的任何裝置的相關資訊。資源裝置是測試待測裝置的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以成功連線到該裝置。資源裝置是選用的，您可以視需要要求任意數量的資源裝置。身為測試寫入器，您可以使用 test.json 檔案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 檔案來提供具有必要功能的資源裝置集區。請務必將此資訊提供給將執行您撰寫之測試套件的使用者。

測試執行器可以使用位於 *<device-tester-extract-location>*/configs/ 資料夾中的下列範本resource.json檔案提供此資訊。

```
[
    {
        "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。此欄位所需的資訊會在測試套件的 test.json 檔案中定義，並決定要執行哪些測試，以及如何執行這些測試。如果測試套件不需要任何功能，則不需要此欄位。

`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的測試數量。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，uart唯一支援的值為實體裝置的 ssh和 ，以及 Docker 容器docker的 和 。

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 連線的連接埠號碼。

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

用於驗證的燈入資料。

`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。裝置所連接的序列連接埠。

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。容器內使用者的名稱。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

`config.json` 檔案包含 IDT 的組態資訊。一般而言，除了為 IDT 提供 AWS 使用者登入資料，以及選擇性的區域之外，測試執行器不需要修改此檔案 AWS 。如果 AWS 提供具有必要許可的登入資料，會 AWS IoT Device Tester 收集並提交用量指標給 AWS。這是選擇加入功能，用於改善 IDT 功能。如需詳細資訊，請參閱IDT 用量指標。

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：

- 憑證檔案

  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該工作階段關閉後無法使用。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，測試執行器會在 資料夾的 `config.json` 檔案中編輯 `auth`區段*<device-tester-extract-location>*`/configs/`。

```
{
    "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**
>
> 此檔案中的所有路徑都是相對於 *<device-tester-extract-location>* 所定義。

`log.location`

<device-tester-extract-location> 中日誌資料夾的路徑。

`configFiles.root`

包含組態檔案之資料夾的路徑。

`configFiles.device`

device.json 檔案的路徑。

`testPath`

包含測試套件的資料夾路徑。

`reportPath`

IDT 執行測試套件後，將包含測試結果的資料夾路徑。

`awsRegion`

選用。測試套件將使用 AWS 的區域。如果未設定，則測試套件將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。支援的值是從登入資料檔案`file`擷取登入資料，並使用環境變數`environment`擷取登入資料。

`auth.credentials.profile`

從登入資料檔案使用的登入資料設定檔。只有當 `auth.method` 設為 `file` 時，才會套用此屬性。

偵錯並執行自訂測試套件

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

設定必要的組態後，IDT 可以執行您的測試套件。完整測試套件的執行時間取決於硬體和測試套件的組成。如需參考，在 Raspberry Pi 3B 上完成完整的 FreeRTOS 資格測試套件大約需要 30 分鐘。 3B

當您撰寫測試套件時，您可以使用 IDT 在偵錯模式下執行測試套件，在執行程式碼之前先檢查程式碼，或將其提供給測試執行器。

在偵錯模式下執行 IDT

由於測試套件依賴 IDT 與裝置互動、提供內容並接收結果，因此您無法在沒有任何 IDT 互動的情況下，在 IDE 中對測試套件進行偵錯。若要這樣做，IDT CLI 會提供 `debug-test-suite`命令，讓您以偵錯模式執行 IDT。執行下列命令以檢視 的可用選項`debug-test-suite`：

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

當您在偵錯模式下執行 IDT 時，IDT 實際上不會啟動測試套件或執行狀態機器；而是與您的 IDE 互動，以回應從 IDE 中執行的測試套件發出的請求，並將日誌列印到主控台。IDT 不會逾時，並等待結束，直到手動中斷為止。在偵錯模式中，IDT 也不會執行狀態機器，也不會產生任何報告檔案。若要偵錯測試套件，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 讀取此資訊`suite.json`。
- 選取資源裝置的引數。IDT 不會從 讀取此資訊`test.json`。

若要偵錯您的測試套件，請完成下列步驟：

1. 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`，請務必視需要設定所有測試套件。
2. 執行下列命令，將 IDT 置於偵錯模式，然後選取執行測試所需的任何裝置。

   ```
   devicetester_[linux | mac | win_x86-64] debug-test-suite [options]
   ```

在您執行此命令後，IDT 會等待來自測試套件的請求，然後回應這些請求。IDT 也會產生 IDT 用戶端 SDK 案例程序所需的環境變數。

3. 在您的 IDE 中，使用 `run`或 `debug`組態來執行下列動作：

   a. 設定 IDT 產生之環境變數的值。

   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 IDE 中執行測試套件。

   您可以視需要多次偵錯並重新執行測試套件。IDT 不會在偵錯模式下逾時。

5. 完成偵錯後，請中斷 IDT 以結束偵錯模式。

用於執行測試的 IDT CLI 命令

下節說明 IDT CLI 命令：

IDT v4.0.0

`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`

列出您 IDT 版本的支援產品，在此情況下，為目前 IDT 版本提供 FreeRTOS 版本和 FreeRTOS 資格測試套件版本。

`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。以下是一些常用的選項：

- `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。如果測試執行器在您的 `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 為具有使用者定義乘數的測試修改 `test.json` 檔案中指定的測試執行逾時。
- `stop-on-first-failure`。 設定 IDT 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 `suite.json` 檔案中`userdataRequired`設定為 true 時，才需要這樣做。

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`

在除錯模式下執行測試套件。如需詳細資訊，請參閱在偵錯模式下執行 IDT。

檢閱 IDT 測試結果和日誌

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

本節說明 IDT 產生主控台日誌和測試報告的格式。

主控台訊息格式

AWS IoT Device Tester 啟動測試套件時， 會使用標準格式將訊息列印到主控台。以下摘錄顯示 IDT 產生的主控台訊息範例。

```
time="2000-01-02T03:04:05-07:00" level=info msg=Using suite: MyTestSuite_1.0.0
executionId=9a52f362-1227-11eb-86c9-8c8590419f30
```

大多數主控台訊息包含下列欄位：

`time`

已記錄事件的完整 ISO 8601 時間戳記。

`level`

所記錄事件的訊息層級。一般而言，記錄的訊息層級是 `info`、 `warn`或 之一`error`。如果 IDT 遇到導致它提早結束的預期事件，則發出 `fatal`或 `panic` 訊息。

`msg`

記錄的訊息。

`executionId`

目前 IDT 程序的唯一 ID 字串。此 ID 用於區分個別 IDT 執行。

從測試套件產生的主控台訊息提供有關待測裝置的額外資訊，以及 IDT 執行的測試套件、測試群組和測試案例。以下摘錄顯示從測試套件產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
```

主控台訊息的測試套件特定部分包含下列欄位：

`suiteId`

目前正在執行的測試套件名稱。

`groupId`

目前正在執行的測試群組 ID。

`testCaseId`

目前執行的測試案例 ID。

`deviceId`

目前測試案例使用的待測裝置 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 Report 狀態。測試摘要包含測試套件的相關資訊、執行的每個群組的測試結果，以及產生的日誌和報告檔案的位置。下列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` 是已簽章的報告，其中包含下列資訊：

- IDT 版本。
- 測試套件版本。
- 用於簽署報告的報告簽章和金鑰。
- `device.json` 檔案中指定的裝置 SKU 和裝置集區名稱。
- 已測試的產品版本和裝置功能。
- 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。測試套件`required`需要標記為 的功能，才能驗證裝置。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` 的功能不需要驗證。以下程式碼片段顯示選用功能。

```
<feature name="hsi" value="supported" type="optional"></feature>
<feature name="mqtt" value="not-supported" type="optional"></feature>
```

測試套件報告結構描述

*suite-name*_Result.xml 報告採用 JUnit XML 格式。您可以將它整合到持續整合和部署平台，例如 Jenkins、Bamboo 等。報告包含測試結果的彙總摘要。

```
<testsuites name="<suite-name> results" time="<run-duration>" tests="<number-of-test>"
 failures="<number-of-tests>" skipped="<number-of-tests>" errors="<number-of-tests>"
 disabled="0">
    <testsuite name="<test-group-id>" package="" tests="<number-of-tests>"
 failures="<number-of-tests>" skipped="<number-of-tests>" errors="<number-of-tests>"
 disabled="0">
        <!--success-->
        <testcase classname="<classname>" name="<name>" time="<run-duration>"/>
        <!--failure-->
        <testcase classname="<classname>" name="<name>" time="<run-duration>">
            <failure type="<failure-type>">
                reason
            </failure>
        </testcase>
        <!--skipped-->
        <testcase classname="<classname>" name="<name>" time="<run-duration>">
            <skipped>
                reason
            </skipped>
        </testcase>
        <!--error-->
        <testcase classname="<classname>" name="<name>" time="<run-duration>">
            <error>
                reason
```

```
            </error>
        </testcase>
    </testsuite>
</testsuites>
```

awsiotdevicetester_report.xml 或 中的報告區段會*suite-name*_report.xml列出已執行的測試和結果。

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

執行測試套件所需的時間，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

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

如果您提供具有必要許可的 AWS 登入資料， 會 AWS IoT Device Tester 收集並提交用量指標給 AWS。這是選擇加入功能，用於改善 IDT 功能。IDT 會收集下列資訊：

- 用來執行 IDT AWS 的帳戶 ID
- 用於執行測試的 IDT CLI 命令
- 執行的測試套件
- *<device-tester-extract-location>* 資料夾中的測試套件
- 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，例如測試是否通過、失敗、發生錯誤或被略過
- 測試的產品功能
- IDT 結束行為，例如非預期或提早結束

IDT 傳送的所有資訊也會記錄到 *<device-tester-extract-location>*/results/*<execution-id>*/ 資料夾中metrics.log的檔案。您可以檢視日誌檔案，以查看在測試執行期間收集的資訊。只有在您選擇收集用量指標時，才會產生此檔案。

若要停用指標集合，您不需要採取其他動作。只要不要存放您的 AWS 登入資料，而且如果您有存放的 AWS 登入資料，請勿設定 config.json 檔案來存取它們。

設定您的 AWS 登入資料

如果您還沒有 AWS 帳戶，則必須建立一個。如果您已經有 AWS 帳戶，您只需要為您的帳戶設定必要的許可，以允許 IDT AWS 代表您將用量指標傳送至 。

註冊 AWS 帳戶

如果您沒有 AWS 帳戶，請完成下列步驟來建立一個。

註冊 AWS 帳戶

1. 開啟 https://portal.aws.amazon.com/billing/signup。
2. 請遵循線上指示進行。

   註冊程序的一部分包括接聽電話或文字訊息，以及在電話鍵盤上輸入驗證碼。

   當您註冊 時 AWS 帳戶，AWS 帳戶根使用者會建立 。根使用者有權存取該帳戶中的所有 AWS 服務 和資源。作為安全最佳實務，請將管理存取權指派給使用者，並且僅使用根使用者來執行需要根使用者存取權的任務。

AWS 會在註冊程序完成後傳送確認電子郵件給您。您可以隨時登錄 https://aws.amazon.com/ 並選擇我的帳戶，以檢視您目前的帳戶活動並管理帳戶。

建立具有管理存取權的使用者

註冊 後 AWS 帳戶，請保護 AWS 帳戶根使用者、啟用 AWS IAM Identity Center和建立管理使用者，以免將根使用者用於日常任務。

保護您的 AWS 帳戶根使用者

1. 選擇根使用者並輸入 AWS 帳戶 您的電子郵件地址，以帳戶擁有者AWS Management Console身分登入 。在下一頁中，輸入您的密碼。

   如需使用根使用者登入的說明，請參閱 AWS 登入 使用者指南中的以根使用者身分登入。

2. 若要在您的根使用者帳戶上啟用多重要素驗證 (MFA)。

   如需說明，請參閱《IAM 使用者指南》中的為您的 AWS 帳戶 根使用者 （主控台） 啟用虛擬 MFA 裝置。

建立具有管理存取權的使用者

1. 啟用 IAM Identity Center。

   如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的啟用 AWS IAM Identity Center。

2. 在 IAM Identity Center 中，將管理存取權授予使用者。

   如需使用 IAM Identity Center 目錄 做為身分來源的教學課程，請參閱AWS IAM Identity Center《 使用者指南》中的使用預設值設定使用者存取 IAM Identity Center 目錄。

以具有管理存取權的使用者身分登入

- 若要使用您的 IAM Identity Center 使用者簽署，請使用建立 IAM Identity Center 使用者時傳送至您電子郵件地址的簽署 URL。

  如需使用 IAM Identity Center 使用者登入的說明，請參閱AWS 登入 《 使用者指南》中的登入 AWS 存取入口網站。

指派存取權給其他使用者

1. 在 IAM Identity Center 中，建立一個許可集來遵循套用最低權限的最佳實務。

   如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的建立許可集。

2. 將使用者指派至群組，然後對該群組指派單一登入存取權。

   如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的新增群組。

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 Management Console 或 AWS Command Line Interface (AWS CLI) 來建立 IAM 政策和 IDT 的使用者，然後將政策連接到使用者。

- 設定 IDT (主控台) 的許可
- 步驟 2：設定 IDT (AWS CLI) 的許可

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，選擇政策，然後選擇建立政策。
   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. 選擇檢閱政策。
   d. 對於名稱，輸入 **IDTUsageMetricsIAMPermissions**。在 Summary (摘要) 下，檢閱您的政策所授與的許可。
   e. 選擇建立政策。
3. 建立 IAM 使用者並將許可連接到使用者。

   若要提供存取權，請新增權限至您的使用者、群組或角色：

- 中的使用者和群組 AWS IAM Identity Center：

  建立權限合集。請按照 AWS IAM Identity Center 使用者指南 中的 建立權限合集 說明進行操作。

- 透過身分提供者在 IAM 中管理的使用者：

  建立聯合身分的角色。遵循「IAM 使用者指南」的為第三方身分提供者 (聯合) 建立角色中的指示。

- IAM 使用者：
  - 建立您的使用者可擔任的角色。請按照「IAM 使用者指南」的為 IAM 使用者建立角色中的指示。
  - (不建議) 將政策直接附加至使用者，或將使用者新增至使用者群組。請遵循 IAM 使用者指南的新增許可到使用者 (主控台) 中的指示。

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 ，請在電腦上安裝和設定 AWS CLI 。請遵循AWS Command Line Interface 《 使用者指南》中的安裝 AWS CLI 中的步驟。

   > **Note**
   >
   > AWS CLI 是開放原始碼工具，可用來與命令列 shell 中的 AWS 服務互動。

2. 建立下列客戶受管政策，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 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IDTUsageMetricsIAMPermissions的政策連接至 IAM 使用者。將 *user-name* 取代為您的 IAM 使用者名稱，並將 命令中的 *<account-id>* 取代為您 AWS 帳戶的 ID。

   ```
   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。您稍後會使用此資訊來設定您的 AWS 登入資料檔案。

提供 AWS 登入資料給 IDT

若要允許 IDT 存取您的 AWS 登入資料並提交指標 AWS，請執行下列動作：

1. 將 IAM 使用者的 AWS 登入資料儲存為環境變數或儲存在登入資料檔案中：

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，請將下列資訊新增至 .aws/credentials file:

```
[profile-name]
aws_access_key_id=access-key
aws_secret_access_key=secret-access-key
```

2. 設定 config.json 檔案的 auth區段。如需詳細資訊，請參閱（選用）設定 config.json。

故障診斷

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

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 主控台輸出來尋找失敗之測試案例的執行 ID、測試案例 ID 和測試群組 ID，然後開啟名為 之測試案例的日誌檔案results/*execution-id*/logs/*test_group_id*__*test_case_id*.log。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。
- 更多詳細的 IDT for FreeRTOS 主控台輸出。

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「*###/##*未獲授權存取此資源」，請確定您設定 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 FRQ_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 下的日誌檔案/results/*execution-id*/logs。
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。
   - 請確定您有乾淨、複製版本的 FreeRTOS 原始程式碼。FreeRTOS 版本會根據 FreeRTOS 版本進行標記。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

當您使用 IDT for FreeRTOS 時，您必須先取得正確的組態檔案，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 IDT for FreeRTOS 產生的錯誤代碼：

錯誤程式碼	錯誤代碼名稱	可能的根本原因	故障診斷
201	InvalidInputError	device.json 、config.json	請確定列出的檔案中未遺漏必要欄位且為

錯誤程式碼	錯誤代碼名稱	可能的根本原因	故障診斷
		on 、或 userdata.json 中的欄位遺失或格式不正確。	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。
202	ValidationError	device.js on 、config.js on 、或 userdata.json 中的欄位包含無效的值。	請查看報告中錯誤代碼右手方的錯誤訊息： • 無效 AWS 區域 - 在config.js on 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 - 在測試機器上設定有效的 AWS 登入資料（透過環境變數或登入資料檔案）。確認身分驗證欄位設定正確。如需詳細資訊，請參閱???。

錯誤程式碼	錯誤代碼名稱	可能的根本原因	故障診斷
203	CopySourceCodeError	無法將 FreeRTOS 原始程式碼複製到指定的目錄。	請確認下列項目： • 檢查有效的 `sourcePath` 在您指定的 `userdata.json` 檔案中。 • 如果存在，請刪除 FreeRTOS 原始程式碼目錄下的 `build`資料夾。如需詳細資訊，請參閱設定建置、刷新和測試設定。

錯誤程式碼	錯誤代碼名稱	可能的根本原因	故障診斷
204	BuildSourceError	無法編譯 FreeRTOS 原始程式碼。	請確認下列項目： • 檢查您 `userdata.json` 檔案底下的 `buildTool` 資訊是否正確。 • 如果您使用 `cmake` 做為建置工具，請確定 `{{enableTests}}` 已指定在 `buildTool` 命令中。如需詳細資訊，請參閱設定建置、刷新和測試設定。 • 如果您已將 IDT for FreeRTOS 解壓縮到系統上包含空格的檔案路徑，例如 `C:\Users\My Name\Desktop\`，則建置命令內部可能需要額外的引號，以確保正確剖析路徑。您的快閃記憶體命令可能需要相同的物件。

錯誤程式碼	錯誤代碼名稱	可能的根本原因	故障診斷
205	FlashOrRunTestError	IDT FreeRTOS 無法在您的 DUT 上刷新或執行 FreeRTOS。	確認您 `userdata.json` 檔案底下的 `flashTool` 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。
206	StartEchoServerError	IDT FreeRTOS 無法啟動適用於 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 `userdata.json` 檔案 `echoServerConfiguration` 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 驗證並列印偵錯資訊。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，但若仍無法找到問題所在，您可以在 IDE 和文字編輯器 (如 Atom 或 Sublime) 中手動執行驗證，也可以透過 JSONLint 等線上工具完成驗證。

偵錯完整性檢查失敗

當您執行 FreeRTOSIntegrity 測試群組並遇到失敗時，請先確定您尚未修改任何*freertos*目錄檔案。如果您尚未，且仍然遇到問題，請確定您使用的是正確的分支。如果您執行 IDT 的`list-supported-products`命令，您可以找到應使用哪個標記*freertos*的儲存庫分支。

如果您複製了`freertos`儲存庫的正確標記分支，但仍發生問題，請確定您已執行 `submodule update`命令。`freertos` 儲存庫的複製工作流程如下所示。

```
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 *freertos*目錄中的 checksums.json 檔案中。若要在不修改檔案和資料夾結構的情況下限定 FreeRTOS 連接埠，請確定檔案的「exhaustive」和「minimal」區段中列出的checksums.json任何檔案都未遭到修改。若要在設定 SDK 的情況下執行 ，請確認 'minimal' 區段下的任何檔案都未修改。

如果您使用 SDK 執行 IDT 並已修改 *freertos*目錄中的一些檔案，請確定您已在 userdata 檔案中正確設定 SDK。否則，完整性檢查程式會驗證 *freertos*目錄中的所有檔案。

偵錯 FullWiFi 測試群組失敗

如果您在 FullWiFi 測試群組中看到失敗，且「AFQP_WiFiConnectMultipleAP」測試失敗，這可能是因為兩個存取點都不在與執行 IDT 的主機電腦相同的子網路中。請確定兩個存取點都與執行 IDT 的主機電腦位於相同的子網路中。

偵錯必要參數遺漏錯誤

由於新功能已新增至 IDT for FreeRTOS，因此可能會引入組態檔案的變更。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 會驗證您的 JSON 組態檔案結構描述，以確保已使用最新的支援版本。

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

您可能會在終端機輸出或 下的 test_manager.log 檔案中看到錯誤「*###/##*無權存取此資源」/results/*execution-id*/logs。若要解決這個問題，請將 AWSIoTDeviceTesterForFreeRTOSFullAccess 受管政策連接到您的測試使用者。如需詳細資訊，請參閱???。

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。如果在 WiFi 或安全通訊端測試中因逾時或無法連線而發生錯誤，請確認網路已設定為允許流量傳送至 1024 - 49151 範圍內的已設定連接埠。

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。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 測試案例因為執行 OTA 後裝置上的相同版本而失敗，可能是因為您的建置系統（例如 cmake) 未注意到 IDT 對 FreeRTOS 原始程式碼的變更，也未建置更新的二進位檔。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。如果發生這種情況，可在日誌中找到下列錯誤訊息：「測試花費不到 60 秒 (URL 到期時間) 即已完成。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*

檢查裝置 ID：

- 若為 Linux/macOS，請開啟終端機並執行 `ls /dev/tty*`。
- 若為 macOS，請開啟終端機並執行 `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：

- 若為 Linux，請確認已安裝 udev 套件，接著執行 `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，請開啟 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，而指派給該裝置的兩個連接埠為 `/dev/ttyUSB0` 和 `/dev/ttyUSB1`，則請在 `device.json` 檔案中使用 `/dev/ttyUSB1`。

  使用 Windows 時，也是遵循相同的邏輯操作。

讀取裝置資料

IDT for FreeRTOS 使用個別裝置建置和快閃記憶體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

這些設定由 IDT for FreeRTOS 處理。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` 命令執行此操作。在 Windows 上，則可使用 TeraTerm 等程式。

```
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 無法建置或刷新您的裝置。
- 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。

日誌

IDT for FreeRTOS 日誌會放置在單一位置。從根 IDT 目錄中，這些檔案可在 `results/`*`execution-id`*`/` 下取得：

- `FRQ_Report.xml`
- `awsiotdevicetester_report.xml`
- `logs/`*`test_group_id`*`__`*`test_case_id`*`.log`

FRQ_Report.xml 和 logs/*test_group_id*__*test_case_id*.log 是要檢查的最重要日誌。FRQ_Report.xml 包含關於哪些測試案例失敗並出現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

AWS IoT Device Tester 執行 時，失敗會以簡短訊息回報至主控台。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。個別測試群組日誌位於 results/*execution-id*/logs 目錄下。使用 IDT for FreeRTOS 主控台的輸出來尋找失敗之測試案例的執行 ID、測試案例 ID 和測試群組 ID。然後使用此資訊來尋找和開啟名為 的測試案例的日誌檔案results/*execution-id*/logs/*test_group_id*__*test_case_id*.log。此檔案中的資訊包含完整的建置和快閃記憶體命令輸出、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

如果您在執行 IDT CTRL+C時按下 ，IDT 將啟動清除程序。清理的一部分是移除已在 IDT 測試中建立的 Amazon S3 資源。如果清除無法完成，您可能會遇到已建立太多 Amazon S3 儲存貯體的問題。這表示您下次執行 IDT 時，測試將會開始失敗。

如果您按 CTRL+C 停止 IDT，則必須讓它完成清除程序以避免此問題。您也可以從您的帳戶刪除手動建立的 Amazon S3 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 FRQ_1.0.1 --pool-id DevicePool1 --timeout-
multiplier 2.5
```

IDT v1.7.0 and earlier

```
./devicetester_linux run-suite --suite-id FRQ_1 --pool-id DevicePool1 --timeout-
multiplier 2.5
```

行動功能和 AWS 費用

當您Yes的 device.JSON 檔案中將Cellular功能設定為 時，FullSecureSockets 將使用 t.micro EC2 執行個體來執行測試，這可能會對 AWS 您的帳戶產生額外費用。如需詳細資訊，請參閱 Amazon EC2 定價。

資格報告產生政策

資格報告僅由 AWS IoT Device Tester (IDT) 版本產生，該版本支援過去兩年內發行的 FreeRTOS 版本。如果您對支援政策有任何疑問，請聯絡 AWS 支援。

適用於 FreeRTOS AWS IoT Device Tester 的 支援政策

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

AWS IoT Device Tester for FreeRTOS 是一種測試自動化工具，可驗證 FreeRTOS 裝置並使其符合包含在 AWS Partner Device Catalog 中的資格。我們建議您使用最新版本的 FreeRTOS 和 AWS IoT Device Tester 來測試或限定您的裝置。我們支援 AWS IoT Device Tester 取得最新版本的 FreeRTOS，以及過去六個月內發行的 FreeRTOS 版本。 AWS IoT Device Tester 支援 LTS 型 FreeRTOS 版本的 版本支援兩年。目前，僅支援 FreeRTOS 202012.00 AWS IoT Device Tester 的 版本支援兩年。最新版本的 FreeRTOS 可在 GitHub 上取得。如需 AWS IoT Device Tester 的支援版本，請參閱 FreeRTOS AWS IoT Device Tester 支援的 版本 FreeRTOS。

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。

您也可以將任何支援的 版本 AWS IoT Device Tester 與對應的 FreeRTOS 版本搭配使用，以測試或限定您的裝置。雖然您可以繼續使用 FreeRTOS 不支援的 IDT 版本，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。

中的安全性 AWS

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

的雲端安全性 AWS 是最高優先順序。身為 AWS 客戶，您可以受益於資料中心和網路架構，這些架構專為滿足最安全敏感組織的需求而建置。

安全性是 AWS 與您之間的共同責任。共同責任模型 將此描述為雲端的安全和雲端內的安全：

- 雲端的安全性 – AWS 負責保護在 AWS Cloud 中執行 AWS 服務的基礎設施。 AWS 也為您提供可安全使用的服務。第三方稽核人員定期檢測及驗證安全的效率也是我們 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。
- 雲端的安全性 – 您的責任取決於您使用 AWS 的服務。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。

本文件將協助您了解如何在使用 時套用共同責任模型 AWS。下列主題說明如何設定 AWS 以符合您的安全與合規目標。您也將了解如何使用可協助您監控和保護 AWS 資源的 AWS 服務。

如需 AWS IoT 安全性的更深入資訊，請參閱 的安全與身分 AWS IoT。

主題

- FreeRTOS 的 Identity and Access Management
- 法規遵循驗證
- 中的彈性 AWS
- FreeRTOS 中的基礎設施安全性

FreeRTOS 的 Identity and Access Management

AWS Identity and Access Management (IAM) 是 AWS 服務 ，可協助管理員安全地控制對 AWS 資源的存取。IAM 管理員可控制誰可以進行身分驗證 （登入） 和授權 （具有許可），以使用 FreeRTOS 資源。IAM 是 AWS 服務 您可以免費使用的 。

主題

- 目標對象
- 使用身分驗證
- 使用政策管理存取權
- FreeRTOS 如何與 IAM 搭配使用
- FreeRTOS 的身分型政策範例
- 疑難排解 FreeRTOS 身分和存取

目標對象

您使用 AWS Identity and Access Management (IAM) 的方式會有所不同，取決於您在 FreeRTOS 中執行的工作。

服務使用者 – 如果您使用 FreeRTOS 服務來執行任務，您的管理員會為您提供所需的登入資料和許可。當您使用更多 FreeRTOS 功能來執行工作時，您可能需要額外的許可。了解存取許可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 FreeRTOS 中的功能，請參閱 疑難排解 FreeRTOS 身分和存取。

服務管理員 – 如果您在公司負責 FreeRTOS 資源，您可能擁有 FreeRTOS 的完整存取權。您的任務是判斷服務使用者應存取的 FreeRTOS 功能和資源。接著，您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。若要進一步了解貴公司如何搭配 FreeRTOS 使用 IAM，請參閱 FreeRTOS 如何與 IAM 搭配使用。

IAM 管理員 – 如果您是 IAM 管理員，建議您了解撰寫政策以管理 FreeRTOS 存取的詳細資訊。若要檢視您可以在 IAM 中使用的 FreeRTOS 身分型政策範例，請參閱 FreeRTOS 的身分型政策範例。

使用身分驗證

身分驗證是您 AWS 使用身分憑證登入 的方式。您必須以 AWS 帳戶根使用者身分、IAM 使用者身分或擔任 IAM 角色來驗證 （ 登入 AWS)。

您可以使用透過身分來源提供的憑證，以聯合身分 AWS 身分身分登入 。 AWS IAM Identity Center (IAM Identity Center) 使用者、您公司的單一登入身分驗證，以及您的 Google 或 Facebook 登入資料，都是聯合身分的範例。您以聯合身分登入時，您的管理員先前已設定使用 IAM 角色的聯合身分。當您使用聯合 AWS 身分存取 時，您會間接擔任角色。

根據您的使用者類型，您可以登入 AWS Management Console 或 AWS 存取入口網站。如需登入的詳細資訊 AWS，請參閱AWS 登入 《 使用者指南》中的如何登入您的 AWS 帳戶 。

如果您以 AWS 程式設計方式存取 ， AWS 會提供軟體開發套件 (SDK) 和命令列界面 (CLI)，以使用您的 憑證以密碼編譯方式簽署您的請求。如果您不使用 AWS 工具，則必須自行簽署請求。如需使用建議的方法自行簽署請求的詳細資訊，請參閱《IAM 使用者指南》中的適用於 API 請求的AWS Signature 第 4 版。

無論您使用何種身分驗證方法，您可能都需要提供額外的安全性資訊。例如， AWS 建議您使用多重驗證 (MFA) 來提高帳戶的安全性。如需更多資訊，請參閱《AWS IAM Identity Center 使用者指南》中的多重要素驗證和《IAM 使用者指南》中的 IAM 中的AWS 多重要素驗證。

AWS 帳戶 根使用者

當您建立 時 AWS 帳戶，您會從一個登入身分開始，該身分可以完整存取帳戶中的所有 AWS 服務 和資源。此身分稱為 AWS 帳戶 Theroot 使用者，可透過使用您用來建立帳戶的電子郵件地址和密碼登入來存取。強烈建議您不要以根使用者處理日常任務。保護您的根使用者憑證，並將其用來執行只能由根使用者執行的任務。如需這些任務的完整清單，了解需以根使用者登入的任務，請參閱 IAM 使用者指南中的需要根使用者憑證的任務。

聯合身分

最佳實務是， 要求人類使用者，包括需要管理員存取權的使用者，使用臨時 AWS 服務 憑證與身分提供者聯合來存取 。

聯合身分是來自您企業使用者目錄、Web 身分提供者、 AWS Directory Service、 Identity Center 目錄或任何使用透過身分來源提供的登入資料 AWS 服務 存取的使用者。當聯合身分存取時 AWS 帳戶，它們會擔任 角色，而角色會提供臨時登入資料。

對於集中式存取權管理，我們建議您使用 AWS IAM Identity Center。您可以在 IAM Identity Center 中建立使用者和群組，也可以連接並同步到您自己的身分來源中的一組使用者 AWS 帳戶 和群組，以便在所有 和應用程式中使用。如需 IAM Identity Center 的詳細資訊，請參閱 AWS IAM Identity Center 使用者指南中的什麼是 IAM Identity Center？。

IAM 使用者和群組

IAM 使用者是 中的身分 AWS 帳戶 ，具有單一人員或應用程式的特定許可。建議您盡可能依賴臨時憑證，而不是擁有建立長期憑證 (例如密碼和存取金鑰) 的 IAM 使用者。但是如果特定使用案例需要擁有長期憑證的 IAM 使用者，建議您輪換存取金鑰。如需更多資訊，請參閱 IAM 使用者指南中的為需要長期憑證的使用案例定期輪換存取金鑰。

IAM 群組是一種指定 IAM 使用者集合的身分。您無法以群組身分簽署。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程序變得更為容易。例如，您可以擁有一個名為 IAMAdmins 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憑證，但角色僅提供臨時憑證。如需更多資訊，請參閱《IAM 使用者指南》中的 IAM 使用者的使用案例。

IAM 角色

IAM 角色是 中具有特定許可 AWS 帳戶 的身分。它類似 IAM 使用者，但不與特定的人員相關聯。若要暫時在 中擔任 IAM 角色 AWS Management Console，您可以從使用者切換至 IAM 角色（主控台）。您可以透過呼叫 AWS CLI 或 AWS API 操作或使用自訂 URL 來擔任角色。如需使用角色的方法詳細資訊，請參閱《IAM 使用者指南》中的擔任角色的方法。

使用臨時憑證的 IAM 角色在下列情況中非常有用：

- 聯合身分使用者存取 — 如需向聯合身分指派許可，請建立角色，並為角色定義許可。當聯合身分進行身分驗證時，該身分會與角色建立關聯，並獲授予由角色定義的許可。如需有關聯合角色的相關資訊，請參閱《IAM 使用者指南》中的為第三方身分提供者 (聯合) 建立角色。如果您使用 IAM Identity Center，則需要設定許可集。為控制身分驗證後可以存取的內容，IAM Identity Center 將許可集與 IAM 中的角色相關聯。如需有關許可集的資訊，請參閱 AWS IAM Identity Center 使用者指南中的許可集。
- 暫時 IAM 使用者許可 – IAM 使用者或角色可以擔任 IAM 角色來暫時針對特定任務採用不同的許可。
- 跨帳戶存取權：您可以使用 IAM 角色，允許不同帳戶中的某人 (信任的主體) 存取您帳戶的資源。角色是授予跨帳戶存取權的主要方式。不過，對於某些 AWS 服務，您可以直接將政策連接到資源（而不是使用角色做為代理）。如需了解使用角色和資源型政策進行跨帳戶存取之間的差異，請參閱《IAM 使用者指南》中的 IAM 中的跨帳戶資源存取。
- 跨服務存取 – 有些 AWS 服務 使用其他 中的功能 AWS 服務。例如，當您在服務中進行呼叫時，該服務通常會在 Amazon EC2 中執行應用程式或將物件儲存在 Amazon Simple Storage Service (Amazon S3) 中。服務可能會使用呼叫主體的許可、使用服務角色或使用服務連結角色來執行此作業。
  - 轉送存取工作階段 (FAS) – 當您使用 IAM 使用者或角色在其中執行動作時 AWS，您會被視為委託人。使用某些服務時，您可能會執行某個動作，進而在不同服務中啟動另一個動作。FAS 使用呼叫 的委託人許可 AWS 服務，結合 AWS 服務 請求向下游服務提出請求。只有當服務收到需要與其他 AWS 服務 或 資源互動才能完成的請求時，才會提出 FAS 請求。在此情況下，您必須具有執行這兩個動作的許可。如需提出 FAS 請求時的政策詳細資訊，請參閱《轉發存取工作階段》。
  - 服務角色 – 服務角色是服務擔任的 IAM 角色，可代表您執行動作。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》中的建立角色以委派許可權給 AWS 服務。

- 服務連結角色 – 服務連結角色是連結至 的一種服務角色 AWS 服務。服務可以擔任代表您執行動作的角色。服務連結角色會出現在您的 中 AWS 帳戶 ，並由服務擁有。IAM 管理員可以檢視，但不能編輯服務連結角色的許可。
- 在 Amazon EC2 上執行的應用程式 – 您可以使用 IAM 角色來管理在 EC2 執行個體上執行之應用程式的臨時登入資料，以及提出 AWS CLI 或 AWS API 請求。這是在 EC2 執行個體內儲存存取金鑰的較好方式。若要將 AWS 角色指派給 EC2 執行個體並將其提供給其所有應用程式，您可以建立連接至執行個體的執行個體描述檔。執行個體設定檔包含該角色，並且可讓 EC2 執行個體上執行的程式取得臨時憑證。如需詳細資訊，請參閱《IAM 使用者指南》中的使用 IAM 角色來授予許可權給 Amazon EC2 執行個體上執行的應用程式。

使用政策管理存取權

您可以透過建立政策並將其連接到身分或資源 AWS 來控制 AWS 中的存取。政策是 中的物件，當與身分或資源相關聯時， AWS 會定義其許可。當委託人（使用者、根使用者或角色工作階段）發出請求時， 會 AWS 評估這些政策。政策中的許可決定是否允許或拒絕請求。大多數政策會以 JSON 文件 AWS 的形式存放在 中。如需 JSON 政策文件結構和內容的詳細資訊，請參閱 IAM 使用者指南中的 JSON 政策概觀。

管理員可以使用 AWS JSON 政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

預設情況下，使用者和角色沒有許可。若要授予使用者對其所需資源執行動作的許可，IAM 管理員可以建立 IAM 政策。然後，管理員可以將 IAM 政策新增至角色，使用者便能擔任這些角色。

IAM 政策定義該動作的許可，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。具有該政策的使用者可以從 AWS Management Console AWS CLI、 或 AWS API 取得角色資訊。

身分型政策

身分型政策是可以附加到身分 (例如 IAM 使用者、使用者群組或角色) 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。如需了解如何建立身分型政策，請參閱《IAM 使用者指南》中的透過客戶管理政策定義自訂 IAM 許可。

身分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。受管政策是獨立的政策，您可以連接到 中的多個使用者、群組和角色 AWS 帳戶。受管政策包括 AWS 受管政策和客戶受管政策。如需了解如何在受管政策及內嵌政策之間選擇，請參閱《IAM 使用者指南》中的在受管政策和內嵌政策間選擇。

資源型政策

資源型政策是連接到資源的 JSON 政策文件。資源型政策的最常見範例是 IAM 角色信任政策和 Amazon S3 儲存貯體政策。在支援資源型政策的服務中，服務管理員可以使用它們來控制對特定資源的存取權限。對於附加政策的資源，政策會定義指定的主體可以對該資源執行的動作以及在何種條件下執行的動作。您必須在資源型政策中指定主體。委託人可以包含帳戶、使用者、角色、聯合身分使用者或 AWS 服務。

資源型政策是位於該服務中的內嵌政策。您無法在資源型政策中使用來自 IAM 的 AWS 受管政策。

存取控制清單 (ACL)

存取控制清單 (ACL) 可控制哪些主體 (帳戶成員、使用者或角色) 擁有存取某資源的許可。ACL 類似於資源型政策，但它們不使用 JSON 政策文件格式。

Amazon S3 AWS WAF和 Amazon VPC 是支援 ACLs的服務範例。如需進一步了解 ACL，請參閱 Amazon Simple Storage Service 開發人員指南中的存取控制清單 (ACL) 概觀。

其他政策類型

AWS 支援其他較不常見的政策類型。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 – 許可範圍是一種進階功能，可供您設定身分型政策能授予 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，請參閱 IAM 使用者指南中的 IAM 實體許可界限。
- 服務控制政策 SCPs) – SCPs是 JSON 政策，可指定 中組織或組織單位 (OU) 的最大許可 AWS Organizations。 AWS Organizations 是一種用於分組和集中管理您企業擁有 AWS 帳戶 之多個 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。SCP 會限制成員帳戶中實體的許可，包括每個實體 AWS 帳戶根使用者。如需 Organizations 和 SCP 的詳細資訊，請參閱《AWS Organizations 使用者指南》中的服務控制政策。
- 資源控制政策 (RCP) - RCP 是 JSON 政策，可用來設定您帳戶中資源的可用許可上限，採取這種方式就不需要更新附加至您所擁有的每個資源的 IAM 政策。RCP 會限制成員帳戶中資源的許可，並可能影響身分的有效許可，包括 AWS 帳戶根使用者，無論它們是否屬於您的組織。如需 Organizations 和 RCPs的詳細資訊，包括支援 RCPs AWS 服務 的 清單，請參閱AWS Organizations 《 使用者指南》中的資源控制政策 (RCPs)。
- 工作階段政策 – 工作階段政策是一種進階政策，您可以在透過撰寫程式的方式建立角色或聯合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型政策和工作

階段政策的交集。許可也可以來自資源型政策。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，請參閱 IAM 使用者指南中的工作階段政策。

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。若要了解如何 AWS 決定是否在涉及多個政策類型時允許請求，請參閱《IAM 使用者指南》中的政策評估邏輯。

FreeRTOS 如何與 IAM 搭配使用

在您使用 IAM 管理 FreeRTOS 的存取權之前，請先了解哪些 IAM 功能可與 FreeRTOS 搭配使用。

您可以搭配 FreeRTOS 使用的 IAM 功能

IAM 功能	FreeRTOS 支援
身分型政策	是
資源型政策	否
政策動作	是
政策資源	是
政策條件索引鍵 (服務特定)	是
ACL	否
ABAC(政策中的標籤)	部分
臨時憑證	是
主體許可	是
服務角色	是
服務連結角色	否

若要全面了解 FreeRTOS 和其他 AWS 服務如何與大多數 IAM 功能搭配使用，請參閱《AWS IAM 使用者指南》中的與 IAM 搭配使用的 服務。

FreeRTOS 的身分型政策

支援身分型政策：是

身分型政策是可以附加到身分 (例如 IAM 使用者、使用者群組或角色) 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。如需了解如何建立身分型政策，請參閱《IAM 使用者指南》中的透過客戶管理政策定義自訂 IAM 許可。

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。您無法在身分型政策中指定主體，因為這會套用至連接的使用者或角色。如要了解您在 JSON 政策中使用的所有元素，請參閱《IAM 使用者指南》中的 IAM JSON 政策元素參考。

FreeRTOS 的身分型政策範例

若要檢視 FreeRTOS 身分型政策的範例，請參閱 FreeRTOS 的身分型政策範例。

FreeRTOS 中的資源型政策

支援資源型政策：否

資源型政策是附加到資源的 JSON 政策文件。資源型政策的最常見範例是 IAM 角色信任政策和 Amazon S3 儲存貯體政策。在支援資源型政策的服務中，服務管理員可以使用它們來控制對特定資源的存取權限。對於附加政策的資源，政策會定義指定的主體可以對該資源執行的動作以及在何種條件下執行的動作。您必須在資源型政策中指定主體。委託人可以包含帳戶、使用者、角色、聯合身分使用者或 AWS 服務。

如需啟用跨帳戶存取權，您可以指定在其他帳戶內的所有帳戶或 IAM 實體，做為資源型政策的主體。新增跨帳戶主體至資源型政策，只是建立信任關係的一半。當委託人和資源位於不同位置時 AWS 帳戶，信任帳戶中的 IAM 管理員也必須授予委託人實體 （使用者或角色） 存取資源的許可。其透過將身分型政策連接到實體來授與許可。不過，如果資源型政策會為相同帳戶中的主體授予存取，這時就不需要額外的身分型政策。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM 中的快帳戶資源存取。

FreeRTOS 的政策動作

支援政策動作：是

管理員可以使用 AWS JSON 政策來指定誰可以存取內容。也就是說，哪個主體在什麼条件下可以對什麼資源執行哪些動作。

JSON 政策的 Action 元素描述您可以用來允許或拒絕政策中存取的動作。政策動作通常具有與相關聯 AWS API 操作相同的名稱。有一些例外狀況，例如沒有相符的 API 操作的僅限許可動作。也有一些作業需要政策中的多個動作。這些額外的動作稱為相依動作。

政策會使用動作來授予執行相關聯動作的許可。

若要查看 FreeRTOS 動作清單，請參閱《服務授權參考》中的 FreeRTOS 定義的動作。

FreeRTOS 中的政策動作在動作之前使用以下字首：

```
awes
```

若要在單一陳述式中指定多個動作，請用逗號分隔。

```
"Action": [
      "awes:action1",
      "awes:action2"
         ]
```

若要檢視 FreeRTOS 身分型政策的範例，請參閱 FreeRTOS 的身分型政策範例。

FreeRTOS 的政策資源

支援政策資源：是

管理員可以使用 AWS JSON 政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

Resource JSON 政策元素可指定要套用動作的物件。陳述式必須包含 Resource 或 NotResource 元素。最佳實務是使用其 Amazon Resource Name (ARN) 來指定資源。您可以針對支援特定資源類型的動作 (稱為資源層級許可) 來這麼做。

對於不支援資源層級許可的動作 (例如列出操作)，請使用萬用字元 (*) 來表示陳述式適用於所有資源。

```
"Resource": "*"
```

若要查看 FreeRTOS 資源類型及其 ARNs，請參閱《服務授權參考》中的 FreeRTOS 定義的資源。若要了解您可以使用哪些動作指定每個資源的 ARN，請參閱 FreeRTOS 定義的動作。

若要檢視 FreeRTOS 身分型政策的範例，請參閱 FreeRTOS 的身分型政策範例。

FreeRTOS 的政策條件索引鍵

支援服務特定政策條件金鑰：是

管理員可以使用 AWS JSON 政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

Condition 元素 (或 Condition 區塊) 可讓您指定使陳述式生效的條件。Condition 元素是選用項目。您可以建立使用條件運算子的條件運算式 (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 元素中指定多個索引鍵， AWS 會使用邏輯 AND 操作評估他們。如果您為單一條件索引鍵指定多個值， 會使用邏輯OR操作 AWS 評估條件。必須符合所有條件，才會授與陳述式的許可。

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。如需更多資訊，請參閱 IAM 使用者指南中的 IAM 政策元素：變數和標籤。

AWS 支援全域條件金鑰和服務特定條件金鑰。若要查看所有 AWS 全域條件索引鍵，請參閱《IAM 使用者指南》中的AWS 全域條件內容索引鍵。

若要查看 FreeRTOS 條件金鑰清單，請參閱服務授權參考中的 FreeRTOS 條件金鑰。若要了解您可以使用條件金鑰的動作和資源，請參閱 FreeRTOS 定義的動作。

若要檢視 FreeRTOS 身分型政策的範例，請參閱 FreeRTOS 的身分型政策範例。

FreeRTOS ACLs

支援 ACL：否

存取控制清單 (ACL) 可控制哪些主體 (帳戶成員、使用者或角色) 擁有存取某資源的許可。ACL 類似於資源型政策，但它們不使用 JSON 政策文件格式。

ABAC 搭配 FreeRTOS

支援 ABAC (政策中的標籤)：部分

屬性型存取控制 (ABAC) 是一種授權策略，可根據屬性來定義許可。在 中 AWS，這些屬性稱為標籤。您可以將標籤連接到 IAM 實體 （使用者或角色） 和許多 AWS 資源。為實體和資源加上標籤是 ABAC 的第一步。您接著要設計 ABAC 政策，允許在主體的標籤與其嘗試存取的資源標籤相符時操作。

ABAC 在成長快速的環境中相當有幫助，並能在政策管理變得繁瑣時提供協助。

如需根據標籤控制存取，請使用 `aws:ResourceTag/key-name`、`aws:RequestTag/key-name` 或 `aws:TagKeys` 條件索引鍵，在政策的條件元素中，提供標籤資訊。

如果服務支援每個資源類型的全部三個條件金鑰，則對該服務而言，值為 Yes。如果服務僅支援某些資源類型的全部三個條件金鑰，則值為 Partial。

如需 ABAC 的詳細資訊，請參閱《IAM 使用者指南》中的使用 ABAC 授權定義許可。如要查看含有設定 ABAC 步驟的教學課程，請參閱 IAM 使用者指南中的使用屬性型存取控制 (ABAC)。

搭配 FreeRTOS 使用臨時登入資料

支援臨時憑證：是

當您使用臨時登入資料登入時，有些 AWS 服務 無法運作。如需詳細資訊，包括哪些 AWS 服務 使用臨時登入資料，請參閱《AWS 服務 IAM 使用者指南》中的 使用 IAM。

如果您 AWS Management Console 使用使用者名稱和密碼以外的任何方法登入 ，則會使用臨時登入資料。例如，當您 AWS 使用公司的單一登入 (SSO) 連結存取 時，該程序會自動建立臨時登入資料。當您以使用者身分登入主控台，然後切換角色時，也會自動建立臨時憑證。如需切換角色的詳細資訊，請參閱《IAM 使用者指南》中的從使用者切換至 IAM 角色 (主控台)。

您可以使用 AWS CLI 或 AWS API 手動建立臨時登入資料。然後，您可以使用這些臨時登入資料來存取 AWS。 AWS 建議您動態產生臨時登入資料，而不是使用長期存取金鑰。如需詳細資訊，請參閱 IAM 中的暫時性安全憑證。

FreeRTOS 的跨服務主體許可

支援轉寄存取工作階段 (FAS)：是

當您使用 IAM 使用者或角色在 中執行動作時 AWS，您會被視為委託人。使用某些服務時，您可能會執行某個動作，進而在不同服務中啟動另一個動作。FAS 使用呼叫 的委託人許可 AWS 服務，結合 AWS 服務 請求向下游服務提出請求。只有在服務收到需要與其他 AWS 服務 或 資源互動才能完成的請求時，才會提出 FAS 請求。在此情況下，您必須具有執行這兩個動作的許可。如需提出 FAS 請求時的政策詳細資訊，請參閱轉發存取工作階段。

FreeRTOS 的服務角色

支援服務角色：是

服務角色是服務擔任的 IAM 角色，可代您執行動作。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》中的建立角色以委派許可權給 AWS 服務。

> **Warning**
>
> 變更服務角色的許可可能會中斷 FreeRTOS 功能。只有在 FreeRTOS 提供指引時，才能編輯服務角色。

FreeRTOS 的服務連結角色

支援服務連結角色：否

服務連結角色是連結至 的一種服務角色 AWS 服務。服務可以擔任代表您執行動作的角色。服務連結角色會出現在您的 中 AWS 帳戶 ，並由服務擁有。IAM 管理員可以檢視，但不能編輯服務連結角色的許可。

如需建立或管理服務連結角色的詳細資訊，請參閱可搭配 IAM 運作的AWS 服務。在表格中尋找服務，其中包含服務連結角色欄中的 Yes。選擇是連結，以檢視該服務的服務連結角色文件。

FreeRTOS 的身分型政策範例

根據預設，使用者和角色沒有建立或修改 FreeRTOS 資源的許可。他們也無法使用 AWS Management Console、 AWS Command Line Interface (AWS CLI) 或 AWS API 來執行任務。若要授予使用者對其所需資源執行動作的許可，IAM 管理員可以建立 IAM 政策。然後，管理員可以將 IAM 政策新增至角色，使用者便能擔任這些角色。

如需了解如何使用這些範例 JSON 政策文件建立 IAM 身分型政策，請參閱《IAM 使用者指南》中的建立 IAM 政策 (主控台)。

如需有關 FreeRTOS 定義的動作和資源類型的詳細資訊，包括每種資源類型的 ARNs 格式，請參閱《服務授權參考》中的 FreeRTOS 的動作、資源和條件索引鍵。

主題

- 政策最佳實務
- 使用 FreeRTOS 主控台

- 允許使用者檢視他們自己的許可

政策最佳實務

身分型政策會判斷您帳戶中的某個人員是否可以建立、存取或刪除 FreeRTOS 資源。這些動作可能會讓您的 AWS 帳戶產生費用。當您建立或編輯身分型政策時，請遵循下列準則及建議事項：

- 開始使用 AWS 受管政策並邁向最低權限許可 – 若要開始將許可授予您的使用者和工作負載，請使用將許可授予許多常見使用案例的 AWS 受管政策。它們可在您的 中使用 AWS 帳戶。我們建議您定義特定於使用案例 AWS 的客戶受管政策，以進一步減少許可。如需更多資訊，請參閱 IAM 使用者指南中的 AWS 受管政策或任務職能的AWS 受管政策。
- 套用最低權限許可 – 設定 IAM 政策的許可時，請僅授予執行任務所需的許可。為實現此目的，您可以定義在特定條件下可以對特定資源採取的動作，這也稱為最低權限許可。如需使用 IAM 套用許可的更多相關資訊，請參閱 IAM 使用者指南中的 IAM 中的政策和許可。
- 使用 IAM 政策中的條件進一步限制存取權 – 您可以將條件新增至政策，以限制動作和資源的存取。例如，您可以撰寫政策條件，指定必須使用 SSL 傳送所有請求。如果透過特定 例如 使用服務動作 AWS 服務，您也可以使用條件來授予其存取權 AWS CloudFormation。如需詳細資訊，請參閱 IAM 使用者指南中的 IAM JSON 政策元素：條件。
- 使用 IAM Access Analyzer 驗證 IAM 政策，確保許可安全且可正常運作 – IAM Access Analyzer 驗證新政策和現有政策，確保這些政策遵從 IAM 政策語言 (JSON) 和 IAM 最佳實務。IAM Access Analyzer 提供 100 多項政策檢查及切實可行的建議，可協助您撰寫安全且實用的政策。如需詳細資訊，請參閱《IAM 使用者指南》中的使用 IAM Access Analyzer 驗證政策。
- 需要多重要素驗證 (MFA) – 如果您的案例需要 IAM 使用者或 中的根使用者 AWS 帳戶，請開啟 MFA 以提高安全性。如需在呼叫 API 操作時請求 MFA，請將 MFA 條件新增至您的政策。如需詳細資訊，請參閱《IAM 使用者指南》https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_mfa_configure-api-require.html中的透過 MFA 的安全 API 存取。

如需 IAM 中最佳實務的相關資訊，請參閱 IAM 使用者指南中的 IAM 安全最佳實務。

使用 FreeRTOS 主控台

若要存取 FreeRTOS 主控台，您必須擁有一組最低許可。這些許可必須允許您列出和檢視 中 FreeRTOS 資源的詳細資訊 AWS 帳戶。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 (使用者或角色) 而言，主控台就無法如預期運作。

對於僅呼叫 AWS CLI 或 AWS API 的使用者，您不需要允許最低主控台許可。反之，只需允許存取符合他們嘗試執行之 API 操作的動作就可以了。

為了確保使用者和角色仍然可以使用 FreeRTOS 主控台，請將 FreeRTOS *ConsoleAccess*或*ReadOnly* AWS 受管政策連接到實體。如需詳細資訊，請參閱《IAM 使用者指南》中的新增許可到使用者。

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 使用者檢視附加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或使用 或 AWS CLI AWS API 以程式設計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": "*"
        }
    ]
}
```

疑難排解 FreeRTOS 身分和存取

使用以下資訊來協助您診斷和修正使用 FreeRTOS 和 IAM 時可能遇到的常見問題。

主題

- 我無權在 FreeRTOS 中執行動作
- 我未獲得執行 iam:PassRole 的授權
- 我想要允許 以外的人員 AWS 帳戶 存取我的 FreeRTOS 資源

我無權在 FreeRTOS 中執行動作

如果您收到錯誤，告知您未獲授權執行動作，您的政策必須更新，允許您執行動作。

下列範例錯誤會在mateojackson IAM 使用者嘗試使用主控台檢視一個虛構 *my-example-widget* 資源的詳細資訊，但卻無虛構 awes:*GetWidget* 許可時發生。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 awes:GetWidget on resource: my-example-widget
```

在此情況下，必須更新 mateojackson 使用者的政策，允許使用 awes:*GetWidget* 動作存取 *my-example-widget* 資源。

如果您需要協助，請聯絡您的 AWS 管理員。您的管理員提供您的簽署憑證。

我未獲得執行 iam:PassRole 的授權

如果您收到錯誤，告知您無權執行 iam:PassRole動作，您的政策必須更新，以允許您將角色傳遞至 FreeRTOS。

有些 AWS 服務 可讓您將現有角色傳遞給該服務，而不是建立新的服務角色或服務連結角色。如需執行此作業，您必須擁有將角色傳遞至該服務的許可。

當名為 的 IAM marymajor 使用者嘗試使用主控台在 FreeRTOS 中執行動作時，會發生下列範例錯誤。但是，動作請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 的政策必須更新，允許她執行 `iam:PassRole` 動作。

如果您需要協助，請聯絡您的 AWS 管理員。您的管理員提供您的簽署憑證。

我想要允許 以外的人員 AWS 帳戶 存取我的 FreeRTOS 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物件取得該角色。針對支援基於資源的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的資源的許可。

如需進一步了解，請參閱以下內容：

- 若要了解 FreeRTOS 是否支援這些功能，請參閱 FreeRTOS 如何與 IAM 搭配使用。
- 若要了解如何 AWS 帳戶 在您擁有的 資源之間提供存取權，請參閱《IAM 使用者指南》中的在您擁有 AWS 帳戶 的另一個 中提供存取權給 IAM 使用者。
- 若要了解如何將資源的存取權提供給第三方 AWS 帳戶，請參閱《IAM 使用者指南》中的將存取權提供給第三方 AWS 帳戶 擁有。
- 如需了解如何透過聯合身分提供存取權，請參閱 IAM 使用者指南中的將存取權提供給在外部進行身分驗證的使用者 (聯合身分)。
- 如需了解使用角色和資源型政策進行跨帳戶存取之間的差異，請參閱《IAM 使用者指南》中的 IAM 中的跨帳戶資源存取。

法規遵循驗證

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

若要了解 是否 AWS 服務 在特定合規計劃的範圍內，請參閱AWS 服務 合規計劃範圍內然後選擇您感興趣的合規計劃。如需一般資訊，請參閱 AWS Compliance Programs。

您可以使用 下載第三方稽核報告 AWS Artifact。如需詳細資訊，請參閱在 中下載報告 AWS Artifact。

您使用 時的合規責任 AWS 服務 取決於資料的機密性、您公司的合規目標，以及適用的法律和法規。AWS 提供下列資源來協助合規：

- 安全合規與治理 - 這些解決方案實作指南內容討論了架構考量，並提供部署安全與合規功能的步驟。
- HIPAA 合格服務參考 – 列出 HIPAA 合格服務。並非所有 AWS 服務 都符合 HIPAA 資格。
- AWS 合規資源 – 此工作手冊和指南集合可能適用於您的產業和位置。
- AWS 客戶合規指南 – 透過合規的角度了解共同責任模型。本指南摘要說明跨多個架構 （ 包括國家標準技術研究所 (NIST)、支付卡產業安全標準委員會 (PCI) 和國際標準化組織 (ISO)) 保護 AWS 服務和映射指南至安全控制的最佳實務。
- 《 AWS Config 開發人員指南》中的使用規則評估資源 – AWS Config 服務會評估資源組態符合內部實務、產業準則和法規的程度。
- AWS Security Hub – 這 AWS 服務 可讓您全面檢視其中的安全狀態 AWS。Security Hub 使用安全控制，可評估您的 AWS 資源並檢查您的法規遵循是否符合安全業界標準和最佳實務。如需支援的服務和控制清單，請參閱「Security Hub 控制參考」。
- Amazon GuardDuty – 這會監控您的環境是否有可疑和惡意活動，以 AWS 服務 偵測對您 AWS 帳戶、工作負載、容器和資料的潛在威脅。GuardDuty 可滿足特定合規架構所規定的入侵偵測需求，以協助您因應 PCI DSS 等各種不同的合規需求。
- AWS Audit Manager – 這 AWS 服務 可協助您持續稽核 AWS 用量，以簡化您管理風險的方式，以及符合法規和業界標準的方式。

中的彈性 AWS

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

AWS 全球基礎設施是以 AWS 區域和可用區域為基礎建置。 AWS 區域提供多個分開且隔離的實際可用區域，它們以低延遲、高輸送量和高度備援聯網功能相互連結。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 區域和可用區域的詳細資訊，請參閱 AWS 全球基礎設施。

FreeRTOS 中的基礎設施安全性

> **Important**
>
> 這是 FreeRTOS 使用者指南的封存版本，可與 FreeRTOS 版本 202012.00 搭配使用。如需本文件的最新版本，請參閱 FreeRTOS 使用者指南。

AWS 受管服務受到 Amazon Web Services：安全程序概觀白皮書中所述 AWS 的全球網路安全程序的保護。

您可以使用 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) 1.2 或更新版本。建議使用 TLS 1.3 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 主體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) 以產生暫時安全憑證以簽署請求。

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。