aws

使用者指南

FreeRTOS

FreeRTOS: 使用者指南

Table of Contents

什麼是 FreeRTOS？ 1
下載 FreeRTOS 的原始程式碼 1
自由伺服器版本控制 1
FreeRTOS 長期 Support 1
FreeRTOS 維護計劃 2
FreeRTOS 架構 2
符合免費使用者資格的硬體平台 3
開發工作流程 4
其他資源 4
FreeRTOS 核心基礎 6
FreeRTOS 核心排程器 6
記憶體管理 6
核心記憶體配置 7
應用程式記憶體管理 7
任務間協調 8
佇列 8
旗號與 Mutex 8
Direct-to-task 通知 9
串流緩衝區 9
訊息緩衝區 10
對稱式多重處理 (SMP) 支援 11
修改應用程式以使用自由式 SMP 核心 12
軟體計時器 12
低電力支援 12
FreeRTOSConfig.h 13
適用於 Embedded C 的 AWS IoT 裝置 SDK 14
通用 IO 15
Libraries (程式庫) 15
通用 IO-基本 15
通用 IO 16
亞馬遜通用軟件的通用 IO 17
什麼是 ACS？ 17
資格計劃 17
FreeRTOS 入門 18

使用快速 Connect 開始使用AWS IoT和 FreeRTOS 18
FreeRTOS 無線程式庫 18
了解如何構建安全可靠的AWS IoT產品 18
開發您的AWS IoT應用產品 19
AWS IoT Device Tester對於免費服務 20
自由服務資格套件 20
自訂測試套件 21
FreeRTOS 支援的 IDT 版本 21
最新版本的免 FreeRTOS 22
舊版 IDT 23
不支援的 IDT 版本 27
下載 FreeRTOS 的 IDT 51
手動下載 IDT 52
以編程方式下載 IDT 52
使用 IDT 搭配 FreeRTOS 資格套件 2.0 (FRQ 2.0) 57
先決條件 58
首次準備測試微型控制器主機板 67
使用 IDT 使用者介面來執行 FreeRTOS 資格套件 81
執行 FreeRTOS 資格 2.0 套件 96
了解結果和日誌 99
將 IDT 與 FreeRTOS 資格套件 1.0 搭配使用 (FRQ 1.0) 103
必要條件 104
首次準備測試微型控制器主機板 108
使用 IDT 使用者介面來執行 FreeRTOS 資格套件 126
執行低功耗藍牙測試 135
執行 FreeRTOS 資格套件 140
了解結果和日誌 145
使用 IDT 開發和運行自己的測試套件 149
下載最新版本的 IDT 免 FreeRTOS 150
測試套件建立工作流 150
教學課程：建置並執行範例 IDT 測試套件 151
教學課程：開發簡單的 IDT 測試套件 156
測試套件版本 232
疑難排解 233
故障診斷裝置組態 234
故障診斷逾時錯誤 245

流動網絡功能和AWS收費 245
資格報告生成方針 246
AWS受管理的政策AWS IoT Device Tester 246
受管政策 246
政策更新 253
支援政策 255
中的安全性 AWS 256
身分和存取權管理 256
物件 257
使用身分驗證 257
使用政策管理存取權 260
FreeRTOS 務如何與 IAM 搭配使用 262
身分型政策範例 267
故障診斷 270
法規遵循驗證 271
恢復能力 272
基礎架構安全 273
亞馬遜自由 Github 存儲庫遷移指南 274
附錄 274
存檔 279
自由使用者指南存檔 279
上一頁《自由服務指南》的內容 279
FreeRTOS 入 279
無線更新 458
FreeRTOS 式庫 534
FreeRTOS 示 592
..... dccx

什麼是 FreeRTOS？

FreeRTOS 是與全球領先晶片公司合作開發的 15 年期間，現在每 170 秒下載一次，是市場領先的微控制器和小型微處理器即時作業系統 (RTOS)。FreeRTOS 在 MIT 開放原始碼授權下自由發佈，包含一個核心和一組不斷增長的程式庫，適合所有產業領域使用。FreeRTOS 的構建重點是可靠性和易用性。

FreeRTOS 包含用於連線、安全性和 over-the-air (OTA) 更新的程式庫。FreeRTOS 也包含示範應用程式，可在合格的主機板上顯示 FreeRTOS 功能。

自由軟體是一個開放原始碼的專案。您可以在 https://github.com/FreeRTOS/FreeRTOS 上下載源代碼，進行更改或增強功能，或在 GitHub 網站上報告問題。

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。

我們也歡迎您對 FreeRTOS 文件的貢獻 (FreeRTOS 使用者指南、FreeRTOS 移植指南和 FreeRTOS 資格指南)。若要檢視文件的降價來源，請參閱 https://github.com/awsdocs/aws-freertos-docs。它是根據創用 CC（CC BY-ND）許可證發布的。

下載 FreeRTOS 的原始程式碼

從 freertos.org 的下載頁面下載最新的 FreeRTOS 和長期 Support (LTS) 套件。

自由伺服器版本控制

單個庫使用 x.y.z 樣式的版本號，類似於語義版本。x 是主要版本號，y 是次要版本號，從 2022 開始，z 是補丁號。在 2022 年之前，z 是一個點發布號，這需要第一個 LTS 庫具有形式「x.y.z LTS 補丁 2」的補丁號。

圖書館套件使用 yyyymm.x 樣式的日期戳版本號。yyyy 是年份，mm 是月份，x 是顯示月份內發行順序的可選序號。就 LTS 套件而言，x 是該 LTS 發行版本的連續修補程式編號。套件中包含的個別程式庫是該程式庫在該日期的最新版本。對於 LTS 軟件包，它是最初作為 LTS 版本在該日期發布的 LTS 庫的最新補丁版本。

FreeRTOS 長期 Support

FreeRTOS 長期 Support (LTS) 發行版本會在發行後至少兩年內收到安全性和重要錯誤修正 (應該是必要的)。透過這項持續的維護，您可以在整個開發和部署週期中整合錯誤修正，而不會因為更新到 FreeRTOS 程式庫的新主要版本而造成昂貴的中斷。

使用 FreeRTOS LTS，您可以獲得構建安全連接 IoT 和嵌入式產品所需的完整庫集。LTS 有助於降低與在生產中的設備上更新庫相關的維護和測試成本。

FreeRTOS LTS 包括 FreeRTOS 費伺服器核心和 IoT 程式庫：自由軟體 + TCP、核心 QTT、CorekCS11、CorekCS11、OTA、工作和 Device Shadow。 AWS IoT AWS IoT AWS IoT Device Defender AWS IoT 如需詳細資訊，請參閱 FreeRTOS LTS 程式庫。

FreeRTOS 維護計劃

AWS 此外還提供 FreeRTOS 延伸維護計畫 (EMP)，可在您選擇的 FreeRTOS 長期 Support (LTS) 版本上提供安全性修補程式和重大錯誤修正，最長可延長 10 年。使用 FreeRTOS EMP，您以 FreeRTOS 為基礎的長壽裝置可以仰賴具有功能穩定性並接收多年安全性更新的版本。您會及時收到 FreeRTOS 程式庫上即將進行修補程式的通知，因此您可以規劃在物聯網 (IoT) 裝置上部署安全性修補程式。

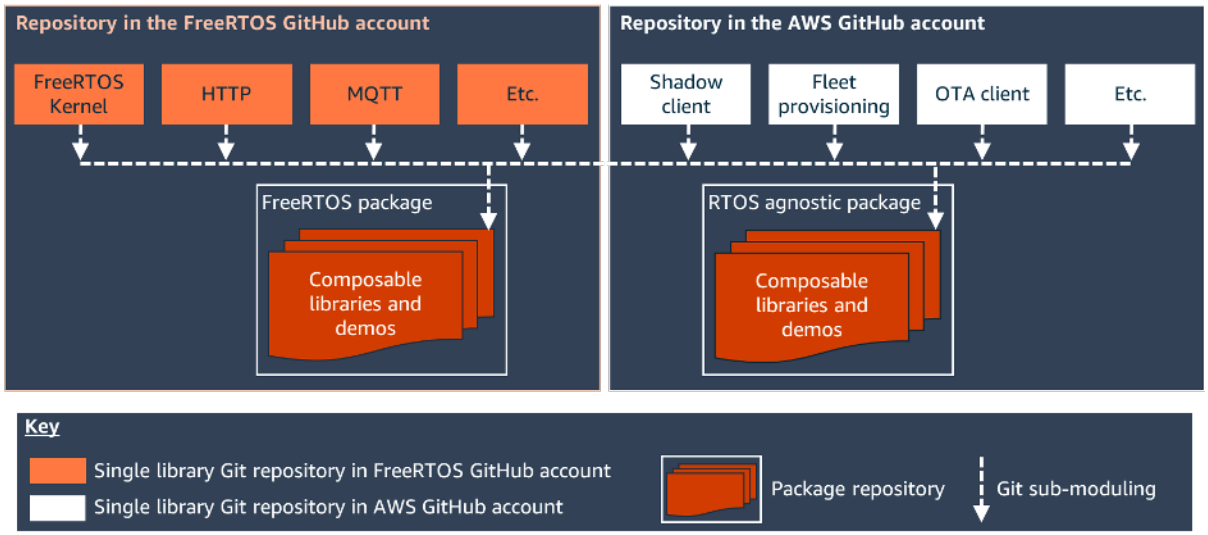
若要進一步了解 FreeRTOS EMP，請參閱功能頁面。

FreeRTOS 架構

FreeRTOS 包含兩種類型的存放庫，單一程式庫存放庫和套件存放庫。每個庫存儲庫都包含一個庫的源代碼，沒有任何構建項目或示例。Package 儲存庫包含多個程式庫，並且可以包含展示程式庫使用情況的預先設定專案。

雖然套件儲存庫包含多個程式庫，但它們不包含這些程式庫的複本。相反，包存儲庫將它們包含的庫引用為 git 子模塊。使用子模塊可確保每個單獨的庫都有一個真實來源。

個別程式庫 git 儲存庫會在兩個 GitHub 組織之間分割。包含 FreeRTOS 特定程式庫 (例如 FreeRTOS 的 +TCP) 或一般程式庫 (例如 CoremQTT，這是雲端不可知的，因為它適用於任何 MQTT 代理程式) 的儲存庫位於 FreeRTOS 組織中。 GitHub 包含 AWS IoT 特定程式庫 (例如 AWS IoT over-the-air 更新用戶端) 的儲存庫位於 AWS GitHub組織中。下圖說明了結構。



符合免費使用者資格的硬體平台

下列硬體平台適用於 FreeRTOS：

- ATECC608A 零接觸佈建套件，適用於 AWS IoT
- Cypress CYW943907AEVAL1F 開發套件
- Cypress CYW954907AEVAL1F 開發套件
- 柏木 CY8CKIT-064S0S2-4343W 套裝
- 濃縮咖啡 ESP32-C DevKit
- Espressif ESP-WROVER-KIT
- 義式濃縮咖啡 ESP-WROOM-32SE
- 意式濃縮咖啡 ESP32-S2-索拉 -1
- 英飛凌 XMC4800 IoT 連接套件
- 瑪維爾 MW320 AWS IoT 入門套件
- 瑪維爾 MW322 AWS IoT 入門套件
- MediaTek 開發套件
- 微晶片好奇心 PIC32MZEF 組合包
- Nordic nRF52840-DK
- NuMaker-物聯網/物聯網M487
- NXP LPC54018 IoT 模組
- 奧迪嘉信任 X 安全解決方案

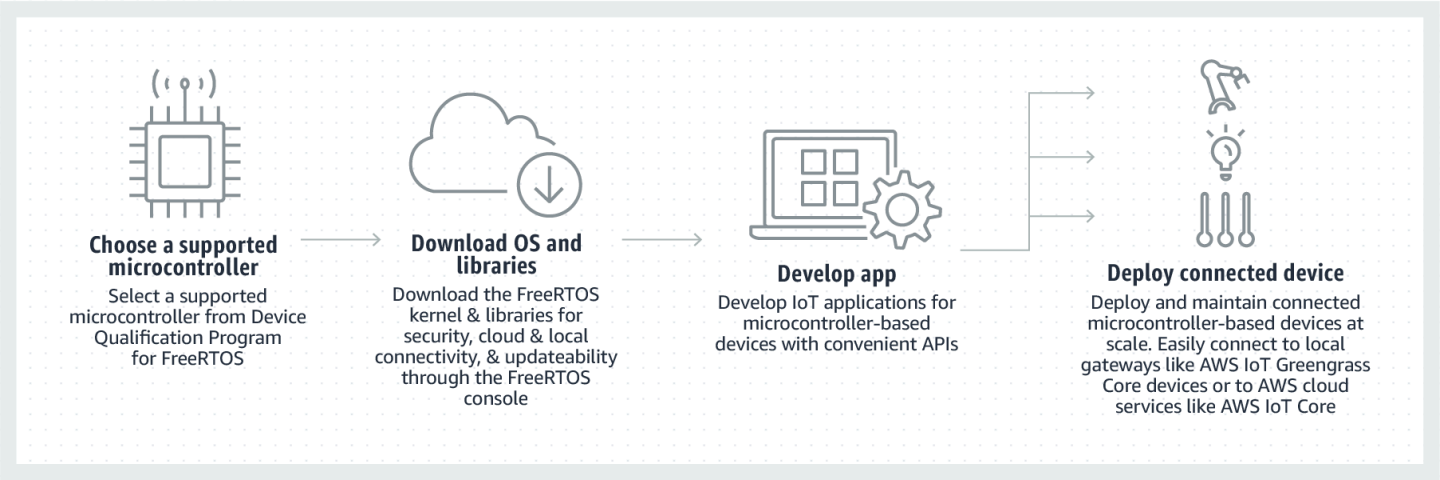
- 瑞薩 RX65N RSK IoT 模組
- 意法國電子 STM32L4 探索套件 IoT 節點
- 德州儀器公司 CC3220SF-LAUNCHXL
- Microsoft Windows 7 或更新版本，含至少一個雙核心和有線乙太網路連線
- 賽靈思安 MicroZed 富利工業 IoT 套件

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

如需有關符合新裝置資格的資訊，請參閱 FreeRTOS 資格指南。

開發工作流程

您可以透過下載 FreeRTOS 來開始開發。您會將套件解壓縮並匯入 IDE。然後，您可以在您選取的硬體平台上開發應用程式，並使用適合您裝置的部署程序製造及部署這些裝置。部署的裝置可以連線至 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。

- 如需其他 FreeRTOS 文件，請參閱 Freer tos.org。
- 如需 FreeRTOS 工程團隊的 FreeRTOS 相關問題，您可以在 FreeRTOS 頁面上開啟問題。 GitHub
- 如需有關 FreeRTOS 的技術問題，請參閱 Free RTOS 社群論壇。
- 如需將裝置連接至的詳細資訊 AWS IoT，請參閱AWS IoT Core 開發人員指南中的裝置佈建。
- 如需的技術 Support AWS，請參閱AWS 支援中心。

- 有關帳 AWS 單、帳戶服務、事件、濫用或其他問題的問題 AWS，請參閱聯絡我們頁面。

FreeRTOS 核心基礎

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。
- Support 多核心微控制器上的對稱式多處理 (SMP)。

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。Direct-to-task 通知允許快速發送任務信號，幾乎沒有 RAM 開銷。它們可以在大多數互任務和 interrupt-to-task 信令方案中使用。

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。

如需有關 FreeRTOS 核心的大部分 up-to-date 文件，請參閱 FreeRTOS 核心。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 原語 (例如 direct-to-task 通知、串流和訊息緩衝區) 為許多常見設計案例中的佇列提供輕量級替代方案。

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。

Direct-to-task 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 direct-to-task 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 direct-to-task 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。

對稱式多重處理 (SMP) 支援

FreeRTOS 核心中的 SMP 支援可讓 FreeRTOS 核心的一個執行個體在多個相同的處理器核心上排程工作。核心架構必須相同且共用相同的記憶體。

修改應用程式以使用自由式 SMP 核心

除了這些額外的 API 之外，FreeRTOS API 在單核和 SMP 版本之間保持大致相同。因此，針對 FreeRTOS 單核心版本撰寫的應用程式應該以 SMP 版本進行編譯，而且毫不費力。但是，可能存在一些功能問題，因為對於單核心應用程序而言，某些假設可能不再適用於多核心應用程序。

一個常見的假設是，當優先順序較高的工作正在執行時，無法執行較低優先順序的工作。雖然這在單核心系統上是如此，但多核心系統不再如此，因為多個工作可以同時執行。如果應用程式依賴相對工作優先順序來提供相互排斥，則可能會在多核心環境中觀察到非預期的結果。

另一個常見的假設是 ISR 無法同時執行彼此或與其他工作一起執行。這在多核心環境中已不再如此。應用程式寫入器需要在存取工作和 ISR 之間共用的資料時，確保適當的相互排斥。

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的FreeRTOSConfig.h檔案位於*freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/FreeRTOSConfig.h和*freertos*/vendors/*vendor*/boards/*board*/aws_tests/config_files/FreeRTOSConfig.h。

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

適用於 Embedded C 的 AWS IoT 裝置 SDK

> **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 開放原始碼授權的 C 原始碼檔案集合，可用於嵌入式應用程式，將 IoT 裝置安全地連接至 AWS IoT Core。其包括 MQTT 用戶端、HTTP 用戶端、HTTP 用戶端、JSON 解析器和AWS IoT Device Shadow、AWS IoT任務、AWS IoT機群佈建和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 通常會針對需要最佳化 C 語言執行階段的資源限制裝置。您可以在任何作業系統上使用軟體開發套件，並將其裝載在任何處理器類型 (例如 MCU 和 MPU) 上。不過，如果裝置有足夠的記憶體和處理資源，建議您使用其中一個較高階的AWS IoT裝置 SDK。

如需詳細資訊，請參閱下列內容：

- 適用於 Embedded C 的 AWS IoT 裝置 SDK
- AWS IoT適用於 Emded 的裝置 SDK GitHub
- 適用於 Embedded C 的 AWS IoT 裝置軟體開發套件讀我檔案
- 適用於 Embedded C 的 AWS IoT 裝置 SDK 範例

通用 IO

通用 IO API 充當硬件抽象層（HAL），在驅動程序和更高級別的應用程序代碼之間提供通用接口。FreeRTOS 通用 IO 提供了一組標準 API，用於存取支援的參考板上的常見序列裝置；這些 API 的實作並不包括在內。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。如果沒有 Common IO，編寫代碼以與低級設備一起工作是矽供應商特定的。

> **Note**
>
> FreeRTOS 不需要通用 IO API 的實作才能運作，但它會嘗試使用通用 IO API 來與微控制器主機板上的特定周邊設備連接，而不是特定於廠商的 API。

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。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，跨多個以微控制器 (MCU-) 為基礎的參考板存取各種裝置驅動程式。

Libraries (程式庫)

目前，FreeRTOS 提供了兩個常見的 IO 庫：通用 IO-基本和通用 IO-BLE。

通用 IO-基本

概要

通用 IO-基本提供 API，可處理基本 I/O 周邊設備和功能，您可能會在基於 MCU 的主機板上找到。通用 IO-基本存儲庫可在中使用GitHub。

支援的周邊裝置

- ADC
- GPIO
- I2C
- PWM
- SPI
- UART
- 看門狗

- 閃光燈
- RTC
- 發揮作用
- 重設
- I2S
- 績效計數器
- 硬體平台資訊

支援的功能

- 同步讀/寫

  該函數在傳輸請求的數據量之前不會返回。

- 異步讀/寫

  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事務。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數據，這意味著寫入和讀取同時發生。

API 參考

如需完整的 API 參考資料，請參閱通用 IO-基本 API 參考資料。

通用 IO

概要

通用 IO-BLE 從製造商的低功耗藍牙堆棧中提供抽象。它提供了可用於控制設備，並執行 GAP 和 GATT 操作以下接口。通用 IO-BLE 存儲庫可在上使用GitHub。

藍牙設備管理器：

這提供了一個接口來控制藍牙設備，執行設備發現操作和其他與連接相關的任務。

BLE 介面卡管理員：

這為 BLE 特有的 GAP API 函數提供了一個接口。

藍牙經典適配器管理器：

這提供了一個接口來控制設備的 BT 經典功能。

關貿總協定伺服器：

這提供了使用藍牙 GATT 伺服器功能的介面。

關貿總協定：

這提供了一個使用藍牙 GATT 客戶端功能的接口。

A2DP 連接介面:

這會為本機裝置的 A2DP 來源設定檔提供介面。

API 參考

如需完整的 API 參考資料，請參閱通用 IO-BLE API 參考資料。

亞馬遜通用軟件的通用 IO

通用 IO API 是 Amazon 裝置通用軟體所需實作的一部分，特別是要在廠商裝置移植套件 (DPK) 中實作。

什麼是 ACS？

Amazon 裝置通用軟體 (ACS) 是一種可讓您更快速地在裝置上整合 Amazon 裝置開發套件的軟體。ACS 提供統一的 API 整合層、預先驗證且具有記憶體效率的元件，適用於連線能力、裝置移植套件 (DPK) 和多層測試套件等常見功能。

資格計劃

Amazon 裝置通用軟體認證計劃會驗證在特定微控制器開發板上執行的 ACS DPK (裝置移植套件) 組建是否與該計畫公佈的最佳實務相容，並且足以通過資格計劃指定的 ACS 強制性測試。

符合此計劃資格的廠商會列在 ACS 晶片組廠商頁面上。

如需符合資格的相關資訊，請聯絡裝置的 ACS。

FreeRTOS 入門

主題：

- 使用快速 Connect 開始使用AWS IoT和 FreeRTOS
- FreeRTOS 無線程式庫
- 了解如何構建安全可靠的AWS IoT產品
- 開發您的AWS IoT應用產品

使用快速 Connect 開始使用AWS IoT和 FreeRTOS

若要快速探索AWS IoT，請從AWS快速 Connect 示範開始。快速 Connect 示範可輕鬆設定並連接合作夥伴提供的 FreeRTOS 合格主機板AWS IoT。

按照AWS IoT入門教程進行操作，以更好地了解控制台AWS IoT和AWS IoT控制台。您可以使用所選主機板的建置系統和工具來連線到您的AWS帳戶，修改 Quick Connect 示範隨附的示範原始程式碼。現在可以看到來自您帳戶AWS IoT控制台的數據流。

FreeRTOS 無線程式庫

一旦您瞭解 IoT 裝置的方式並共同AWS IoT運作，就可以開始探索 FreeRTOS 程式庫和長期 Support (LTS) 程式庫。

以 FreeRTOS 為基礎之AWS IoT裝置的一些常用程式庫包括：

- FreeRTOS 核心
- 中央电视台
- AWS IoT無線 (OTA)

請造訪 freertos.org 以取得程式庫特定的技術文件和示範。

了解如何構建安全可靠的AWS IoT產品

請參閱精選 FreeRTOSAWS IoT 整合，瞭解如何使 IoT 裝置軟體更加安全且可靠的最佳實務。這些 FreeRTOS IoT 整合功能是結合 FreeRTOS 軟體和合作夥伴提供的具有硬體安全性功能的主機板，旨在提高安全性。按原樣在生產中使用它們，或將它們用作您自己設計的模型。

開發您的AWS IoT應用產品

請依照下列步驟為您的AWS IoT產品建立應用程式專案：

1. 從 Freer tos.org 下載最新的免費服務或長期 Support (LTS) 版本，或從 Freertos-LTS GitHub 軟體庫複製。如果有的話，您也可以從 MCU 廠商的工具鏈將所需的 FreeRTOS 程式庫整合到您的專案中。
2. 遵循 FreeRTOS 移植指南來建立專案、設定開發環境，以及將 FreeRTOS 程式庫整合到您的專案中。使用 FreeRTOS 程式庫整合測試 GitHub 存放庫來驗證移植。

AWS IoT Device Tester對於免費服務

FreeRTOS 的 IDT 是一種工具，可以透過 FreeRTOS 作業系統來判定資料傳輸率。設備測試儀（IDT）首先打開與設備的 USB 或 UART 連接。然後，它會閃爍設定為在各種條件下測試裝置功能的 FreeRTOS 影像。 AWS IoT Device Tester套件是可擴展的，IDT 用於客戶AWS IoT測試協調。

免費伺服器的 IDT 會在連接到測試裝置的主機電腦 (視窗、MacOS 或 Linux) 上執行。IDT 配置和協調測試用例，並彙總結果。還會提供可管理測試執行作業的命令列界面。

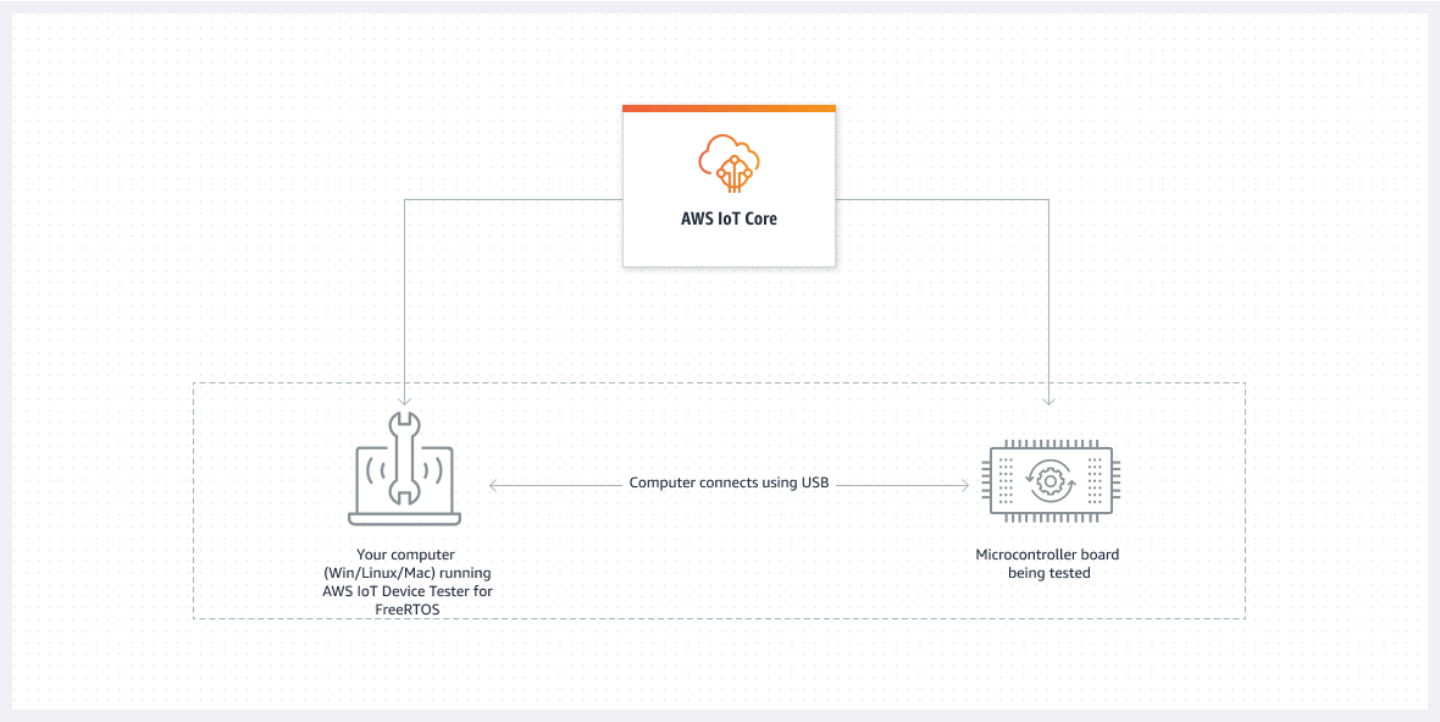
自由服務資格套件

FreeRTOS 的 IDT 會驗證您的微型控制器上 FreeRTOS 的連接埠，以及它是否能以可靠且安全的方式進行有AWS IoT效通訊。具體來說，它會驗證 FreeRTOS 程式庫的移植層介面是否正確實作。也會使用 AWS IoT Core 執行端對端測試。例如，它會驗證您的主機板是否可以傳送和接收 MQTT 訊息，並正確處理它們。

FreeRTOS 資格認證（FRQ）2.0 套件使用 FreeRTOS 資格指南中定義的 FreeRTOS 圖書館集成測試和設備顧問的測試案例。

FreeRTOS 的 IDT 會產生測試報告，您可以將這些報告提交給AWS合作夥伴網路 (APN)，以將 FreeRTOS 裝置包含在合作夥伴裝置目錄中。AWS如需詳細資訊，請參閱 AWS 裝置資格計劃。

下圖顯示 FreeRTOS 資格的測試基礎結構設定。



FreeRTOS 的 IDT 將測試資源組織到測試套件和測試群組中：

- 測試套件是一組測試群組，用來驗證裝置是否可與特定版本的 FreeRTOS 搭配使用。
- 測試組是一組與特定功能相關的單個測試用例，例如 BLE 和 MQTT 消息傳遞。

如需詳細資訊，請參閱 測試套件版本

自訂測試套件

FreeRTOS 的 IDT 結合了標準化的組態設定和結果格式與測試套件環境。此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添加自定義測試，也可以將其提供給客戶進行設備驗證。

配置自定義測試套件的方式決定了您必須提供給用戶才能運行自定義測試套件的設置配置。如需詳細資訊，請參閱使用 IDT 開發和運行自己的測試套件。

支援的版本AWS IoT Device Tester對於 FreeRTOS

本主題列出支援的版本AWS IoT Device Tester對於 FreeRTOS。作為最佳實務，建議您使用最佳實務，適用於 FreeRTOS，並支援目標版本的 FreeRTOS。FreeRTOS 的每個 IDT 版本都有一個或多個

對應的 FreeRTOS 版本，它支援這些版本。我們建議您在新版的 FreeRTOS 發行時，下載新版本的 FreeRTOS 版 IDT。

一旦下載本軟體，即表示您同意AWS IoT Device Tester下載存檔中包含的許可協議。

> **Note**
>
> 當您使用時AWS IoT Device Tester對於 FreeRTOS，我們建議您更新至最新的 FreeRTOS 版本的最新修補程式版本。

> **Important**
>
> 截至二零二二年十月，AWS IoT Device Tester為了AWS IoTFreeRTOS 資格認證 (FRQ) 1.0 不會產生已簽署的資格報告。您無法符合新資格AWS IoT要在中列出的 FreeRTOS 裝置AWS夥伴裝置目錄通過AWS設備認證計劃使用 IDT FRQ 1.0 版本。雖然您無法使用 IDT FRQ 1.0 來限定 FreeRTOS 裝置的資格，但您可以繼續使用 FRQ 1.0 來測試您的 FreeRTOS 裝置。建議您使用代碼先生 2.0以符合資格並列出AWS夥伴裝置目錄。

FreeRTOS 的最新 AWS IoT Device Tester 版本

請使用下列連結來下載 FreeRTOS 的最新版本的 IDT。

FreeRTOS 的最新 AWS IoT Device Tester 版本

AWS IoT Device Tester version (Python 版本)	測試套件版本	支援的 FreeRTOS 版本	下載連結	發行日期	版本備註
印尼盾	FRQ_2.5.0	• 202112.00 • 202212.00 • 202212.01 • 使用 FreeRTOS LTS 程式庫	• Linux • macOS • 視窗	2023.04.04	• 支援針對 FreeRTOS 的測試202112,202212,2 以及 FreeRTOS 的所有修補程式二

AWS IoT Device Tester version (Python 版本)	測試套件版本	支援的 FreeRTOS 版本	下載連結	發行日期	版本備註
		的所有修補程式。			2210 公升使用 FreeRTOS 式庫。請參閱雷德美.MD以取得更多資訊。您必須在您的`manifest.yml` 。 • 改進了 OTA E2E 測試的運行時間。 • 限制中所列設備的數量`device.json` 至 1。 • 少量错误修复和改进。

> **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。這種做法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。

適用於 FreeRTOS 務的早期 IDT 版本

此外，也支援下列舊版的 FreeRTOS IDT。

早期版本AWS IoT Device Tester對於 FreeRTOS

AWS IoT Device Tester version (Python 版本)	測試套件版本	支援的 FreeRTOS 版本	下載連結	發行日期	版本備註
印尼盾 v4.8.1	FRQ_2.4.0	• 202112.00 • 202212.00 • 202212.01 • 使用 FreeRTOS LTS 程式庫的所有修補程式。	• Linux • macOS • 視窗	2023.01.23	• 請參閱閱讀我的 .MD以獲取更多信息。您必須在您的manifest.yml 。 • 少量错误修复和改进。
印尼盾	FRQ_2.3.0	• 202112.00 • 202212.00 • 202212.01 • 202210-使用 FreeRTOS LTS 程式庫的 LTS。	• Linux • macOS • 視窗	2022.11.16	• 請參閱閱讀我的 .MD以獲取更多信息。您必須在您的manifest.yml 。 • 如需有關 FreeRTOS 所包含內容的更多資訊二 2210 公升發行版本，請參閱更新日誌.MD檔案於 GitHub。 • 添加配置和運行的

AWS IoT Device Tester version (Python 版本)	測試套件版本	支援的 FreeRTOS 版本	下載連結	發行日期	版本備註
					能力AWS IoT Device Tester通過基於網絡的用戶界面獲得FreeRTOS。請參閱使用免FreeRTOS使用者介面執行FreeRTOS資格套件2.0 (FRQ 2.0)開始。 • 添加一個選項，以保留在運行時創建和用於後測試調試的源代碼的修改副本。如需詳細資訊，請參閱設定建置、刷新和測試設定。 • 添加對 Java 的 IDT 客戶

AWS IoT Device Tester version (Python 版本)	測試套件版本	支援的 FreeRTOS 版本	下載連結	發行日期	版本備註
					端 SDK 支持。如需有關 IDT 用戶端 SDK 的詳細資訊，請參閱使用 IDT 開發和運行自己的測試套件。
印尼盾 v4.5.11	FRQ_2.2.0	• 202112.00 • 202212.00 • 202212.01 • 202210-使用 FreeRTOS LTS 程式庫的 LTS。	• Linux • macOS • 視窗	2022.10.14	• 請參閱閱讀我的 .MD以獲取更多信息。您必須在您的`manifest.yml` 。 • 如需有關 FreeRTOS 所包含內容的更多資訊二 2210 公升發行版本，請參閱更新日誌.MD檔案於 GitHub。 • 少量错误修复和改进。

如需詳細資訊，請參閱 免費服務的AWS IoT Device Tester支援政策。

FreeRTOS 不支援的 IDT 版本

本節列出了 FreeRTOS 不受支援的 IDT 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，請參閱免費服務的AWS IoT Device Tester支援政策。

不再支援下列版本的 IDT-FreeRTOS。

不支援的免費伺AWS IoT Device Tester服器版本

AWS IoT Device Tester version (Python 版本)	測試套件版本	支援的 FreeRTOS 版本	發行日期	版本備註
IDT v4.5.10	FRQ_2.1.4	• 202112.00 • 使用 FreeRTOS LTS 程式庫的 LTS。	2022.09.02	• 如需有關 FreeRTOS 202012-LTS 發行版本中包含哪些內容的詳細資訊，請參閱中的變更記錄檔 .md 檔案。 GitHub • 已解決影響OTA End to End測試群組的問題。 • 已FullTransportInterfacePlaintext 從資格賽中執行移除。純文本仍然可以通過使用-\-group-id 標誌作為

AWS IoT Device Tester version (Python 版本)	測試套件版本	支援的 FreeRTOS 版本	發行日期	版本備註
				開發測試組運行。 • 改進了控制台和文件輸出的日誌記錄和可讀性。 • 少量错误修复和改进。

AWS IoT Device Tester version (Python 版本)	測試套件版本	支援的 FreeRTOS 版本	發行日期	版本備註
印尼盾 v4.5.9	FRQ_2.1.3	• 202112.00 • 使用免費伺服器 LTS 程式庫的 LTS。	2022.08.17	• 如需有關 FreeRTOS 202012.04-LTS 發行版本所包含內容的詳細資訊，請參閱中的變更記錄檔 .md 檔案。 GitHub • 已解決影響 `FreeRTOSIntegrity` 測試群組的問題。 • 通過刪除「MQTT Connect 指數輪詢重 `FullCloudIoT` 試」測試用例來更新測試組。 • 少量错误修复和改进。

AWS IoT Device Tester version (Python 版本)	測試套件版本	支援的 FreeRTOS 版本	發行日期	版本備註
印尼盾 4.5.6	FRQ_2.1.2	• 202112.00 • 使用免費伺服器 LTS 程式庫的 LTS。	2022.06.29	• 如需有關 FreeRTOS 202012.04-LTS 發行版本所包含內容的詳細資訊，請參閱中的變更記錄檔 .md 檔案。 GitHub • 添加測試板的新測試組`FullCloudIoT` AWS IoT Core Device Advisor。 • 解決了影響 OTA E2E 測試用例的問題。 • 少量错误修复和改进。

AWS IoT Device Tester version (Python 版本)	測試套件版本	支援的 FreeRTOS 版本	發行日期	版本備註
印尼盾 v4.5.5	FRQ_2.1.1	• 202112.00 • 使用免費伺服器 LTS 程式庫的 LTS。	2022.06.06	• 如需有關 FreeRTOS 202012.04-LTS 發行版本所包含內容的詳細資訊，請參閱中的變更記錄檔 .md 檔案。 GitHub • 添加測試板的新測試組FullCloudIoT AWS IoT Core Device Advisor。 • 已解決影響自由版本和免費版本完整性測試案例的問題。 • 少量错误修复和改进。

AWS IoT Device Tester version (Python 版本)	測試套件版本	支援的 FreeRTOS 版本	發行日期	版本備註
印尼盾 v4.5.5	FRQ_2.1.0	• 202107.00 • 202112.00 • 使用免費伺服器 LTS 程式庫的 LTS。	2022.05.31	• 如需有關 FreeRTOS 202012.04-LTS 發行版本所包含內容的詳細資訊，請參閱中的變更記錄檔 .md 檔案。 GitHub • 添加測試板的新測試組`FullCloudIoT` AWS IoT Core Device Advisor。 • 少量错误修复和改进。

AWS IoT Device Tester version (Python 版本)	測試套件版本	支援的 FreeRTOS 版本	發行日期	版本備註
IDT v4.5.4	FRQ_2.0.0	• 202112.00 • 使用免費伺服器 LTS 程式庫的 LTS。	2022.05.09	• 如需請參閱中包含項目的詳細資訊，請參閱中包含項目的詳細資訊，請參閱中的。GitHub • 移除僅使用儲存庫中 Amazon FreeRTOS 版本的主機板資格要求。`aws/amazon-freertos` GitHub • 少量错误修复和改进。

AWS IoT Device Tester version (Python 版本)	測試套件版本	支援的 FreeRTOS 版本	發行日期	版本備註
印尼盾版 4.5.2	FRQ_1.6.2	202107.00	2022.01.25	• 如需請參閱中包含項目的詳細資訊，請參閱中包含項目的詳細資訊，請參閱中的。GitHub • 實作新的 IDT 測試協調器，以設定自訂測試套件。如需詳細資訊，請參閱設定 IDT 測試協調程式。 • 少量错误修复和改进。
印尼盾 v4.0.3	FRQ_1.5.1	202012.00	2021.07.30	• Support 硬體安全模組上具有鎖定認證的裝置鑑定。 • 少量错误修复和改进。

AWS IoT Device Tester version (Python 版本)	測試套件版本	支援的 FreeRTOS 版本	發行日期	版本備註
印尼盾	FRQ_1.6.1	202107.00	2021.07.26	• 如需請參閱中包含項目的詳細資訊，請參閱中包含項目的詳細資訊，請參閱中的。 GitHub • 添加通過基於 Web 的用戶界面配置和運行 AWS IoT Device Tester FreeRTOS 的功能。請參閱使用免費伺服器的 IDT 使用者介面來執行 FreeRTOS 資格套件以開始使用。

AWS IoT Device Tester version (Python 版本)	測試套件版本	支援的 FreeRTOS 版本	發行日期	版本備註

印尼盾版 4.1.0	FRQ_1.6.0	202107.00	2021.07.21	• 如需請參閱中包含項目的詳細資訊，請參閱中包含項目的詳細資訊，請參閱中的。GitHub
				• 從 OTA 資格中刪除以下測試用例：
				  • OTA 代理程式
				  • OTA 遺失檔名
				  • OTA 最大配置塊數
				• 從 OTA 資格中刪除 OTA 數據通道Both測試組。在device.json 檔案中，OTADataPlaneProtocol 組態現在只接受HTTP或MQTT作為支援的值。
				• 對 FreeRTOS 原始程式碼的變更，

AWS IoT Device Tester version (Python 版本)	測試套件版本	支援的 FreeRTOS 版本	發行日期	版本備註
				對userdata.json 檔案中的freertosFileConfiguration 組態實作下列變更： • 將為otaAgentTestsConfig 和otaAgentDemosConfig 從指定的檔案名稱變更aws_ota_agent_config.h 為ota_config.h 。 • 添加新的otaDemosConfig 可選配置以指定新文件的ota_demo_config.h 文件路徑。 • 將新欄位新增testStartDelayms userdata.json 至，以指定裝置刷新以執行

AWS IoT Device Tester version (Python 版本)	測試套件版本	支援的 FreeRTOS 版本	發行日期	版本備註
				FreeRTOS 測試群組與開始執行測試之間的延遲時間。數值應為單位為毫秒。這種延遲可用於為 IDT 提供連接的機會，以便不會錯過任何測試輸出。

AWS IoT Device Tester version (Python 版本)	測試套件版本	支援的 FreeRTOS 版本	發行日期	版本備註
印尼盾 v4.0.1	弗爾 Q_1.4.1	202012.00	2021.01.19	• 如需請參閱中包含項目的詳細資訊，請參閱中包含項目的詳細資訊。GitHub • 引入其他 OTA（Over-the-air）E 2E（端到端）測試用例。 • 支援執行使用 FreeRTOS LTS 程式庫之開發主機板的認證。 • 添加對使用蜂窩連接的 FreeRTOS 開發板認證的支持。 • 修復了 echo 服務器配置中的錯誤。 • 可讓您使AWS IoT Device Tester用 FreeRTOS 開發和執行您自己的自訂測試套件。如需詳

AWS IoT Device Tester version (Python 版本)	測試套件版本	支援的 FreeRTOS 版本	發行日期	版本備註
				細資訊，請參閱使用 IDT 開發和運行自己的測試套件。 • 提供代碼簽名的 IDT 應用程序，因此在 Windows 或 macOS 下運行時不需要授予權限。 • 完善了 BLE 測試結果剖析邏輯。

AWS IoT Device Tester version (Python 版本)	測試套件版本	支援的 FreeRTOS 版本	發行日期	版本備註
印尼盾	FRQ_1.3.0	202011.01	2020.11.05	• 如需詳細資訊，請參閱 中的變更記錄檔。 GitHub • 修復了「RSA」不是有效的 PKCS11 配置選項的錯誤。 • 修復了 OTA 測試後 Amazon S3 存儲桶未正確清理的錯誤。 • 更新以支援 FullMQtt 測試群組內部的新測試案例。

AWS IoT Device Tester version (Python 版本)	測試套件版本	支援的 FreeRTOS 版本	發行日期	版本備註
印尼盾	FRQ_1.2.0	202007.00	2020.09.17	• 如需詳細資訊，請參閱 中的變更記錄檔。 GitHub • 新的端到端測試，以驗證空中（OTA）更新暫停和恢復功能。 • 修正了導致歐盟中心 -1 區域中的用戶無法通過 OTA 測試的配置驗證的錯誤。 • 已將--update-idt 參數新增至run-suite 指令。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 指令。您可以使用此選項來設定受

AWS IoT Device Tester version (Python 版本)	測試套件版本	支援的 FreeRTOS 版本	發行日期	版本備註
				管理原則更新提示的回應。 • 內部改進和錯誤修復，包括： • 對於自動測試套件更新，改進了配置文件升級。

AWS IoT Device Tester version (Python 版本)	測試套件版本	支援的 FreeRTOS 版本	發行日期	版本備註

印尼盾	弗爾 Q_1.0.1	202002.00		• 如需詳細資訊，請參閱中的變更記錄檔。 GitHub
				• 新增 IDT 內的測試套件自動更新。IDT 現在可以下載適用於 FreeRTOS 版本的最新測試套件。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

AWS IoT Device Tester version (Python 版本)	測試套件版本	支援的 FreeRTOS 版本	發行日期	版本備註
				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 IDT 版本所支援的 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

AWS IoT Device Tester version (Python 版本)	測試套件版本	支援的 FreeRTOS 版本	發行日期	版本備註
				major.minor.patch 格式進行版本化。 • 新增命list-supported-products 令 — 列出已安裝的 IDT 版本所支援的 FreeRTOS 和測試套件版本。 • 新增list-test-cases命令 — 列出測試群組中可用的測試案例。 • 為run-suite 命令添加test-id選項 — 使用此選項可在測試組中運行單個測試用例。

AWS IoT Device Tester version (Python 版本)	測試套件版本	支援的 FreeRTOS 版本	發行日期	版本備註
印尼盾	FRQ_1.0.0	202002.00		• 如需詳細資訊，請參閱 中的變更記錄檔。 GitHub • 支持over-the-air（OTA）端到端測試用例的自定義代碼簽名方法，以便您可以使用自己的代碼簽名命令和腳本對 OTA 有效負載進行簽名。 • 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。 • 已新增具有執行所需權限`AWSIoTDeviceTesterForFreeRTOSFullAccess` 的AWS受管理原則AWS

AWS IoT Device Tester version (Python 版本)	測試套件版本	支援的 FreeRTOS 版本	發行日期	版本備註
				IoT Device Tester。如果新版本需要其他許可，我們會將它們新增至此受管政策，以便您不必手動更新 IAM 許可。 • 結果目錄命名為 `AFQ_Report.xml` 的檔案現在是 `FRQ_Report.xml` 。
印尼盾版	FRQ_1.0.0	202002.00		• 支援透過 HTTPS 進行 OTA 的選擇性測試，以符合您的 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。

AWS IoT Device Tester version (Python 版本)	測試套件版本	支援的 FreeRTOS 版本	發行日期	版本備註
印尼盾版	FRQ_1.0.0	201910.00		• 支援使用安全元件 (內建金鑰) 的 FreeRTOS 裝置進行認證。 • 支援 SecureSocket 和 Wifi 測試群組的可設定 echo 伺服器連接埠。 • 支持超時乘數標誌以增加超時，這在您對超時相關錯誤進行故障排除時非常有用。 • 新增用於日誌剖析的錯誤修正。 • 在測試中支援 IoT ATS 端點。

AWS IoT Device Tester version (Python 版本)	測試套件版本	支援的 FreeRTOS 版本	發行日期	版本備註
印尼盾	FRQ_1.0.0	201908.00		• 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。
印尼盾 V1.3.2	FRQ_1.0.0	201906.00		• 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	FRQ_1.0.0	• FreeRTOS 托斯版本 1.4.8 • FreeRTOS 行		增加了對使用 CMAKE 構建系統測試 FreeRTOS 設備的支持。

AWS IoT Device Tester version (Python 版本)	測試套件版本	支援的 FreeRTOS 版本	發行日期	版本備註
IDT-FreeRTOS 1.1 版	FRQ_1.0.0			
IDT-FreeRTOS 1.0 版	FRQ_1.0.0			

下載 FreeRTOS 的 IDT

本主題說明下載適用於 FreeRTOS 之 IDT 的選項。您可以使用下列其中一個軟體下載連結，也可以按照指示以程式設計方式下載 IDT。

> **Important**
>
> 截至二零二二年十月，AWS IoT Device Tester為了AWS IoTFreeRTOS 資格認證 (FRQ) 1.0 不會產生已簽署的資格報告。您無法符合新資格AWS IoT要在中列出的 FreeRTOS 裝置AWS 合作夥伴裝置目通過AWS設備認證計劃使用 IDT FRQ 1.0 版本。雖然您無法使用 IDT FRQ 1.0 來限定 FreeRTOS 裝置的資格，但您可以繼續使用 FRQ 1.0 來測試您的 FreeRTOS 裝置。我們建議您使用外匯編碼 2.0以符合資格並列出AWS合作夥伴裝置目。

主題

- 手動下載 IDT
- 以編程方式下載 IDT

一旦下載本軟體，即表示您同意AWS IoT Device Tester下載存檔中包含的許可協議。

> **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。

手動下載 IDT

本主題列出 FreeRTOS 支援的 IDT 版本。我們建議您使用最新版本的AWS IoT Device Tester支援您目標版本的 FreeRTOS。新版本的 FreeRTOS 可能會要求您下載新版本的AWS IoT Device Tester。當您開始測試運行時，您會收到通知AWS IoT Device Tester與您使用的 FreeRTOS 版本不相容。

請參閱 支援的版本AWS IoT Device Tester對於 FreeRTOS

以編程方式下載 IDT

IDT 提供了一個 API 操作，您可以使用它來檢索 URL，您可以在其中以編程方式下載 IDT。您也可以使用此 API 操作檢查您是否擁有最新版本的 IDT。此 API 操作具有以下端點。

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

若要呼叫此 API 操作，您必須具有執行**iot-device-tester:LatestIdt**動作。包括您的AWS簽名，使用`iot-device-tester`作為服務名稱

API 請求

HostOs — 主機的作業系統。您可以從以下選項中選擇：

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

TestSuiteType — 測試套件的類型。選擇下列選項：

FR— IDT 的 FreeRTOS

ProductVersion

(選用) FreeRTOS 的版本。此服務會傳回該 FreeRTOS 版本的最新相容版本的 IDT。如果您未指定此選項，服務會傳回最新版本的 IDT。

API 回應

API 回應格式如下。該`DownloadURL`包括一個 zip 文件。

```
{
    "Success": True or False,
    "Message": Message,
```

```
    "LatestBk": {
        "Version": The version of the IDT binary,
        "TestSuiteVersion": The version of the test suite,
        "DownloadURL": The URL to download the IDT Bundle, valid for one hour
    }
}
```

範例

您可以參考下列範例，以程式設計方式下載 IDT。這些範例使用您儲存在AWS_ACCESS_KEY_ID和AWS_SECRET_ACCESS_KEY環境變數。若要遵循最佳安全性做法，請勿將憑證儲存在程式碼中。

Example

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

如果您有 cURL 版本 7.75.0 或更高版本，您可以使用aws-sigv4用來簽署 API 要求的旗標。此範例使用JQ從響應中解析下載 URL。

> **Warning**
>
> 該aws-sigv4標誌要求 curl GET 請求的查詢參數按順序HostOs/ProductVersion/TestSuiteType或者HostOs/TestSuiteType。不符合的訂單將導致從 API Gateway 取得標準字串不符合的簽章時發生錯誤。
> 如果可選參數ProductVersion已包含，您必須使用支援的產品版本，如中所述支援的版本AWS IoT Device Tester對於 FreeRTOS。

- 取代*##-## -2*與您的AWS 區域。如需區域代碼清單，請參閱區域端點。
- 取代*Linux*與您的主機的操作系統。
- 取代*202107.00*使用您的 FreeRTOS 版本。

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

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

Example

範例：使用較早版本的 cURL (Mac 和 Linux) 下載

您可以使用以下 cURL 命令AWS您簽署並計算的簽名。如需如何簽署和計算AWS簽名，請參閱簽署AWSAPI 請求。

- 取代*Linux*與您的主機的操作系統。
- 取代*###*與日期和時間，如**20220210T004606Z**。
- 取代*##*與日期，例如**20220210**。
- 取代*AWSRegion*與您的AWS 區域。如需區域代碼清單，請參閱區域端點。
- 取代*AWSSignature*與AWS簽名您生成的。

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

Example

範例：使用 Python 指令碼下載

這個例子使用了 Python要求圖書館。此範例是從 Python 範例改編而成簽署一個AWSAPI 請求在AWS一般參考。

- 取代*##-## -2*與您所在的地區。如需區域代碼清單，請參閱區域端點。
- 取代*Linux*與您的主機的操作系統。

```
# Copyright 2010-2022 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#
# This file is licensed under the Apache License, Version 2.0 (the "License").
# You may not use this file except in compliance with the License. A copy of the
#License is located at
#
```

```
# http://aws.amazon.com/apache2.0/
#
# This file is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS
# OF ANY KIND, either express or implied. See the License for the specific
# language governing permissions and limitations under the License.

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

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

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

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

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

```
# ************* TASK 1: CREATE A CANONICAL REQUEST *************
# http://docs.aws.amazon.com/general/latest/gr/sigv4-create-canonical-request.html
# Step 1 is to define the verb (GET, POST, etc.)--already done.
# Step 2: Create canonical URI--the part of the URI from domain to query
# string (use '/' if no path)
canonical_uri = '/latestidt'
# Step 3: Create the canonical query string. In this example (a GET request),
# request parameters are in the query string. Query string values must
# be URL-encoded (space=%20). The parameters must be sorted by name.
# For this example, the query string is pre-formatted in the request_parameters
 variable.
canonical_querystring = request_parameters
# Step 4: Create the canonical headers and signed headers. Header names
# must be trimmed and lowercase, and sorted in code point order from
# low to high. Note that there is a trailing \n.
canonical_headers = 'host:' + host + '\n' + 'x-amz-date:' + amzdate + '\n'
# Step 5: Create the list of signed headers. This lists the headers
# in the canonical_headers list, delimited with ";" and in alpha order.
# Note: The request can include any headers; canonical_headers and
# signed_headers lists those that you want to be included in the
# hash of the request. "Host" and "x-amz-date" are always required.
signed_headers = 'host;x-amz-date'
# Step 6: Create payload hash (hash of the request body content). For GET
# requests, the payload is an empty string ("").
payload_hash = hashlib.sha256(('').encode('utf-8')).hexdigest()
# Step 7: Combine elements to create canonical request
canonical_request = method + '\n' + canonical_uri + '\n' + canonical_querystring + '\n'
 + canonical_headers + '\n' + signed_headers + '\n' + payload_hash

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

```
# ************* TASK 4: ADD SIGNING INFORMATION TO THE REQUEST *************
# The signing information can be either in a query string value or in
# a header named Authorization. This code shows how to use a header.
# Create authorization header and add to request headers
authorization_header = algorithm + ' ' + 'Credential=' + access_key + '/' +
 credential_scope + ', ' +  'SignedHeaders=' + signed_headers + ', ' + 'Signature=' +
 signature
# The request can include any headers, but MUST include "host", "x-amz-date",
# and (for this scenario) "Authorization". "host" and "x-amz-date" must
# be included in the canonical_headers and signed_headers, as noted
# earlier. Order here is not significant.
# Python note: The 'host' header is added automatically by the Python 'requests'
 library.
headers = {'x-amz-date':amzdate, 'Authorization':authorization_header}

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

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

使用 IDT 搭配 FreeRTOS 資格套件 2.0 (FRQ 2.0)

FreeRTOS 資格套件 2.0 是 FreeRTOS 資格套件的更新版本。我們建議開發人員使用 FRQ 2.0，因為它包含相關的測試案例，以限定執行 FreeRTOS 長期 Support (LTS) 程式庫的裝置。

FreeRTOS 的 IDT 會驗證您的微型控制器上 FreeRTOS 的連接埠，以及是否可以有效地與之通訊。AWS IoT具體來說，它會使用 FreeRTOS 程式庫驗證移植層介面，以及 FreeRTOS 測試儲存庫是否正確實作。也會使用 AWS IoT Core 執行端對端測試。IDT 針對 FreeRTOS 執行的測試是在 FreeRTOS 儲存庫中定義的。GitHub

FreeRTOS 的 IDT 會以嵌入式應用的形式執行測試，並在被測微控制器裝置上閃爍。應用程式二進位影像包括 FreeRTOS、移植的 FreeRTOS 介面，以及主機板裝置驅動程式。測試的目的是確認移植的 FreeRTOS 介面在您的裝置驅動程式之上是否正常運作。

FreeRTOS 的 IDT 會產生測試報告，您可以提交這些報告，讓您AWS IoT的硬體列在AWS合作夥伴裝置目錄中。如需詳細資訊，請參閱 AWS 裝置資格計劃。

FreeRTOS 的 IDT 會在連接至待測試裝置的主機電腦 (視窗、macOS 或 Linux) 上執行。IDT 可配置和協調測試用例並彙總結果。它還提供了一個命令行界面來管理運行的測試。

為了測試您的設備，IDT 的 FreeRTOS 會創建諸如AWS IoT事物，FreeRTOS 組，Lambda 函數之類的資源。若要建立這些資源，FreeRTOS 的 IDT 會使用中設定的AWS認證代表您`config.json`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 FreeRTOS 的 IDT 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。

主題

- 先決條件
- 首次準備測試微型控制器主機板
- 使用免 FreeRTOS 使用者介面執行 FreeRTOS 資格套件 2.0 (FRQ 2.0)
- 執行 FreeRTOS 資格 2.0 套件
- 了解結果和日誌

先決條件

本節說明使AWS IoT Device Tester用測試微控制器的先決條件。

準備接受 FreeRTOS 的資格

> **Note**
>
> AWS IoT Device Tester對於 FreeRTOS，強烈建議您使用最新的 FreeRTOS 版本的最新修補程式版本。

FRQ 2.0 的 IDT 是 FreeRTOS 的資格。在執行 IDT FRQ 2.0 取得資格之前，您必須先完成 FreeRTOS 資格指南中的主機板資格資格。若要移植程式庫、測試和設定`manifest.yml`，請參閱《FreeRTOS 移植指南》中的〈移植 Fre e RTOS 程式庫〉。FRQ 2.0 包含一個不同的資格過程。如需詳細資訊，請參閱 FreeRTOS 資格指南中的最新資格變更。

必須存在 FreeRTOS 庫集成測試存儲庫才能運行 IDT。有關如何克隆此存儲庫並將其移植到源項目的信息，請參閱 Readme.md。Freertos 庫集成測試必須包含`manifest.yml`位於項目根目錄中的內容，以便 IDT 運行。

> **Note**
>
> IDT 依賴於測試存儲庫的`UNITY_OUTPUT_CHAR`實現。測試輸出日誌和設備日誌不得相互交錯。如需進一步的詳細資訊，請參閱 FreeRTOS 移植指南中的程式庫記錄巨集一節。

下載 FreeRTOS 的 IDT

每個版本的 FreeRTOS 都有相應的 IDT 版本，供 FreeRTOS 執行資格測試。從支援的 FreeRTOS 版本下載適用於 FreeRTOS 的適用 IDT 版本AWS IoT Device Tester。

將 FreeRTOS 的 IDT 擷取至檔案系統上您具有讀取和寫入權限的位置。由於微軟視窗的路徑長度有一個字元限制，因此請將 FreeRTOS 的 IDT 解壓縮到根目錄中，例如或。`C:\` `D:\`

> **Note**
>
> 多個使用者不得從共用位置執行 IDT，例如 NFS 目錄或 Windows 網路共用資料夾。這將導致崩潰或數據損壞。建議您將 IDT 套件解壓縮至本機磁碟機。

下載 Git

IDT 必須先安裝 Git 作為確保原始程式碼完整性的先決條件。

按照指GitHub南中的說明安裝 Git。要驗證當前安裝的 Git 版本，請在終端機`git --version`上輸入命令。

> **Warning**
>
> IDT 使用 Git 與目錄的乾淨或臟狀態保持一致。如果未安裝 Git，`FreeRTOSIntegrity`測試群組將會失敗，或無法如預期般執行。如果 IDT 傳回或之類的錯誤`git command not found`，請安裝`git executable not found`或重新安裝 Git，然後再試一次。

建立並設定 AWS 帳戶

> **Note**
>
> 完整的 IDT 資格套件僅在以下情況下受到支援 AWS 區域
>
> - 美國東部 (維吉尼亞北部)
> - 美國西部 (奧勒岡)
> - 亞太區域 (東京)
> - 歐洲 (愛爾蘭)

為了測試您的裝置，免費伺服器的 IDT 會建立類似AWS IoT事物、FreeRTOS 群組和 Lambda 函數等資源。若要建立這些資源，FreeRTOS 的 IDT 需要您建立和設定AWS帳戶，以及授予 FreeRTOS IDT 權限的 IAM 政策，以便在執行測試時代表您存取資源。

以下步驟是建立和設定您的AWS帳戶。

1. 若您已經擁有 AWS 帳戶，請跳至下一項任務，否則創建一個AWS帳戶。
2. 請依照建立 IAM 角色中的步驟進行。此時請勿新增權限或原則。
3. 要運行 OTA 資格測試，請轉到步驟 4。否則請轉到步驟 5。
4. 將 OTA IAM 許可內嵌政策附加到您的 IAM 角色。

   a.

   > **Important**
   >
   > 下列政策範本會授予 IDT 許可，允許使用者建立角色、建立政策，以及將政策附加至角色。FreeRTOS 的 IDT 會將這些權限用於建立角色的測試。雖然政策範本不會為使用者提供系統管理員權限，但這些權限可以用來取得AWS帳戶的管理員存取權。

   b. 按照以下步驟將必要的許可連接到 IAM 角色：

i. 在 [權限] 頁面上，選擇 [新增權限]。

ii. 選擇 [建立內嵌原則]。

iii. 選擇 JSON 索引標籤，並將以下權限複製到 JSON 文字方塊中。如果您不在中國地區，請使用「大多數區域」下的範本。如果您位於中國地區，請使用北京和寧夏地區下的範本。

Most Regions

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

```
                "iot:DescribeJob",
                "iot:DescribeThing",
                "iot:GetPolicy",
                "iot:ListAttachedPolicies",
                "iot:ListCertificates",
                "iot:ListPrincipalPolicies",
                "iot:ListThingPrincipals",
                "iot:ListThings",
                "iot:Publish",
                "iot:UpdateThingShadow",
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:DescribeLogGroups",
                "logs:DescribeLogStreams",
                "logs:PutLogEvents",
                "logs:PutRetentionPolicy"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iotdeviceadvisor:*",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "logs:DeleteLogGroup",
            "Resource": "arn:aws:logs:*:*:log-group:/aws/iot/
deviceadvisor/*"
        },
        {
            "Effect": "Allow",
            "Action": "logs:GetLogEvents",
            "Resource": "arn:aws:logs:*:*:log-group:/aws/iot/
deviceadvisor/*:log-stream:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreatePolicy",
                "iam:DetachRolePolicy",
                "iam:DeleteRolePolicy",
                "iam:DeletePolicy",
                "iam:CreateRole",
```

```
                "iam:DeleteRole",
                "iam:AttachRolePolicy"
            ],
            "Resource": [
                "arn:aws:iam::*:policy/idt*",
                "arn:aws:iam::*:role/idt*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ssm:GetParameters"
            ],
            "Resource": [
                "arn:aws:ssm:*::parameter/aws/service/ami-amazon-linux-
latest/amzn2-ami-hvm-x86_64-gp2"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeInstances",
                "ec2:RunInstances",
                "ec2:CreateSecurityGroup",
                "ec2:CreateTags",
                "ec2:DeleteTags"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateKeyPair",
                "ec2:DeleteKeyPair"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:key-pair/idt-ec2-ssh-key-*"
            ]
        },
        {
            "Effect": "Allow",
            "Condition": {
```

```
                "StringEqualsIgnoreCase": {
                    "aws:ResourceTag/Owner": "IoTDeviceTester"
                }
            },
            "Action": [
                "ec2:TerminateInstances",
                "ec2:DeleteSecurityGroup",
                "ec2:AuthorizeSecurityGroupIngress",
                "ec2:RevokeSecurityGroupIngress"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Beijing and Ningxia Regions

北京和寧夏區域中可使用下列政策範本。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreatePolicy",
                "iam:DetachRolePolicy",
                "iam:DeleteRolePolicy",
                "iam:DeletePolicy",
                "iam:CreateRole",
                "iam:DeleteRole",
                "iam:AttachRolePolicy"
            ],
            "Resource": [
                "arn:aws-cn:iam::*:policy/idt*",
                "arn:aws-cn:iam::*:role/idt*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
```

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

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

iv. 完成時，請選擇 Review policy (檢閱政策)。

v. 輸入 ID Certificate。

vi. 選擇 建立政策。

5. 附加AWSIoTDeviceTesterForFreeRTOSFullAccess到您的 IAM 角色。

   a. 若要將必要的許可附加到 IAM 角色：

      i. 在 [權限] 頁面上，選擇 [新增權限]。

      ii. 選擇 Attach policies (連接政策)。

      iii. 搜尋原AWSIoTDeviceTesterForFreeRTOSFullAccess則。勾選方塊。

   b. 選擇 Add permissions (新增許可)。

6. 匯出 IDT 的身分證明。如需詳細資訊，請參閱取得 CLI 存取的 IAM 角色憑證。

AWS IoT Device Tester 受管政策

受`AWSIoTDeviceTesterForFreeRTOSFullAccess`管理的原則包含下列版本檢查、auto 更新功能及指標集合的AWS IoT Device Tester權限。

- `iot-device-tester:SupportedVersion`

  授AWS IoT Device Tester予獲取支持產品，測試套件和 IDT 版本列表的權限。

- `iot-device-tester:LatestIdt`

  授AWS IoT Device Tester予獲取可供下載的最新 IDT 版本的權限。

- `iot-device-tester:CheckVersion`

  授AWS IoT Device Tester予檢查 IDT、測試套件和產品版本相容性的權限。

- `iot-device-tester:DownloadTestSuite`

  授AWS IoT Device Tester予下載測試套件更新的權限。

- `iot-device-tester:SendMetrics`

授AWS予收集有關AWS IoT Device Tester內部使用量度之指標的權限。

(選用) 安裝 AWS Command Line Interface

您可能偏好使用 AWS CLI 來執行部分操作。如果您沒有安AWS CLI裝，請遵循安裝 AWS CLI.

透過aws configure從指令行執行，AWS CLI為您要使用的AWS區域設定。如需支援 FreeRTOS 之 IDT 之AWS區域的相關資訊，請參閱AWS區域和端點。若要取得有aws configure關的詳細資訊，aws configure請參閱快速組態

首次準備測試微型控制器主機板

您可以針對 FreeRTOS 使用 IDT 來測試您對 FreeRTOS 程式庫的實作。為主機板的裝置驅動程式移植 FreeRTOS 程式庫之後，請使用AWS IoT Device Tester在微控制器主機板上執行認證測試。

新增程式庫移植層並實作 FreeRTOS 測試儲存庫

若要為您的裝置移植 FreeRTOS，請參閱 Free RTOS 移植指南。實作 FreeRTOS 測試儲存庫並移植 FreeRTOS 層時，您必須提供每個程式庫 (包括測試儲存庫) 的路徑。`manifest.yml`此檔案將位於源碼的根目錄中。如需詳細資訊，請參閱資訊清單檔

設定 AWS 憑證

您必須設定AWS認證，AWS IoT Device Tester才能與AWS雲端通訊。如需詳細資訊，請參閱設定AWS認證和開發區域。有效的AWS身分證明會在*devicetester_extract_location*/devicetester_freertos_*[win|mac|linux]*/configs/config.json組態檔中指定。

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

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

該config.json文件的auth屬性有一個控制AWS身份驗證的方法字段，並且可以聲明為文件或環境。將欄位設定為環境會從主機的環境變數中提取您的AWS認證。將欄位設定為檔案會從組態檔匯入指定的設.aws/credentials定檔。

在 IDT 中建立 FreeRTOS 的裝置集區

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。您可以將 FreeRTOS 的 IDT 設定為測試單一裝置或集區中的多個裝置。為了加速認證流程，FreeRTOS 的 IDT 可以同時測試具有相同規格的設備。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。

該device.json文件的頂層有一個數組。每個陣列屬性都是新的裝置集區。每個設備池都有一個設備數組屬性，該屬性具有聲明多個設備。在範本中，有一個裝置集區，該裝置集區中只有一個裝置。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**
>
> 同一集區中的所有裝置必須是相同的技術規格和 SKU。若要為不同的測試群組啟用原始程式碼的 parallel 建置，FreeRTOS 的 IDT 會將原始程式碼複製到結果資料夾中 IDT (FreeRTOS) 解壓縮資料夾中的結果資料夾。您必須使用變數參考組建或 Flash 命令中的原始程式碼路testdata.sourcePath徑。FreeRTOS 的 IDT 會以複製原始程式碼的暫存路徑取代此變數。如需詳細資訊，請參閱FreeRTOS 變數的 IDT。

以下是用來建立包含多個裝置的裝置集區的範例device.json檔案。

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

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

```
          }
        ]
    }
]
```

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**
>
> 如果您要在AWS合作夥伴裝置目錄中刊登主機板，您在此指定的 SKU 必須符合您在刊登程序中使用的 SKU。

features

包含裝置支援功能的陣列。 AWS IoT Device Tester使用此資訊來選取要執行的資格測試。

支援的值如下：

Wifi

指出您的面板是否具有 Wi-Fi 功能。

Cellular

指出您的主機板是否有行動網路功能。

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ECCRSA、和Both。 Both表示主機板同時支援ECC和RSA。

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ImportOnboard、Both和No。 OnboardBoth、或No金鑰佈建需要資格。Import單獨不是資格的有效選擇。

- Import僅當您的主機板允許導入私鑰時才使用。選擇對於合格而言不Import是有效的配置，應該僅用於測試目的，特別是用於 PKCS11 測試用例。 Onboard，Both或者No是資格所必需的。
- 如Onboard果您的主機板支援內建的私密金鑰 (例如，您的裝置具有安全元素，或者您想要產生自己的裝置金 key pair 和憑證)，請使用此選項。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Both果您的主機板同時支援匯入私密金鑰和內建金鑰產生以進行金鑰佈建，請使用
- 在No您的主機板不支援金鑰佈建時使用。 No只有當您的設備也已預先配置時才是有效的選項。

OTA

指出您的主機板是否支援 over-the-air (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。需要具有 HTTP 或 MQTT 數據通道協議的 OTA 才能進行認證。要在測試時跳過運行 OTA 測試，請將 OTA 功能設置為，No並將OtaDataPlaneProtocol屬性設置為None。這將不是資格賽。

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.serialPort

要測試用於連接到裝置之主機電腦的序列埠。

devices.secureElementConfig.PublicKeyAsciiHexFilePath

如果您的主機板未提供pre-provisioned或未提供，則需要PublicDeviceCertificateArn此選項。由於Onboard是必要的金鑰佈建類型，因此 FullTransportInterface TLS 測試群組目前需要此欄位。如果您的設備是pre-provisioned，則PublicKeyAsciiHexFilePath是可選的，不需要包括在內。

下列區塊是檔案的絕對路徑，其中包含從Onboard私密金鑰擷取的十六進位元組公開金鑰。

```
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
```

如果您的公鑰是 .der 格式，則可以直接對公鑰進行十六進制編碼以生成十六進制文件。

若要從 .der 公開金鑰產生十六進位檔案，請輸入下列xxd指令：

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

如果您的公鑰是 .pem 格式，則可以提取 base64 編碼的頁眉和頁腳並將其解碼為二進制格式。然後，您將十六進制字符串編碼以生成十六進制文件。

若要為 .pem 公開金鑰產生十六進位檔案，請執行下列動作：

1. 執行下列base64命令，從公開金鑰移除 base64 頁首和頁尾。然後將解碼的密鑰（名為base64key）輸出到文件pubkey.der中：

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

2. 運行以下xxd命令以轉換為十六pubkey.der進制格式。產生的金鑰會另存為 *outFile*

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

devices.secureElementConfig.PublicDeviceCertificateArn

從您的安全元素上傳到AWS IoT Core的憑證 ARN。如需將憑證上傳至的相關資訊AWS IoT Core，請參閱AWS IoT開發人員指南中的 X.509 用戶端憑證。

devices.secureElementConfig.SecureElementSerialNumber

(選用) 安全元素的序號。序號可選擇性地用於建立 JITR 金鑰佈建的裝置憑證。

devices.secureElementConfig.preProvisioned

(選擇性) 如果裝置具有預先佈建的安全元素，且具有鎖定認證，無法匯入、建立或銷毀物件，則設定為「是」。如果此屬性設定為 「是」，您必須提供對應的 pkcs11 標籤。

devices.secureElementConfig.pkcs11JITPCodeVerifyRootCertSupport

(選擇性) 如果裝置的 CoRepkCS11 實作支援 JITP 儲存，請設定為是。這將在測codeverify試核心 PKCS 11 時啟用 JITP 測試，並且需要提供代碼驗證金鑰、JITP 憑證和根憑證 PKCS 11 標籤。

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

適用於 FreeRTOS 的 IDT 會自動在您的主機板上建立並閃爍測試。若要啟用此功能，您必須設定 IDT 來執行硬體的建置和 Flash 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。下列 JSON 範例顯示如何為 FreeRTOS 設定 IDT 以測試多個裝置：

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

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

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

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

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。

`retainModifiedSourceDirectories`

(選擇性) 檢查是否要保留建置期間所使用的修改來源目錄，以及為了偵錯而進行更新。如果設定為`true`，修改後的來源目錄會命名為 RetainedSRC，並在每次執行測試群組的結果記錄檔資料夾中找到。如果未包含，則欄位預設為`false`。

`freeRTOSTestParamConfigPath`

`test_param_config.h`文件的路徑為自由圖書館集成測試集成。此檔案必須使用`{{testData.sourcePath}}`預留位置變數，使其相對於原始程式碼根目錄。 AWS IoT Device Tester使用此文件中的參數來配置測試。

`freeRTOSTestExecutionConfigPath`

`test_execution_config.h`文件的路徑為自由圖書館集成測試集成。此檔案必須使用`{{testData.sourcePath}}`預留位置變數，使其相對於儲存庫根目錄。 AWS IoT Device Tester使用此檔案來控制必須執行哪些測試。

`freeRTOSVersion`

FreeRTOS 的版本，包括您實作中使用的修補程式版本。請參閱與 FreeRTOS 相容的 AWS IoT Device Tester FreeRTOS 版本的支援版本。AWS IoT Device Tester

buildTool

建置程式碼的命令。在 build 命令中對源代碼路徑的所有引用都必須由AWS IoT Device Tester變量替換{{testData.sourcePath}}。使用{{config.idtRootPath}}預留位置來參考相對於AWS IoT Device Tester根路徑的建置指令碼。

flashTool

將影像閃爍到裝置的指令。ash 命令中對源代碼路徑的所有引用都必須由AWS IoT Device Tester變量{{testData.sourcePath}}替換。使用{{config.idtRootPath}}預留位置來參考相對於AWS IoT Device Tester根路徑的 Flash 指令碼。

> **Note**
>
> 使用 FRQ 2.0 的新集成測試結構不需要路徑變量，例如{{enableTests}}和{{buildImageName}}. OTA 端對端測試使用 FreeRTOS 庫集成測試存儲庫中提供的配置模板運行GitHub。如果GitHub存儲庫中的文件存在於父源代碼項目中，則在測試之間不會更改源代碼。如果需要 OTA 端到端的不同構建映像，則必須在構建腳本中構建此映像並在下指定的userdata.json文件中指定它otaConfiguration。

testStartDelayms

指定 FreeRTOS 測試執行程式在開始執行測試之前等待的毫秒數。如果被測設備在 IDT 因網絡或其他延遲問題而有機會連接並開始日誌記錄之前開始輸出重要的測試信息，這很有用。此值僅適用於 FreeRTOS 測試群組，不適用於不使用 FreeRTOS 測試執行程式的其他測試群組，例如 OTA 測試。如果您收到與預期 10 相關的錯誤，但收到 5，則此欄位應設定為 5000。

echoServerConfiguration

為 TLS 測試設定回應伺服器的組態。此欄位為必填。

keyGenerationMethod

使用此選項配置了 echo 服務器。選項包括 EC 或 RSA。

serverPort

執行 echo 伺服器的連接埠號碼。

otaConfiguration

OTA PAL 和 OTA E2E 測試的配置。此欄位為必填。

otaE2EFirmwarePath

IDT 用於 OTA 端到端測試的 OTA bin 映像的路徑。

otaPALCertificatePath

在設備上進行 OTA PAL 測試的證書的路徑。此憑證用於驗證簽章。例如，埃克薩沙 256-簽名者 .crt.pem。

deviceFirmwarePath

要開機之韌體映像檔之硬式編碼名稱的路徑。如果您的裝置不使用檔案系統進行韌體開機，請將此欄位指定為'NA'。如果您的設備使用檔案系統進行韌體開機，請指定韌體開機映像的路徑或名稱。

codeSigningConfiguration

signingMethod

程式碼簽章方法。可能的值為AWS或自訂。

> **Note**
>
> 對於北京和寧夏區域，請使用 Cuate。 AWS該區域不支援程式碼簽章。

signerHashingAlgorithm

裝置上支援的雜湊演算法。可能的值為 SHA1 或 SHA256。

signerSigningAlgorithm

裝置上支援的簽署演算法。可能的值為 RSA 或 ECDSA。

signerCertificate

用於 OTA 的信任憑證。對於程式AWS碼簽章方法，請使用 Amazon Resource Name (ARN)，上傳到 Certificate Manager 的受信任 Certificate Name (ARN)。AWS對於自訂程式碼簽章方法，請使用簽署者憑證檔案的絕對路徑。如需建立受信任憑證的詳細資訊，請參閱建立程式碼簽署憑證。

untrustedSignerCertificate

在某些 OTA 測試中用作不受信任證書的第二個證書的 ARN 或文件路徑。如需建立憑證的相關資訊，請參閱建立程式碼簽署憑證。

signerCertificateFileName

裝置上程式碼簽章憑證的檔案名稱。此值必須與您在執行aws acm import-certificate指令時提供的檔案名稱相符。

compileSignerCertificate

決定簽章驗證憑證狀態的布林值。有效值為 true 和 false。

如果未佈建或刷新程式碼簽署者簽章驗證憑證，請將此值設定為 true。它必須被編譯到項目中。 AWS IoT Device Tester獲取受信任的證書並將其編譯成。aws_codesigner_certificate.h

signerPlatform

AWS 程式碼簽署者在建立 OTA 更新任務時使用的簽署和雜湊演算法。目前，此欄位的可能值為 AmazonFreeRTOS-TI-CC3220SF 和 AmazonFreeRTOS-Default。

- 如果是 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

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。

pkcs11LabelConfiguration

PKCS11 標籤組態至少需要一組裝置憑證標籤、公開金鑰標籤和私密金鑰標籤的標籤，才能執行 PKCS11 測試群組。所需的 PKCS11 標籤取決於device.json檔案中的裝置配置。如果在中將預先佈建設定為 「是」device.json，則必要的標籤必須是以下其中一個，具體取決於 PKCS11 功能所選擇的標籤。

- PreProvisionedEC
- PreProvisionedRSA

如果預先佈建設定為 No indevice.json，則需要的標籤為：

- pkcs11LabelDevicePrivateKeyForTLS
- pkcs11LabelDevicePublicKeyForTLS

- pkcs11LabelDeviceCertificateForTLS

僅當您pkcs11JITPCodeVerifyRootCertSupport在device.json檔案中選取「是」時，才需要以下三個標籤。

- pkcs11LabelCodeVerifyKey
- pkcs11LabelRootCertificate
- pkcs11LabelJITPCertificate

這些欄位的值應與 FreeRTOS 移植指南中定義的值相符。

pkcs11LabelDevicePrivateKeyForTLS

(選擇性) 此標籤用於私密金鑰的 PKCS #11 標籤。對於具有內建和匯入支援金鑰佈建的裝置，此標籤可用於測試。此標籤可能與為預先佈建案例定義的標籤不同。如果您將金鑰佈建設定設定為否，並將預先佈建設為是，則在中device.json，這將是未定義的。

pkcs11LabelDevicePublicKeyForTLS

(選擇性) 此標籤用於公開金鑰的 PKCS #11 標籤。對於具有內建和匯入支援金鑰佈建的裝置，此標籤可用於測試。此標籤可能與為預先佈建案例定義的標籤不同。如果您將金鑰佈建設定設定為否，並將預先佈建設為是，則在中device.json，這將是未定義的。

pkcs11LabelDeviceCertificateForTLS

(選擇性) 此標籤用於裝置憑證的 PKCS #11 標籤。對於具有內建和匯入支援金鑰佈建的裝置，此標籤將用於測試。此標籤可能與為預先佈建案例定義的標籤不同。如果您將金鑰佈建設定設定為否，並將預先佈建設為是，則在中device.json，這將是未定義的。

pkcs11LabelPreProvisionedECDevicePrivateKeyForTLS

(選擇性) 此標籤用於私密金鑰的 PKCS #11 標籤。對於具有安全元素或硬體限制的裝置，這會有不同的標籤來保留AWS IoT憑證。如果您的裝置支援使用 EC 金鑰預先佈建，請提供此標籤。在device.json中將「預先佈建」設定為 「是」時pkcs11LabelPreProvisionedRSADevicePrivateKeyForTLS，必須提供此標籤或兩者。此標籤可能與針對機載和進口案例定義的標籤不同。

pkcs11LabelPreProvisionedECDevicePublicKeyForTLS

(選擇性) 此標籤用於公開金鑰的 PKCS #11 標籤。對於具有安全元素或硬體限制的裝置，這會有不同的標籤來保留AWS IoT憑證。如果您的裝置支援使用 EC 金鑰預先佈建，請提供此標籤。在device.json中將「預先佈建」設定為 「是」時pkcs11LabelPreProvisionedRSADevicePublicKeyForTLS，必須提供此標籤或兩者。此標籤可能與針對機載和進口案例定義的標籤不同。

pkcs11LabelPreProvisionedECDeviceCertificateForTLS

(選擇性) 此標籤用於裝置憑證的 PKCS #11 標籤。對於具有安全元素或硬體限制的裝置，這會有不同的標籤來保留AWS IoT憑證。如果您的裝置支援使用 EC 金鑰預先佈建，請提供此標籤。在device.json中將「預先佈建」設定為 「是」時pkcs11LabelPreProvisionedRSADeviceCertificateForTLS，必須提供此標籤或兩者。此標籤可能與針對機載和進口案例定義的標籤不同。

pkcs11LabelPreProvisionedRSADevicePrivateKeyForTLS

(選擇性) 此標籤用於私密金鑰的 PKCS #11 標籤。對於具有安全元素或硬體限制的裝置，這會有不同的標籤來保留AWS IoT憑證。如果您的裝置支援使用 RSA 金鑰進行預先佈建，請提供此標籤。在device.json中將「預先佈建」設定為 「是」時pkcs11LabelPreProvisionedECDevicePrivateKeyForTLS，必須提供此標籤或兩者。

pkcs11LabelPreProvisionedRSADevicePublicKeyForTLS

(選擇性) 此標籤用於公開金鑰的 PKCS #11 標籤。對於具有安全元素或硬體限制的裝置，這會有不同的標籤來保留AWS IoT憑證。如果您的裝置支援使用 RSA 金鑰進行預先佈建，請提供此標籤。在device.json中將「預先佈建」設定為 「是」時pkcs11LabelPreProvisionedECDevicePublicKeyForTLS，必須提供此標籤或兩者。

pkcs11LabelPreProvisionedRSADeviceCertificateForTLS

(選擇性) 此標籤用於裝置憑證的 PKCS #11 標籤。對於具有安全元素或硬體限制的裝置，這會有不同的標籤來保留AWS IoT憑證。如果您的裝置支援使用 RSA 金鑰進行預先佈建，請提供此標籤。在device.json中將「預先佈建」設定為 「是」時pkcs11LabelPreProvisionedECDeviceCertificateForTLS，必須提供此標籤或兩者。

pkcs11LabelCodeVerifyKey

(選擇性) 此標籤用於驗證碼金鑰的 PKCS #11 標籤。如果您的裝置具有 JITP 憑證、代碼驗證金鑰和根憑證的 PKCS #11 儲存支援，請提供此標籤。當 pkcs11JITPCodeVerifyRootCertSupport in device.json 設定為 「是」 時，必須提供此標籤。

pkcs11LabelJITPCertificate

(選擇性) 此標籤用於 JITP 憑證的 PKCS #11 標籤。如果您的裝置具有 JITP 憑證、代碼驗證金鑰和根憑證的 PKCS #11 儲存支援，請提供此標籤。當

pkcs11JITPCodeVerifyRootCertSupport in device.json 設定為 「是」 時，必須提供此標籤。

FreeRTOS 變數的 IDT

建立程式碼和快閃記憶體裝置的命令可能需要連線或裝置的其他相關資訊，才能成功執行。 AWS IoT Device Tester允許您在 Flash 中引用設備信息並使用構建命令JsonPath。通過使用簡單的JsonPath表達式，您可以獲取device.json文件中指定的所需信息。

路徑變數

FreeRTOS 的 IDT 會定義下列路徑變數，可用於命令列和組態檔案的路徑變數：

{{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。

{{device.connectivity.serialPort}}

展開至序列埠。

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

展開至裝置的序號。

{{config.idtRootPath}}

展開到AWS IoT Device Tester根路徑。

使用免 FreeRTOS 使用者介面執行 FreeRTOS 資格套件 2.0 (FRQ 2.0)

AWS IoT Device Tester FreeRTOS (FreeRTOS 的 IDT) 包含網頁式使用者介面 (UI)，您可以在其中與 IDT 命令列應用程式及相關的設定檔進行互動。您可以使用 FreeRTOS 使用者介面為您的裝置建立新的組態或修改現有的設定。您也可以使用使用者介面呼叫 IDT 應用程式，並針對您的裝置執行 FreeRTOS 測試。

如需如何使用命令列執行限定測試的相關資訊，請參閱首次準備測試微型控制器主機板。

本節說明 FreeRTOS 使用者介面的 IDT 先決條件，以及如何從 UI 執行資格測試。

主題

- 必要條件
- 配置 AWS 憑證

- 開啟 FreeRTOS 使用者介面的 IDT
- 建立新的模型組態
- 修改現有的模型組態
- 執行資格測試

必要條件

若要透過 FreeRTOS 使用者介面 AWS IoT Device Tester (IDT) 執行測試，您必須完成 IDT FreeRTOS 資格 (FRQ) 2.x 先決條件 頁面上的必要條件。

配置 AWS 憑證

您必須為在中建立的使用者設定 IAM 使 AWS 用者登入資料建立並設定 AWS 帳戶。您可以使用下列兩種方式的其中之一指定登入資料：

- 在認證檔案中
- 作為環境變量

使用 AWS 認證檔設定認證

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。

身份證明檔的位置會根據您使用的作業系統而有所不同：

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

以下列格式將您的 AWS 認證新增至`credentials`檔案：

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

> **Note**
>
> 如果您不使用`default` AWS 設定檔，您必須在 FreeRTOS 使用者介面的 IDT 中指定設定檔名稱。如需有關設定檔的詳細資訊，請參閱具名設定檔

使用環境變數設定 AWS 認證

環境變數是由作業系統維護且由系統命令使用的變數。如果您關閉 SSH 工作階段，則不會儲存它們。FreeRTOS 使用者介面的 IDT 會使用AWS_ACCESS_KEY_ID和AWS_SECRET_ACCESS_KEY環境變數來儲存您的認證。 AWS

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
```

開啟 FreeRTOS 使用者介面的 IDT

若要開啟免費伺服器使用者介面的 IDT

1. 下載 FreeRTOS 版本支援的 IDT。然後將下載的歸檔解壓縮到您具有讀取和寫入權限的目錄中。
2. 瀏覽至 FreeRTOS 安裝目錄的 IDT：

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

3. 執行下列命令以開啟 FreeRTOS 使用者介面的 IDT：

   Linux

   ```
   .devicetester_ui_linux_x86-64
   ```

   Windows

   ```
   ./devicetester_ui_win_x64-64
   ```

   macOS

   ```
   ./devicetester_ui_mac_x86-64
   ```

> **Note**
>
> 在 macOS 中，若要允許您的系統執行 UI，請前往「系統偏好設定」->「安全性與隱私權」。當你運行測試時，你可能需要再做三次。這

FreeRTOS 使用者介面的 IDT 會在您的預設瀏覽器中開啟。以下瀏覽器的最新三個主要版本支援使用者介面：

- Google Chrome
- Mozilla Firefox
- Microsoft Edge
- 適用於 macOS 的 Apple Safari

> **Note**
>
> 為了獲得更好的體驗，我們建議谷歌瀏覽器或火狐瀏覽器訪問 IDT 免費使用者介面。Microsoft 互聯網資源管理器不受 UI 支持。

> **Important**
>
> 在開啟 UI 之前，您必須先設定 AWS 認證。如果您尚未設定認證，請關閉 FreeRTOS 使用者介面的 IDT 瀏覽器視窗，遵循中配置 AWS 憑證的步驟，然後重新開啟 FreeRTOS 使用者介面的 IDT。

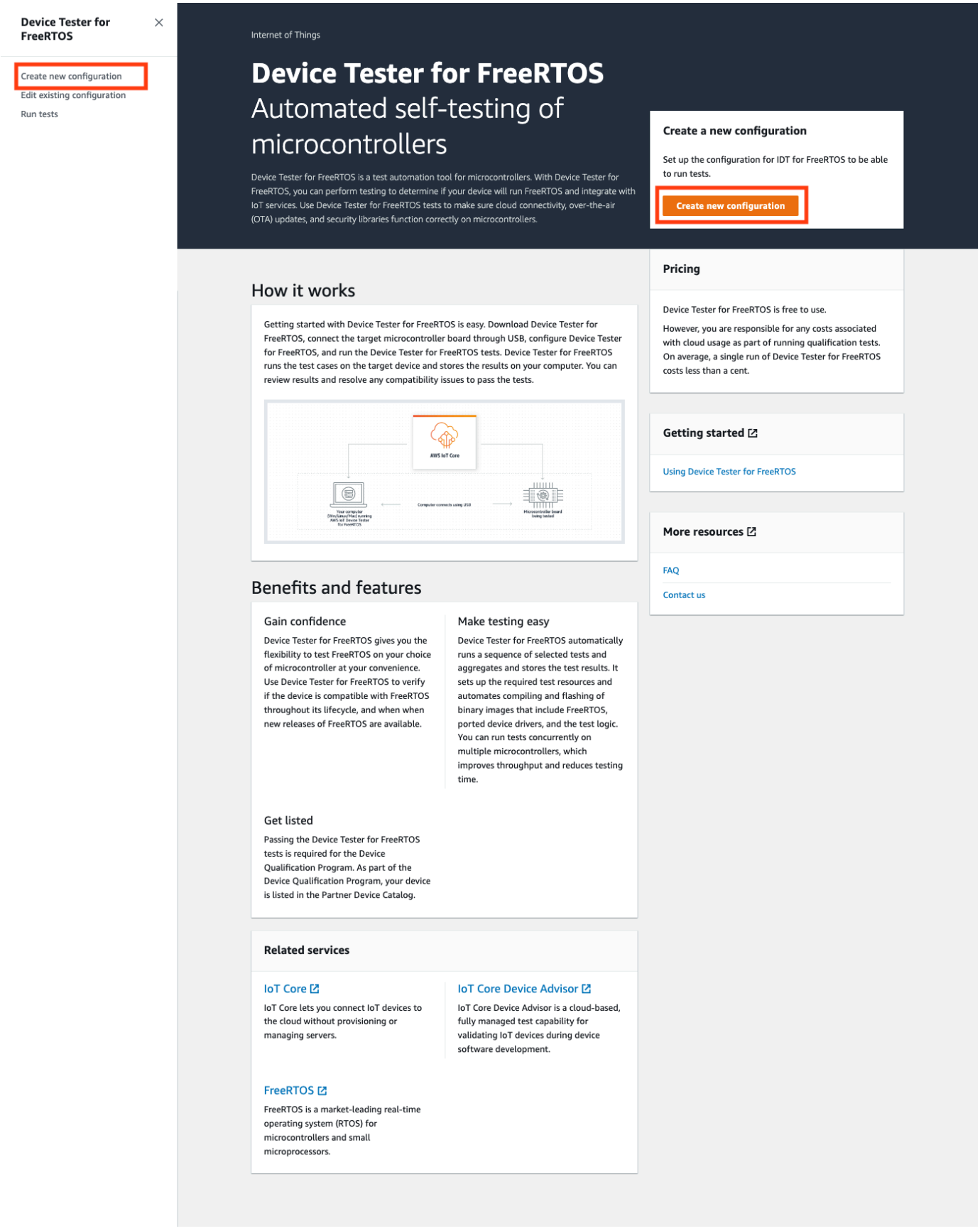
建立新的模型組態

如果您是初次使用的使用者，您必須建立新的組態來設定 FreeRTOS 的 IDT 執行測試所需的 JSON 設定檔。然後，您可以運行測試或修改創建的配置。

如需`config.json`、`device.json`和`userdata.json`檔案的範例，請參閱首次準備測試微型控制器主機板。

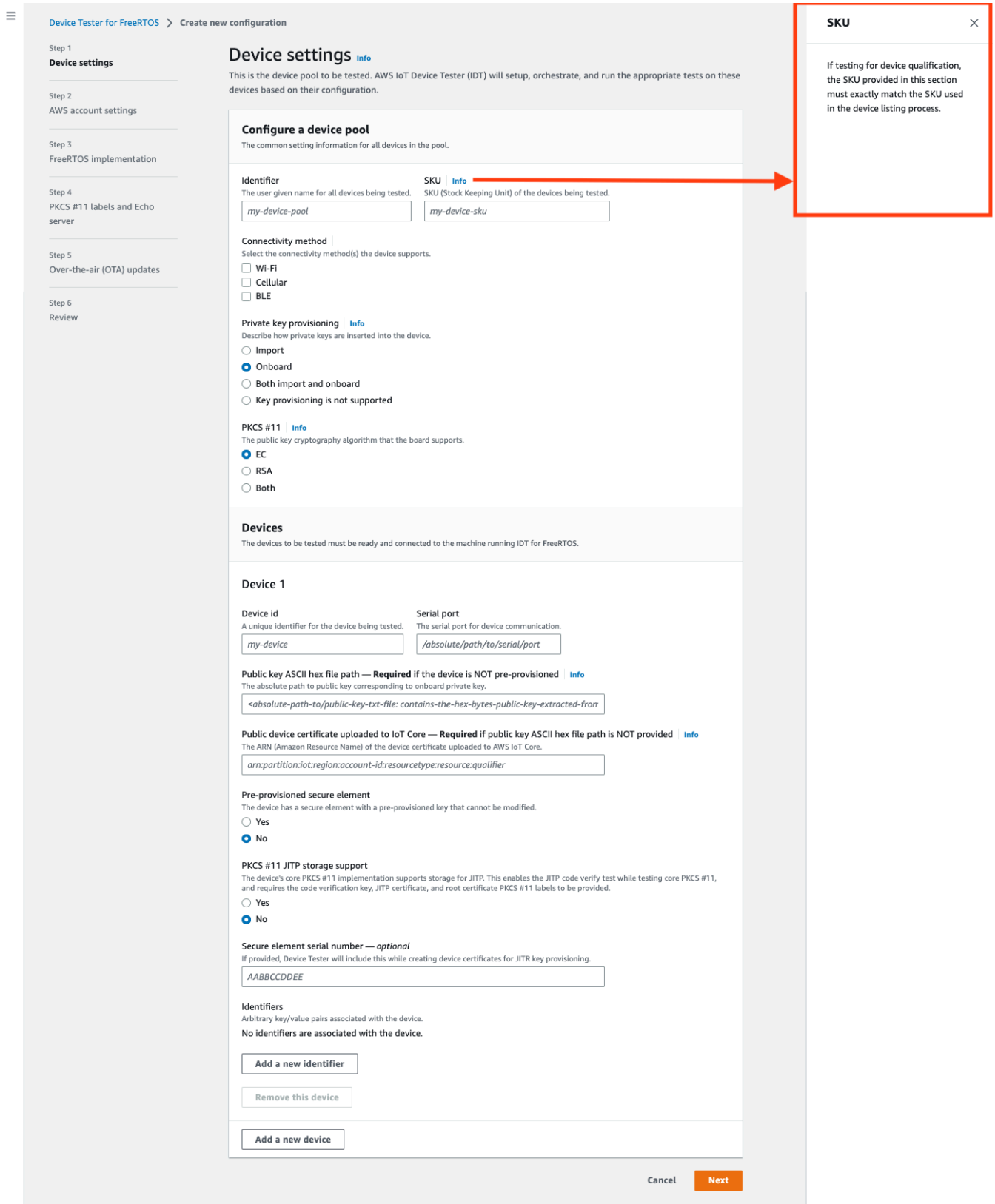
建立新的組態。

1. 在 FreeRTOS 使用者介面的 IDT 中，開啟瀏覽功能表，然後選擇 [建立新的組態]。

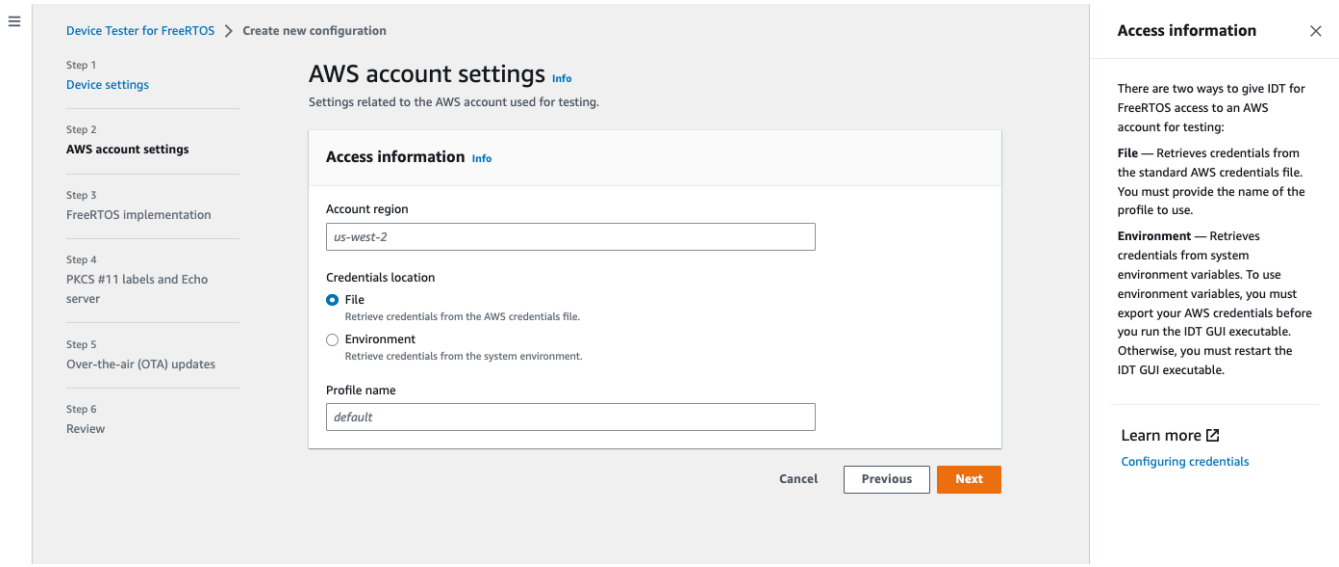

Device Tester for FreeRTOS
Create new configuration
Edit existing configuration
Run tests
Internet of Things
Device Tester for FreeRTOS
Automated self-testing of microcontrollers
Device Tester for FreeRTOS is a test automation tool for microcontrollers. With Device Tester for FreeRTOS, you can perform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

2. 遵循組態精靈，輸入用於執行資格測試的 IDT 組態設定。精靈會在目錄中的 JSON 組態檔案中設定下列設定。*devicetester-extract-location*/config

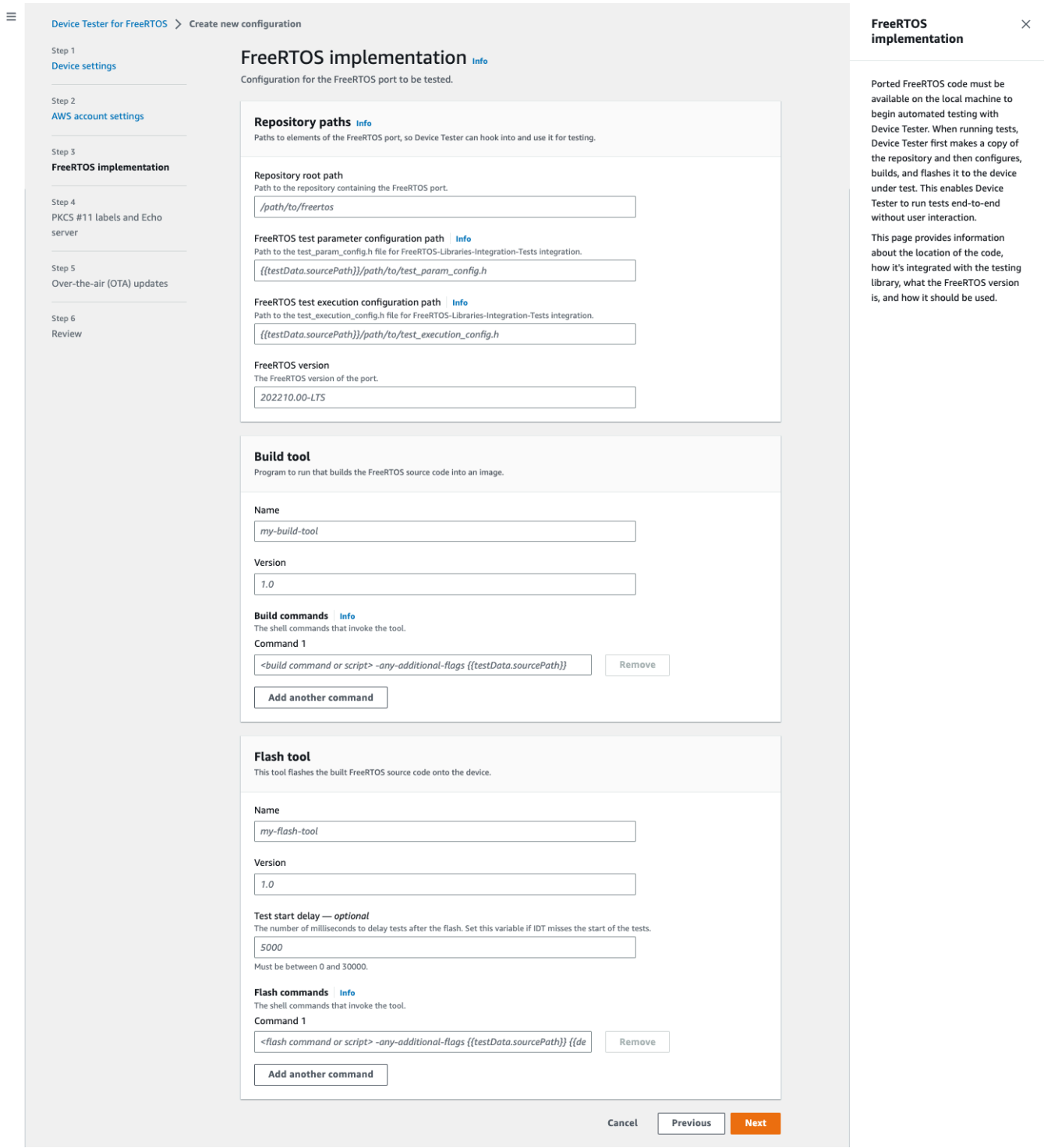
   - [裝置設定] — 要測試之裝置的裝置集區設定。這些設定會在**id**和**sku**欄位中進行設定，且**config.json**檔案中裝置集區的裝置區塊。



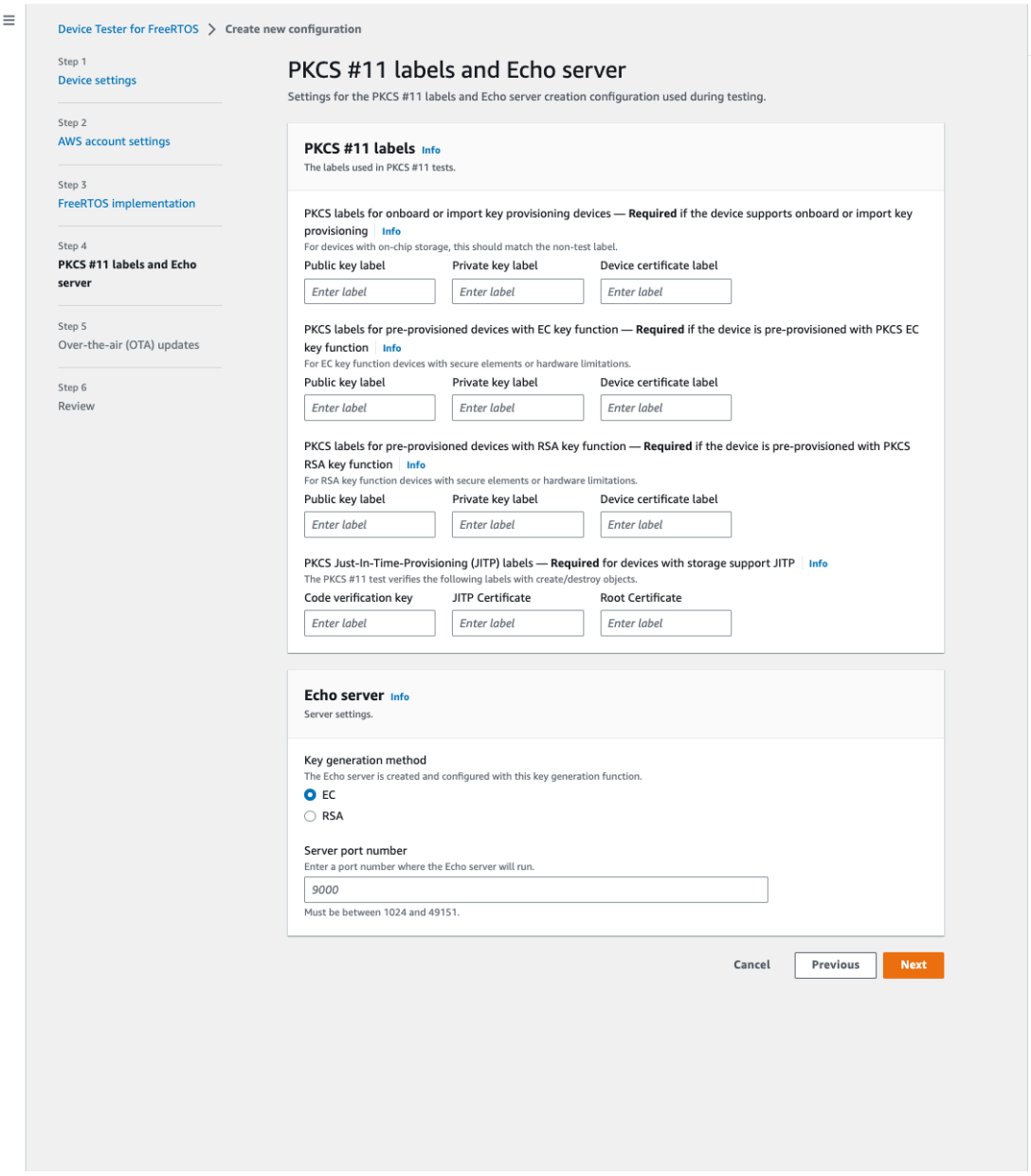
- AWS 帳號設定 — FreeRTOS 的 IDT 用來在測試執行期間建立資 AWS 源的資 AWS 帳戶 訊。這些設定是在`config.json`檔案中設定的。



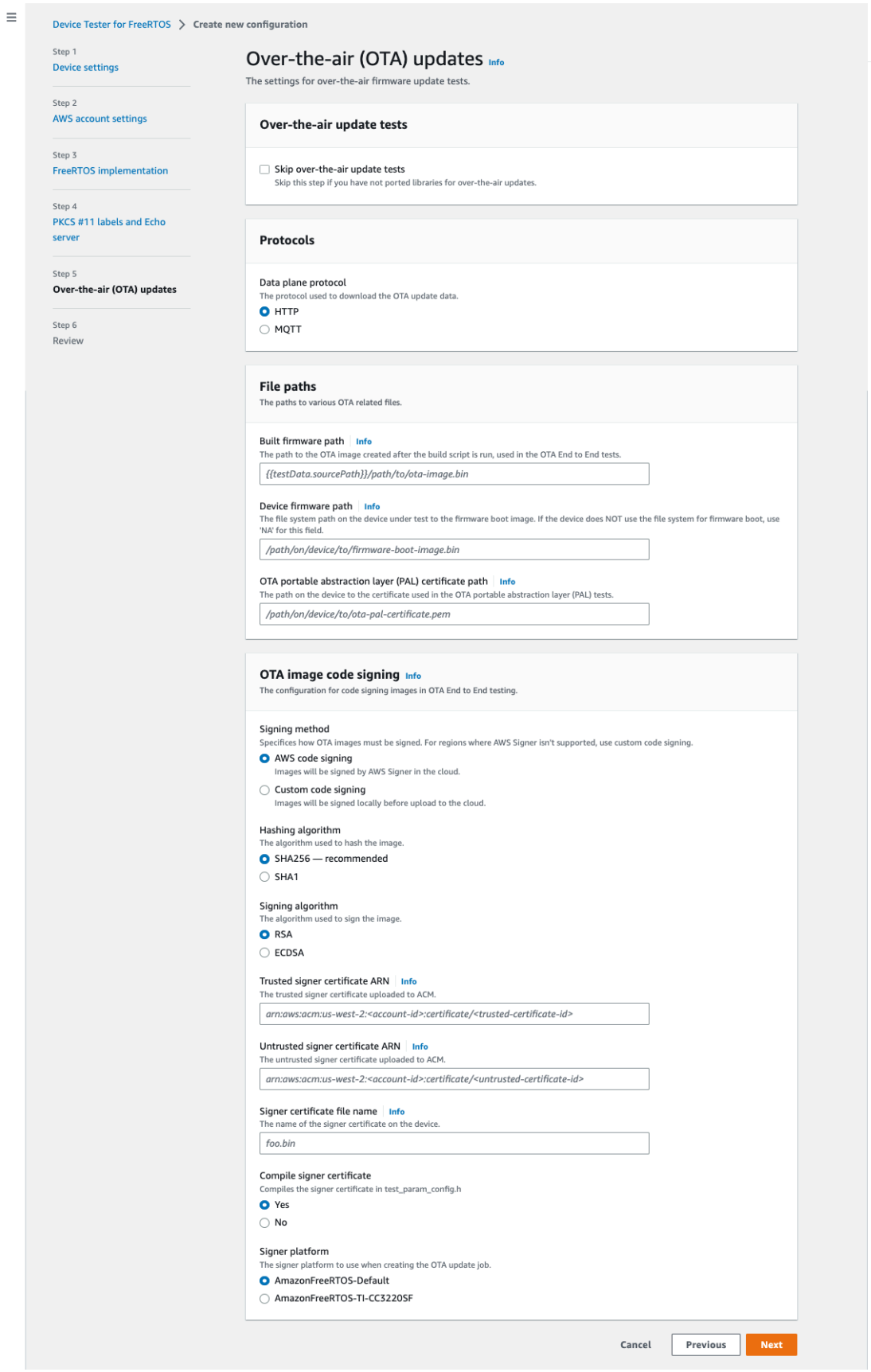
- FreeRTOS 實作 — FreeRTOS 儲存庫和移植程式碼的絕對路徑，以及您要在其上執行 IDT FRQ 的 FreeRTOS 版本。`FreeRTOS-Libraries-Integration-Tests` GitHub 儲存庫中執行和參數組態標頭檔案的路徑。硬件的構建和閃存命令，允許 IDT 自動構建和閃存測試到您的主板。這些設定是在`userdata.json`檔案中設定的。



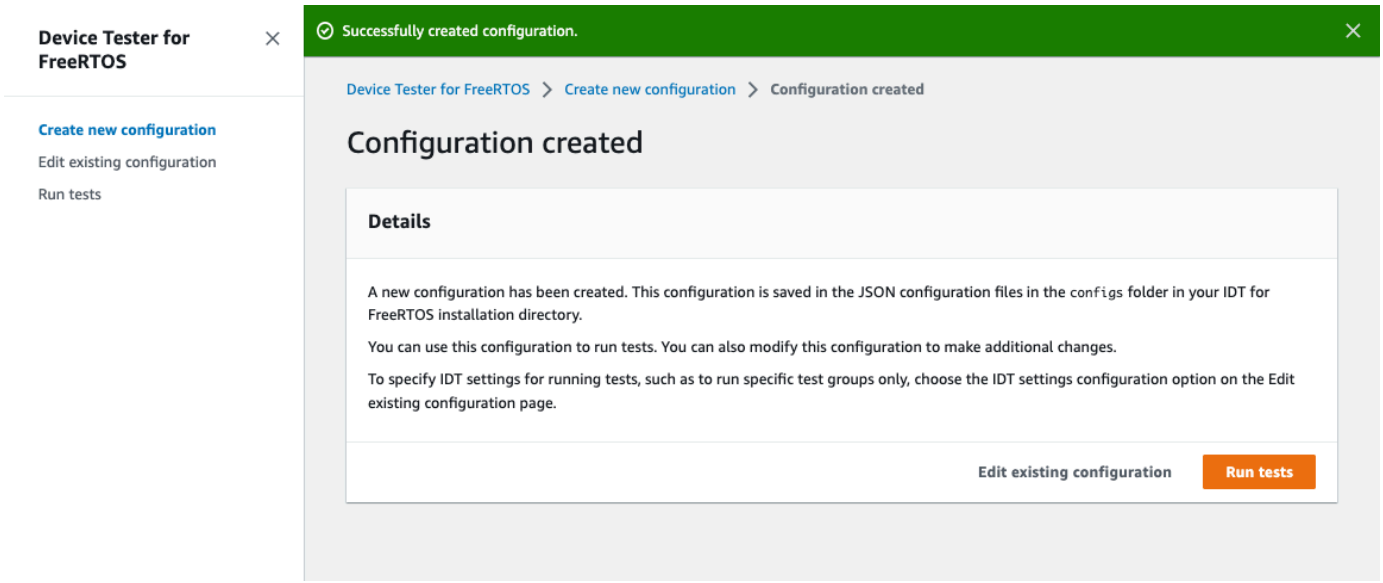
- PKCS #11 標籤和回應伺服器 — PKCS #11 標籤，對應於根據金鑰功能和金鑰佈建方法在硬體中佈建的金鑰。傳輸介面測試的回應伺服器組態設定。這些設定是在`userdata.json`和`device.json`檔案中設定的。



- Over-the-air （OTA）更新-控制 OTA 功能測試的設置。這些設定是在`device.json`和`userdata.json`檔案`features`區塊中進行設定。


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

3. 在「複查」頁面上，確認您的組態資訊。



完成檢閱組態之後，若要執行資格測試，請選擇 [執行測試]。

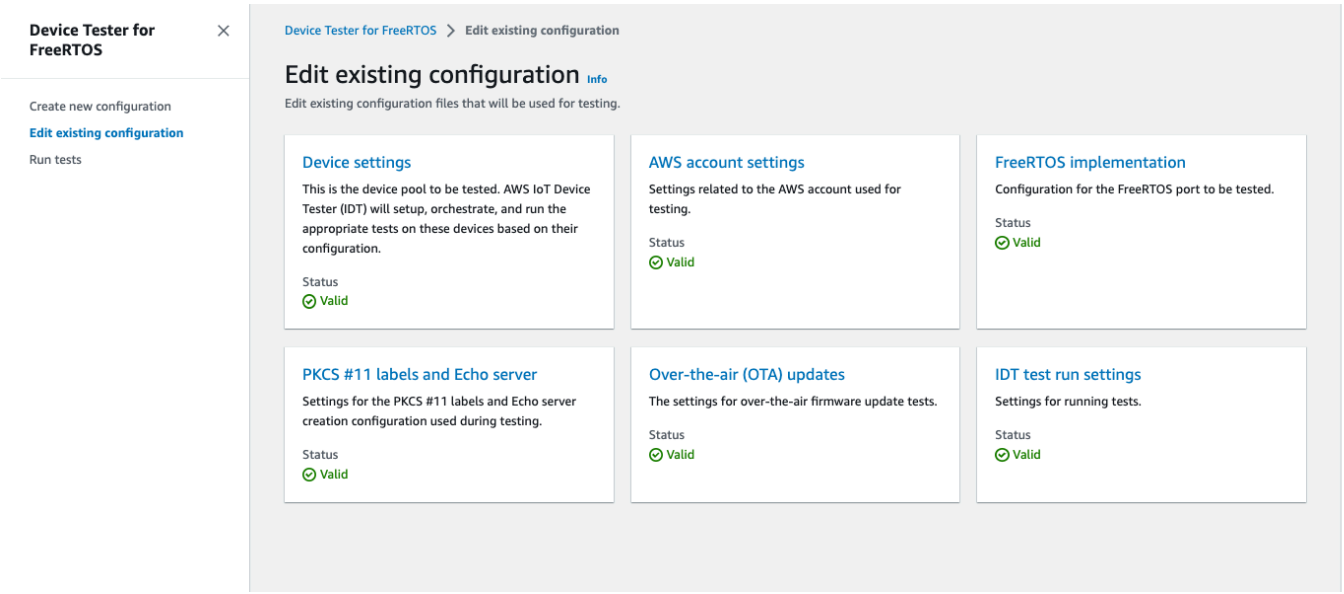
修改現有的模型組態

如果您已經為 FreeRTOS 的 IDT 設定檔，您可以使用 FreeRTOS 使用者介面的 IDT 來修改您現有的組態。現有的組態檔案必須位於目*devicetester-extract-location*/config錄中。

修改模型組態

1. 在 FreeRTOS 使用者介面的 IDT 中，開啟瀏覽功能表，然後選擇 [編輯現有的組態]。

   組態儀表板會顯示現有組態設定的相關資訊。如果組態不正確或無法使用，則該組態的狀態為`Error validating configuration`。

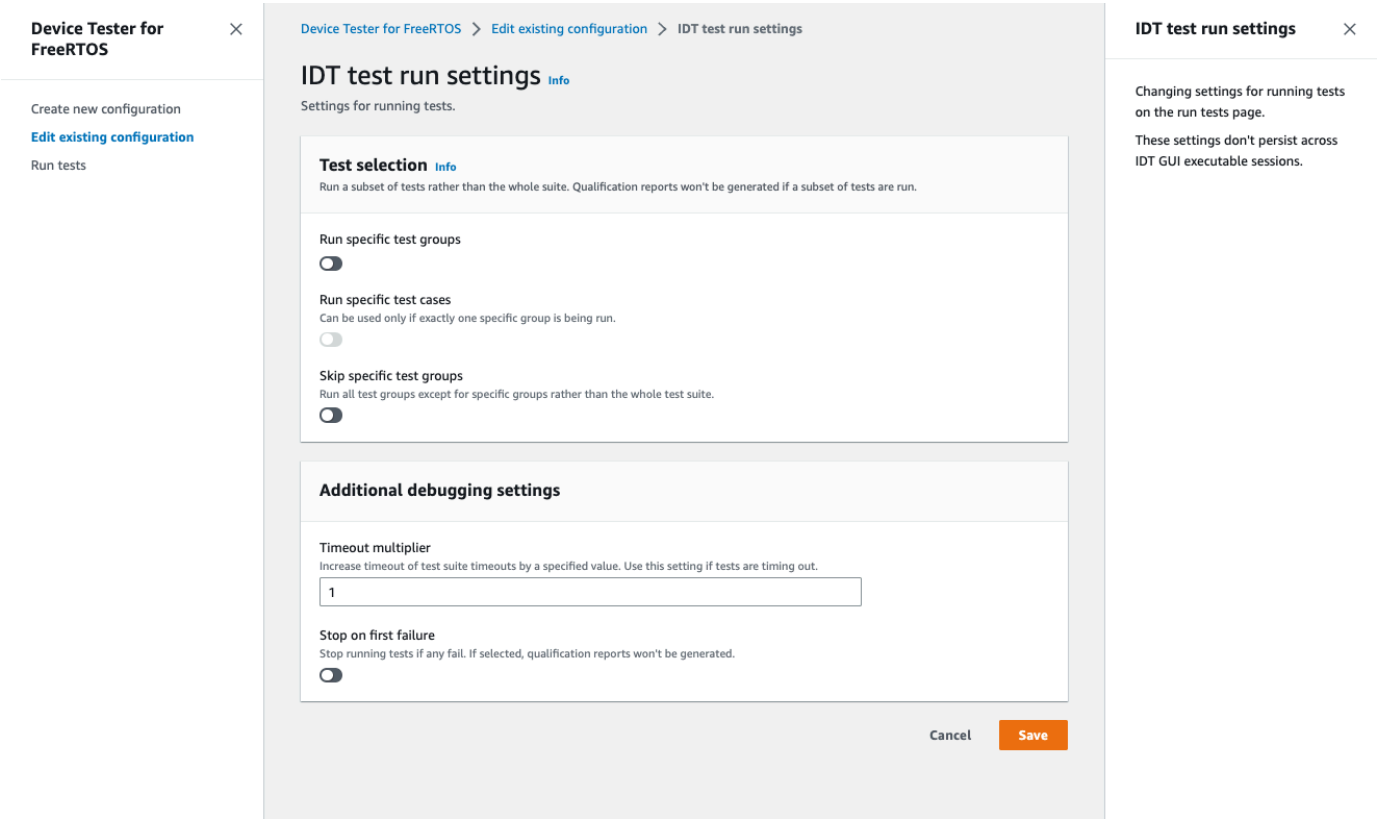


2. 若要修改現有的組態設定，請完成以下步驟：

   a. 選擇組態設定的名稱以開啟其設定頁面。

   b. 修改設定，然後選擇 [儲存] 以重新產生對應的規劃檔案。

3. 若要修改 FreeRTOS 測試回合設定的 IDT，請在編輯檢視中選擇 IDT 測試回合設定：



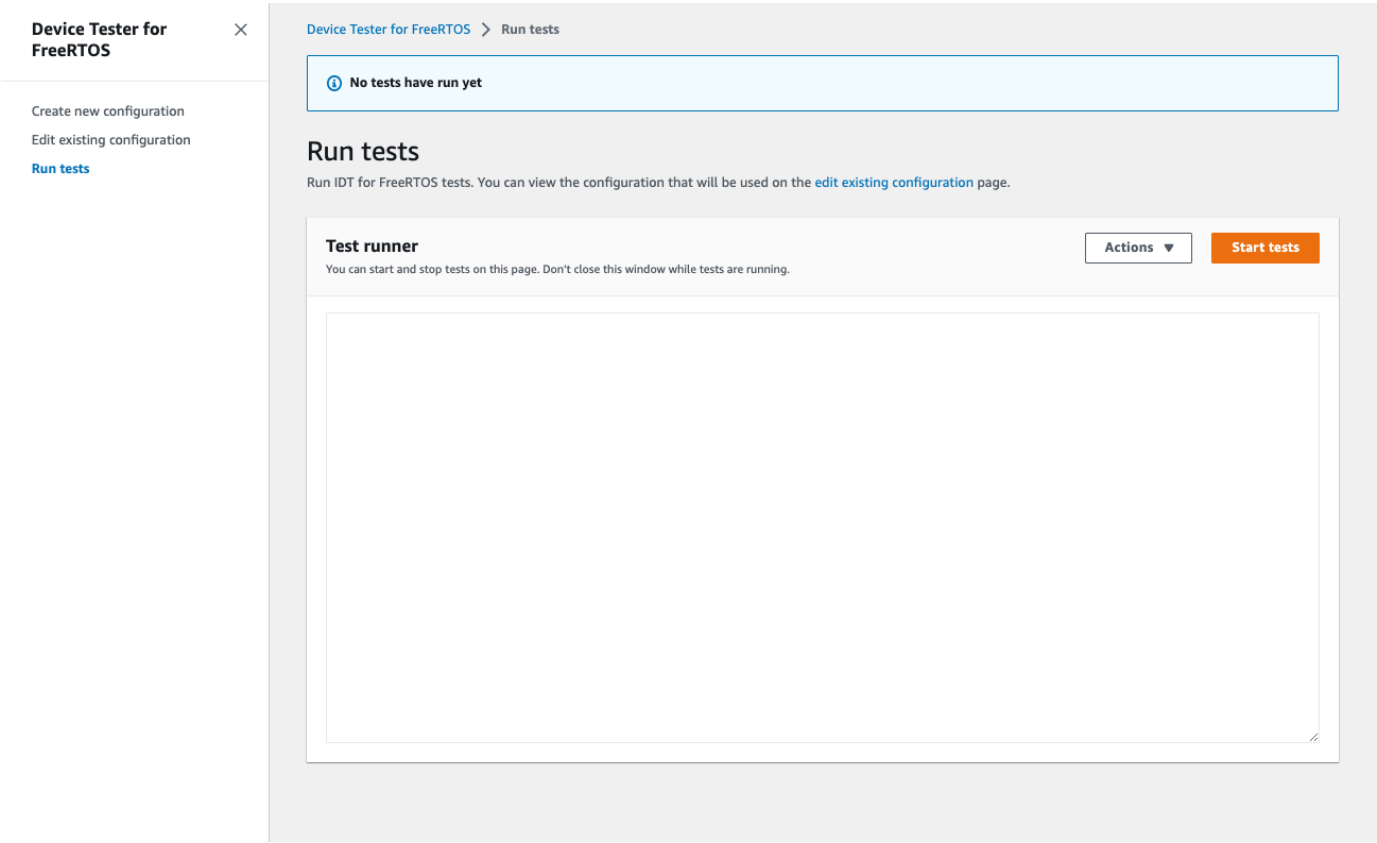
完成修改組態之後，請確認所有組態設定都通過驗證。如果每個組態設定的狀態為Valid，您可以使用此組態執行資格測試。

執行資格測試

建立 FreeRTOS 使用者介面的 IDT 組態之後，您就可以執行您的資格測試。

執行資格測試

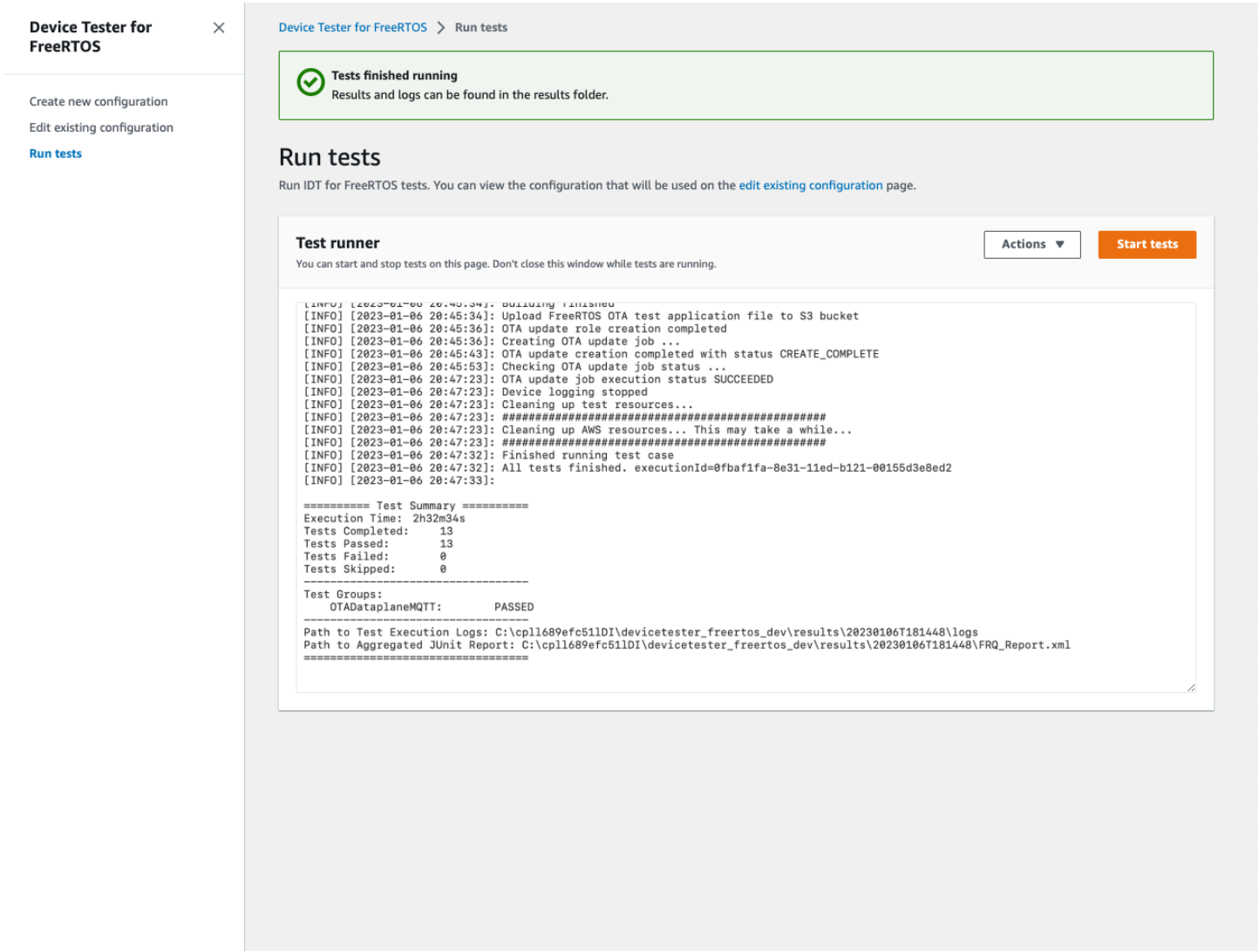
1. 在導覽功能表中，選擇 [執行測試]。
2. 選擇開始測試以開始測試運行。默認情況下，所有適用的測試都會針對您的設備配置運行。FreeRTOS 的 IDT 會在所有測試完成時產生資格報告。



免費伺服器的 IDT 會執行資格測試。然後，它會在 Test Runner 控制台中顯示測試運行摘要和任何錯誤。測試運行完成後，您可以從以下位置查看測試結果和日誌：

- 測試結果位於目*devicetester-extract-location*/results/*execution-id*錄中。
- 測試日誌位於目*devicetester-extract-location*/results/*execution-id*/logs錄中。

如需有關測試結果和記錄檔的詳細資訊，請參閱了解結果和日誌。



執行 FreeRTOS 資格 2.0 套件

使用 FreeRTOS 執行檔與免費伺服器的 IDT 互動。AWS IoT Device Tester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 v4.5.2 and later

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

在裝置集區上執行測試套件。userdata.json 檔案必須位於 *devicetester_extract_location*/devicetester_freertos_*[win|mac|linux]*/configs/ 目錄。

> **Note**
>
> 如果您在 Windows 上執行 FreeRTOS 的 IDT，請使用正斜線 (/) 來指定檔案的路徑。userdata.json

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_1.99.0  \
    --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。

FreeRTOS 命令列選項的 IDT

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。

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 FreeRTOSVersion --
test-id FreeRTOSVersion
```

h

使用說明選項以進一步了解 `run-suite` 選項。

Example

範例

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

FreeRTOS 命令的 IDT

FreeRTOS 命令的 IDT 可支援下列作業：

IDT v4.5.2 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 FreeRTOSVersion --
test-id FreeRTOSVersion
```

> **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。

了解結果和日誌

本節說明如何檢視和解譯 IDT 結果報告與日誌。

檢視結果

執行期間，IDT 會將錯誤寫入主控台、日誌檔和測試報告。IDT 完成資格測試套件後，即會將測試執行摘要寫入主控台，並產生兩份測試報告。您可以在 *devicetester-extract-location*/results/*execution-id*/ 中找到這些報告。這兩份報告都會從資格測試套件執行擷取結果。

這awsiotdevicetester_report.xml是您提交的資格測試報告，以AWS便在AWS合作夥伴裝置目錄中列出您的裝置。該報告包含下列元素：

- FreeRTOS 版本的 IDT。
- 已測試的 FreeRTOS 版本。

- 裝置根據通過的測試所支援的 FreeRTOS 功能。
- `device.json` 檔案中指定的 SKU 和裝置名稱。
- `device.json` 檔案中所指定裝置的功能。
- 測試案例結果的彙總摘要。
- 根據裝置功能進行測試的程式庫測試案例結果明細。

`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 以取得 FreeRTOS 搜尋結果

`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`

FreeRTOS 的 IDT 無法執行的測試案例數目。

disabled

此屬性未使用，可忽略。

如果沒有測試用例故障或錯誤，則您的設備符合運行 FreeRTOS 的技術要求，並且可以與服務互操作。AWS IoT如果您選擇在AWS合作夥伴裝置目錄中列出您的裝置，您可以使用此報告作為資格證據。

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="FreeRTOSVersion" package="" tests="1" failures="0"
time="2" 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="FRQ FreeRTOSVersion" name="FreeRTOSVersion"
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。

features

驗證的功能。標記為 required 的功能為提交主機板獲得資格時所需。以下程式碼片段會說明此項目在awsiotdevicetester_report.xml檔案中的顯示方法，

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

標記為 optional 的功能不需要進行資格測試。以下程式碼片段顯示選用功能。

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

如果所需功能沒有測試失敗或錯誤，則您的裝置符合執行 FreeRTOS 的技術需求，並且可以與服務互通。AWS IoT如果您想在AWS合作夥伴裝置目錄中列出您的裝置，可以使用此報告作為資格證據。

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">
```

格式與<testsuites>標籤類似，但具有未使用且可以忽略的skipped屬性。在每個 <testsuite> XML 標籤內，測試群組每個執行的測試都有 <testcase> 標籤。例如：

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

<testcase>標籤中使用的屬性

name

測試案例的名稱。

attempts

FreeRTOS 的 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="FRQ FreeRTOSVersion" name="FreeRTOSVersion">
    <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，請參閱疑難排解。

檢視 日誌

您可以在中找到 FreeRTOS 的 IDT 從測試執行產生的記錄檔。*devicetester-extract-location*/results/*execution-id*/logs該工具會產生兩組日誌：

- `test_manager.log`

  包含從 FreeRTOS 的 IDT 產生的記錄檔 (例如，與記錄相關的組態設定和報告產生)。

- *`test_group_id`*`/`*`test_case_id`*`/`*`test_case_id`*`.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。

將 IDT 與 FreeRTOS 資格套件 1.0 搭配使用 (FRQ 1.0)

> **Important**
>
> 自 2022 年 10 月起， AWS IoT Device Tester AWS IoT FreeRTOS 資格 (FRQ) 1.0 不會產生已簽署的資格報告。您無法透過使用 IDT FRQ 1.0 版本的裝置資格認證計劃，將新 AWS IoT FreeRTOS AWS 裝置列入AWS 合作夥伴裝置目錄的資格。雖然您無法使用 IDT FRQ 1.0 來限定 FreeRTOS 裝置的資格，但您可以繼續使用 FRQ 1.0 來測試您的 FreeRTOS 裝置。我們建議您使用 IDT FRQ 2.0 來限定和列出合作夥伴裝置目錄中的 FreeRTOS 裝置。AWS

您可以使用 IDT 取得 FreeRTOS 資格認證，驗證 FreeRTOS 作業系統可在您的裝置本機上運作，並且可以與之通訊。 AWS IoT具體而言，它會驗證 FreeRTOS 程式庫的移植層介面是否已正確實作。它還執行 end-to-end 測試與 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 針對 FreeRTOS 執行的測試是在 FreeRTOS 儲存庫中定義的。 GitHub

測試會以主機板內嵌應用程式的方式執行。應用程式二進位影像包括 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之上正常運作。

FreeRTOS 的 IDT 會產生測試報告，您可以提交這些報告，將硬體新增 AWS IoT 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。

FreeRTOS 的 IDT 會在連接至待測試主機板的主機電腦 (視窗、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，FreeRTOS 的 IDT 還會建立資源 (例如， AWS IoT 事物、FreeRTOS 群組、Lambda 函數等) 來促進資格認證程序。若要建立這些資源，FreeRTOS 的 IDT 會使用中設定的 AWS 認證代表您`config.json`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 FreeRTOS 的 IDT 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。

主題

- 必要條件
- 首次準備測試微型控制器主機板
- 使用免費伺服器的 IDT 使用者介面來執行 FreeRTOS 資格套件
- 執行低功耗藍牙測試
- 執行 FreeRTOS 資格套件
- 了解結果和日誌

必要條件

本節說明使 AWS IoT Device Tester用測試微控制器的先決條件。

下載 FreeRTOS

您可以GitHub使用下列命令從下列指令下載 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>是對應於中列出的 IDT 版本的 FreeRTOS 伺服器版本 (例如 202007.00) 的版本。支援的版本AWS IoT Device Tester對於 FreeRTOS這可確保您擁有完整的原始程式碼 (包括子模組)，並且針對 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 的路徑結構很深，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`而不是。`C:\Users\username\programs\projects\myproj\FreeRTOS\`

LTS 資格的注意事項（使用 LTS 程式庫的 FreeRTOS 資格）

- 若要在 AWS 合作夥伴裝置目錄中將您的微控制器指定為支援長期支援 (LTS) 的 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 Fre e RTOS 資格檢查清單。
- 若要驗證您的微控制器是否支援以 LTS 為基礎的 FreeRTOS 版本，並有資格提交至 AWS 合作夥伴裝置目錄，您必須使用 AWS IoT Device Tester (IDT) 搭配 FreeRTOS 認證 (FRQ) 測試套件 v1.4.x 版。
- 對於以 LTS 為基礎的免費伺服器版本的 Support 援僅限於 202012.xx 版的 FreeRTOS。

下載 FreeRTOS 的 IDT

每個版本的 FreeRTOS 都有相應的 IDT 版本，供 FreeRTOS 執行資格測試。請從以下位置下載適用於 FreeRTOS 的適用 IDT 版本。支援的版本AWS IoT Device Tester對於 FreeRTOS

將 FreeRTOS 的 IDT 擷取至檔案系統上您具有讀取和寫入權限的位置。由於 Microsoft 視窗有路徑長度的字元限制，因此請將 FreeRTOS 的 IDT 解壓縮至根目錄，例如或。`C:\` `D:\`

> **Note**
>
> 不建議多位使用者從 NFS 目錄或 Windows 網路共用資料夾等共用位置執行 IDT。這麼做可能會導致當機或資料損毀。建議您將 IDT 套件解壓縮至本機磁碟機。

建立和設定 AWS 帳號

註冊一個 AWS 帳戶

如果您沒有 AWS 帳戶，請完成以下步驟來建立一個。

若要註冊成為 AWS 帳戶

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

   部分註冊程序需接收來電，並在電話鍵盤輸入驗證碼。

當您註冊一個時 AWS 帳戶，將創建AWS 帳戶根使用者一個。根使用者有權存取該帳戶中的所有 AWS 服務 和資源。安全性最佳做法是將管理存取權指派給使用者，並僅使用 root 使用者來執行需要 root 使用者存取權的工作。

AWS 註冊過程完成後，會向您發送確認電子郵件。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。

建立具有管理權限的使用者

註冊後，請保護 AWS 帳戶 AWS 帳戶根使用者、啟用和建立系統管理使用者 AWS IAM Identity Center，這樣您就不會將 root 使用者用於日常工作。

保護您的 AWS 帳戶根使用者

1. 選擇 Root 使用者並輸入您的 AWS 帳戶 電子郵件地址，以帳戶擁有者身分登入。AWS Management Console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)。

   如需指示，請參閱《IAM 使用者指南》中的為 AWS 帳戶 根使用者啟用虛擬 MFA 裝置 (主控台)。

建立具有管理權限的使用者

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 身分中心中，將管理存取權授予使用者。

   若要取得有關使用 IAM Identity Center 目錄 做為身分識別來源的自學課程，請參閱《使用指南》IAM Identity Center 目錄中的「以預設值設定使用AWS IAM Identity Center 者存取」。

以具有管理權限的使用者身分登入

- 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 身分中心使用者登入的說明，請參閱使用AWS 登入 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 身分中心中，建立遵循套用最低權限許可的最佳做法的權限集。

   如需指示，請參閱《AWS IAM Identity Center 使用指南》中的「建立權限集」。
2. 將使用者指派給群組，然後將單一登入存取權指派給群組。

   如需指示，請參閱《AWS IAM Identity Center 使用指南》中的「新增群組」。

AWS IoT Device Tester 受管理政策

受AWSIoTDeviceTesterForFreeRTOSFullAccess管理的原則包含下列版本檢查、auto 更新功能及指標集合的 AWS IoT Device Tester 權限。

- `iot-device-tester:SupportedVersion`

  授 AWS IoT Device Tester 予獲取支持產品，測試套件和 IDT 版本列表的權限。
- `iot-device-tester:LatestIdt`

  授 AWS IoT Device Tester 予獲取可供下載的最新 IDT 版本的權限。
- `iot-device-tester:CheckVersion`

  授 AWS IoT Device Tester 予檢查 IDT、測試套件和產品版本相容性的權限。
- `iot-device-tester:DownloadTestSuite`

  授 AWS IoT Device Tester 予下載測試套件更新的權限。
- `iot-device-tester:SendMetrics`

  授 AWS 予收集有關 AWS IoT Device Tester 內部使用量度之指標的權限。

(選擇性) 安裝 AWS Command Line Interface

您可能更喜歡使用 AWS CLI 來執行某些操作。如果您沒有安 AWS CLI 裝，請按照安裝 AWS CLI.

透過aws configure從指令行執行， AWS CLI 為您要使用的 AWS 區域設定。如需支援 FreeRTOS 之 IDT 之 AWS 區域的相關資訊，請參閱AWS 區域和端點。若要取得有aws configure關的詳細資訊，aws configure請參閱快速組態

首次準備測試微型控制器主機板

您可以在移植 FreeRTOS 介面時，使用 IDT 進行測試。移植主機板裝置驅動程式的 FreeRTOS 介面之後，您可 AWS IoT Device Tester 以用來在微控制器主機板上執行認證測試。

新增程式庫移植層

若要為您的裝置移植 FreeRTOS，請依照 Fre e RTOS 移植指南中的指示操作。

設定您的 AWS 認證

您必須設定 AWS 認證， AWS IoT Device Tester 才能與 AWS 雲端通訊。如需詳細資訊，請參閱設定 AWS 認證和開發區域。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 認證。

在 IDT 中建立 FreeRTOS 的裝置集區

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。您可以將 FreeRTOS 的 IDT 設定為測試集區中的單一裝置或集區中的多個裝置。為了加速認證流程，FreeRTOS 的 IDT 可以同時測試具有相同規格的設備。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原始程式碼的 parallel 建置，FreeRTOS 的 IDT 會將原始程式碼複製到結果資料夾中 IDT (FreeRTOS) 解壓縮資料夾中的結果資料夾。必須使用或變量引用構建或 Flash 命令中的`testdata.sourcePath`源代碼路`sdkPath`徑。FreeRTOS 的 IDT 會以複製原始程式碼的暫存路徑取代此變數。若要取得更多資訊，請參閱FreeRTOS 變數的 IDT。

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"
                    }
                ]
            },
            {
                "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**
>
> 如果您要在 AWS 合作夥伴裝置目錄中刊登主機板，您在此指定的 SKU 必須符合您在刊登程序中使用的 SKU。

`features`

包含裝置支援功能的陣列。 AWS IoT Device Tester 使用此資訊來選取要執行的資格測試。

支援的值如下：

`TCP/IP`

指出您的面板是否支援 TCP/IP 堆疊，以及支援晶載 (MCU) 或轉移到另一個模組。你需要 TCP/IP 才能符合資格。

`WIFI`

指出您的面板是否具有 Wi-Fi 功能。如果設定為，`NoCellular`則必須設定為`Yes`。

`Cellular`

指出您的主機板是否有行動網路功能。如果設定為，`NoWIFI`則必須設定為`Yes`。將此功能設為時`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 將創建一個私鑰並將其構建到 FreeRTOS 源代碼中。
- 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

指出您的主機板是否支援 over-the-air (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。

> **Note**
>
> 從 IDT v4.1.0 開始，僅`OtaDataPlaneProtocol`接受`HTTP`並`MQTT`作為支援的值。

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
```

如果您的公鑰是 .der 格式，則可以直接對公鑰進行十六進制編碼以生成十六進制文件。

用於生成十六進制文件的 .der 公鑰的示例命令：

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

如果您的公鑰是 .pem 格式，則可以提取 base64 編碼的部分，將其解碼為二進制格式，然後對其進行十六進制編碼以生成十六進制文件。

例如，使用下列命令為 .pem 公開金鑰產生十六進位檔案：

1. 取出密鑰的 base64 編碼部分（去除頁眉和頁腳）並將其存儲在文件中，例如命名它base64key，運行此命令將其轉換為 .der 格式：

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

2. 執行命xxd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，請在列印序號和裝置公開金鑰時提供此欄位。

devices.secureElementConfig.preProvisioned

(選擇性) 如果裝置具有預先佈建的安全元素，且具有鎖定認證，無法匯入、建立或銷毀物件，則設定為「是」。只有當KeyProvisioning設定為「內建」features 且設定為「ECC」時，PKCS11此組態才會生效。

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 FreeRTOS 的 IDT 自動在主機板上建置並進行快閃測試，您必須設定 IDT 來執行硬體的建置和 Flash 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。我們通過在中加載客戶端和服務器證書以及密鑰來支持 Echo 服務器配置customPath。如需詳細資訊，請參閱《FreeRTOS 移植指南》中的〈設定回應伺服器〉。下列 JSON 範例顯示如何為 FreeRTOS 設定 IDT 以測試多個裝置：

```
{
    "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": [
            "{{config.idtRootPath}}/relative-path-to/build-parallel.sh
 {{testData.sourcePath}} {{enableTests}}"
        ]
    },
    "flashTool": {
        "name": "your-flash-tool-name",
        "version": "your-flash-tool-version",
        "command": [
            "/{{config.idtRootPath}}/relative-path-to/flash-parallel.sh
 {{testData.sourcePath}} {{device.connectivity.serialPort}} {{buildImageName}}"
        ],
        "buildImageInfo" : {
            "testsImageName": "tests-image-name",
            "demosImageName": "demos-image-name"
        }
    },
    "testStartDelayms": 0,
    "clientWifiConfig": {
        "wifiSSID": "ssid",
        "wifiPassword": "password",
```

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

```
        "codeSigningConfiguration": {
            "signingMethod": "AWS | Custom",
            "signerHashingAlgorithm": "SHA1 | SHA256",
            "signerSigningAlgorithm": "RSA | ECDSA",
            "signerCertificate": "arn:partition:service:region:account-
id:resource:qualifier | /absolute-path-to/signer-certificate-file",
            "signerCertificateFileName": "signerCertificate-file-name",
            "compileSignerCertificate": boolean,
            // ***********Use signerPlatform if you choose aws for
 signingMethod***************
            "signerPlatform": "AmazonFreeRTOS-Default | AmazonFreeRTOS-TI-CC3220SF",
            "untrustedSignerCertificate": "arn:partition:service:region:account-
id:resourcetype:resource:qualifier",
            // ***********Use signCommand if you choose custom for
 signingMethod***************
            "signCommand": [
                "/absolute-path-to/sign.sh {{inputImageFilePath}}
 {{outputSignatureFilePath}}"
            ]
        }
    },
    // ***********Remove the section below if you're not configuring
 CMake***************
    "cmakeConfiguration": {
        "boardName": "board-name",
        "vendorName": "vendor-name",
        "compilerName": "compiler-name",
        "frToolchainPath": "/path/to/freertos/toolchain",
        "cmakeToolchainPath": "/path/to/cmake/toolchain"
    },
    "freertosFileConfiguration": {
        "required": [
            {
                "configName": "pkcs11Config",
                "filePath": "{{testData.sourcePath}}/vendors/vendor-name/boards/board-
name/aws_tests/config_files/core_pkcs11_config.h"
            },
            {
                "configName": "pkcs11TestConfig",
                "filePath": "{{testData.sourcePath}}/vendors/vendor-name/boards/board-
name/aws_tests/config_files/iot_test_pkcs11_config.h"
            }
        ],
        "optional": [
```

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

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。對於使用 SDK 進 parallel 測試，sourcePath可以使用預留{{userData.sdkConfiguration.path}}位置進行設置。例如：

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

vendorPath

供應商特定 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" }
```

只有在沒有 SDK 的情況下運行時才需要該vendorPath變量，否則可以將其刪除。

> **Note**
>
> 在沒有 SDK 的情況下 parallel 執行測試時，必須在vendorPath、buildToolflashTool欄{{testData.sourcePath}}位中使用預留位置。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，、和flashTool指令中必須使用{{sdkPath}}預留位置。sourcePath buildTool

sdkConfiguration

如果您對 FreeRTOS 進行任何修改的檔案和資料夾結構超出移植所需要的限制，那麼您將需要在此區塊中設定 SDK 資訊。如果您沒有在 SDK 中使用移植的 FreeRTOS 的資格，那麼您應該完全省略此塊。

sdkConfiguration.name

您與 FreeRTOS 搭配使用的開發套件名稱。如果您不使用 SDK，則應省略整個sdkConfiguration塊。

sdkConfiguration.version

您與 FreeRTOS 搭配使用的開發套件版本。如果您不使用 SDK，則應省略整個sdkConfiguration塊。

sdkConfiguration.path

包含 FreeRTOS 程式碼的 SDK 目錄的絕對路徑。如果您不使用 SDK，則應省略整個sdkConfiguration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。build 命令中對源代碼路徑的所有引用都必須由 AWS IoT Device Tester 變量替換，{{testdata.sourcePath}}並且對 SDK 路徑的引用應替換為{{sdkPath}}。使用{{config.idtRootPath}}預留位置來參考絕對或相對 IDT 路徑。

testStartDelayms

指定 FreeRTOS 測試執行程式在開始執行測試之前等待的毫秒數。如果被測設備在 IDT 因網絡或其他延遲而有機會連接並開始日誌記錄之前開始輸出重要的測試信息，這很有用。允許的最大值為 3 萬毫秒（30 秒）。此值僅適用於 FreeRTOS 測試群組，不適用於不使用 FreeRTOS 測試執行程式的其他測試群組，例如 OTA 測試。

flashTool

包含用於裝置的刷入命令之刷入指令碼 (.sh 或 .bat) 的完整路徑。ash 命令中對原始程式碼路徑的所有參照都必須由 FreeRTOS 變數的 IDT 取代，而{{testdata.sourcePath}}且 SDK 路徑的所有參照都必須由 FreeRTOS 變數的 IDT 取代{{sdkPath}}。使用{{config.idtRootPath}}預留位置來參照絕對或相對 IDT 路徑。

buildImageInfo

testsImageName

從文件*freertos-source*/tests夾構建測試時，由 build 命令生成的文件的名稱。

demosImageName

從文件*freertos-source*/demos夾構建測試時，由 build 命令生成的文件的名稱。

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

用於安全套接字測試的可配置 echo 服務器證書生成佔位符。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「自定義」，certificatePath並且privateKeyPath是必需的。

certificatePath

指定伺服器憑證的檔案路徑。

privateKeyPath

指定私密金鑰的檔案路徑。

eccCurveFormat

指定電路板支援的曲線格式。在device.json中設定PKCS11為「ecc」時需要。有效值為「P224」、「256」、「384」或「P520」。

echoServerConfiguration

可配置的 echo 服務器端口，用於 WiFi 安全套接字測試。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

用於在沒有 TLS 的情況下設置 echo 服務器進行 WiFi 測試的端口。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 固件所在 MCU 設備上的完整文件路徑。有些裝置不使用此欄位，但您仍必須提供值。

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**
>
> 對於北京和寧夏地區，請使用Custom。 AWS這些區域不支援程式碼簽章。

signerHashingAlgorithm

裝置上支援的雜湊演算法。可能的值為 SHA1 或 SHA256。

signerSigningAlgorithm

裝置上支援的簽署演算法。可能的值為 RSA 或 ECDSA。

signerCertificate

用於 OTA 的信任憑證。

對於 AWS 程式碼簽章方法，請針對上傳到 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

裝置上程式碼簽章憑證的檔案名稱。此值必須與您在執行aws acm import-certificate指令時提供的檔案名稱相符。

如需詳細資訊，請參閱 建立程式碼簽署憑證。

compileSignerCertificate

true如果未佈建或刷新程式碼簽署者簽名驗證憑證，則設定為，因此必須將其編譯到專案中。AWS IoT Device Tester 獲取受信任的證書並將其編譯成。aws_codesigner_certifiate.h

untrustedSignerCertificate

在某些 OTA 測試中用作不受信任證書的第二個證書的 ARN 或文件路徑。如需有關建立憑證的詳細資訊，請參閱建立程式碼簽署憑證。

signerPlatform

AWS 代碼簽名者在創建 OTA 更新作業時使用的簽名和哈希算法。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。

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 檔案的組態。

required

本節指定您已移動其設定檔的必要測試，例如 PKCS11、TLS 等。

configName

正在設定的測試名稱。

filePath

*freertos*存儲庫中配置文件的絕對路徑。使用{{testData.sourcePath}}變數來定義路徑。

optional

本節指定了您已移動其配置文件的可選測試，例如 OTA WiFi，等等。

configName

正在設定的測試名稱。

filePath

*freertos*存儲庫中配置文件的絕對路徑。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。

FreeRTOS 變數的 IDT

建立程式碼和快閃記憶體裝置的指令可能需要連線或裝置的其他相關資訊，才能成功執行。 AWS IoT Device Tester 允許您在 Flash 中引用設備信息並使用構建命令JsonPath。通過使用簡單的 JsonPath 表達式，您可以獲取device.json文件中指定的所需信息。

路徑變數

FreeRTOS 的 IDT 定義了下列可用於命令列和組態檔案的路徑變數：

{{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}}

擴展為在構建和 Flash 命令中使用userData.sdkConfiguration.path時的值。

{{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 代碼簽名者的 PEM 文件。

{{config.idtRootPath}}

展開到 AWS IoT Device Tester 根路徑。當建置和 Flash 命令使用時，此變數會取代 IDT 的絕對路徑。

使用免費伺服器的 IDT 使用者介面來執行 FreeRTOS 資格套件

自 IDT v4.3.0 開始， AWS IoT Device Tester 針對 FreeRTOS (IDT-免費使用者)，包含網頁式使用者介面，可讓您與 IDT 命令列執行檔和相關組態檔進行互動。您可以使用 IDT-FreeRTOS 使用者介面來建立新的組態以執行 IDT 測試，或修改現有的組態。您還可以使用 UI 調用 IDT 可執行文件並運行測試。

IDT 免費使用者介面提供下列功能：

- 簡化 IDT 免費伺服器測試的設定檔案的設定。
- 簡化使用 IDT FreeRTOS 執行資格測試的程序。

如需使用指令列執行限定測試的相關資訊，請參閱首次準備測試微型控制器主機板。

本節說明使用 IDT-FreeRTOS 使用者介面的先決條件，並說明如何開始在 UI 中執行資格測試。

主題

- 必要條件
- 開始使用 IDT 免費使用者介面

必要條件

本節說明使 AWS IoT Device Tester用測試微控制器的先決條件。

主題

- 使用支援的網頁瀏覽器
- 下載 FreeRTOS
- 下載 FreeRTOS 的 IDT
- 建立和設定 AWS 帳號

- AWS IoT Device Tester 受管理政策

使用支援的網頁瀏覽器

IDT 免費使用者介面支援下列網頁瀏覽器。

瀏覽器	版本
Google Chrome	最新的三個主要版本
Mozilla Firefox	最新的三個主要版本
Microsoft Edge	最新的三個主要版本
適用於 macOS 的 Apple Safari	最新的三個主要版本

我們建議您使用谷歌瀏覽器或火狐瀏覽器為了更好的體驗。

> **Note**
>
> ID 免費使用者介面不支援 Microsoft IE 瀏覽器。

下載 FreeRTOS

您可以GitHub使用下列命令從下列指令下載 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>是對應於中列出的 IDT 版本的 FreeRTOS 伺服器版本 (例如 202007.00) 的版本。支援的版本AWS IoT Device Tester對於 FreeRTOS這可確保您擁有完整的原始程式碼 (包括子模組)，並且針對 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 的路徑結構有許多層級深，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`而不是。`C:\Users\username\programs\projects\myproj\FreeRTOS\`

LTS 資格的注意事項（使用 LTS 程式庫的 FreeRTOS 資格）

- 若要將您的微控制器指定為支援 AWS 合作夥伴裝置目錄中 FreeRTOS 的長期支援 (LTS) 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 Fre e RTOS 資格檢查清單。
- 若要驗證您的微控制器是否支援以 LTS 為基礎的 FreeRTOS 版本，並有資格提交至 AWS 合作夥伴裝置目錄，您必須使用 AWS IoT Device Tester (IDT) 搭配 FreeRTOS 認證 (FRQ) 測試套件 v1.4.x 版。
- 對於以 LTS 為基礎的免費伺服器版本的 Support 援僅限於 202012.xx 版本的 FreeRTOS。

下載 FreeRTOS 的 IDT

每個版本的 FreeRTOS 都有一個對應的 IDT 版本，適用於執行資格測試。請從以下位置下載適用於 FreeRTOS 的適用 IDT 版本。支援的版本AWS IoT Device Tester對於 FreeRTOS

將 FreeRTOS 的 IDT 擷取至檔案系統上您具有讀取和寫入權限的位置。由於 Microsoft 視窗有路徑長度的字元限制，因此請將 FreeRTOS 的 IDT 解壓縮至根目錄，例如或。`C:\` `D:\`

> **Note**
>
> 建議您將 IDT 套件解壓縮至本機磁碟機。允許多個使用者從共用位置 (例如 NFS 目錄或 Windows 網路共用資料夾) 執行 IDT，可能會導致系統沒有回應或資料損毀。

建立和設定 AWS 帳號

註冊一個 AWS 帳戶

如果您沒有 AWS 帳戶，請完成以下步驟來建立一個。

若要註冊成為 AWS 帳戶

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

   部分註冊程序需接收來電，並在電話鍵盤輸入驗證碼。

   當您註冊一個時 AWS 帳戶，將創建AWS 帳戶根使用者一個。根使用者有權存取該帳戶中的所有 AWS 服務 和資源。安全性最佳做法是將管理存取權指派給使用者，並僅使用 root 使用者來執行需要 root 使用者存取權的工作。

AWS 註冊過程完成後，會向您發送確認電子郵件。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。

建立具有管理權限的使用者

註冊後，請保護 AWS 帳戶 AWS 帳戶根使用者、啟用和建立系統管理使用者 AWS IAM Identity Center，這樣您就不會將 root 使用者用於日常工作。

保護您的 AWS 帳戶根使用者

1. 選擇 Root 使用者並輸入您的 AWS 帳戶 電子郵件地址，以帳戶擁有者身分登入。AWS Management Console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)。

   如需指示，請參閱《IAM 使用者指南》中的為 AWS 帳戶 根使用者啟用虛擬 MFA 裝置 (主控台)。

建立具有管理權限的使用者

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 身分中心中，將管理存取權授予使用者。

   若要取得有關使用 IAM Identity Center 目錄 做為身分識別來源的自學課程，請參閱《使用指南》IAM Identity Center 目錄中的「以預設值設定使用AWS IAM Identity Center 者存取」。

以具有管理權限的使用者身分登入

- 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 身分中心使用者登入的說明，請參閱使用AWS 登入 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 身分中心中，建立遵循套用最低權限許可的最佳做法的權限集。

如需指示，請參閱《AWS IAM Identity Center 使用指南》中的「建立權限集」。

2. 將使用者指派給群組，然後將單一登入存取權指派給群組。

   如需指示，請參閱《AWS IAM Identity Center 使用指南》中的「新增群組」。

AWS IoT Device Tester 受管理政策

若要讓裝置測試人員能夠執行並收集指標，受AWSIoTDeviceTesterForFreeRTOSFullAccess管理的策略包含下列權限：

- `iot-device-tester:SupportedVersion`

  授予取得 IDT 支援的 FreeRTOS 版本清單和測試套件版本清單的權限，以便從. AWS CLI
- `iot-device-tester:LatestIdt`

  授予取得可供下載的最新 AWS IoT Device Tester 版本的權限。
- `iot-device-tester:CheckVersion`

  授予檢查產品、測試套件和 AWS IoT Device Tester 版本之組合是否相容的許可。
- `iot-device-tester:DownloadTestSuite`

  授予下載測試套 AWS IoT Device Tester 件的權限。
- `iot-device-tester:SendMetrics`

  授與發佈 AWS IoT Device Tester 使用量度資料的權限。

開始使用 IDT 免費使用者介面

本節說明如何使用 IDT-FreeRTOS 使用者介面來建立或修改您的組態，然後說明如何執行測試。

主題

- 設定 AWS 認證
- 開啟 IDT 免費使用者介面
- 建立新的模型組態
- 修改現有的模型組態
- 執行資格測試

設定 AWS 認證

您必須為在中建立的 AWS 使用者設定認證建立和設定 AWS 帳號。您可以使用下列兩種方式的其中之一指定登入資料：

- 在認證檔案中
- 作為環境變量

使用 AWS 認證檔設定認證

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。

憑證檔案的位置會根據您使用的作業系統而有所不同：

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

以下列格式將您的 AWS 認證新增至`credentials`檔案：

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

> **Note**
>
> 如果您不使用設定`default` AWS 檔，請務必在 IDT-FreeRTOS 使用者介面中指定設定檔名稱。如需有關設定檔的詳細資訊，請參閱具名設定檔

使用環境變數設定 AWS 認證

環境變數是由作業系統維護且由系統命令使用的變數。如果您關閉 SSH 工作階段，則不會儲存它們。IDT-免費使用者介面會使用`AWS_ACCESS_KEY_ID`和`AWS_SECRET_ACCESS_KEY`環境變數來儲存您的認證。 AWS

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>
```

開啟 IDT 免費使用者介面

若要開啟 ID 免費使用者介面

1. 下載支援的 IDT-FreeRTOS 版本，並將下載的封存解壓縮至檔案系統上您具有讀取和寫入權限的位置。
2. 執行下列命令以導覽至 IDT-FreeRTOS 裝目錄：

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

3. 執行下列命令以開啟 IDT 免費使用者介面：

   Linux

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

   Windows

   ```
   ./devicetestergui_win_x64-64
   ```

   macOS

   ```
   ./devicetestergui_mac_x86-64
   ```

   > **Note**
   >
   > 在 Mac 上，若要允許您的系統執行 UI，請前往 「系統偏好設定」->「安全性與隱私權」。當您執行測試時，您可能需要再執行三次。

   IDT 免費使用者介面會在您的預設瀏覽器中開啟。如需有關支援瀏覽器的資訊，請參閱使用支援的網頁瀏覽器。

建立新的模型組態

如果您是初次使用的使用者，則必須建立新的組態來設定 IDT-FreeRTOS 執行測試所需的 JSON 組態檔。然後，您可以運行測試或修改已創建的配置。

如需config.json、device.json和userdata.json檔案的範例，請參閱首次準備測試微型控制器主機板。如需僅用於執行藍牙低功耗 (BLE) 測試的resource.json檔案範例，請參閱執行低功耗藍牙測試。

建立新的模型組態

1. 在 IDT-FreeRTOS 使用者介面中，開啟瀏覽功能表，然後選擇 [建立新的組態]。

   > **Important**
   >
   > 在開啟 UI 之前，您必須先設定 AWS 認證。如果您尚未設定認證，請關閉 IDT-FreeRTOS 使用者介面瀏覽器視窗，遵循中設定 AWS 認證的步驟，然後重新開啟 IDT-FreeRTOS 使用者介面。

2. 遵循組態精靈，輸入用於執行資格測試的 IDT 組態設定。精靈會在目錄中的 JSON 組態檔中設定下列設定。*devicetester-extract-location*/config

   - AWS 設定 — IDT-FreeRTOS 在測試執行期間用來建立 AWS 資源的資 AWS 帳戶 訊。這些設定是在config.json檔案中設定的。
   - FreeRTOS 儲存庫 — FreeRTOS 儲存庫和移植的程式碼的絕對路徑，以及您要執行的資格類型。這些設定是在userdata.json檔案中設定的。

     您必須先為您的裝置移植 FreeRTOS，才能執行資格測試。如需詳細資訊，請參閱 FreeRTOS 移植指南
   - 構建和閃存-硬件的構建和閃存命令，允許 IDT 自動構建和閃存測試到您的主板。這些設定是在userdata.json檔案中設定的。
   - 裝置 — 要測試之裝置的裝置集區設定。這些設定會在id和sku欄位中設定，以及device.json檔案中裝置集區的devices區塊。
   - 網路 — 測試裝置網路通訊支援的設定。這些設定是在檔案區features塊以及device.json檔案中的clientWifiConfig和testWifiConfig區塊中進行userdata.json設定。
   - Echo 伺服器 — 用於安全通訊端測試的回應伺服器組態設定 這些設定是在userdata.json檔案中設定的。

如需有關 echo 伺服器組態的詳細資訊，請參閱https://docs.aws.amazon.com/freertos/latest/portingguide/afr-echo-server.html。

- CMake — (可選) 運行 CMake 構建功能測試的設置。僅當您使用 CMake 作為構建系統時，才需要此配置。這些設定是在userdata.json檔案中設定的。
- BLE — 執行藍牙低功耗功能測試的設定。這些設定會在檔案features區塊和device.json檔案中進行resource.json設定。
- OTA-運行 OTA 功能測試的設置。這些設定會在檔案features區塊和device.json檔案中進行userdata.json設定。

3. 在「複查」頁面上，確認您的組態資訊。

完成檢閱組態之後，若要執行資格測試，請選擇 [執行測試]。

修改現有的模型組態

如果您已經設定 IDT 的組態檔，則可以使用 IDT-FreeRTOS 使用者介面來修改您現有的組態。請確定您現有的組態檔案位於目*devicetester-extract-location*/config錄中。

修改新的模型組態

1. 在 IDT-FreeRTOS 使用者介面中，開啟瀏覽功能表，然後選擇 [編輯現有的組態]。

   組態儀表板會顯示現有組態設定的相關資訊。如果組態不正確或無法使用，則該組態的狀態為Error validating configuration。
2. 若要修改現有的組態設定，請完成以下步驟：

   a. 選擇組態設定的名稱以開啟其設定頁面。
   b. 修改設定，然後選擇 [儲存] 以重新產生對應的規劃檔案。

完成修改組態之後，請確認所有組態設定都通過驗證。如果每個組態設定的狀態為Valid，您可以使用此組態執行資格測試。

執行資格測試

建立 IDT FreeRTOS 的組態之後，您就可以執行您的資格測試。

執行資格測試

1. 驗證您的組態。

2. 在導覽功能表中，選擇 [執行測試]。
3. 要開始測試運行，請選擇開始測試。

IDT-FreeRTOS 運行資格測試，並在測試運行器控制台中顯示測試運行摘要和任何錯誤。測試運行完成後，您可以從以下位置查看測試結果和日誌：

- 測試結果位於目*devicetester-extract-location*/results/*execution-id*錄中。
- 測試日誌位於目*devicetester-extract-location*/results/*execution-id*/logs錄中。

如需有關測試結果和記錄檔的詳細資訊，請參閱了解結果和日誌。

執行低功耗藍牙測試

本節說明如何使 AWS IoT Device Tester 用 FreeRTOS 來設定和執行藍牙測試。核心資格不需要進行藍牙測試。如果您不想在 FreeRTOS 藍牙支援下測試您的裝置，您可以略過此設定，請務必將 device.json 中的 BLE 功能設定為。No

必要條件

- 請遵循中的說明進行首次準備測試微型控制器主機板
- 一個樹莓派 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

要測試被測設備（DUT）的 BLE 功能，您必須具有樹莓派型號 4B 或 3B +。

設定您的 Raspberry Pi 以執行 BLE 測試

1. 下載其中一個包含執行測試所需軟體的自訂 Yocto 映像檔。
    - 樹莓派 4B 的圖像
    - 圖片為樹莓派 3B +

> **Note**
>
> Yocto 映像只能用於與 FreeRTOS 進行測試，而不能用 AWS IoT Device Tester 於任何其他目的。

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經有 SSK key pair，最佳做法是建立新的，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. 要將 SSH 密鑰添加到樹莓派上，以便 AWS IoT Device Tester FreeRTOS 可以登錄到設備，請使用主機計算機上的`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 裝置測試儀的目錄中，建立名為的檔案resource.json。

ii. 將有關樹莓派的以下信息添加到文件中，*rasp-pi-ip-address*替換為樹莓派的 IP 地址。

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

iii. 如果您沒有選擇使用 SSH 的公開金鑰驗證，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

您可以使用免 AWS IoT Device Tester 費伺服器執行檔來與 IDT 的 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 FreeRTOS 的 IDT，請使用正斜線 (/) 來指定檔案的路徑。userdata.json

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。

FreeRTOS 命令列選項的 IDT

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
```

更新 IDT

(選擇性) 如果未設定此參數，且有較新的 IDT 版本可供使用，系統將提示您更新 IDT。如果將此參數設定為Y，IDT 會在偵測到有更新的版本可用時停止測試執行。如果將此參數設定為N，IDT 將繼續執行測試。

update-managed-policy

(選擇性) 如果未使用此參數，且 IDT 偵測到您的受管理原則不存在 up-to-date，系統會提示您更新受管理的原則。如果將此參數設定為Y，IDT 會在偵測到您的受管政策不 up-to-date是時停止測試執行。如果將此參數設定為N，IDT 將繼續執行測試。

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

範例

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

範例

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

IDT v1.7.0 and earlier

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

userdata.json 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 FreeRTOS 的 IDT，請使用正斜線 (/) 來指定檔案的路徑。userdata.json

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。

FreeRTOS 命令列選項的 IDT

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。

FreeRTOS 命令的 IDT

FreeRTOS 命令的 IDT 支援下列作業：

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

隨著 FreeRTOS 認證測試的新版 IDT 發行，或者當您更新主機板特定的套件或裝置驅動程式時，您可以使用 FreeRTOS 的 IDT 來測試您的微控制器主機板。對於後續的資格，請確定您擁有 FreeRTOS 和最新版本的 FreeRTOS 和 IDT，然後再次執行資格測試。

了解結果和日誌

本節說明如何檢視和解譯 IDT 結果報告與日誌。

檢視結果

執行期間，IDT 會將錯誤寫入主控台、日誌檔和測試報告。IDT 完成資格測試套件後，即會將測試執行摘要寫入主控台，並產生兩份測試報告。您可以在 *devicetester-extract-location*/results/*execution-id*/ 中找到這些報告。這兩份報告都會從資格測試套件執行擷取結果。

這`awsiotdevicetester_report.xml`是您提交的資格測試報告，以 AWS 便在 AWS 合作夥伴裝置目錄中列出您的裝置。該報告包含下列元素：

- FreeRTOS 版本的 IDT。
- 已測試的 FreeRTOS 版本。
- 裝置根據通過的測試所支援的 FreeRTOS 功能。
- `device.json` 檔案中指定的 SKU 和裝置名稱。
- `device.json` 檔案中所指定裝置的功能。
- 測試案例結果的彙總摘要。
- 根據裝置功能 (例如 FullmQtt 等) 測試的程式庫來劃分測試用例結果。 FullWiFi
- 這個 FreeRTOS 的資格是否適用於使用 LTS 程式庫的 202012.00 版本。

`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 以取得 FreeRTOS 搜尋結果

`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

FreeRTOS 的 IDT 無法執行的測試案例數目。

disabled

此屬性未使用，可忽略。

如果沒有測試用例故障或錯誤，則您的設備符合運行 FreeRTOS 的技術要求，並且可以與服務互操作。 AWS IoT 如果您選擇在 AWS 合作夥伴裝置目錄中列出裝置，您可以使用此報告作為資格證據。

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，則此區塊會包含 SDK 的名稱和版本。如果您沒有使用 SDK 運行 IDT，則此塊包含：

```
<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。下面的代碼片段顯示了它在awsiotdevicetester_report.xml文件中的顯示方式。

```
<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>
```

如果所需功能沒有測試失敗或錯誤，則您的裝置符合執行 FreeRTOS 的技術需求，並可與服務互通。 AWS IoT 如果您想在AWS 合作夥伴裝置目錄中列出您的裝置，可以使用此報告作為資格證據。

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">
```

格式與<testsuites>標籤類似，但具有未使用且可以忽略的skipped屬性。在每個 <testsuite> XML 標籤內，測試群組每個執行的測試都有 <testcase> 標籤。例如：

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

lts

如果您符合使用 LTS 庫的 FreeRTOS 版本的資格，則為真，否則為 false。

<testcase>標籤中使用的屬性

name

測試案例的名稱。

attempts

FreeRTOS 的 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="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 疑難排解。

檢視 日誌

您可以在中找到 FreeRTOS 的 IDT 從測試執行產生的記錄檔。*devicetester-extract-location*/results/*execution-id*/logs該工具會產生兩組日誌：

test_manager.log

包含從 FreeRTOS 的 IDT 產生的記錄檔 (例如，與記錄相關的組態設定和報告產生)。

***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 開發和運行自己的測試套件

從 IDT v4.0.0 開始，FreeRTOS 的 IDT 將標準化的組態設定和結果格式與測試套件環境相結合，讓您能夠為您的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添加自定義測試，也可以將其提供給客戶進行設備驗證。

使用 IDT 開發和執行自訂測試套件，如下所示：

若要開發自訂測試套件

- 使用要測試的設備的自定義測試邏輯創建測試套件。
- 為 IDT 提供您的自定義測試套件以測試運行者。包括有關測試套件的特定設置配置的信息。

執行自訂測試套件

- 設定您要測試的裝置。
- 根據要使用的測試套件的要求實施設置配置。
- 使用 IDT 運行您的自定義測試套件。
- 檢視 IDT 執行之測試的測試結果和執行記錄。

下載最新版本的 FreeRT AWS IoT OS 裝置測試程式

下載最新版本的 IDT，並將軟體解壓縮至檔案系統上具有讀取和寫入權限的位置。

> **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

測試套件由三種類型的文件組成：

- 為 IDT 提供有關如何執行測試套件的信息的配置文件。
- 測試 IDT 用於運行測試用例的可執行文件。
- 運行測試所需的其他文件。

完成下列基本步驟以建立自訂 IDT 測試：

1. 為您的測試套件創建配置文件。
2. 創建包含測試套件測試邏輯的測試用例可執行文件。
3. 驗證並記錄測試運行程序運行測試套件所需的配置信息。
4. 驗證 IDT 可以按預期運行測試套件並產生測試結果。

若要快速建置範例自訂套件並執行它，請遵循中的指示教學課程：建置並執行範例 IDT 測試套件。

要開始在 Python 中創建自定義測試套件，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

此下 AWS IoT Device Tester 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，以瞭解如何讓 FreeRTOS 執行自訂測試套件。 AWS IoT Device Tester 雖然本教學課程使用 SSH，但學習如何 AWS IoT Device Tester 搭配 FreeRTOS 裝置使用是很有用的。

在本教學課程中，您將完成以下步驟：

1. 建置範例測試套件
2. 使用 IDT 運行示例測試套件

必要條件

為了完成本教學，您需要以下項目：

- 主機電腦需求
  - 最新版本的 AWS IoT Device Tester
  - Python 3.7 或更高版本

    若要檢查電腦上安裝的 Python 版本，請執行下列命令：

    ```
    python3 --version
    ```

    在 Windows 上，如果使用此命令返回錯誤，則`python --version`改用。如果傳回的版本號碼為 3.7 或更高版本，請在 PowerShell 終端機中執行下列命令，以設定`python3`為`python`命令的別名。

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

    如果沒有傳回任何版本資訊，或是版本號碼小於 3.7，請依照下載 Python 中的指示安裝 Python 3.7+。如需詳細資訊，請參閱 Python 文件。
  - urllib3

    若要確認`urllib3`是否已正確安裝，請執行下列命令：

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

如果urllib3未安裝，請執行下列命令進行安裝：

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

- 裝置要求
  - 具有 Linux 作業系統以及與主機電腦相同網路的網路連線的裝置。

    我們建議您使用樹莓派與樹莓派 OS. 確保您在樹莓派上設置了 SSH 以遠程連接到它。

設定 IDT 的裝置資訊

配置 IDT 的設備信息以運行測試。您必須使用下列資訊更新位於資*<device-tester-extract-location>*/configs料夾中的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碼。

connectivity.ip

您裝置的 IP 位址。

connectivity.port

選用。用於連線至裝置的 SSH 連線的連接埠號碼。

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

用於登入裝置的使用者名稱。

connectivity.auth.credentials.privKeyPath

用於登入裝置之私密金鑰的完整路徑。

只有當 connectivity.auth.method 設為 pki 時，才會套用此值。

devices.connectivity.auth.credentials.password

用於登入裝置的密碼。

只有當 connectivity.auth.method 設為 password 時，才會套用此值。

> **Note**
>
> 如果 `method` 是設定為 `pki`，則指定 `privKeyPath`
> 如果 `method` 是設定為 `password`，則指定 `password`

建置範例測試套件

該文件*<device-tester-extract-location>*/samples/python夾包含示例配置文件，源代碼和 IDT Client SDK，您可以使用提供的構建腳本將其合併到測試套件中。下列目錄樹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
```

若要建立測試套件，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
```

這會在資料夾內的IDTSampleSuitePython_1.0.0資料夾中建立範例測試套*<device-tester-extract-location>*/tests件。檢閱IDTSampleSuitePython_1.0.0資料夾中的檔案，瞭解範例測試套件的結構方式，並查看測試案例可執行檔和測試組態檔的各種範例。

> **Note**
>
> 範例測試套件包含 Python 原始程式碼。請勿在測試套件代碼中包含敏感信息。

下一步：使用 IDT 運行您創建的示例測試套件。

使用 IDT 運行示例測試套件

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 運行示例測試套件並將結果流式傳輸到控制台。測試執行完成後，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

使用下列資訊可協助解決完成自學課程時的任何問題。

測試用例未成功運行

- 如果測試未成功運行，IDT 將錯誤日誌流式傳輸到控制台，以幫助您對測試運行進行故障排除。請確定您符合本教學課程的所有先決條件。

無法連接到被測設備

請確認下列內容：

- 您的`device.json`檔案包含正確的 IP 位址、連接埠和驗證資訊。
- 您可以通過 SSH 從主機連接到設備。

教學課程：開發簡單的 IDT 測試套件

測試套件結合了以下內容：

- 測試包含測試邏輯的可執行文件
- 描述測試套件的配置文件

本教程將向您展示如何使用 IDT 的 FreeRTOS 開發包含單個測試用例的 Python 測試套件。雖然本教學課程使用 SSH，但學習如何 AWS IoT Device Tester 搭配 FreeRTOS 裝置使用是很有用的。

在本教學課程中，您將完成以下步驟：

1. 創建測試套件目錄
2. 建立組態檔案
3. 創建測試用例可執行文件
4. 運行測試套件

必要條件

為了完成本教學，您需要以下項目：

- 主機電腦需求
  - 最新版本的 AWS IoT Device Tester
  - Python 3.7 或更高版本

    若要檢查電腦上安裝的 Python 版本，請執行下列命令：

```
python3 --version
```

在 Windows 上，如果使用此命令返回錯誤，則python --version改用。如果傳回的版本號碼為 3.7 或更高版本，請在 PowerShell 終端機中執行下列命令，以設定python3為python命令的別名。

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

如果沒有傳回任何版本資訊，或是版本號碼小於 3.7，請依照下載 Python 中的指示安裝 Python 3.7+。如需詳細資訊，請參閱 Python 文件。

- urllib3

  若要確認urllib3是否已正確安裝，請執行下列命令：

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

  如果urllib3未安裝，請執行下列命令進行安裝：

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

- 裝置要求
  - 具有 Linux 作業系統以及與主機電腦相同網路的網路連線的裝置。

    我們建議您使用樹莓派與樹莓派 OS. 確保您在樹莓派上設置了 SSH 以遠程連接到它。

創建測試套件目錄

IDT 邏輯上將測試用例分離為每個測試套件中的測試組。每個測試用例必須位於測試組內。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：

```
MyTestSuite_1.0.0
### suite
    ### myTestGroup
        ### myTestCase
```

建立組態檔案

您的測試套件必須包含以下必要的配置文件：

必要的檔案

suite.json

包含測試套件的相關資訊。請參閱設定套件。

group.json

包含測試群組的相關資訊。您必須為測試套group.json件中的每個測試組創建一個文件。請參閱設定群組。

test.json

包含有關測試用例的信息。您必須為測試套test.json件中的每個測試用例創建一個文件。請參閱配置測試。

1. 在MyTestSuite_1.0.0/suite資料夾中，建立具有下列結構的suite.json檔案：

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

2. 在MyTestSuite_1.0.0/myTestGroup資料夾中，建立具有下列結構的group.json檔案：

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

3. 在MyTestSuite_1.0.0/myTestGroup/myTestCase資料夾中，建立具有下列結構的test.json檔案：

```
{
    "id": "MyTestCase",
```

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

MyTestSuite_1.0.0資料夾的目錄樹現在應該如下所示：

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

取得 IDT 用戶端開發套件

您可以使用 IDT 用戶端 SDK 來啟用 IDT 與待測裝置互動，並報告測試結果。在本教程中，您將使用 Python 版本的 SDK。

從*<device-tester-extract-location>*/sdks/python/資料夾中，將資idt_client料夾複製到您的MyTestSuite_1.0.0/suite/myTestGroup/myTestCase資料夾。

若要驗證 SDK 是否已成功複製，請執行下列命令。

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

創建測試用例可執行文件

測試用例可執行文件包含要運行的測試邏輯。一個測試套件可以包含多個測試用例可執行文件。在本教程中，您將只創建一個測試用例可執行文件。

1. 建立測試套件檔案。

   在MyTestSuite_1.0.0/suite/myTestGroup/myTestCase資料夾中，建立包含下列內容的myTestCase.py檔案：

   ```
   from idt_client import *

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

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

2. 使用客戶端 SDK 函數將以下測試邏輯添加到您的myTestCase.py文件中：

   a. 在被測設備上運行 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. 將測試結果發送給 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

配置 IDT 的設備信息以運行測試。您必須使用下列資訊更新位於資*<device-tester-extract-location>*/configs料夾中的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碼。

connectivity.ip

您裝置的 IP 位址。

connectivity.port

選用。用於連線至裝置的 SSH 連線的連接埠號碼。

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

用於登入裝置的使用者名稱。

connectivity.auth.credentials.privKeyPath

用於登入裝置之私密金鑰的完整路徑。

只有當 connectivity.auth.method 設為 pki 時，才會套用此值。

devices.connectivity.auth.credentials.password

用於登入裝置的密碼。

只有當 connectivity.auth.method 設為 password 時，才會套用此值。

> **Note**
>
> 如果 method 是設定為 pki，則指定 privKeyPath
> 如果 method 是設定為 password，則指定 password

運行測試套件

創建測試套件後，要確保它按預期運行。完成以下步驟以使用現有設備池運行測試套件以執行此操作。

1. 將您的資MyTestSuite_1.0.0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 運行您的測試套件並將結果流式傳輸到控制台。測試執行完成後，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

使用下列資訊可協助解決完成自學課程時的任何問題。

測試用例未成功運行

如果測試未成功運行，IDT 將錯誤日誌流式傳輸到控制台，以幫助您對測試運行進行故障排除。檢查錯誤記錄檔之前，請先確認下列事項：

- 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。

無法連接到被測設備

請確認下列內容：

- 您的`device.json`檔案包含正確的 IP 位址、連接埠和驗證資訊。
- 您可以通過 SSH 從主機連接到設備。

建立 IDT 測試套件組態檔

本節說明您在撰寫自訂測試套件時所使用的建立組態檔案的格式。

必要的組態檔案

`suite.json`

包含測試套件的相關資訊。請參閱設定套件。

`group.json`

包含測試群組的相關資訊。您必須為測試套`group.json`件中的每個測試組創建一個文件。請參閱設定群組。

`test.json`

包含有關測試用例的信息。您必須為測試套`test.json`件中的每個測試用例創建一個文件。請參閱配置測試。

可選配置文件

`test_orchestrator.yaml` 或 `state_machine.json`

定義 IDT 執行測試套件時如何執行測試。上交所設定測試協調工具。

> **Note**
>
> 從 IDT v4.5.2 開始，您可以使用`test_orchestrator.yaml`檔案來定義測試工作流程。在舊版 IDT 中，您可以使用該`state_machine.json`檔案。如需狀態機的相關資訊，請參閱設定 IDT 狀態機器。

`userdata_schema.json`

定義測試運行程序可以包含在其設置配置中的`userdata.json`文件的模式。該`userdata.json`文件用於運行測試所需的任何其他配置信息，但不存在於`device.json`文件中。請參閱配置用戶數據模式。

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

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

設定套件

該suite.json文件設置環境變量，並確定是否需要用戶數據來運行測試套件。使用下列範本來設定*<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必須與檔案所在的測試套suite.json件資料夾名稱相符。套件名稱和套件版本也必須符合下列需求：

- *<suite-name>*不能包含底線。
- *<suite-version>*被表示為*x.x.x*，其中x是一個數字。

ID 會顯示在 IDT 產生的測試報告中。

title

此測試套件所測試之產品或功能的使用者定義名稱。測試執行程式的名稱會顯示在 IDT CLI 中。

details

測試套件用途的簡短描述。

userDataRequired

定義測試執行程式是否需要在userdata.json檔案中包含自訂資訊。如果將此值設定為true，則還必須在測試套userdata_schema.json件資料夾中包含該檔案。

environmentVariables

選用。要為此測試套件設置的環境變量數組。

environmentVariables.key

環境變數的名稱。

environmentVariables.value

環境變數的值。

設定群組

該group.json文件定義測試組是必需的還是可選的。使用下列範本來設定*<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檔案所在之測試群組資料夾的名稱，且不應有底線 (_)。ID 會用於 IDT 產生的測試報告中。

title

測試群組的描述性名稱。測試執行程式的名稱會顯示在 IDT CLI 中。

details

測試群組用途的簡短描述。

optional

選用。設定`true`為可在 IDT 完成執行必要測試後，將此測試群組顯示為選用群組。預設值為`false`。

配置測試

該`test.json`文件確定測試用例可執行文件和由測試用例使用的環境變量。如需建立測試案例可執行檔的詳細資訊，請參閱創建 IDT 測試用例可執行文件。

使用下列範本來設定*<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

測試用例的唯一用戶定義 ID。的值`id`必須與檔案所在的測試`test.json`案例資料夾名稱相符，且不應有底線 (_)。ID 會用於 IDT 產生的測試報告中。

title

測試用例的描述性名稱。測試執行程式的名稱會顯示在 IDT CLI 中。

details

測試用例目的的簡短描述。

requireDUT

選用。設定為是`true`否需要裝置才能執行此測試，否則將設定為`false`。預設值為 `true`。測試運行者將配置他們將用於在其`device.json`文件中運行測試的設備。

requiredResources

選用。陣列，提供執行此測試所需資源裝置的相關資訊。

requiredResources.name

此測試運行時為資源設備提供的唯一名稱。

requiredResources.features

使用者定義的資源裝置功能陣列。

requiredResources.features.name

特徵的名稱。您要使用此裝置的裝置功能。此名稱與`resource.json`文件中測試運行器提供的功能名稱匹配。

requiredResources.features.version

選用。功能的版本。該值與`resource.json`文件中的測試運行程序提供的功能版本匹配。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次數。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1此值設定為。

execution.timeout

IDT 等待測試完成執行的時間 (以毫秒為單位)。若要取得有關設定此值的更多資訊，請參閱創建 IDT 測試用例可執行文件。

execution.*os*

根據執行 IDT 之主機電腦的作業系統，要執行的測試案例可執行檔。支援的值為 `linux`、`mac` 和 `win`。

execution.*os*.cmd

您要針對指定作業系統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。要提供執行測試案例可執行檔的引數。

environmentVariables

選用。為此測試用例設置的環境變量數組。

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，則test.json檔案中的值優先。suite.json

設定測試協調工具

測試協調器是一種控制測試套件執行流程的構造。它會判斷測試套件的開始狀態、根據使用者定義的規則管理狀態轉換，並繼續透過這些狀態進行轉換，直到達到結束狀態為止。

如果您的測試套件不包含用戶定義的測試協調器，IDT 將為您生成測試協調器。

默認測試協調器執行以下功能：

- 為測試運行者提供選擇和運行特定測試組的功能，而不是整個測試套件。
- 如果未選擇特定的測試組，請以隨機順序運行測試套件中的每個測試組。
- 生成報告並打印顯示每個測試組和測試用例的測試結果的控制台摘要。

如需 IDT 測試協調器如何運作的詳細資訊，請參閱。設定 IDT 測試協調器

配置用戶數據模式

該userdata_schema.json文件確定測試運行程序提供用戶數據的模式。如果您的測試套件需要device.json文件中不存在的信息，則需要用戶數據。例如，您的測試可能需要使用者必須提供的 Wi-Fi 網路認證、特定開放連接埠或憑證。此資訊可以作為使用者在其資*<device-tester-extract-location>*/config料夾中建立的名為userdatauserdata.json檔案的輸入參數提供給IDT。該userdata.json文件的格式基於您包含在測試套userdata_schema.json件中的文件。

為了表示測試運行者必須提供一個userdata.json文件：

1. 在suite.json檔案中，設定userDataRequired為true。
2. 在您的*<custom-test-suite-folder>*中建立userdata_schema.json檔案。
3. 編輯該userdata_schema.json文件以創建一個有效的 IETF 草案 v4 JSON 模式。

當 IDT 運行測試套件時，它會自動讀取模式並使用它來驗證測試運行器提供的`userdata.json`文件。如果有效，則`userdata.json`檔案內容可在 IDT 前後關聯和測試協調器內容中使用。

設定 IDT 測試協調器

從 IDT v4.5.2 開始，IDT 包含新的測試協調器元件。測試協調器是一個 IDT 組件，用於控制測試套件執行流程，並在 IDT 完成運行所有測試後生成測試報告。測試協調器會根據使用者定義的規則，決定測試選擇和執行測試的順序。

如果您的測試套件不包含用戶定義的測試協調器，IDT 將為您生成測試協調器。

默認測試協調器執行以下功能：

- 為測試運行者提供選擇和運行特定測試組的功能，而不是整個測試套件。
- 如果未選擇特定的測試組，請以隨機順序運行測試套件中的每個測試組。
- 生成報告並打印顯示每個測試組和測試用例的測試結果的控制台摘要。

測試協調器會取代 IDT 狀態機器。我們強烈建議您使用測試協調器來開發測試套件，而不是 IDT 狀態機器。測試協調器提供了以下改進的功能：

- 與 IDT 狀態機器使用的命令式格式相比，使用宣告式格式。這使您可以指定要運行的測試以及何時運行它們。
- 管理特定群組處理、報告產生、錯誤處理和結果追蹤，因此您不需要手動管理這些動作。
- 使用 YAML 格式，預設支援註解。
- 需要比測試協調器少 80% 的磁碟空間來定義相同的工作流程。
- 添加測試前驗證以驗證您的工作流程定義不包含錯誤的測試 ID 或循環依賴關係。

測試協調器格式

您可以使用下列範本來設定您自己的*custom-test-suite-folder*/suite/test_orchestrator.yaml檔案：

```
Aliases:
  string: context-expression

ConditionalTests:
  - Condition: context-expression
    Tests:
```

```
      - test-descriptor

Order:
  - - group-descriptor
    - group-descriptor

Features:
  - Name: feature-name
    Value: support-description
    Condition: context-expression
    Tests:
        - test-descriptor
    OneOfTests:
        - test-descriptor
    IsRequired: boolean
```

如下所述，包含值的所有欄位皆為必要：

`Aliases`

選用。對應至上下文運算式的使用者定義字串 別名可讓您產生易記的名稱，以識別測試協調器組態中的內容運算式。如果您要建立複雜的內容運算式或在多個位置使用的運算式，這個選項特別有用。

您可以使用前後關聯運算式來儲存前後關聯查詢，以便從其他 IDT 組態存取資料。如需詳細資訊，請參閱 存取上下文中的資料。

Example

範例

```
Aliases:
    FizzChosen: "'{{$pool.features[?(@.name == 'Fizz')].value[0]}}' == 'yes'"
    BuzzChosen: "'{{$pool.features[?(@.name == 'Buzz')].value[0]}}' == 'yes'"
    FizzBuzzChosen: "'{{$aliases.FizzChosen}}' && '{{$aliases.BuzzChosen}}'"
```

`ConditionalTests`

選用。條件清單，以及滿足每個條件時執行的對應測試案例。每個條件都可以有多個測試用例；但是，您只能將給定的測試用例分配給一個條件。

默認情況下，IDT 運行未分配給此列表中條件的任何測試用例。如果您未指定此部分，IDT 會運行測試套件中的所有測試組。

`ConditionalTests`清單中的每個項目都包含下列參數：

`Condition`

評估為布林值的上下文運算式。如果評估值為 true，IDT 會執行`Tests`參數中指定的測試案例。

`Tests`

測試描述符的列表。

每個測試描述符都使用測試組 ID 和一個或多個測試用例 ID 來識別要從特定測試組運行的單個測試。測試描述符使用以下格式：

```
GroupId: group-id
CaseIds: [test-id, test-id] # optional
```

Example

範例

下列範例使用您可以定義為的泛用前後關聯運算式`Aliases`。

```
ConditionalTests:
    - Condition: "{{$aliases.Condition1}}"
      Tests:
          - GroupId: A
          - GroupId: B
    - Condition: "{{$aliases.Condition2}}"
      Tests:
          - GroupId: D
    - Condition: "{{$aliases.Condition1}} || {{$aliases.Condition2}}"
      Tests:
          - GroupId: C
```

根據定義的條件，IDT 選取測試群組，如下所示：

- 如果`Condition1`為 true，IDT 會在測試群組 A、B 和 C 中執行測試。
- 如果`Condition2`為 true，IDT 會在測試群組 C 和 D 中執行測試。

`Order`

選用。執行測試的順序。您可以在測試群組層級指定測試順序。如果您未指定此部分，IDT 會以隨機順序運行所有適用的測試組。的值`Order`是群組描述元清單的清單。您未列出的任何測試組都可以與任何其他列出的測試組 parallel 運行。`Order`

每個群組描述元清單都包含其中一個以上的群組描述元，並識別執行在每個描述元中指定之群組的順序。您可以使用下列格式來定義個別的群組描述元：

- *group-id*現有測試群組的群組 ID。
- [*group-id*, *group-id*]可以以任何相對順序執行的測試群組清單。
- "*"萬用字元。這等同於目前群組描述元清單中尚未指定的所有測試群組清單。

的值也Order必須符合下列需求：

- 您在組描述符中指定的測試組 ID 必須存在於測試套件中。
- 每個群組描述元清單必須包含至少一個測試群組。
- 每個群組描述元清單必須包含唯一的群組 ID。您無法在個別群組描述元中重複測試群組 ID。
- 群組描述元清單最多只能有一個萬用字元群組描述元。萬用字元群組描述元必須是清單中的第一個或最後一個項目。

Example

範例

對於包含測試群組 A、B、C、D 和 E 的測試套件，下列範例清單顯示了指定 IDT 應先執行測試群組 A，然後執行測試群組 B，然後以任何順序執行測試群組 C、D 和 E 的不同方法。

- ```
  Order:
      - - A
        - B
        - [C, D, E]
  ```
- ```
  Order:
      - - A
        - B
        - "*"
  ```
- ```
  Order:
      - - A
        - B

      - - B
        - C

      - - B
        - D
  ```
  ```

```
  - - B
    - E
```

Features

選用。您希望 IDT 新增至awsiotdevicetester_report.xml檔案的產品功能清單。如果您未指定此區段，IDT 不會在報告中新增任何產品功能。

產品功能是有關設備可能符合的特定條件的用戶定義信息。例如，MQTT 產品功能可指定裝置正確發佈 MQTT 訊息。在中awsiotdevicetester_report.xml，產品功能會根據指定的測試是否通過 supportednot-supported，將產品功能設定為、或自訂使用者定義值。

Features清單中的每個項目都包含下列參數：

Name

特徵的名稱。

Value

選用。您要在報表中使用的自訂值，而不是supported。如果未指定此值，則以 IDT 為基礎將特徵值設定為supported或not-supported根據測試結果。如果您使用不同的條件測試相同的特徵，則可以為Features清單中該特徵的每個例證使用自訂值，並且 IDT 會為支援的條件連接特徵值。如需詳細資訊，請參閱

Condition

評估為布林值的上下文運算式。如果評估值為 true，IDT 會在測試報告完成執行測試套件後，將該功能新增至測試報告。如果評估值為 false，則測試不會包含在報告中。

Tests

選用。測試描述符的列表。此清單中指定的所有測試都必須通過，才能支援該功能。

此列表中的每個測試描述符都使用測試組 ID 和一個或多個測試用例 ID 來識別要從特定測試組運行的單個測試。測試描述符使用以下格式：

```
GroupId: group-id
CaseIds: [test-id, test-id] # optional
```

您必須為Features清單中OneOfTests的每個特徵指定Tests或。

`OneOfTests`

選用。測試描述符的列表。此清單中指定的至少一項測試必須通過，才能支援此功能。

此列表中的每個測試描述符都使用測試組 ID 和一個或多個測試用例 ID 來識別要從特定測試組運行的單個測試。測試描述符使用以下格式：

```
GroupId: group-id
CaseIds: [test-id, test-id] # optional
```

您必須為`Features`清單中`OneOfTests`的每個特徵指定`Tests`或。

`IsRequired`

布爾值，用於定義測試報告中是否需要該功能。預設值為 `false`。

測試協調器上下文

測試協調器內容是唯讀的 JSON 文件，其中包含測試協調器在執行期間可供測試協調器使用的資料。測試協調器上下文只能從測試協調器訪問，並包含確定測試流程的信息。例如，您可以使用測試運行程序在`userdata.json`文件中配置的信息來確定是否需要運行特定測試。

測試協調器內容使用下列格式：

```
{
    "pool": {
        <device-json-pool-element>
    },
    "userData": {
        <userdata-json-content>
    },
    "config": {
        <config-json-content>
    }
}
```

`pool`

有關為測試運行選擇的設備池的信息。對於選取的裝置集區，會從`device.json`檔案中定義的對應頂層裝置池陣列元素擷取此資訊。

`userData`

userdata.json檔案中的資訊。

`config`

config.json檔案中的資訊。

您可以使用 JSONPath 符號查詢上下文。在狀態定義中的 JSONPath 查詢的語法是。{{*query*}}當您從測試協調器內容存取資料時，請確定每個值評估為字串、數字或布林值。

如需有關使用 JsonPath 標記法存取內容中資料的詳細資訊，請參閱。使用 IDT 上下文

設定 IDT 狀態機器

> **Important**
>
> 從 IDT v4.5.2 開始，此狀態機器已被棄用。強烈建議您使用新的測試協調器。如需詳細資訊，請參閱 設定 IDT 測試協調器。

狀態機是控制測試套件執行流程的結構。它會判斷測試套件的開始狀態、根據使用者定義的規則管理狀態轉換，並繼續透過這些狀態進行轉換，直到達到結束狀態為止。

如果您的測試套件不包含用戶定義的狀態機，IDT 將為您生成一個狀態機。預設狀態機執行下列功能：

- 為測試運行者提供選擇和運行特定測試組的功能，而不是整個測試套件。
- 如果未選擇特定的測試組，請以隨機順序運行測試套件中的每個測試組。
- 生成報告並打印顯示每個測試組和測試用例的測試結果的控制台摘要。

IDT 測試套件的狀態機必須符合以下條件：

- 每個狀態都對應於 IDT 要採取的動作，例如執行測試群組或產生報告檔案。
- 轉移至某個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移規則。
- 結束狀態必須是`Succeed`或`Fail`。

狀態機格式

您可以使用下列範本來設定您自己的*<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

將使用者定義的狀態名稱對映至有效 IDT 狀態的物件。每個州。 ##########

States物件必須包含Succeed和Fail狀態。如需有效狀態的資訊，請參閱有效的狀態和狀態定義。

有效的狀態和狀態定義

本節說明 IDT 狀態機器中可以使用的所有有效狀態的狀態定義。以下某些狀態支持測試用例級別的配置。不過，除非絕對必要，否則建議您在測試群組層級而非測試案例層級設定狀態轉換規則。

狀態定義

- RunTask
- Choice
- 平行
- AddProductFeatures
- 報告
- LogMessage
- SelectGroup
- 失敗
- Succeed

RunTask

該RunTask狀態從測試套件中定義的測試組運行測試用例。

```
{
    "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之後要轉換至的狀態名稱。

`TestGroup`

選用。要執行之測試群組的 ID。如果未指定此值，則 IDT 運行測試運行器選擇的測試組。

TestCases

選用。來自中指定之群組的測試用例 ID 陣列TestGroup。根據TestGroup和的值TestCases，IDT 會判斷測試執行行為，如下所示：

- 當同時指TestCases定TestGroup和時，IDT 會從測試組運行指定的測試用例。
- TestCases如果指定但TestGroup未指定，IDT 會執行指定的測試案例。
- 如果TestGroup指定但TestCases未指定，IDT 會執行指定測試群組內的所有測試案例。
- 如果兩者都沒TestCases有指定TestGroup或，IDT 會從測試運行程序從 IDT CLI 中選擇的測試組中運行所有測試用例。若要啟用測試執行程式的群組選取，您必須在statemachine.json檔案中同時包含RunTask和Choice狀態。如需其運作方式的範例，請參閱範例狀態機器：執行使用者選取的測試群組。

  如需針對測試執行程式啟用 IDT CLI 命令的詳細資訊，請參閱〈〉the section called “啟用 IDT 命 CLI”。

ResultVar

要與測試運行結果一起設置的上下文變量的名稱。如果您未指定的值，請勿指定此值TestGroup。IDT 會false根據下列項目，將您在中定義的變數值設定ResultVar為true或：

- 如果變量名是形式的*text*_*text*_passed，則該值被設置為是否通過或被跳過的第一個測試組中的所有測試。
- 在所有其他情況下，該值被設置為是否通過或跳過所有測試組中的所有測試。

通常，您將使用 RunTask state 指定測試組 ID 而不指定單獨的測試用例 ID，以便 IDT 將運行指定測試組中的所有測試用例。由此狀態執行的所有測試案例會以隨機順序 parallel 執行。但是，如果所有測試用例都需要運行設備，並且只有一個設備可用，則測試用例將按順序運行。

錯誤處理

如果任何指定的測試組或測試用例 ID 無效，則此狀態會發出RunTaskError執行錯誤。如果狀態遇到執行錯誤，那麼它也將狀態機上下文中的hasExecutionError變量設置為true。

Choice

狀Choice態可讓您根據使用者定義的條件動態設定要轉換為的下一個狀態。

```
{
    "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`Choices`定義的任何運算式都不能評估為，則要轉換到的預設狀態`true`。

`FallthroughOnError`

選用。指定狀態在評估運算式時遇到錯誤時的行為。`true`如果您想要在評估結果發生錯誤時略過表示式，則設定為。如果沒有符合的運算式，則狀態機會轉換到`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，可決定在目前狀態下執行動作之後要轉換為哪個狀態。

`Choices.Expression`

計算結果為布林值的運算式字串。如果表示式評估為`true`，則狀態機會轉換為中定義的狀態`Choices.Next`。運算式字串會從狀態機器內容擷取值，然後對它們執行作業以達到布林值。如需有關存取狀態機器前後關聯的資訊，請參閱状态机上下文。

`Choices.Next`

如果中定義的運算式`Choices.Expression`評估為，則要轉換至的狀態名稱`true`。

錯誤處理

在下列情況下，`Choice`狀態可能需要錯誤處理：

- 選擇運算式中的某些變數不存在於狀態機器上下文中。
- 運算式的結果不是布林值。
- JSON 查閱的結果不是字串、數字或布林值。

您無法使用圖Catch塊來處理此狀態下的錯誤。如果您想要在發生錯誤時停止執行狀態機，則必須FallthroughOnError將設定為false。不過，我們建議您FallthroughOnError將設定為true，並根據您的使用案例，執行下列其中一項作業：

- 如果您正在存取的變數在某些情況下預期不存在，請使用的值Default和其他Choices區塊來指定下一個狀態。
- 如果您正在訪問的變量應始終存在，則將狀Default態設置為Fail。

平行

此Parallel狀態可讓您彼此 parallel 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之後要轉換至的狀態名稱。

Branches

要執行的狀態機定義陣列。每個狀態機定義都必須包含其本身的StartAtSucceed、、和Fail狀態。此陣列中的狀態機定義無法參考其定義之外的狀態。

> **Note**
>
> 由於每個分支狀態機器共享相同的狀態機器上下文，因此在一個分支中設置變量，然後從另一個分支讀取這些變量可能會導致意外行為。

狀Parallel態只會在執行所有分支狀態機器之後才會移至下一個狀態。需要設備的每個狀態都將等待運行，直到設備可用為止。如果有多個設備可用，則此狀態會從多個組並行運 parallel 測試用例。如果

足夠的設備不可用，那麼測試用例將按順序運行。由於測試用例在並行運行時以隨機順序運 parallel，因此可能會使用不同的設備從同一測試組運行測試。

錯誤處理

確保分支狀態機器和父狀態機器都轉換為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到父狀態機器，因此您無法使用Catch區塊來處理分支狀態機器中的執行錯誤。而是在共用狀態機器上下文中使用該hasExecutionErrors值。如需其運作方式的範例，請參閱狀態機示例：並行運 parallel 兩個測試組。

AddProductFeatures

狀AddProductFeatures態可讓您將產品功能新增至 IDT 所產生的awsiotdevicetester_report.xml檔案。

產品功能是有關設備可能符合的特定條件的用戶定義信息。例如，MQTT產品功能可指定裝置正確發佈 MQTT 訊息。在報告中，產品功能會根據指定的測試是否通過，設定為supported、或自訂值。not-supported

> **Note**
>
> 狀AddProductFeatures態本身不會產生報告。此狀態必須轉移到狀Report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之後要轉換至的狀態名稱。

Features

要在awsiotdevicetester_report.xml檔案中顯示的產品功能陣列。

Feature

特徵的名稱

FeatureValue

選用。要在報表中使用的自訂值，而不是supported。如果未指定此值，則根據測試結果，特徵值會設定為supported或not-supported。

如果您將自訂值用於FeatureValue，則可以使用不同的條件測試相同的圖徵，並且 IDT 會為支援的條件連接特徵值。例如，以下摘錄展示了具有兩個單獨MyFeature特徵值的特徵：

```
...
{
    "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，則特徵值會設定為first-feature-supported, second-feature-supported。

Groups

選用。測試群組 ID 的陣列。必須通過每個指定測試組中的所有測試才能支持該功能。

OneOfGroups

選用。測試群組 ID 的陣列。至少一個指定測試群組中的所有測試都必須通過，才能支援該功能。

TestCases

選用。測試用例 ID 的數組。如果您指定此值，則適用下列項目：

- 必須通過所有指定的測試用例才能支持該功能。
- Groups只能包含一個測試群組 ID。
- OneOfGroups不得指定。

IsRequired

選用。設定false為可將此功能標記為報表中的選用功能。預設值為 true。

ExecutionMethods

選用。與device.json檔案中指定的protocol值相符的執行方法陣列。如果指定此值，則測試執行者必須指定protocol與此陣列中的其中一個值相符的值，才能在報表中包含該功能。如果未指定此值，則該特徵將永遠包括在報告中。

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：

- 如果您指定了單獨的測試用例 ID，則ResultVar將設置為*group-id_test-id*_passed。
- 如果您未指定個別測試案例 ID，則ResultVar將設定為*group-id*_passed。

狀AddProductFeatures態會以下列方式檢查測試結果：

- 如果您未指定任何測試用例 ID，則每個測試組的結果將根據狀態機上下文中的*group-id*_passed變量值確定。
- 如果您確實指定了測試用例 ID，則每個測試的結果都是從狀態機器上下文中的*group-id_test-id*_passed變量值確定的。

錯誤處理

如果在此狀態下提供的群組 ID 不是有效的群組識別碼，則此狀態會導致AddProductFeaturesError執行錯誤。如果狀態遇到執行錯誤，那麼它也將狀態機上下文中的hasExecutionErrors變量設置為true。

報告

狀Report態會產生*suite-name*_Report.xml和awsiotdevicetester_report.xml檔案。此狀態也會將報表串流至主控台。

```
{
    "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之後要轉換至的狀態名稱。

您應該始終轉換到測試執行流程結束時的Report狀態，以便測試運行程序可以查看測試結果。通常，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

狀LogMessage態會生成test_manager.log文件並將日誌消息流式傳輸到控制台。

```
{
    "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之後要轉換至的狀態名稱。

Level

要在其中建立記錄訊息的錯誤層級。如果您指定的層級無效，此狀態會產生錯誤訊息並捨棄它。

Message

要記錄的訊息。

SelectGroup

狀SelectGroup態會更新狀態機器前後關聯，以指示已選取哪些群組。此狀態所設定的值會由任何後續狀Choice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之後要轉換至的狀態名稱。

TestGroups

將標記為已選取的測試群組陣列。對於此數組中的每個測試組 ID，*group-id*_selected變量在上下文true中設置為。請確定您提供有效的測試群組 ID，因為 IDT 不會驗證指定的群組是否存在。

失敗

狀Fail態表示狀態機未正確執行。這是狀態機器的結束狀態，且每個狀態機器定義都必須包含此狀態。

```
{
```

```
    "Type": "Fail"
}
```

Succeed

狀Succeed態表示狀態機正確執行。這是狀態機器的結束狀態，且每個狀態機器定義都必須包含此狀態。

```
{
    "Type": "Succeed"
}
```

状态机上下文

狀態機器內容是唯讀的 JSON 文件，其中包含在執行期間可供狀態機器使用的資料。狀態機上下文只能從狀態機訪問，並包含確定測試流程的信息。例如，您可以使用測試運行程序在userdata.json文件中配置的信息來確定是否需要運行特定測試。

狀態機器上下文使用以下格式：

```
{
    "pool": {
        <device-json-pool-element>
    },
    "userData": {
        <userdata-json-content>
    },
    "config": {
        <config-json-content>
    },
    "suiteFailed": true | false,
    "specificTestGroups": [
        "<group-id>"
    ],
    "specificTestCases": [
        "<test-id>"
    ],
    "hasExecutionErrors": true
}
```

pool

有關為測試運行選擇的設備池的信息。對於選取的裝置集區，會從device.json檔案中定義的對應頂層裝置池陣列元素擷取此資訊。

userData

userdata.json檔案中的資訊。

config

資訊釘選config.json檔案。

suiteFailed

此值設定為狀態機啟動false時。如果測試群組在某個RunTask狀態下失敗，則此值會在狀態機器執true行的剩餘持續時間內設定為。

specificTestGroups

如果測試運行程序選擇要運行的特定測試組而不是整個測試套件，則會創建此密鑰並包含特定測試組 ID 的列表。

specificTestCases

如果測試運行程序選擇要運行的特定測試用例而不是整個測試套件，則會創建此密鑰並包含特定測試用例 ID 的列表。

hasExecutionErrors

狀態機啟動時不退出。如果有任何狀態遇到執行錯誤，則會在狀態機器執行的剩餘持續時間內建立此變數並將其設定true為。

您可以使用 JSONPath 符號查詢上下文。在狀態定義中的 JSONPath 查詢的語法是。{{$.*query*}}您可以在某些狀態中使用 JSONPath 查詢作為佔位符字符串。IDT 將預留位置字串取代為從上下文中評估的 JsonPath 查詢的值。您可以為下列值使用預留位置：

- RunTask狀態中的TestCases值。
- Expression值Choice狀態。

當您從狀態機器前後關聯存取資料時，請確定符合下列條件：

- 您的 JSON 路徑必須以 $.

- 每個值必須評估為字串、數字或布林值。

如需有關使用 JsonPath 標記法存取內容中資料的詳細資訊，請參閱。使用 IDT 上下文

執行錯誤

執行錯誤是狀態機在執行狀態時遇到的狀態機器定義中的錯誤。IDT 會記錄test_manager.log檔案中每個錯誤的相關資訊，並將記錄訊息串流至主控台。

您可以使用下列方法來處理執行錯誤：

- 在狀態定義中加入Catch圖塊。
- 檢查狀態機器上下文中的hasExecutionErrors值。

捕捉

要使用Catch，請將以下內容添加到狀態定義中：

```
"Catch": [
    {
        "ErrorEquals": [
            "<error-type>"
        ]
        "Next": "<state-name>"
    }
]
```

如下所述，包含值的所有欄位皆為必要：

Catch.ErrorEquals

要 catch 的錯誤類型的數組。如果執行錯誤符合其中一個指定值，則狀態機會轉換至中指定的狀態Catch.Next。請參閱每個狀態定義，以取得其所產生錯誤類型的相關資訊。

Catch.Next

當目前狀態遇到符合中指定值之一的執行錯誤時，要轉換到的下一個狀態Catch.ErrorEquals。

Catch 區塊會依序處理，直到一個相符項目為止。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。因為執行錯誤是由於不正確的狀態定義所造成，因此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，它們還將hasExecutionError值設置為狀態機器上下文true中。您可以使用此值來偵測何時發生錯誤，然後使用Choice狀態將狀態機轉換為狀Fail態。

此方法具有以下特徵。

- 狀態機不會以指派給的任何值開始hasExecutionError，並且在特定狀態設定之前無法使用此值。這表示您必須false針對存取此值的FallthroughOnErrorChoice狀態明確設定為，以防止狀態機在沒有發生任何執行錯誤時停止。
- 設定為後true，hasExecutionError永遠不會設定為 false 或從前後關聯中移除。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，它不會提供有意義的值。
- 該hasExecutionError值會與狀態中的所有分支狀態機器Parallel共用，這可能會導致非預期的結果，具體取決於其存取順序。

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。

示例狀態機

本節提供一些狀態機器組態的範例。

範例

- 示例狀態機：運行單個測試組
- 狀態機器範例：執行使用者選取的測試群組
- 狀態機器範例：執行具有產品功能的單一測試群組
- 狀態機示例：並行運 parallel 兩個測試組

示例狀態機：運行單個測試組

這種狀態機：

- 使用 id 運行測試組GroupA，該 ID 必須存在於文件中的套group.json件中。
- 檢查執行錯誤和轉換是Fail否有找到。
- Succeed如果沒有錯誤，則生成報告並轉換為，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

這種狀態機：

- 檢查測試運行者是否選擇了特定的測試組。狀態機不檢查特定的測試用例，因為測試運行程序無法選擇測試用例，而無需選擇測試組。

- 如果選取了測試群組：
  - 運行所選測試組中的測試用例。為此，狀態機不會明確指定狀RunTask態中的任何測試組或測試用例。
  - 運行所有測試並退出後生成報告。
- 如果未選取測試群組：
  - 在測試組中運行測試GroupA。
  - 生成報告並退出。

```
{
    "Comment": "Runs specific groups if the test runner chose to do that, otherwise
 runs GroupA.",
    "StartAt": "SpecificGroupsCheck",
    "States": {
        "SpecificGroupsCheck": {
            "Type": "Choice",
            "Default": "RunGroupA",
            "FallthroughOnError": true,
            "Choices": [
                {
                    "Expression": "{{$.specificTestGroups[0]}} != ''",
                    "Next": "RunSpecificGroups"
                }
            ]
        },
        "RunSpecificGroups": {
            "Type": "RunTask",
            "Next": "Report",
            "Catch": [
                {
                    "ErrorEquals": [
                        "RunTaskError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "RunGroupA": {
            "Type": "RunTask",
            "Next": "Report",
            "TestGroup": "GroupA",
```

```
            "Catch": [
                {
                    "ErrorEquals": [
                        "RunTaskError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "Report": {
            "Type": "Report",
            "Next": "Succeed",
            "Catch": [
                {
                    "ErrorEquals": [
                        "ReportError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "Succeed": {
            "Type": "Succeed"
        },
        "Fail": {
            "Type": "Fail"
        }
    }
}
```

狀態機器範例：執行具有產品功能的單一測試群組

這種狀態機：

- 執行測試群組GroupA。
- 檢查執行錯誤和轉換是Fail否有找到。
- 將FeatureThatDependsOnGroupA功能添加到文awsiotdevicetester_report.xml件中：
  - 如果GroupA通過，特徵會設定為supported。
  - 該功能在報告中未標記為可選。
- Succeed如果沒有錯誤，則生成報告並轉換為，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"
        }
    }
}
```

狀態機示例：並行運 parallel 兩個測試組

這種狀態機：

- 並行執 parallel GroupA 和GroupB測試群組。由分支狀態機器中的RunTask狀態儲存在前後關聯中的ResultVar變數可供AddProductFeatures狀態使用。
- 檢查執行錯誤和轉換是Fail否有找到。此狀態機不使用Catch塊，因為該方法不會檢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，特徵會設定為supported。
  - 該功能在報告中未標記為可選。
- Succeed如果沒有錯誤，則生成報告並轉換為，Fail否則

如果設備池中配置了兩個設備，則GroupA和GroupB可以同時運行。但是，如果其中一個GroupA或GroupB具有多個測試，則可以將兩個設備分配給這些測試。如果僅配置了一個設備，則測試組將按順序運行。

```
{
    "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"
        }
    }
}
```

創建 IDT 測試用例可執行文件

您可以通過以下方式在測試套件文件夾中創建並放置測試用例可執行文件：

- 對於使用文件中的參數或環境變量來確定要運行哪些測試的測試套test.json件，您可以為整個測試套件創建單個測試用例可執行文件，或為測試套件中的每個測試組創建一個測試可執行文件。
- 對於要根據指定命令運行特定測試的測試套件，您可以為測試套件中的每個測試用例創建一個可執行文件。

作為測試編寫者，您可以確定哪種方法適合您的用例，並相應地構建測試用例可執行文件。確保您在每個test.json文件中提供了正確的測試用例可執行路徑，並且指定的可執行文件正確運行。

當所有設備都準備好運行測試用例時，IDT 會讀取以下文件：

- 所選測試案例的決定要啟動的程序以及要設定的環境變數。test.json
- 測試套件的決定要設定的環境變數。suite.json

IDT 會根據test.json檔案中指定的命令和引數啟動所需的測試可執行程序，並將必要的環境變數傳遞至處理序。

使用 IDT 用戶端開發套件

IDT 客戶端 SDK 使您可以使用 API 命令簡化在測試可執行文件中編寫測試邏輯的方式，這些命令可以與 IDT 和受測設備進行交互。IDT 目前提供下列軟體開發套件：

- 用於 Python 的 IDT 客戶端開發套件
- 適用於移動的 IDT 客戶端 SDK
- 用於 Java 的 IDT 客戶端開發套件

這些 SDK 位於*<device-tester-extract-location>*/sdks資料夾中。當您創建新的測試用例可執行文件時，必須將要使用的 SDK 複製到包含測試用例可執行文件的文件夾中，並在代碼中引用 SDK。本節提供了可用於測試用例可執行文件中的可用 API 命令的簡要說明。

本節內容

- 裝置互動
- IDT 互動
- 主機互動

裝置互動

以下命令使您可以與被測設備進行通信，而無需實現任何其他設備交互和連接管理功能。

`ExecuteOnDevice`

允許測試套件在支持 SSH 或 Docker 外殼連接的設備上運行 shell 命令。

`CopyToDevice`

允許測試套件將本地文件從運行 IDT 的主機複製到支持 SSH 或 Docker shell 連接的設備上的指定位置。

`ReadFromDevice`

允許測試套件從支持 UART 連接的設備的串行端口讀取。

> **Note**
>
> 由於 IDT 不管理與使用內容中的設備訪問信息進行的設備的直接連接，因此我們建議您在測試用例可執行文件中使用這些設備交互 API 命令。但是，如果這些命令不符合您的測試用例要求，則可以從 IDT 上下文中檢索設備訪問信息，並使用它從測試套件直接連接到設備。若要建立直接連線，請分別擷取待測裝置的`device.connectivity`和`resource.devices.connectivity`欄位中的資訊，以及資源裝置的資訊。若要取得有關使用 IDT 前後關聯的更多資訊，請參閱使用 IDT 上下文。

IDT 互動

以下命令使您的測試套件能夠與 IDT 進行通信。

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 上下文中檢索值。如需詳細資訊，請參閱 使用 IDT 上下文。

SendResult

允許測試套件向 IDT 報告測試用例結果。必須在測試套件中的每個測試用例結束時調用此命令。

主機互動

以下命令使您的測試套件能夠與主機進行通信。

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 上下文中檢索值。如需詳細資訊，請參閱 使用 IDT 上下文。

ExecuteOnHost

允許測試套件在本地計算機上運行命令，並允許 IDT 管理測試用例可執行生命週期。

啟用 IDT 命 CLI

命`run-suite`令 IDT CLI 提供了幾個選項，讓測試運行器自定義測試執行。為了允許測試運行程序使用這些選項來運行自定義測試套件，請實現對 IDT CLI 的支持。如果您沒有實現支持，測試運行程序仍然可以運行測試，但是某些 CLI 選項將無法正常工作。為了提供理想的客戶體驗，我們建議您在 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，此值將在執行測試時套用至所有逾時。

測試運行者可以使用此引數來增加他們想要運行的測試用例的超時時間。當測試運行程序在其`run-suite`命令中指定此引數時，IDT 會使用它來計算 IDT_TEST_TIMEOUT 環境變量的值，並在 IDT 上下文中設置該`config.timeoutMultiplier`字段。若要支援此引數，您必須執行下列動作：

- 而不是直接使用test.json檔案中的逾時值，而是讀取 IDT_TEST_TIMEOUT 環境變數以取得正確計算的逾時值。
- 從 IDT 內容擷取config.timeoutMultiplier值，並將其套用至長時間執行逾時。

如需因逾時事件而提前退出的詳細資訊，請參閱指定退出行為。

stop-on-first-failure

指定 IDT 遇到失敗時應停止執行所有測試。

當測試運行程序在其run-suite命令中指定此參數時，IDT 將在遇到故障時立即停止運行測試。但是，如果測試用例並行運 parallel，那麼這可能會導致意外的結果。要實現支持，請確保如果 IDT 遇到此事件，則測試邏輯會指示所有正在運行的測試用例停止，清理臨時資源並向 IDT 報告測試結果。如需失敗提前退出的詳細資訊，請參閱指定退出行為。

group-id 和 **test-id**

指定 IDT 應該只執行選取的測試群組或測試案例。

測試運行者可以將這些引數與其run-suite命令一起使用，以指定以下測試執行行為：

- 運行指定的測試組內的所有測試。
- 從指定的測試組中運行選擇的測試。

為了支持這些參數，您的測試套件的狀態機必須在狀態機中包含一組特定的RunTask和Choice狀態。如果您不使用自訂狀態機器，則預設 IDT 狀態機器會為您包含必要的狀態，而且您不需要採取其他動作。但是，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作為範例，在狀態機中新增所需的狀態。

如需 IDT CLI 指令的詳細資訊，請參閱偵錯並執行自訂測試套件。

寫入事件記錄

在測試運行時，您將數據發送stderr到stdout並將事件日誌和錯誤消息寫入控制台。如需有關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，資*<devicetester-extract-location>*/results/*<execution-id>*/logs料夾中的test_manager.log檔案也會提供此資訊。

您可以配置每個測試用例以將日誌從其測試運行（包括來自被測設備的日誌）寫入到*<group-id>*_*<test-id>*文件*<device-tester-extract-location>*/results/*execution-id*/logs夾中的文件。若要這麼做，請使用testData.logFilePath查詢從 IDT 內容擷取記錄檔的路

徑，在該路徑上建立檔案，然後將您想要的內容寫入。IDT 會根據正在執行的測試案例自動更新路徑。如果您選擇不為測試用例創建日誌文件，則不會為該測試用例生成任何文件。

您也可以設定文字執行檔，視需要在*<device-tester-extract-location>*/logs資料夾中建立其他記錄檔。建議您為記錄檔名稱指定唯一的前置詞，這樣您的檔案就不會被覆寫。

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。這些報告檔案位於中*<device-tester-extract-location>*/results/*<execution-id>*/。這兩個報告都會捕獲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 查看 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指令將測試結果報告給 IDT。如果 IDT 找不到測試結果，則會針對測試用例發出錯誤。以下 Python 摘錄顯示了將測試結果發送到 IDT 的命令：

```
request-variable = SendResultRequest(TestResult(result))
client.send_result(request-variable)
```

如果您未透過 API 報告結果，IDT 會在測試成品資料夾中尋找測試結果。此資料夾的路徑儲存在 IDT 前後關聯中的testData.testArtifactsPath欄位中。在此文件夾中，IDT 使用它定位的第一個按字母順序排序的 XML 文件作為測試結果。

如果您的測試邏輯產生 JUnit XML 結果，則可以將測試結果寫入成品文件夾中的 XML 文件，以直接將結果提供給 IDT，而不是解析結果，然後使用 API 將其提交給 IDT。

如果您使用此方法，請確保您的測試邏輯準確地總結測試結果，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，但下列例外：

- IDT 會忽略testsuites標籤的所有性質。相反地，它會從其他報告的測試群組結果中計算標籤屬性。
- 中必須至少有一個testsuite標籤存在testsuites。

由於 IDT 對所有測試用例使用相同的工件文件夾，並且在測試運行之間不會刪除結果文件，因此如果 IDT 讀取不正確的文件，此方法也可能導致錯誤的報告。我們建議您在所有測試用例中為生成的 XML 結果文件使用相同的名稱，以覆蓋每個測試用例的結果，並確保 IDT 可以使用正確的結果。儘管您可以使用混合方法在測試套件中進行報告，也就是說，對某些測試用例使用 XML 結果文件，並通過 API 為其他測試用例提交結果，但我們不建議使用此方法。

指定退出行為

將您的文字可執行檔設定為永遠以 0 結束代碼結束，即使測試案例報告失敗或錯誤結果也是如此。僅使用非零退出代碼來指示測試用例未運行，或者測試用例可執行文件無法向 IDT 傳達任何結果。當 IDT 收到非零退出代碼時，它標誌著測試用例遇到了阻止其運行的錯誤。

IDT 可能會要求或預期測試用例在下列事件中完成之前停止執行。使用此信息配置測試用例可執行文件，以檢測測試用例中的每個事件：

Timeout (逾時)

當測試用例運行時超過`test.json`文件中指定的超時值時發生。如果測試執行程式使用`timeout-multiplier`引數來指定逾時乘數，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。當測試運行程序啟動測試時，IDT 將 IDT_TEST_TIMEOUT 環境變量的值設置為計算的超時值（以秒為單位），並將該變量傳遞給測試用例可執行文件。您可以讀取變數值來設定適當的計時器。

中斷

當測試運行器中斷 IDT 時發生。例如，按下Ctrl+C。

由於終端機將信號傳播到所有子進程，因此您只需在測試用例中配置信號處理程序即可檢測中斷信號。

或者，您可以定期輪詢 API 以檢查 PollForNotifications API 回應中的`CancellationRequested`布林值。當 IDT 接收到中斷信號，它將`CancellationRequested`布爾值設置為`true`。

第一次失敗時停止

當與當前測試用例並行運 parallel 的測試用例失敗，並且測試運行程序使用`stop-on-first-failure`參數來指定 IDT 在遇到任何故障時應停止時發生。

若要偵測此事件，您可以定期輪詢 API，以檢查 PollForNotifications API 回應中的`CancellationRequested`布林值。當 IDT 遇到失敗並配置為在第一次失敗時停止時，它將`CancellationRequested`布林值設置為`true`。

當發生任何這些事件時，IDT 會等待 5 分鐘，讓任何當前正在運行的測試用例完成運行。如果所有正在運行的測試用例都未在 5 分鐘內退出，則 IDT 會強制其每個進程停止。如果 IDT 在進程結束之前沒有收到測試結果，則會將測試用例標記為已超時。作為最佳實踐，您應該確保您的測試用例遇到其中一個事件時執行以下操作：

1. 停止運行正常測試邏輯。
2. 清理所有臨時資源，例如被測設備上的測試工件。
3. 向 IDT 報告測試結果，例如測試失敗或錯誤。
4. 退出。

使用 IDT 上下文

當 IDT 運行測試套件時，測試套件可以訪問一組數據，這些數據可用於確定每個測試的運行方式。此資料稱為 IDT 內容。例如，測試運行程序在`userdata.json`文件中提供的用戶數據配置可用於 IDT 上下文中的測試套件。

IDT 上下文可以被視為只讀 JSON 文檔。測試套件可以使用標準 JSON 數據類型（如對象，數組，數字等）檢索數據並將數據寫入上下文。

上下文模式

IDT 前後關聯使用下列格式：

```
{
    "config": {
        <config-json-content>
        "timeoutMultiplier": timeout-multiplier,
        "idtRootPath": <path/to/IDT/root>
    },
    "device": {
        <device-json-device-element>
    },
    "devicePool": {
        <device-json-pool-element>
    },
    "resource": {
        "devices": [
            {
                <resource-json-device-element>
                "name": "<resource-name>"
            }
        ]
    },
    "testData": {
        "awsCredentials": {
```

```
            "awsAccessKeyId": "<access-key-id>",
            "awsSecretAccessKey": "<secret-access-key>",
            "awsSessionToken": "<session-token>"
        },
        "logFilePath": "/path/to/log/file"
    },
    "userData": {
        <userdata-json-content>
    }
}
```

config

來自config.json文件的信息。該字config段還包含以下附加字段：

config.timeoutMultiplier

測試套件使用的任何超時值的乘數。此值由 IDT CLI 的測試執行程式指定。預設值為 1。

config.idRootPath

此值是配置檔案時 IDT 絕對路徑值的預留位userdata.json置。這是由構建和閃存命令使用。

device

有關為測試運行選擇的設備的信息。此資訊相當於所選裝置的device.json檔案中的devices陣列元素。

devicePool

有關為測試運行選擇的設備池的信息。此資訊相當於在device.json檔案中為所選裝置集區定義的頂層裝置集區陣列元素。

resource

resource.json檔案中資源裝置的相關資訊。

resource.devices

這項資訊等同於resource.json檔案中定義的devices陣列。每個devices元素都包含下列其他欄位：

resource.device.name

資源裝置的名稱。此值會設定為test.json檔案中的requiredResource.name值。

testData.awsCredentials

測試用來連線至 AWS 雲端的 AWS 認證。此資訊是從config.json檔案中取得的。

testData.logFilePath

測試案例寫入記錄訊息的記錄檔路徑。如果測試套件不存在，則會創建此文件。

userData

由測試運行程序在userdata.json文件中提供的信息。

存取上下文中的資料

您可以使用配置文件中的 JSONPath 表示法以及使用和 API 的文本可執行文件查詢上下GetContextValue文。GetContextString用於訪問 IDT 上下文的 JSONPath 字符串的語法變化如下：

- 在suite.json和test.json，您使用{{*query*}}. 也就是說，請勿使用根元素$.來啟動運算式。
- 在中statemachine.json，您使用{{$.*query*}}.
- 在 API 命令中，您可以根據命令使用*query*或{{$.*query*}}。如需詳細資訊，請參閱 SDK 中的內嵌文件。

下表描述了一個典型的包括 JSONPath 表達式的運算符：

運算子	描述
$	根元素。由於 IDT 的頂層前後關聯值是物件，因此您通常會用$.來啟動查詢。
.childName	從物件存取具有名稱childName 的子元素。如果施加到一個數組，產生一個新的數組，並應用於每個元素這個運算符。元素名稱區分大小寫。例如，存取config物件中awsRegion 值的查詢為$.config.awsRegion 。
[start:end]	從數組過濾元素，檢索從索start引開始和上到索end引，都包括在內的項目。
[index1, index2, ... , indexN]	從數組過濾元素，僅從指定的索引中檢索項目。

運算子	描述
[?(expr)]	使用expr運算式篩選陣列中的元素。此運算式必須評估為布林值。

若要建立篩選運算式，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 JSON 路徑。
- value是任何使用標準 JSON 語法的自訂值。
- operator是下列其中一個運算子：
  - <(小於)
  - <=(小於或等於)
  - ==(等於)

    如果運算式中的 JSONPath 或值是陣列、布林值或物件值，則這是您可以使用的唯一受支援的二進位運算子。
  - >=(大於或等於)
  - >(大於)
  - =~（正則表達式匹配）。若要在篩選運算式中使用這個運算子，運算式左側的 JSONPath 或值必須評估為字串，而右側必須是遵循 RE 2 語法的模式值。

您可以在 {{*query}} #### JSONPath ##*作為檔案args和environmentVariables欄位內的預留位置字串，以及test.json檔案中的environmentVariables欄位內。suite.jsonIDT 會執行前後關聯查詢，並使用查詢的評估值填入欄位。例如，在suite.json文件中，您可以使用佔位符字符串來指定隨每個測試用例而變化的環境變量值，IDT 將為每個測試用例填充正確的值的環境變量。但是，當您在test.json和 suite.json file 中使用預留位置字串時，下列考量適用於您的查詢：

- 您必須在查詢中每次出現devicePool索引鍵的全部小寫。也就是說，使用devicepool代替。
- 對於數組，您只能使用字符串數組。此外，陣列使用非標準item1, item2,...,itemN格式。如果數組只包含一個元素，那麼它被序列化為item，使其與字符串字段無法區分。
- 您無法使用預留位置從前後關聯擷取物件。

由於這些考量因素，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。但是，在某些情況下，使用 JSONPath 佔位符來檢索要設置為環境變量的單個字符串可能會更方便。

設定測試執行程式的設定

要運行自定義測試套件，測試運行者必須根據要運行的測試套件配置其設置。根據*<device-tester-extract-location>*`/configs/`資料夾中的規劃檔案樣板來指定設定。如果需要，測試運行程序還必須設置 IDT 將用於連接到 AWS 雲的 AWS 憑據。

作為測試編寫者，您將需要配置這些文件來調試您的測試套件。您必須提供測試運行程序的說明，以便他們可以根據需要配置以下設置以運行測試套件。

設定 device.json

該`device.json`文件包含有關運行測試的設備的信息（例如，IP 地址，登錄信息，操作系統和 CPU 架構）。

測試運行者可以使用位於該文件*<device-tester-extract-location>*`/configs/`夾中的以下模板`device.json`文件提供此信息。

```
[
    {
        "id": "<pool-id>",
        "sku": "<pool-sku>",
        "features": [
            {
                "name": "<feature-name>",
                "value": "<feature-value>",
                "configs": [
                    {
                        "name": "<config-name>",
                        "value": "<config-value>"
                    }
                ],
            }
        ],
        "devices": [
            {
                "id": "<device-id>",
                "pairedResource": "<device-id>", //used for no-op protocol
                "connectivity": {
```

```
                    "protocol": "ssh | uart | docker | no-op",
                    // ssh
                    "ip": "<ip-address>",
                    "port": <port-number>,
                    "publicKeyPath": "<public-key-path>",
                    "auth": {
                        "method": "pki | password",
                        "credentials": {
                            "user": "<user-name>",
                            // pki
                            "privKeyPath": "/path/to/private/key",

                            // password
                            "password": "<password>",
                        }
                    },

                    // uart
                    "serialPort": "<serial-port>",

                    // docker
                    "containerId": "<container-id>",
                    "containerUser": "<container-user-name>",
                }
            }
        ]
    }
]
```

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**
>
> 如果您要在 AWS 合作夥伴裝置目錄中刊登您的主機板，您在此指定的 SKU 必須與您在刊登程序中使用的 SKU 相符。

`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。您必須向測試執行者提供有關要包含在`device.json`檔案中的功能名稱和值的資訊。例如，如果您要測試裝置作為其他裝置的 MQTT 伺服器運作，您可以設定測試邏輯，以驗證名為的功能的特定支援層級。`MQTT_QoS`測試運行者提供此功能名稱，並將功能值設置為其設備支持的 QoS 級別。您可以使用查詢從 IDT 前後關聯中擷取提供的資訊，或從具有`devicePool.features`查詢的狀態機器前後關聯中擷取提供的資訊。`pool.features`

`features.name`

特徵的名稱。

`features.value`

支援的特徵值。

`features.configs`

功能的組態設定 (如果需要)。

`features.config.name`

組態設定的名稱。

`features.config.value`

支援的設定值。

`devices`

池中要測試的一系列設備。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。

`devices.pairedResource`

資源裝置的使用者定義唯一識別碼。使用`no-op`連線通訊協定測試裝置時，需要此值。

connectivity.protocol

用來與此裝置通訊的通訊協定。池中的每個設備都必須使用相同的協議。

目前，唯一支援的值是ssh實體裝置、docker Docker 容器，以及no-op與 IDT 主機沒有直接連線但需要資源裝置做為實體中介軟體才能與主機通訊的裝置。uart

對於無作業裝置，您可以在devices.pairedResource中配置資源裝置 ID。您也必須在resource.json檔案中指定此 ID。配對的設備必須是與被測設備實際配對的設備。IDT 識別並連接到配對的資源裝置後，IDT 將不會根據檔案中描述的功能連線到其他資源裝置。test.json

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.publicKeyPath

選用。用於驗證與被測設備的連接的公鑰的完整路徑。當您指定時publicKeyPath，IDT 會在設備與被測設備建立 SSH 連接時驗證設備的公鑰。如果未指定此值，IDT 會建立 SSH 連線，但不會驗證裝置的公開金鑰。

強烈建議您指定公開金鑰的路徑，並使用安全方法來擷取此公開金鑰。對於基於命令行的標準 SSH 客戶端，公鑰在known_hosts文件中提供。如果您指定個別的公開金鑰檔案，則此檔案必須使用與known_hosts檔案相同的格式，也就是ip-address key-type public-key。

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。容器內的使用者對使用者的名稱。預設值是 Docker 檔案中提供的使用者。

預設值為 22。

只有當 connectivity.protocol 設為 docker 時，才會套用此屬性。

> **Note**
>
> 要檢查測試運行程序是否為測試配置了不正確的設備連接，您可以pool.Devices[0].Connectivity.Protocol從狀態機上下文中檢索並將其

與Choice狀態中的預期值進行比較。如果使用了不正確的協議，則使用LogMessage狀態和轉換到狀態打印消息。Fail
或者，您可以使用錯誤處理代碼來報告錯誤設備類型的測試失敗。

（可選）配置用戶數據.json

該userdata.json文件包含測試套件所需但未在device.json文件中指定的任何其他信息。這個文件的格式取決於在測試套userdata_scheme.json件中定義的文件。如果您是測試編寫者，請確保將此信息提供給將運行您編寫的測試套件的用戶。

(選擇性) 設定資源 .json

該resource.json文件包含有關將用作資源設備的任何設備的信息。資源設備是測試被測設備的某些功能所需的設備。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能成功連線。資源設備是可選的，您可以根據需要任意數量的資源設備。作為測試編寫者，您可以使用 test.json 文件來定義測試所需的資源設備功能。然後，測試運行程序使用該resource.json文件提供具有所需功能的資源設備池。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。

測試運行者可以使用位於該文件*<device-tester-extract-location>*/configs/夾中的以下模板resource.json文件提供此信息。

```
[
    {
        "id": "<pool-id>",
        "features": [
            {
                "name": "<feature-name>",
                "version": "<feature-value>",
                "jobSlots": <job-slots>
            }
        ],
        "devices": [
            {
                "id": "<device-id>",
                "connectivity": {
                    "protocol": "ssh | uart | docker",
                    // ssh
                    "ip": "<ip-address>",
                    "port": <port-number>,
                    "publicKeyPath": "<public-key-path>",
```

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

                            // password
                            "password": "<password>",
                        }
                    },

                    // uart
                    "serialPort": "<serial-port>",

                    // docker
                    "containerId": "<container-id>",
                    "containerUser": "<container-user-name>",
                }
            }
        ]
    }
]
```

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。此字段中所需的信息在測試套件的 test.json 文件中定義，並確定要運行哪些測試以及如何運行這些測試。如果測試套件不需要任何功能，則不需要此字段。

features.name

特徵的名稱。

features.version

功能版本。

features.jobSlots

設定以指出可同時使用裝置的測試數目。預設值為 1。

devices

池中要測試的一系列設備。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。池中的每個設備都必須使用相同的協議。

目前，唯一支援的值是ssh實uart體裝置和 docker Docker 容器。

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.publicKeyPath

選用。用於驗證與被測設備的連接的公鑰的完整路徑。當您指定時publicKeyPath，IDT 會在設備與被測設備建立 SSH 連接時驗證設備的公鑰。如果未指定此值，IDT 會建立 SSH 連線，但不會驗證裝置的公開金鑰。

強烈建議您指定公開金鑰的路徑，並使用安全方法來擷取此公開金鑰。對於基於命令行的標準 SSH 客戶端，公鑰在known_hosts文件中提供。如果您指定個別的公開金鑰檔案，則此檔案必須使用與known_hosts檔案相同的格式，也就是ip-address key-type public-key。

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。容器內的使用者對使用者的名稱。預設值是 Docker 檔案中提供的使用者。

預設值為 22。

只有當 connectivity.protocol 設為 docker 時，才會套用此屬性。

（可選）配置配置。JSON

config.json檔案包含 IDT 的組態資訊。通常，測試運行者不需要修改此文件，除非為 IDT 提供 AWS 用戶憑據以及可選 AWS 地區提供用戶憑據。如果提供具有必要權限的 AWS 認證，則 AWS IoT Device Tester 會收集使用狀況測量結果並提交給。 AWS這是一項選擇加入功能，用於改善 IDT 功能。如需詳細資訊，請參閱 IDT 使用量測量結果。

測試運行者可以通過以下方式之一配置其 AWS 憑據：

- 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 工作階段關閉後，無法使用在 SSH 工作階段期間定義的變數。IDT 可以使用AWS_ACCESS_KEY_ID和AWS_SECRET_ACCESS_KEY環境變數來儲存 AWS 認證

  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>
  ```

要配置 IDT 的 AWS 憑據，測試運行程序編輯文件*<device-tester-extract-location>*/configs/夾中config.json文件中的auth部分。

```
{
```

```
    "log": {
        "location": "logs"
    },
    "configFiles": {
        "root": "configs",
        "device": "configs/device.json"
    },
    "testPath": "tests",
    "reportPath": "results",
    "awsRegion": "<region>",
    "auth": {
        "method": "file | environment",
        "credentials": {
            "profile": "<profile-name>"
        }
    }
}
]
```

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 >* 來定義的。

log.location

< device-tester-extract-location > 中記錄檔資料夾的路徑。

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

包含測試套件之資料夾的路徑。

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。測試套件將使用的 AWS 區域。如果未設置，則測試套件將使用每個測試套件中指定的默認區域。

auth.method

IDT 用來擷取 AWS 認證的方法。支援的值是`file`從認證檔案擷取認證，`environment`以及使用環境變數擷取認證。

auth.credentials.profile

要從身份證明檔案使用的身份證明設定檔。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

設置所需的配置後，IDT 可以運行您的測試套件。完整測試套件的運行時取決於硬件和測試套件的組合。作為參考, 它需要大約 30 分鐘來完成完整的 FreeRTOS 資格測試套件上樹莓派 3B.

在編寫測試套件時，可以使用 IDT 在調試模式下運行測試套件，以在運行代碼之前檢查代碼或將其提供給測試運行程序。

在除錯模式下執行 IDT

由於測試套件依賴 IDT 與設備交互，提供上下文並接收結果，因此您無法在沒有任何 IDT 交互的情況下簡單地在 IDE 中調試測試套件。為此，IDT CLI 提供了可讓您在除錯模式下執行 IDT 的`debug-test-suite`命令。執行下列命令以檢視下列項目的可用選項`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 實際上並不會啟動測試套件或執行測試協調程式；相反，它會與 IDE 互動，以回應從 IDE 中執行的測試套件發出的要求，並將記錄檔列印到主控台。IDT 不會逾時並等待結束，直到手動中斷為止。在偵錯模式下，IDT 也不會執行測試協調程式，也不會產生任何報告檔案。要調試您的測試套件，您必須使用 IDE 來提供 IDT 通常從配置文件中獲取的一些信息。請務必提供下列資訊：

- 每個測試的環境變數和引數。IDT 不會從`test.json`或`suite.json`讀取此資訊。
- 用於選取資源裝置的引數。IDT 不會從`test.json`中讀取此信息。

若要偵錯測試套件，請完成以下步驟：

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`，請確保您根據需要配置所有這些。
2. 執行下列命令，將 IDT 置於偵錯模式，並選取執行測試所需的任何裝置。

   ```
   devicetester_[linux | mac | win_x86-64] debug-test-suite [options]
   ```

   運行此命令後，IDT 會等待來自測試套件的請求，然後對其進行響應。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 產生的環境變數的值。

   b. 設定您在`test.json`和`suite.json`檔案中指定的任何環境變數或引數的值。

   c. 視需要設定中斷點。
4. 在 IDE 中執行測試套件。

   您可以根據需要多次調試和重新運行測試套件。IDT 在除錯模式下不會逾時。
5. 完成除錯之後，請中斷 IDT 以結束偵錯模式。

用於運行測試的 IDT CLI 命令

下一節將說明 IDT CLI 指令：

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 (在此例中為 FreeRTOS 版本)，以及目前 IDT 版本可用的 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。 要運行的測試套件版本。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，以逗號分隔的清單形式。如果未指定，IDT 會執行測試套件中的所有測試群組。
- test-id。 要運行的測試用例，以逗號分隔的列表。指定時，group-id 必須指定單一群組。
- pool-id。 要測試的裝置集區。如果測試運行程序在您的device.json文件中定義了多個設備池，則必須指定一個池。
- timeout-multiplier。 設定 IDT 以修改test.json檔案中指定的測試執行逾時，使用使用者定義的乘數進行測試。
- stop-on-first-failure。 設定 IDT 以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。 設置包含運行測試套件所需的用戶數據信息的文件。只有在測試套件userdataRequired的suite.json文件中設置為 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

在調試模式下運行測試套件。如需詳細資訊，請參閱 在除錯模式下執行 IDT。

查看 IDT 測試結果和日誌

本節說明 IDT 產生主控台記錄檔和測試報告的格式。

主控台訊息格式

AWS IoT Device Tester 使用標準格式在啟動測試套件時將消息打印到控制台。以下摘錄顯示 IDT 產生的主控台訊息範例。

```
[INFO] [2000-01-02 03:04:05]: Using suite: MyTestSuite_1.0.0
 executionId=9a52f362-1227-11eb-86c9-8c8590419f30
```

大多數控制台消息包含以下字段：

time

記錄事件的完整 ISO 8601 時間戳記。

level

記錄事件的訊息層級。通常，記錄的消息級別是`info`、`warn`或之一`error`。如果 IDT 遇到導致其提前退出的預期事件，則會發出`fatal`或`panic`消息。

msg

記錄的消息。

executionId

目前 IDT 處理程序的唯一 ID 字串。此 ID 用於區分個別 IDT 執行。

從測試套件生成的控制台消息提供有關被測設備以及 IDT 運行的測試套件，測試組和測試用例的其他信息。以下摘錄顯示了從測試套件生成的控制台消息的示例。

```
[INFO] [2000-01-02 03:04:05]: Hello world! suiteId=MyTestSuitegroupId=myTestGroup
 testCaseId=myTestCase deviceId=my-
deviceexecutionId=9a52f362-1227-11eb-86c9-8c8590419f30
```

控制台消息的測試套件特定部分包含以下字段：

suiteId

當前正在運行的測試套件的名稱。

groupId

目前執行之測試群組的 ID。

testCaseId

當前正在運行的測試用例的 ID。

deviceId

當前測試用例正在使用的被測設備的 ID。

測試摘要包含有關測試套件、每個執行之群組的測試結果，以及產生的記錄檔和報告檔案位置的相關資訊。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 報表結構描

`awsiotdevicetester_report.xml`是包含下列資訊的已簽署報表：

- IDT 版本。
- 測試套件版本。
- 用於簽署報表的報表簽名和金鑰。
- 裝置 SKU 和`device.json`檔案中指定的裝置集區名稱。
- 已測試的產品版本和裝置功能。
- 測試結果的彙總摘要。此資訊與檔*`suite-name`*`_report.xml`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。標記為的功能required是測試套件驗證設備所需的。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>
```

測試套裝報表架構

*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

運行測試套件所花費的時間（以秒為單位）。

tests

執行的測試次數。

failures

已執行但未通過的測試次數。

errors

IDT 無法執行的測試次數。

disabled

此屬性未使用，可忽略。

如果測試發生失敗或錯誤，您可以檢閱 <testsuites> XML 標籤來識別失敗的測試。<testsuites> 標籤內的 <testsuite> XML 標籤會顯示測試群組的測試結果摘要。例如：

```
<testsuite name="combination" package="" tests="1" failures="0" time="161" disabled="0"
 errors="0" skipped="0">
```

其格式類似於 <testsuites> 標籤，但有不使用且可忽略的 skipped 屬性。在每個 <testsuite> XML 標籤內，測試群組每個執行的測試都有 <testcase> 標籤。例如：

```
<testcase classname="Security Test" name="IP Change Tests" attempts="1"></testcase>
```

<testcase>標籤中使用的屬性

name

測試的名稱。

attempts

IDT 執行測試案例的次數。

當測試案例失敗或發生錯誤時，系統就會將 <failure> 或 <error> 標籤新增至 <testcase> 標籤，其中附有相關資訊以利故障診斷。例如：

```
<testcase classname="mcu.Full_MQTT" name="MQTT_TestCase" attempts="1">
 <failure type="Failure">Reason for the test failure</failure>
 <error>Reason for the test execution error</error>
</testcase>
```

IDT 使用量測量結果

如果您提供具有必要權限的 AWS 認證，請 AWS IoT Device Tester 收集使用狀況測量結果並將 AWS 其提交給。這是一項選擇加入功能，用於改善 IDT 功能。IDT 會收集以下資訊：

- 用來執行 IDT 的 AWS 帳號識別碼
- 用於執行測試的 IDT CLI 命令
- 正在運行的測試套件
- *< device-tester-extract-location >* 文件夾中的測試套件
- 裝置集區中設定的裝置數目
- 測試用例名稱和運行時間
- 測試結果資訊，例如測試是否通過、失敗、遇到錯誤或被略過
- 產品功能測試
- IDT 退出行為，例如意外或提前退出

IDT 傳送的所有資訊也會記錄到資*<device-tester-extract-location>*/results/*<execution-id>*/料夾中的metrics.log檔案。您可以檢視記錄檔，以查看測試執行期間收集的資訊。只有當您選擇收集使用狀況測量結果時，才會產生此檔案。

若要停用測量結果收集，您不需要採取其他動作。只需不要存儲您的 AWS 憑據，如果您確實有存儲的 AWS 憑據，則不要配置`config.json`文件以訪問它們。

註冊一個 AWS 帳戶

如果您沒有 AWS 帳戶，請完成以下步驟來建立一個。

若要註冊成為 AWS 帳戶

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

   部分註冊程序需接收來電，並在電話鍵盤輸入驗證碼。

   當您註冊時 AWS 帳戶，會建立AWS 帳戶根使用者一個。根使用者有權存取該帳戶中的所有 AWS 服務 和資源。安全性最佳做法是將管理存取權指派給使用者，並僅使用 root 使用者來執行需要 root 使用者存取權的工作。

AWS 註冊過程完成後，會向您發送確認電子郵件。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。

建立具有管理權限的使用者

註冊後，請保護您的 AWS 帳戶 AWS 帳戶根使用者 AWS IAM Identity Center、啟用和建立系統管理使用者，這樣您就不會將 root 使用者用於日常工作。

保護您的 AWS 帳戶根使用者

1. 選擇 Root 使用者並輸入您的 AWS 帳戶 電子郵件地址，以帳戶擁有者身分登入。AWS Management Console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)。

   如需指示，請參閱《IAM 使用者指南》中的為 AWS 帳戶 根使用者啟用虛擬 MFA 裝置 (主控台)。

建立具有管理權限的使用者

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 身分中心中，將管理存取權授予使用者。

   若要取得有關使用 IAM Identity Center 目錄 做為身分識別來源的自學課程，請參閱《使用指南》IAM Identity Center 目錄中的「以預設值設定使用AWS IAM Identity Center 者存取」。

以具有管理權限的使用者身分登入

- 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 身分中心使用者登入的說明，請參閱使用AWS 登入 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 身分中心中，建立遵循套用最低權限許可的最佳做法的權限集。

   如需指示，請參閱《AWS IAM Identity Center 使用指南》中的「建立權限集」。
2. 將使用者指派給群組，然後將單一登入存取權指派給群組。

   如需指示，請參閱《AWS IAM Identity Center 使用指南》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：

- 使用者和群組位於 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。請按照 IAM 使用者指南 的 為第三方身分提供者 (聯合) 建立角色 中的指示進行操作。
- 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 中的指示進行操作。
  - (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。

向 IDT 提供 AWS 憑證

若要允許 IDT 存取您的 AWS 認證並提交量度 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. 若要使用認證檔案，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。如需更多詳細資訊，請參閱 （可選）配置配置。JSON。

AWS IoT Device Tester適用於免費伺服器測試套件版本

FreeRTOS 的 IDT 將測試資源組織到測試套件和測試群組中：

- 測試套件是一組測試群組，用來驗證裝置是否可與特定版本的 FreeRTOS 搭配使用。
- 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。

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，請參閱免費服務的AWS IoT Device Tester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。

您也可以執行命list-supported-versions令來列出目前 IDT 版本所支援的 FreeRTOS 和測試套件版本。

新測試可能會引入新的 IDT 組態設定。如果這些是選用設定，IDT 會通知您並繼續執行測試。如果這些是必要設定，IDT 會通知您並停止執行。完成設定後，您可以繼續執行測試。

疑難排解

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 FreeRTOS 控制台輸出的 IDT 來查找失敗的測試用例的執行 ID，測試用例 ID 和測試組 ID，然後打開名為的測試用例的日誌文件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。
- 更詳細的 IDT，用於 FreeRTOS 控制台輸出。

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」*###/##*沒有授權訪問此資源」，請確保您按照中的指定配置權限建立和設定 AWS 帳號。
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. 查看下面的日誌文件/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。

- 對於 FRQ 1.x.x，請確定可以使用 FreeRTOS 原始程式碼的完整複製版本。免費伺服器版本會根據 FreeRTOS 版本進行標記。若要克隆特定版本的代碼，請使用以下命令：

```
git clone --branch version-number https://github.com/aws/amazon-freertos.git
cd amazon-freertos
git submodule update --checkout --init --recursive
```

故障診斷裝置組態

當您將 IDT 用於 FreeRTOS 時，您必須先取得正確的組態檔，才能執行二進位檔案。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 針對 FreeRTOS 產生的錯誤碼：

錯誤代碼	錯誤代碼名稱	可能的根本原因	疑難排解
201	InvalidInputError	device.json 、config.json 、或 userdata.json 中的欄位遺失或格式不正確。	請確定列出的檔案中未遺漏必要欄位且為必要格式。如需詳細資訊，請參閱首次準備測試微型控制器主機板。

錯誤代碼	錯誤代碼名稱	可能的根本原因	疑難排解
202	ValidationError	`device.json` 、`config.json` 、或 `userdata.json` 中的欄位包含無效的值。	請查看報告中錯誤代碼右手方的錯誤訊息： • 無效AWS區域-指定有效AWS您的區域`config.json` 文件。有關更多信息AWS區域，請參閱區域和端點。 • 無效AWS認證-設定有效AWS測試機器上的認證（通過環境變量或認證文件）。確認身分驗證欄位設定正確。如需詳細資訊，請參閱建立和設定 AWS 帳號。

錯誤代碼	錯誤代碼名稱	可能的根本原因	疑難排解
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。 • 刪除`buildFreeRTOS`原始程式碼目錄下的資料夾 (如果存在的話)。如需詳細資訊，請參閱設定建置、刷新和測試設定。 • Windows 對於檔案路徑名稱有一個字元限制。較長的檔案路徑名稱會導致錯誤。

錯誤代碼	錯誤代碼名稱	可能的根本原因	疑難排解
204	BuildSourceError	無法編譯免費伺服器的原始程式碼。	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。 • 例如，如果您已將 FreeRTOS 的 IDT 解壓縮到系統上包含空格的檔案路徑`C:\Users\My Name\Desktop\`，您可能需要在構建命令中額外的引號，以確保正確解析路徑。你的 Flash 命令可能需要相同的東西。

錯誤代碼	錯誤代碼名稱	可能的根本原因	疑難排解
205	FlashOrRunTestError	IDT 免費伺服器無法在您的 DUT 上閃存或執行 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 自由伺服器無法啟動回應伺服器WiFi或安全套接字測試。	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。

調試配置文件解析錯誤

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。免費伺服器的 IDT 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。

偵錯測試結果剖析錯誤

從運行測試組時自由圖書館集成測試，例如FullTransportInterfaceTLS, 全核心, 全板 CS11 核心, 全板上網, 全板 CS11_PreProvisioned_ ECC, 全包 11 _PreProvisionedRSA，或奧塔科，FreeRTOS 的 IDT 會透過序列連線剖析測試裝置的測試結果。有時，設備上的額外串行輸出可能會干擾測試結果的分析。

在上述情況下，輸出奇怪的測試用例失敗原因，例如源自不相關設備輸出的字符串。FreeRTOS 測試用例記錄檔的 IDT (包括測試期間接收到的 FreeRTOS 的所有序列輸出 IDT) 可能會顯示下列內容：

```
<unrelated device output>
TEST(Full_PKCS11_Capabilities, PKCS11_Capabilities)<unrelated device output>
```

```
<unrelated device output>
 PASS
```

在上面的例子中，不相關的設備輸出會阻止 FreeRTOS 的 IDT 檢測測試結果通過。

檢查以下內容以確保最佳測試。

- 確保設備上使用的日誌宏是線程安全的。請參閱實現庫日誌宏以取得更多資訊。
- 在測試期間，請確保串行連接的輸出最小。即使您的記錄宏正確地是線程安全的，其他設備輸出也可能會出現問題，因為測試期間測試結果將在單獨的調用中輸出。

FreeRTOS 測試用例日誌的 IDT 理想情況下會顯示不間斷的測試結果輸出，如下所示：

```
---------STARTING TESTS---------
TEST(Full_OTA_PAL, otaPal_CloseFile_ValidSignature) PASS
TEST(Full_OTA_PAL, otaPal_CloseFile_InvalidSignatureBlockWritten) PASS
-----------------------
2 Tests 0 Failures 0 Ignored
```

偵錯完整性檢查失敗

如果使用 FRQ 1.x.x 版的 FreeRTOS，則適用下列完整性檢查。

當您執行 FreeRTosIntegrity 測試群組且您遇到失敗時，請先確定您尚未修改任何*freertos*目錄檔案。如果您還沒有，並且仍然看到問題，請確保您使用的是正確的分支。如果你運行 IDT 的`list-supported-products`命令，你可以找到哪個標記分支*freertos*你應該使用的回購。

如果您克隆了正確的標記分支`freertos`回購，仍然有問題，請確保你也運行`submodule update`指令。的複製工作流程`freertos`回購如下。

```
git clone --branch version-number https://github.com/aws/amazon-freertos.git
cd amazon-freertos
git submodule update --checkout —init —recursive
```

完整性檢查器查找的文件列表位於`checksums.json`在您的文件*freertos*目錄。要在不修改文件和文件夾結構的情況下限定 FreeRTOS 端口，請確保沒有列在 '`exhaustive`'和'`minimal`'部

分checksums.json檔案已被修改。要使用配置的 SDK 運行，請驗證 '下沒有任何文件minimal'區段已修改。

如果您使用 SDK 運行 IDT 並修改了您的某些文件*freertos*目錄，然後確保您在您的 SDK 中正確配置userdata文件。否則，完整性檢查程式將驗證*freertos*目錄。

除錯FullWiFi測試群組失敗

如果您正在使用 FRQ 1.x.x，並在FullWiFi測試組，以及」AFQP_WiFiConnectMultipleAP「測試失敗，這可能是因為兩個接入點與運行 IDT 的主機不在同一子網中。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中。

偵錯必要參數遺漏錯誤

由於 FreeRTOS 的 IDT 新增了新功能，因此可能會導入組態檔的變更。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。FreeRTOS 的 IDT 會驗證您的 JSON 組態檔結構描述，以確保已使用最新的受支援版本。

調試「測試無法啟動」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。

調試「無法找到測試結果的開始」錯誤

當 IDT 嘗試解析被測設備輸出的結果時，您可能會看到錯誤。有幾種可能的原因，因此請檢查以下幾個方面的正確性：

- 確保被測設備與主機具有穩定的連接。您可以檢查日誌文件以查看顯示這些錯誤的測試，以查看 IDT 正在接收的內容。
- 如果使用 FRQ 1.x.x，且待測裝置是透過緩慢的網路或其他介面連接，或是您在 FreeRTOS 測試群組記錄中看不到「----------」旗標以及其他 Freertos 測試群組輸出，您可以嘗試增加testStartDelayms在您的用戶數據配置中。如需詳細資訊，請參閱設定建置、刷新和測試設定。

偵錯「測試失敗：預期 __ 個結果，但看到 ___」錯誤

在測試期間，您可能會看到指向測試失敗的錯誤。測試需要一定數量的結果，並且在測試過程中看不到它。某些 FreeRTOS 測試會在 IDT 看到裝置的輸出之前執行。如果看到此錯誤，則可以嘗試增加`testStartDelayms`在您的用戶數據配置。如需詳細資訊，請參閱設定建置、刷新和測試設定。

調試「_____ 由於未選擇ConditionalTests約束」錯誤

這意味著您正在與測試不兼容的設備池上運行測試。OTA E2E 測試可能會發生這種情況。例如，在執行`OTADataplaneMQTT`測試組並在您的`device.json`配置文件，您選擇了 OTA 作為沒有或者`OTADataPlaneProtocol`如HTTP。選擇執行的測試群組必須符合您的`device.json`能力選取。

在設備輸出監視期間調試 IDT 超時

IDT 可能會由於多種原因而超時。如果在測試的設備輸出監視階段發生超時，並且您可以在 IDT 測試用例日誌中看到結果，則表示 IDT 對結果進行了錯誤的解析。其中一個原因可能是測試結果中間的交錯日誌消息。如果是這種情況，請參閱自由移植指南有關如何設置 UNITY 日誌的更多詳細信息。

設備輸出監視期間超時的另一個原因可能是在單個 TLS 測試用例失敗後重新啟動設備。然後，設備運行刷新的映像並導致在日誌中看到的無限循環。如果發生這種情況，請確保您的設備在測試失敗後不會重新啟動。

偵錯「未授權存取資源」錯誤

您可能會看到錯誤」*###/##*未授權存取此資源「在終端輸出中或`test_manager.log`下的檔案`/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，請參閱建立和設定 AWS 帳號。

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。如果由於超時或無法使用的連接而遇到錯誤WiFi或安全通訊端測試，請確定您的網路設定為允許在 1024-49151 範圍內設定的連接埠流量。

根據預設，安全通訊端測試會使用連接埠 33333 和 33334。該WiFi在預設情況下，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 測試用例在執行 OTA 後由於設備上存在相同版本而失敗，則可能是由於您的構建系統（例如 cmake）沒有註意到 IDT 對 FreeRTOS 源代碼的更改，而不是構建更新的二進製文件。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。

支援的平台

IDT 支援 Linux、macOS 和 Windows，這三個平台對與其連接的序列裝置有不同的命名機制：

- Linux：`/dev/tty*`
- macOS：`/dev/tty.*` 或 `/dev/cu.*`
- Windows：COM*

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

FreeRTOS 的 IDT 會使用個別的裝置組建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 針對 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。在視窗上，您可以使用諸如TeraTerm。

`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 無法構建或閃存到您的設備。
- 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

免費伺服器記錄的 IDT 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

何時AWS IoT Device Tester正在執行時，會向主控台報告失敗，並附上簡短訊息。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 FreeRTOS 主控台的 IDT 輸出來尋找失敗的測試案例的執行識別碼、測試案例識別碼和測試群組識別碼。然後使用此信息查找並打開名為的測試用例的日誌文件`results/execution-id/logs/test_group_id__test_case_id.log`此文件中的信息包括完整的構建和 flash 命令輸出，測試執行輸出以及更詳細AWS IoT Device Tester控制台輸出。

S3 儲存貯體問題

如果你按CTRL+C在運行 IDT 時，IDT 將啟動清理過程。清理的一部分是移除作為 IDT 測試一部分而建立的 Amazon S3 資源。如果清理無法完成，您可能會遇到已建立太多 Amazon S3 儲存貯體的問題。這意味著下次運行 IDT 時，測試將開始失敗。

如果你按CTRL+C要停止 IDT，您必須讓它完成清理過程以避免此問題。您也可以從您的帳戶中刪除手動建立的 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
```

流動網絡功能和AWS收費

當`Cellular`功能設定為`Yes`在您的`device.JSON`文件，FullSecureSockets將使用 t.micro EC2 實例進行測試，這可能會給您帶來額外費用AWS帳戶。如需詳細資訊，請參閱 Amazon EC2 定價。

資格報告生成方針

資格報告僅由AWS IoT Device Tester(IDT) 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 Support。

AWS受管理的政策AWS IoT Device Tester

AWS 受管政策是由 AWS 建立和管理的獨立政策。AWS 受管政策的設計在於為許多常見使用案例提供許可，如此您就可以開始將許可指派給使用者、群組和角色。

請記住，AWS 受管政策可能不會授與您特定使用案例的最低權限許可，因為它們可供所有 AWS 客戶使用。我們建議您定義使用案例專屬的客戶管理政策，以便進一步減少許可。

您無法更改 AWS 受管政策中定義的許可。如果 AWS 更新 AWS 受管政策中定義的許可，更新會影響政策連接的所有主體身分 (使用者、群組和角色)。在推出新的 AWS 服務 或有新的 API 操作可供現有服務使用時，AWS 很可能會更新 AWS 受管政策。

如需詳細資訊，請參閱《IAM 使用者指南》https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies_managed-vs-inline.html#aws-managed-policies中的 AWS 受管政策。

主題

- AWS受管理的策略：AWS物聯網DeviceTesterForFreeRTOSFullAccess
- AWS 受管政策的 AWS IoT Device Tester 更新項目

AWS受管理的策略：AWS物聯網DeviceTesterForFreeRTOSFullAccess

該AWSIoTDeviceTesterForFreeRTOSFullAccess受管理策略包含以下內容AWS IoT Device Tester版本檢查、自動更新功能和指標集合的權限。

權限詳情

此政策包含以下許可：

- `iot-device-tester:SupportedVersion`

  補助金AWS IoT Device Tester獲取支持產品，測試套件和 IDT 版本列表的權限。
- `iot-device-tester:LatestIdt`

  補助金AWS IoT Device Tester獲取可供下載的最新 IDT 版本的權限。

- `iot-device-tester:CheckVersion`

  補助金AWS IoT Device Tester檢查 IDT，測試套件和產品的版本兼容性的權限。

- `iot-device-tester:DownloadTestSuite`

  補助金AWS IoT Device Tester下載測試套件更新的權限。

- `iot-device-tester:SendMetrics`

  補助金AWS收集指標的權限AWS IoT Device Tester內部使用。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::*:role/idt-*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "iot.amazonaws.com"
                }
            }
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "iot:DeleteThing",
                "iot:AttachThingPrincipal",
                "iot:DeleteCertificate",
                "iot:GetRegistrationCode",
                "iot:CreatePolicy",
                "iot:UpdateCACertificate",
                "s3:ListBucket",
                "iot:DescribeEndpoint",
                "iot:CreateOTAUpdate",
                "iot:CreateStream",
                "signer:ListSigningJobs",
                "acm:ListCertificates",
                "iot:CreateKeysAndCertificate",
```

```
                "iot:UpdateCertificate",
                "iot:CreateCertificateFromCsr",
                "iot:DetachThingPrincipal",
                "iot:RegisterCACertificate",
                "iot:CreateThing",
                "iam:ListRoles",
                "iot:RegisterCertificate",
                "iot:DeleteCACertificate",
                "signer:PutSigningProfile",
                "s3:ListAllMyBuckets",
                "signer:ListSigningPlatforms",
                "iot-device-tester:SendMetrics",
                "iot-device-tester:SupportedVersion",
                "iot-device-tester:LatestIdt",
                "iot-device-tester:CheckVersion",
                "iot-device-tester:DownloadTestSuite"
            ],
            "Resource": "*"
        },
        {
            "Sid": "VisualEditor2",
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "signer:StartSigningJob",
                "acm:GetCertificate",
                "signer:DescribeSigningJob",
                "s3:CreateBucket",
                "execute-api:Invoke",
                "s3:DeleteBucket",
                "s3:PutBucketVersioning",
                "signer:CancelSigningProfile"
            ],
            "Resource": [
                "arn:aws:execute-api:us-east-1:098862408343:9xpmnvs5h4/prod/POST/
metrics",
                "arn:aws:signer:*:*:/signing-profiles/*",
                "arn:aws:signer:*:*:/signing-jobs/*",
                "arn:aws:iam::*:role/idt-*",
                "arn:aws:acm:*:*:certificate/*",
                "arn:aws:s3:::idt-*",
                "arn:aws:s3:::afr-ota*"
            ]
        },
```

```
        {
            "Sid": "VisualEditor3",
            "Effect": "Allow",
            "Action": [
                "iot:DeleteStream",
                "iot:DeleteCertificate",
                "iot:AttachPolicy",
                "iot:DetachPolicy",
                "iot:DeletePolicy",
                "s3:ListBucketVersions",
                "iot:UpdateCertificate",
                "iot:GetOTAUpdate",
                "iot:DeleteOTAUpdate",
                "iot:DescribeJobExecution"
            ],
            "Resource": [
                "arn:aws:s3:::afr-ota*",
                "arn:aws:iot:*:*:thinggroup/idt*",
                "arn:aws:iam::*:role/idt-*"
            ]
        },
        {
            "Sid": "VisualEditor4",
            "Effect": "Allow",
            "Action": [
                "iot:DeleteCertificate",
                "iot:AttachPolicy",
                "iot:DetachPolicy",
                "s3:DeleteObjectVersion",
                "iot:DeleteOTAUpdate",
                "s3:PutObject",
                "s3:GetObject",
                "iot:DeleteStream",
                "iot:DeletePolicy",
                "s3:DeleteObject",
                "iot:UpdateCertificate",
                "iot:GetOTAUpdate",
                "s3:GetObjectVersion",
                "iot:DescribeJobExecution"
            ],
            "Resource": [
                "arn:aws:s3:::afr-ota*/*",
                "arn:aws:s3:::idt-*/*",
                "arn:aws:iot:*:*:policy/idt*",
```

```
                "arn:aws:iam::*:role/idt-*",
                "arn:aws:iot:*:*:otaupdate/idt*",
                "arn:aws:iot:*:*:thing/idt*",
                "arn:aws:iot:*:*:cert/*",
                "arn:aws:iot:*:*:job/*",
                "arn:aws:iot:*:*:stream/*"
            ]
        },
        {
            "Sid": "VisualEditor5",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::afr-ota*/*",
                "arn:aws:s3:::idt-*/*"
            ]
        },
        {
            "Sid": "VisualEditor6",
            "Effect": "Allow",
            "Action": [
                "iot:CancelJobExecution"
            ],
            "Resource": [
                "arn:aws:iot:*:*:job/*",
                "arn:aws:iot:*:*:thing/idt*"
            ]
        },
        {
            "Sid": "VisualEditor7",
            "Effect": "Allow",
            "Action": [
                "ec2:TerminateInstances"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:instance/*"
            ],
            "Condition": {
                "StringEquals": {
                    "ec2:ResourceTag/Owner": "IoTDeviceTester"
                }
```

```
            }
        },
        {
            "Sid": "VisualEditor8",
            "Effect": "Allow",
            "Action": [
                "ec2:AuthorizeSecurityGroupIngress",
                "ec2:DeleteSecurityGroup"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:security-group/*"
            ],
            "Condition": {
                "StringEquals": {
                    "ec2:ResourceTag/Owner": "IoTDeviceTester"
                }
            }
        },
        {
            "Sid": "VisualEditor9",
            "Effect": "Allow",
            "Action": [
                "ec2:RunInstances"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:instance/*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/Owner": "IoTDeviceTester"
                }
            }
        },
        {
            "Sid": "VisualEditor10",
            "Effect": "Allow",
            "Action": [
                "ec2:RunInstances"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:image/*",
                "arn:aws:ec2:*:*:security-group/*",
                "arn:aws:ec2:*:*:volume/*",
                "arn:aws:ec2:*:*:key-pair/*",
```

```
                "arn:aws:ec2:*:*:placement-group/*",
                "arn:aws:ec2:*:*:snapshot/*",
                "arn:aws:ec2:*:*:network-interface/*",
                "arn:aws:ec2:*:*:subnet/*"
            ]
        },
        {
            "Sid": "VisualEditor11",
            "Effect": "Allow",
            "Action": [
                "ec2:CreateSecurityGroup"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:security-group/*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/Owner": "IoTDeviceTester"
                }
            }
        },
        {
            "Sid": "VisualEditor12",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeInstances",
                "ec2:DescribeSecurityGroups",
                "ssm:DescribeParameters",
                "ssm:GetParameters"
            ],
            "Resource": "*"
        },
        {
            "Sid": "VisualEditor13",
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:security-group/*",
                "arn:aws:ec2:*:*:instance/*"
            ],
            "Condition": {
                "ForAnyValue:StringEquals": {
```

```
                    "aws:TagKeys": [
                        "Owner"
                    ]
                },
                "StringEquals": {
                    "ec2:CreateAction": [
                        "RunInstances",
                        "CreateSecurityGroup"
                    ]
                }
            }
        }
    ]
}
```

AWS 受管政策的 AWS IoT Device Tester 更新項目

您可以檢視更新的詳細資訊AWS受管理的政策AWS IoT Device Tester從這項服務開始追蹤這些變更的時間。

版本	變更	描述	日期
7 (最新)	重新架構`ec2:CreateTags` 條件。	移除使用`ForAnyValues` 。	6/14/2023
6	已移除`freertos:ListHardwarePlatforms` 從政策。	自 2023 年 3 月 1 日起，由於此動作移除權限已淘汰。	6/2/2023
5	添加了使用 EC2 運行 echo 服務器測試的權限。	這是為了啟動和停止客戶中的 EC2 實例AWS帳戶。	12/15/2020
4	已新增`iot:CancelJobExecution` 。	此權限取消 OTA 工作。	7/17/2020
3	已新增下列權限：	• `iot-device-tester:`	3/23/2020

版本	變更	描述	日期
	• iot-device-tester: DownloadTestSuite， • iot-device-tester: CheckVersion， • iot-device-tester: LatestIdt， • iot-device-tester: SupportedVersion。	DownloadTestSuite — 補助金AWS IoT Device Tester下載測試套件更新的權限， • iot-device-tester: CheckVersion — 補助金AWS IoT Device Tester允許檢查 IDT，測試套件和產品的版本兼容性， • iot-device-tester: LatestIdt — 補助金AWS IoT Device Tester獲取可供下載的最新 IDT 版本的權限， • iot-device-tester: SupportedVersion — 補助金AWS IoT Device Tester獲取支持產品，測試套件和 IDT 版本列表的權限。	
2	已新增iot-device-tester: SendMetrics 權限。	補助金AWS收集指標的權限AWS IoT Device Tester內部使用。	2/18/2020

版本	變更	描述	日期
1	初始版本。		2/12/2020

免費服務的AWS IoT Device Tester支援政策

> **Important**
>
> AWS IoT自 2022 年 10 月起，AWS IoT Device Tester自由式資格 (FRQ) 1.0 不會產生已簽署的資格報告。您無法透過使用 IDT FRQ 1.0 版本的裝置資格認證計劃，將新 AWS IoT FreeRTOS AWS 裝置列入AWS合作夥伴裝置目錄的資格。雖然您無法使用 IDT FRQ 1.0 來限定 FreeRTOS 裝置的資格，但您可以繼續使用 FRQ 1.0 來測試您的 FreeRTOS 裝置。我們建議您使用 IDT FRQ 2.0 來限定和列出合作夥伴裝置目錄中的 FreeRTOS 裝置。AWS

AWS IoT Device TesterFreeRTOS 是一種測試自動化工具，用於驗證 FreeRTOS 連接埠到裝置。此外，您還可以限定 FreeRTOS 裝置的資格，並將它們列在AWS合作夥伴裝置目錄中。FreeRTOS 支援在 FreeRTOS /免費托斯LTS 上提供的免費伺服器長期支援 (LTS) 程式庫的驗證和限定，以及 FreeRTOS 主線可GitHub在自由軟體/免費伺服器上使用。AWS IoT Device Tester我們建議您使用 FreeRTOS 和 FreeRTOS 的最新版本來驗證和AWS IoT Device Tester限定您的裝置。

對於免費托斯 LTS，IDT 支援免費托斯 202210 LTS 版本的驗證和認證。有關 FreeRTOS LTS 發行版本及其維護時間表的更多資訊，請參閱此處。這些 LTS 版本的支持期結束後，您仍然可以繼續驗證，但 IDT 不會生成報告，這將允許您提交設備以獲得資格。

對於 FreeRTOS 上提供的主線 Fre eRTOS，我們支持過去六個月內發行的所有版本的驗證和認證，或者如果相隔超過六個月，則支持 Fre eRTOS 的前兩個版本的驗證和認證。請參閱此處瞭解目前支援的版本。對於不受支援的 FreeRTOS 版本，您仍然可以繼續驗證，但 IDT 不會產生報告，讓您能夠提交裝置以取得資格。

如需支援支援的版本AWS IoT Device Tester對於 FreeRTOS的最新 IDT 和 FreeRTOS 版本，請參閱。您可以使用任何支援的版本AWS IoT Device Tester搭配對應的 FreeRTOS 版本來測試或限定您的裝置。如果您繼續使用FreeRTOS 不支援的 IDT 版本，將不會收到最新的錯誤修正或更新。

如有關支援政策的問題，請聯絡AWS客戶支援。

中的安全性 AWS

雲安全 AWS 是最高的優先級。身為 AWS 客戶，您可以從資料中心和網路架構中獲益，該架構專為滿足對安全性最敏感的組織的需求而打造。

安全是 AWS 與您之間共同承擔的責任。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 雲端中執行 AWS 服務的基礎架構。 AWS 還為您提供可以安全使用的服務。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。若要瞭解適用於某項 AWS 服務的規範遵循計劃，請參閱合規計劃範圍內的AWS 服務。
- 雲端中的安全性 — 您的責任取決於您使用的 AWS 服務。您也必須對資料敏感度、組織要求，以及適用法律和法規等其他因素負責。

本文檔將幫助您了解如何在使用時應用共同責任模型 AWS。下列主題說明如何設定 AWS 以符合安全性與合規性目標。您還將學習如何使用可以幫助您監控和保護 AWS 資源的 AWS 服務。

如需有關 AWS IoT 安全性的詳細資訊，請參閱 AWS IoT.

主題

- FreeRTOS 的 Identity and Access Management
- 法規遵循驗證
- 韌性 AWS
- FreeRTOS 的基礎架構安全性

FreeRTOS 的 Identity and Access Management

AWS Identity and Access Management (IAM) 可協助管理員安全地控制 AWS 資源存取權。 AWS 服務 IAM 管理員控制哪些人可以通過身份驗證 (登入) 和授權 (具有權限) 來使用 FreeRTOS 資源。IAM 是您可以使用的 AWS 服務 ，無需額外付費。

主題

- 物件
- 使用身分驗證
- 使用政策管理存取權

- FreeRTOS 務如何與 IAM 搭配使用
- FreeRTOS 的身分識別原則範例
- 疑難排解 FreeRTOS 身分識別與存取

物件

您使用 AWS Identity and Access Management (IAM) 的方式會有所不同，這取決於您在 FreeRTOS 中執行的工作。

服務使用者 — 如果您使用 FreeRTOS 服務執行工作，則管理員會為您提供所需的認證和權限。當您使用更多 FreeRTOS 功能來完成工作時，您可能需要額外的權限。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。如果您無法在 FreeRTOS 中存取某項功能，請參閱。疑難排解 FreeRTOS 身分識別與存取

服務管理員 — 如果您負責公司的 FreeRTOS 資源，您可能擁有 FreeRTOS 的完整存取權。決定您的服務使用者應該存取哪些 FreeRTOS 功能和資源是您的工作。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。若要進一步瞭解貴公司如何將 IAM 與 FreeRTOS 搭配使用，請參閱。FreeRTOS 務如何與 IAM 搭配使用

IAM 管理員 — 如果您是 IAM 管理員，您可能想要瞭解如何撰寫政策來管理 FreeRTOS 存取權的詳細資訊。若要檢視可在 IAM 中使用的 FreeRTOS 身分型政策範例，請參閱。FreeRTOS 的身分識別原則範例

使用身分驗證

驗證是您 AWS 使用身分認證登入的方式。您必須以 IAM 使用者身分或假設 IAM 角色進行驗證 (登入 AWS)。 AWS 帳戶根使用者

您可以使用透過 AWS 身分識別來源提供的認證，以聯合身分識別身分登入。 AWS IAM Identity Center (IAM 身分中心) 使用者、貴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識別的範例。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 AWS 用同盟存取時，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。如需有關登入的詳細資訊 AWS，請參閱《AWS 登入 使用指南》 AWS 帳戶中的如何登入您的。

如果您 AWS 以程式設計方式存取，請 AWS 提供軟體開發套件 (SDK) 和命令列介面 (CLI)，以使用您的認證以加密方式簽署您的要求。如果您不使用 AWS 工具，則必須自行簽署要求。如需使用建議的方法自行簽署請求的詳細資訊，請參閱 IAM 使用者指南中的簽署 AWS API 請求。

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 建議您使用多重要素驗證 (MFA) 來增加帳戶的安全性。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 使用者指南中的在 AWS中使用多重要素驗證 (MFA)。

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，該身分可完整存取該帳戶中的所有資源 AWS 服務和資源。此身分稱為 AWS 帳戶 root 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

最佳作法是要求人類使用者 (包括需要系統管理員存取權的使用者) 使用與身分識別提供者的同盟，才能使用臨時認證 AWS 服務 來存取。

聯合身分識別是來自企業使用者目錄的使用者、Web 身分識別提供者、Identi ty Center 目錄，或使用透過身分識別來源提供的認證進行存取 AWS 服務 的任何使用者。 AWS Directory Service同盟身分存取時 AWS 帳戶，他們會假設角色，而角色則提供臨時認證。

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，也可以連線並同步到自己身分識別來源中的一組使用者和群組，以便在所有應用程式 AWS 帳戶 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 使用者是您內部的身分，具 AWS 帳戶 有單一人員或應用程式的特定許可。建議您盡可能依賴暫時憑證，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，但角色僅提供暫時憑證。如需進一步了解，請參閱IAM 使用者指南中的建立 IAM 使用者 (而非角色) 的時機。

IAM 角色

IAM 角色是您 AWS 帳戶 內部具有特定許可的身分。它類似 IAM 使用者，但不與特定的人員相關聯。您可以切換角色，在中暫時擔任 IAM 角色。 AWS Management Console 您可以透過呼叫 AWS CLI 或 AWS API 作業或使用自訂 URL 來擔任角色。如需使用角色的方法詳細資訊，請參閱 IAM 使用者指南中的使用 IAM 角色。

使用暫時憑證的 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。但是，對於某些策略 AWS 服務，您可以將策略直接附加到資源（而不是使用角色作為代理）。若要了解跨帳戶存取角色和以資源為基礎的政策之間的差異，請參閱 IAM 使用者指南中的 IAM 中的跨帳戶資源存取。
- 跨服務訪問 — 有些 AWS 服務 使用其他 AWS 服務功能。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。
  - 轉寄存取工作階段 (FAS) — 當您使用 IAM 使用者或角色在中執行動作時 AWS，您會被視為主體。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 會使用主體呼叫的權限 AWS 服務，並結合要求 AWS 服務 向下游服務發出要求。只有當服務收到需要與其 AWS 服務 他資源互動才能完成的請求時，才會發出 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給 AWS 服務服務。
  - 服務連結角色 — 服務連結角色是連結至. AWS 服務服務可以擔任代表您執行動作的角色。服務連結角色會顯示在您的中， AWS 帳戶 且屬於服務所有。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 執行個體上執行的應用程式以及發出 AWS CLI 或 AWS API 請求的臨時登入資料。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 執行個體並提供給其所有應用程式，請建立連接至執行個體

的執行個體設定檔。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暫時憑證。如需詳細資訊，請參閱 IAM 使用者指南中的利用 IAM 角色來授予許可給 Amazon EC2 執行個體上執行的應用程式。

如需了解是否要使用 IAM 角色或 IAM 使用者，請參閱 IAM 使用者指南中的建立 IAM 角色 (而非使用者) 的時機。

使用政策管理存取權

您可以透 AWS 過建立原則並將其附加至 AWS 身分識別或資源來控制中的存取。原則是一個物件 AWS ，當與身分識別或資源相關聯時，會定義其權限。 AWS 當主參與者 (使用者、root 使用者或角色工作階段) 提出要求時，評估這些原則。政策中的許可決定是否允許或拒絕請求。大多數原則會 AWS 以 JSON 文件的形式儲存在中。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哪些內容。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。具有該原則的使用者可以從 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。若要了解如何建立身分類型政策，請參閱 IAM 使用者指南中的建立 IAM 政策。

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。受管理的策略是獨立策略，您可以將其附加到您的 AWS 帳戶. 受管政策包括 AWS 受管政策和客戶管理的策略。如需了解如何在受管政策及內嵌政策間選擇，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。主參與者可以包括帳戶、使用者、角色、同盟使用者或。 AWS 服務

資源型政策是位於該服務中的內嵌政策。您無法在以資源為基礎的政策中使用 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 和 Amazon VPC 是支援 ACL 的服務範例。 AWS WAF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原則類型。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。
- 服務控制策略（SCP）— SCP 是 JSON 策略，用於指定中組織或組織單位（OU）的最大權限。AWS Organizations AWS Organizations 是一種用於分組和集中管理您企業擁 AWS 帳戶 有的多個服務。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 限制成員帳戶中實體的權限，包括每個 AWS 帳戶根使用者帳戶。如需 Organizations 和 SCP 的詳細資訊，請參閱 AWS Organizations 使用者指南中的 SCP 運作方式。
- 工作階段政策 – 工作階段政策是一種進階政策，您可以在透過編寫程式的方式建立角色或聯合使用者的暫時工作階段時，作為參數傳遞。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。要了解如何在涉及多個政策類型時 AWS 確定是否允許請求，請參閱《IAM 使用者指南》中的政策評估邏輯。

FreeRTOS 務如何與 IAM 搭配使用

在您使用 IAM 管理 FreeRTOS 的存取權限之前，請先了解哪些 IAM 功能可用於 FreeRTOS。

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

若要深入瞭解 FreeRTOS 和其他 AWS 服務如何搭配大多數 IAM 功能運作，請參閱 IAM 使用者指南中的搭配 IAM 使用的AWS 服務。

FreeRTOS 的身分識別原則

支援身分型政策	是

身分型政策是可以連接到身分 (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。若要了解如何建立身分類型政策，請參閱《IAM 使用者指南》中的建立 IAM 政策。

使用 IAM 身分型政策，您可以指定允許或拒絕的動作和資源，以及在何種條件下允許或拒絕動作。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 的身分識別原則範例

若要檢視 FreeRTOS 身分識別型原則的範例，請參閱。FreeRTOS 的身分識別原則範例

FreeRTOS 中以資源為基礎的政策

支援以資源基礎的政策	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。主參與者可以包括帳戶、使用者、角色、同盟使用者或。 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，作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。當主體和資源位於不同時 AWS 帳戶，受信任帳戶中的 IAM 管理員也必須授與主體實體 (使用者或角色) 權限，才能存取資源。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 中的跨帳戶資源存取。

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哪些內容。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。

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。原則動作通常與關聯的 AWS API 作業具有相同的名稱。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 FreeRTOS 定義的動作。

FreeRTOS 中的原則動作會在動作之前使用下列前置詞：

```
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 身分識別型原則的範例，請參閱。FreeRTOS 的身分識別原則範例

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哪些內容。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 ARN 的清單，請參閱服務授權參考中 FreeRTOS 定義的資源。若要瞭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 身分識別型原則的範例，請參閱。FreeRTOS 的身分識別原則範例

FreeRTOS 的原則條件金鑰

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哪些內容。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，請使用邏輯OR運算來 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 全域條件金鑰，請參閱《IAM 使用者指南》中的AWS 全域條件內容金鑰。

若要查看 FreeRTOS 條件金鑰清單，請參閱服務授權參考中 FreeRTOS 的條件金鑰。若要瞭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 身分識別型原則的範例，請參閱。FreeRTOS 的身分識別原則範例

FreeRTOS 中的 ACL

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 與 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。您可以將標籤附加到 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 使用者指南中的什麼是 ABAC?。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)。

使用臨時登入資料搭配 FreeRTOS

支援臨時憑證	是

當您使用臨時憑據登錄時，某些 AWS 服務 不起作用。如需其他資訊，包括哪些 AWS 服務 與臨時登入資料搭配AWS 服務 使用，請參閱 IAM 使用者指南中的 IAM。

如果您使用除了使用者名稱和密碼以外的任何方法登入，則您正在 AWS Management Console 使用臨時認證。例如，當您 AWS 使用公司的單一登入 (SSO) 連結存取時，該程序會自動建立暫時認證。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 使用者指南中的切換至角色 (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，而非使用長期存取金鑰。如需詳細資訊，請參閱 IAM 中的暫時性安全憑證。

免費伺服器的跨服務主體權限

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，您會被視為主體。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 會使用主體呼叫的權限 AWS 服務，並結合要求 AWS 服務 向下游服務發出要求。只有當服務收到需要與其 AWS 服務 他資源互動才能完成的請求時，才會發出 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》。

免費伺服器的服務角色

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給 AWS 服務服務。

> **Warning**
>
> 變更服務角色的權限可能會中斷 FreeRTOS 的功能。只有在 FreeRTOS 提供指引時才編輯服務角色。

免費伺服器的服務連結角色

支援服務連結角色。	否

服務連結角色是連結至. AWS 服務服務可以擔任代表您執行動作的角色。服務連結角色會顯示在您的中， AWS 帳戶 且屬於服務所有。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 的身分識別原則範例

依預設，使用者和角色沒有建立或修改 FreeRTOS 資源的權限。他們也無法使用 AWS Management Console、 AWS Command Line Interface (AWS CLI) 或 AWS API 來執行工作。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。

若要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。

如需 FreeRTOS 定義的動作和資源類型的詳細資訊，包括每個資源類型的 ARN 格式，請參閱服務授權參考中 FreeRTOS 的動作、資源和條件金鑰。

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

以身分識別為基礎的原則會決定某人是否可以在您的帳戶中建立、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 受管原則並邁向最低權限權限 — 若要開始授與使用者和工作負載的權限，請使用可授與許多常見使用案例權限的AWS 受管理原則。它們可用在您的 AWS 帳戶. 建議您透過定義特定於您使用案例的 AWS 客戶管理政策，進一步降低使用權限。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。您也可以使用條件來授與對服務動作的存取權 (如透過特定 AWS 服務) 使用 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，可協助您編寫安全且實用的政策。如需更多資訊，請參閱 IAM 使用者指南中的 IAM Access Analyzer 政策驗證。
- 需要多因素身份驗證 (MFA) — 如果您的案例需要 IAM 使用者或根使用者 AWS 帳戶，請開啟 MFA 以獲得額外的安全性。如需在呼叫 API 操作時請求 MFA，請將 MFA 條件新增至您的政策。如需更多資訊，請參閱 IAM 使用者指南中的設定 MFA 保護的 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，您必須擁有最少一組權限。這些權限必須允許您列出並檢視您的. 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。

您不需要為僅對 AWS CLI 或 AWS API 進行呼叫的使用者允許最低主控台權限。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。

若要確保使用者和角色仍可使用 FreeRTOS 主控台，請同時將 FreeRTOS *ConsoleAccess* 或*ReadOnly* AWS 受管理的原則附加至實體。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。此原則包含在主控台上或以程式設計方式使用 AWS CLI 或 AWS API 完成此動作的權限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
```

```
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

疑難排解 FreeRTOS 身分識別與存取

使用下列資訊可協助您診斷並修正使用 FreeRTOS 和 IAM 時可能會遇到的常見問題。

主題

- 我沒有在 FreeRTOS 中執行動作的授權
- 我沒有授權執行 iam：PassRole
- 我想允許我以外的人訪問我 AWS 帳戶 的 FreeRTOS 資源

我沒有在 FreeRTOS 中執行動作的授權

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 系統管理員。您的管理員提供您的簽署憑證。

我沒有授權執行 iam：PassRole

如果您收到未獲授權執行iam:PassRole動作的錯誤訊息，則必須更新您的原則以允許您將角色傳遞給 FreeRTOS。

有些 AWS 服務 允許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 使用者marymajor嘗試使用主控台在 FreeRTOS 中執行動作時，就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 系統管理員。您的管理員提供您的簽署憑證。

我想允許我以外的人訪問我 AWS 帳戶 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：

- 若要瞭解 FreeRTOS 是否支援這些功能，請參閱。FreeRTOS 務如何與 IAM 搭配使用
- 若要了解如何提供對您所擁有資源 AWS 帳戶 的存取權，請參閱《IAM 使用者指南》中您擁有的另一 AWS 帳戶 個 IAM 使用者提供存取權限。
- 若要了解如何將資源存取權提供給第三方 AWS 帳戶，請參閱 IAM 使用者指南中的提供第三方 AWS 帳戶 擁有的存取權。
- 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)。
- 若要了解跨帳戶存取使用角色和以資源為基礎的政策之間的差異，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

若要瞭解 AWS 服務 是否屬於特定規範遵循方案的範圍內，請參閱AWS 服務 遵循規範計劃方案中的，並選擇您感興趣的合規方案。如需一般資訊，請參閱AWS 規範計劃AWS。

您可以使用下載第三方稽核報告 AWS Artifact。如需詳細資訊，請參閱下載中的報告中的 AWS Artifact。

您在使用時的合規責任取決 AWS 服務 於資料的敏感性、公司的合規目標以及適用的法律和法規。AWS 提供下列資源以協助遵循法規：

- 安全性與合規性快速入門指南 — 這些部署指南討論架構考量，並提供部署以安全性和合規性 AWS 為重點的基準環境的步驟。
- 在 Amazon Web Services 上架構 HIPAA 安全性與合規性 — 本白皮書說明公司如何使用建立符合 HIPAA 資格的應 AWS 用程式。

> **Note**
>
> 並非所有人 AWS 服務 都符合 HIPAA 資格。如需詳細資訊，請參閱 HIPAA 資格服務參照。

- AWS 合規資源AWS — 此工作簿和指南集合可能適用於您的產業和所在地。
- AWS 客戶合規指南 — 透過合規的角度瞭解共同的責任模式。這份指南總結了在多個架構 (包括美國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)) 中，保 AWS 服務 護指引並對應至安全控制的最佳實務。
- 使用AWS Config 開發人員指南中的規則評估資源 — 此 AWS Config 服務會評估您的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 提供了內部安全狀態的全面視圖 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 controls reference。
- Amazon GuardDuty — 透過監控環境中的 AWS 帳戶可疑和惡意活動，藉此 AWS 服務 偵測您的工作負載、容器和資料的潛在威脅。 GuardDuty 可協助您因應各種合規性需求，例如 PCI DSS，滿足特定合規性架構所規定的入侵偵測需求。
- AWS Audit Manager— 這 AWS 服務 有助於您持續稽核您的 AWS 使用情況，以簡化您管理風險的方式，以及遵守法規和業界標準的方式。

韌性 AWS

AWS 全球基礎架構是圍繞區 AWS 域和可用區域建立的。 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。

如需區域和可用區域的相關 AWS 資訊，請參閱AWS 全域基礎結構。

FreeRTOS 的基礎架構安全性

AWS 受管理服務受 Amazon Web Services：安 AWS 全程序概觀白皮書中所述的全球網路安全程序保護。

您可以使用 AWS 已發佈的 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。

亞馬遜自由 Github 存儲庫遷移指南

如果您有一個現有的 FreeRTOS 專案以目前已取代的 Amazon-freertos 儲存庫為基礎，請依照下列步驟執行：

1. 隨時獲得最新。檢查自由的 LTS 程式庫更新頁面，或訂閱自由式-LTS GitHub 用於接收包含重大和安全錯誤修復的最新 LTS 修補程序的存儲庫。您可以直接從個人下載或克隆所需的最新 FreeRTOS LTS 修補程序 GitHub 儲存庫。
2. 請考慮重構網路傳輸介面實作，以最佳化您的硬體平台。抽象的 API 就像安全通訊端和無線網絡 API不是最新的要求中央网络圖書館。請參閱傳輸介面有關更多詳細信息。

附錄

下表為亞馬遜 FreeRTOS 存儲庫中的所有示範專案、舊版程式庫和抽象 API 提供建議。

移轉的資源庫和示範

名稱	類型	建議
核心 HTTP	示範和圖書館	直接從現為複製或下載核心 HTTP存儲庫（子模塊，如果使用 git）FreeRTOS 組織。核心 HTTP 示範位於主要 FreeRTOS 發行版。如需詳細資訊，請參閱核心網頁。
中央网络	示範和圖書館	直接從現為複製或下載中央网络存儲庫（子模塊，如果使用 git）FreeRTOS 組織。科雷姆 QTT 演示是在主要 FreeRTOS 發行版。如需詳細資訊，請參閱酷网页。
核心 QTT 代理程式	示範和圖書館	直接從中複製或下載 CoremQTT 代理程式程式庫核心 QTT 代理程式存儲庫（子模塊，如果使用 git）FreeRTOS 組織。CoremQTT 代理程式示範位於核心 QTT 代理程式演示儲存庫。如

名稱	類型	建議	
		需詳細資訊，請參閱核心 QTT 代理程式頁面。	
设备防御者	示範和圖書館	該AWS IoT設備後衛程序庫位於其存儲庫中AWS GitHub 組織。克隆或下載它（子模塊，如果使用 git）直接從AWS IoT設備後衛儲存庫。該AWS IoT設備後衛演示在主要 FreeRTOS 發行版。如需詳細資訊，請參閱AWS IoT裝置後衛頁面。	
設備影子的 AWS	示範和圖書館	該AWS IoTDevice Shadow 程式庫位於其儲存庫中AWS GitHub 組織。克隆或下載它（子模塊，如果使用 git）直接從AWS IoTDevice Shadow) 儲存庫。該AWS IoTDevice Shadow 演示都在主要 FreeRTOS 發行版。如需詳細資訊，請參閱AWS IoTDevice Shadow 頁面。	
工作	示範和圖書館	該AWS IoT工作庫位於其儲存庫中AWS GitHub 組織。克隆或下載它（子模塊，如果使用 git）直接從AWS IoT工作儲存庫。該AWS IoT工作演示都在主要 FreeRTOS 發行版。如需詳細資訊，請參閱AWS IoT工作頁面。	
OTA	示範和圖書館	該AWS IoT空中（OTA）更新庫的存儲現為AWS GitHub 組織。克隆或下載它（子模塊，如果使用 git）直接從AWS IoT大田區儲存庫。該AWS IoTOTA 演示在主要 FreeRTOS 發行版。如需詳細資訊，請參閱AWS IoT大田頁。	

名稱	類型	建議	
CLI 和自由式加入	示範和圖書館	有一個 CLI 示例 WinSim。請參閱FreeRTOS 為最新頁面了解更多詳細信息。功能齊全的 FreeRTOS IoT 參考整合恩智浦智浦智浦 RT1060和STM32U5平台也提供實際硬體的 CLI 範例。	
日誌	巨集	某些 FreeRTOS 程式庫所使用的特定硬體平台有記錄巨集的實作。請參閱記錄頁面了解如何實現日誌宏。請參閱其中一個 FreeRTOS 精選 IoT 參考資料以取得在實際硬體上執行的範例。	
格蘭加拉斯連通	演示	[移轉進行中] 此示範專案假設在連線到AWS IoTGreengrass 的設備。展示本機驗證和探索功能的新專案正在開發中。預計新的演示項目將很快發布在FreeRTOS 組織。	

已淘汰的程式庫和示

名稱	類型	建議	
BLE	示範和程式庫	FreeRTOS BLE 程式庫實作專有的 MQTT 通訊協定，並支援透過代理裝置 (例如行動電話) 透過藍牙低功耗 (BLE) 發佈和訂閱 MQTT 主題。現為強制。使用您自己的 BLE 堆疊或第三方選項，例如靈活以最佳化您的專案。	
開發模式開發	示範	功能齊全的 FreeRTOS IoT 參考整合恩智浦智浦智浦	

名稱	類型	建議
		RT1060,STM32U5，或ESP32平台提供使用 CLI 進行關鍵佈建的範例。
POSIX	抽象和演示	不建議使用。
無線佈建	example	這個例子演示了如何提供 WiFi 使用亞馬遜 FreeRTOS 使用 BLE 庫的設備上的憑據。請參閱《FreeRTOS 精選 IoT》參考資料ESP32C3 平台以下為範例 WiFi 透過 BLE 進行佈建。
舊的抽象 API	code	這些 API 是為了提供各種協力廠商軟體堆疊、連線模組和 MCU 平台的抽象介面而建立的 API。例如，有接口 WiFi 抽象，安全的套接字等。它們在亞馬遜 FreeRTOS 存儲庫中受支持，並且位於文件夾中`/libraries/abstractions/` 。使用時不需要這些 API自由的 LTS 程式庫。

上表中的程式庫和示範不會取得安全性修補程式或錯誤修正。

第三方庫

當 Amazon-FreeRTOS 中的演示使用第三方庫時，我們建議您直接從其第三方存儲庫對其進行子模塊。

- 吊床：克隆它（子模塊，如果你使用 git）直接從吊床儲存庫。
- 傑斯門：不建議現為支援。
- lwip：克隆它（子模塊，如果你使用 git）直接從路易普儲存庫。
- 奧薩爾：請參閱《FreeRTOS 特色參考整合》（英文）i.MX RT1060或者STM32U5了解如何在您的硬件平台/板上實現 lwip_osal。
- mbedtls：克隆它（子模塊，如果你使用 git）直接從MBED-TLS儲存庫。mbedtls 配置和實用程序可以重複使用; 在這種情況下製作本地副本。
- pkcs11: 克隆它（子模塊，如果你使用 git）直接從海洋公園 11圖書館或綠洲公園 11儲存庫。

- 丁硝石：克隆它（子模塊，如果你使用 git）直接從丁硝石儲存庫。
- 靜脈密碼：我們建議您使用 MCU 平台上的加密加速器（如果可用）。如果您想繼續使用 tinycrypt，請直接從靜脈密碼儲存庫。
- 追蹤器記錄器：克隆它（如果你使用 git 的子模塊）直接從 Procepio 的追蹤記錄儲存庫。
- 團結：克隆它（子模塊，如果你使用 git）直接從ThrowTheSwitch/合一儲存庫。
- 贏帽：不再維護不再維護。我們建議您改用 libslirp、利比帽 (POSIX) 或 npcap。

移植測試和集成測試

下的所有測試`/tests`驗證 FreeRTOS 程式庫整合所需的資料夾已移轉至自由圖書館集成測試儲存庫。這些可用於測試 PAL 實現和庫集成。使用相同的測試AWS IoT裝置測試器 (IDT)AWSFreeRTOS 器的裝置資格認證計劃。

免費伺服器封存文件

自由使用者指南存檔

這些舊版的 FreeRTOS 使用者指南可與 FreeRTOS LTS (長期支援) 發行版本搭配使用。

- 免費伺服器使用者指南
- 免費伺服器使用者指南 20 2012.00 版

上一頁《自由服務指南》的內容

此內容已過時，但此處提供以供參考。

如需最近內容FreeRTOS 入門的連結，請參閱。

FreeRTOS 入

> **Important**
>
> 此頁面指的是已棄用的亞馬遜 FreeRTOS 存儲庫。建議您在建立新專案時入 如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱亞馬遜自由 Github 存儲庫遷移指南.

這個 FreeRTOS 入門教學課程會示範如何在主機上下載和設定 FreeRTOS，然後在合格的微控制器主機板上編譯並執行簡單的示範應用程式。

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

主題：

- 自由演示應用程序
- 首要步驟
- 故障診斷入門
- 搭配使用 FreeRTOS 體

- 開發人員模式金鑰佈建
- 主機板特定的入門指南
- FreeRTOS 務的後續步驟

自由演示應用程序

> **Important**
>
> 此頁面指的是已棄用的亞馬遜 FreeRTOS 存儲庫。建議您在建立新專案時入 如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱亞馬遜自由 Github 存儲庫遷移指南.

本教學課程中的示範應用程式是*freertos*/demos/coreMQTT_Agent/mqtt_agent_task.c檔案中定義的 CoremQtt 代理程式示範。它會使用連線通訊協定圖書館至AWS雲端，然後定期將訊息發佈至 MQTT 代理程式主控的AWS IoT 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就是執行的應用程式。在本教學課程中，您不需要啟用或停用任何示範。CoremQtt 代理程式示

如需關於 FreeRTOS 的詳細資訊，請參閱FreeRTOS 示。

首要步驟

> **Important**
>
> 此頁面指的是已棄用的亞馬遜 FreeRTOS 存儲庫。我們建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱. 亞馬遜自由 Github 存儲庫遷移指南

若要開始使用 FreeRTOS AWS IoT，您必須擁有一個 AWS 帳戶、具有存取權限的使用者 AWS IoT 和 FreeRTOS 雲端服務。您也必須下載 FreeRTOS 並設定您主機板的 FreeRTOS 示範專案才能使用。AWS IoT以下章節將逐步引導您完成這些要求。

> **Note**
>
> - 如果您使用的是義式濃縮咖啡 ESP32-DevKit C，ESP-勞弗套件，或 ESP32-WROOM-32SE，請跳過這些步驟並轉到。開始使用意式濃縮咖啡 ESP32-DevKit C 和歐洲環保工具組
> - 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. 設定您的 AWS 帳戶和權限
2. 註冊您的 MCU 主機板 AWS IoT
3. 下載 FreeRTOS
4. 設定 FreeRTOS 範

設定您的 AWS 帳戶和權限

註冊一個 AWS 帳戶

如果您沒有 AWS 帳戶，請完成以下步驟來建立一個。

若要註冊成為 AWS 帳戶

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

   部分註冊程序需接收來電，並在電話鍵盤輸入驗證碼。

   當您註冊一個時 AWS 帳戶，將創建AWS 帳戶根使用者一個。根使用者有權存取該帳戶中的所有 AWS 服務 和資源。安全性最佳做法是將管理存取權指派給使用者，並僅使用 root 使用者來執行需要 root 使用者存取權的工作。

AWS 註冊過程完成後，會向您發送確認電子郵件。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。

建立具有管理權限的使用者

註冊後，請確保您的安全 AWS 帳戶 AWS 帳戶根使用者 AWS IAM Identity Center、啟用和建立系統管理使用者，這樣您就不會將 root 使用者用於日常工作。

保護您的 AWS 帳戶根使用者

1. 選擇 Root 使用者並輸入您的 AWS 帳戶 電子郵件地址，以帳戶擁有者身分登入。AWS Management Console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)。

   如需指示，請參閱《IAM 使用者指南》中的為 AWS 帳戶 根使用者啟用虛擬 MFA 裝置 (主控台)。

建立具有管理權限的使用者

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 身分中心中，將管理存取權授予使用者。

   若要取得有關使用 IAM Identity Center 目錄 做為身分識別來源的自學課程，請參閱《使用指南》IAM Identity Center 目錄中的「以預設值設定使用AWS IAM Identity Center 者存取」。

以具有管理權限的使用者身分登入

- 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 身分中心使用者登入的說明，請參閱使用AWS 登入 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 身分中心中，建立遵循套用最低權限許可的最佳做法的權限集。

   如需指示，請參閱《AWS IAM Identity Center 使用指南》中的「建立權限集」。

2. 將使用者指派給群組，然後將單一登入存取權指派給群組。

   如需指示，請參閱《AWS IAM Identity Center 使用指南》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：

- 使用者和群組位於 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。請按照 IAM 使用者指南 的 為第三方身分提供者 (聯合) 建立角色 中的指示進行操作。

- 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 中的指示進行操作。
  - (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。

註冊您的 MCU 主機板 AWS IoT

您的主機板必須註冊， AWS IoT 才能與 AWS 雲端通訊。若要註冊您的主機板 AWS IoT，您必須具備：

一個 AWS IoT 政策

此原 AWS IoT 則會授與您的裝置存取 AWS IoT 資源的權限。它存儲在 AWS 雲中。

一 AWS IoT 件事

允許您在中管理設備的 AWS IoT 東西 AWS IoT。它存儲在 AWS 雲中。

私密金鑰和 X.509 憑證

私密金鑰和憑證可讓您的裝置進行驗證 AWS IoT。

若要註冊您的主機板，請遵循下列程序。

若要建立 AWS IoT 策略

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

   若要尋找您的 AWS 帳號，請開啟AWS 管理主控台，找出並展開右上角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。

若要尋找您 AWS 帳戶的 AWS 地區，請使用 AWS Command Line Interface。若要安裝 AWS CLI，請遵循使AWS Command Line Interface 用者指南中的指示。安裝之後 AWS CLI，開啟命令提示字元視窗，然後輸入下列命令：

```
aws iot describe-endpoint --endpoint-type=iot:Data-ATS
```

輸出應如下所示：

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

在此範例中，區域為 us-west-2。

> **Note**
>
> 我們建議使用 ATS 端點，如範例所示。

2. 瀏覽至 AWS IoT 主控台。
3. 在瀏覽窗格中，選擇 [安全]，選擇 [原則]，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。*aws-account*以您的 AWS 地區和帳戶 ID 取*aws-region*代和。

```
{
    "Version": "2012-10-17",
    "Statement": [
    {
        "Effect": "Allow",
        "Action": "iot:Connect",
        "Resource":"arn:aws:iot:aws-region:aws-account-id:*"
    },
    {
        "Effect": "Allow",
        "Action": "iot:Publish",
        "Resource": "arn:aws:iot:aws-region:aws-account-id:*"
    },
    {
         "Effect": "Allow",
```

```
            "Action": "iot:Subscribe",
            "Resource": "arn:aws:iot:aws-region:aws-account-id:*"
        },
        {
            "Effect": "Allow",
            "Action": "iot:Receive",
            "Resource": "arn:aws:iot:aws-region:aws-account-id:*"
        }
        ]
}
```

此政策可授予下列許可：

iot:Connect

授予您的裝置使用任何用戶端 ID 連線至 AWS IoT 訊息代理程式的權限。

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 [附加原則]，將原則附加至憑證，以授予裝置 AWS IoT 作業存取權。
10. 選擇您剛建立的政策，然後選擇 Register thing (註冊實物)。

在註冊您的主機板之後 AWS IoT，您可以繼續執行下載 FreeRTOS。

下載 FreeRTOS

您可以從 FreeRTOS 軟體庫下載 Free GitHub RTOS。

下載 FreeRTOS 之後，您可以繼續執行。設定 FreeRTOS 範

設定 FreeRTOS 範

您必須先編輯 FreeRTOS 目錄中的一些設定檔，才能在主機板上編譯並執行任何示範。

設定您的 AWS IoT 端點

您必須將 FreeRTOS 與 AWS IoT 端點一起提供，以便在主機板上執行的應用程式可以將要求傳送到正確的端點。

1. 瀏覽至 AWS IoT 主控台。
2. 在左側的導覽窗格中，選擇設定。

   您的 AWS IoT 端點會顯示在裝置資料端點中。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 連線連線到網際網路，您必須提供具有 Wi-Fi 認證的 FreeRTOS 才能連線到網路。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 認證

FreeRTOS 必須具有與您註冊物件相關聯的 AWS IoT 憑證和私密金鑰及其權限原則，才能代表您的裝置成功通訊。 AWS IoT

> **Note**
>
> 若要設定身分證 AWS IoT 明，您必須擁有註冊裝置時從 AWS IoT 主機下載的私密金鑰和憑證。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，但無法擷取私密金鑰。

FreeRTOS 是一個 C 語言專案，憑證和私密金鑰必須經過特殊格式化才能新增至專案。

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的入門指南，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。在入門教學課程中使用的示範應用程式是 CoremQtt 相互驗證示範，位於。demos/coreMQTT/mqtt_demo_mutual_auth.c

故障診斷入門

> **Important**
>
> 此頁面指的是已棄用的亞馬遜 FreeRTOS 存儲庫。建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱亞馬遜自由 Github 存儲庫遷移指南.

下列方案可以幫助您對 FreeRTOS 開始使用方案發生的問題進行故障診斷：

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 指南。

一般入門故障診斷提示

執行 Hello World 示範專案之後，AWS IoT主控台中沒有任何訊息出現。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。

運行演示時在我的終端中顯示的日誌被截斷。我怎樣才能增加他們的長度？

將您正在執`configLOGGING_MAX_MESSAGE_LENGTH`行的示範`FreeRTOSconfig.h`檔案中的值增加到 255：

```
#define configLOGGING_MAX_MESSAGE_LENGTH    255
```

安裝終端機模擬器

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

您必須將電路板連接到電腦，然後再嘗試建立電路板與終端機模擬器的序列連線。

請使用以下設定來設定終端機模擬器：

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

尋找您的開發板的序列埠

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

Windows

```
chgport
```

Linux

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

macOS

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

搭配使用 FreeRTOS 體

> **Important**
>
> 此頁面指的是已棄用的亞馬遜 FreeRTOS 存儲庫。我們建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱. 亞馬遜自由 Github 存儲庫遷移指南

您可以使用 CMake 從 FreeRTOS 應用程序源代碼生成項目構建文件，並構建和運行源代碼。

您也可以使用 IDE 在符合 FreeROS 資格的裝置上編輯、偵錯、編譯、快閃記憶體和執行程式碼。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：

- 濃縮咖啡 ESP32-C DevKit
- Espressif ESP-WROVER-KIT
- Infineon XMC4800 IoT 連線套件
- 瑪維爾 MW320 AWS IoT 入門套件
- 瑪維爾 MW322 AWS IoT 入門套件
- Microchip Curiosity PIC32MZEF 套件
- Nordic nRF52840 DK Development kit
- STMicroelectronicsSTM32L4 Discovery Kit IoT Node
- Texas Instruments CC3220SF-LAUNCHXL

- Microsoft Windows 模擬器

請參閱下列主題，以取得有關搭配 FreeRTOS 使用 CMake 的詳細資訊。

主題

- 必要條件
- 使用協力廠商程式碼編輯器和偵錯工具開發 FreeRTOS 應用
- 使用 CMake 構建 FreeRTOS 務

必要條件

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**
  >
  > 在軟件包管理器中為許多 Linux 發行版提供的 CMake 軟件包版本是 out-of-date。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。

使用協力廠商程式碼編輯器和偵錯工具開發 FreeRTOS 應用

您可以使用程式碼編輯器和偵錯延伸模組或協力廠商偵錯工具來開發 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。如需有關使用 CMake 建置 FreeRTOS 應用程式的詳細資訊，請參閱。使用 CMake 構建 FreeRTOS 務

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 構建 FreeRTOS 務

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。有關 CMake 中交叉編譯的更多信息，請參閱 CMake 官CrossCompiling方維基。

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。

   如需有關產生 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。

   如需建立 FreeRTOS 建置檔案的詳細資訊，請參閱。從生成的構建文件構建 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 的組建檔案。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`— 指定原始程式碼的位置。
- `-B`— 指定產生之建置檔案的位置。

> **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 將調試標誌添加到編譯選項中，並使用調試符號構建 FreeRTOS。

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

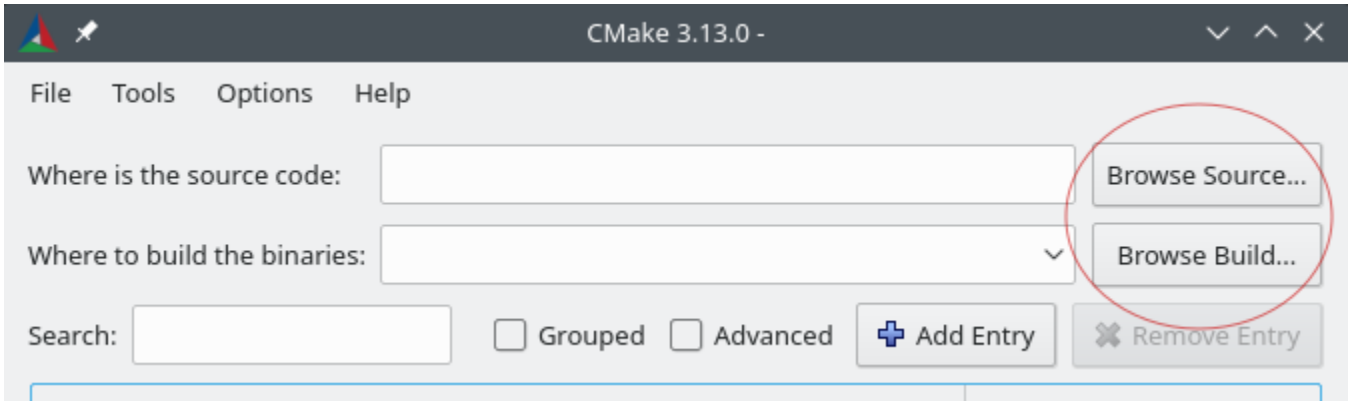
您也可以將 CMAKE_BUILD_TYPE 設為 release，以將最佳化旗標新增到編譯選項。

產生建置檔案 (CMake GUI)

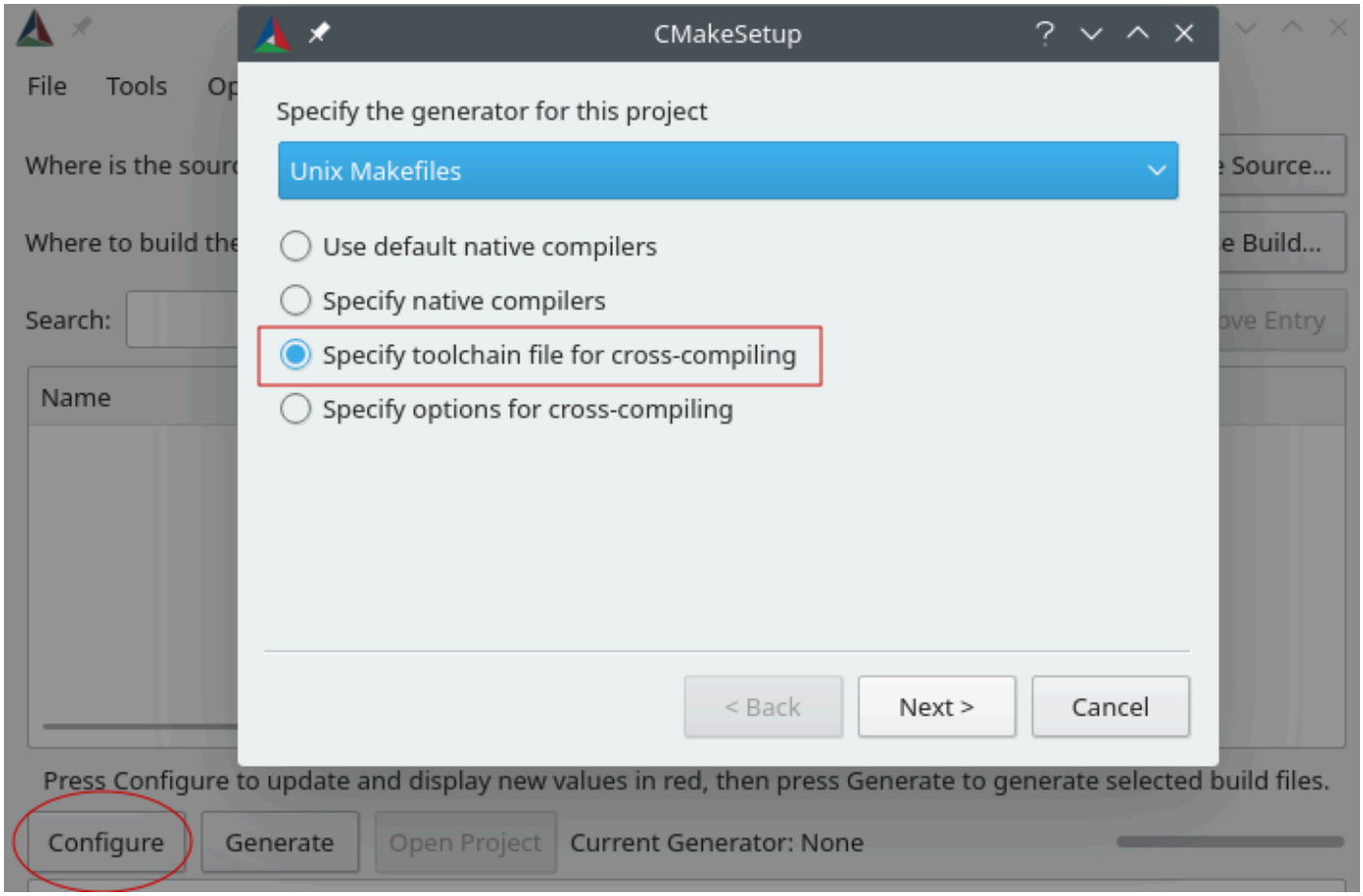
您可以使用 CMake 圖形用戶界面來生成 FreeRTOS 構建文件。

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。在這種情況下，請從「檔案」功能表中選擇「刪除快取」來刪除 CMake 快取。



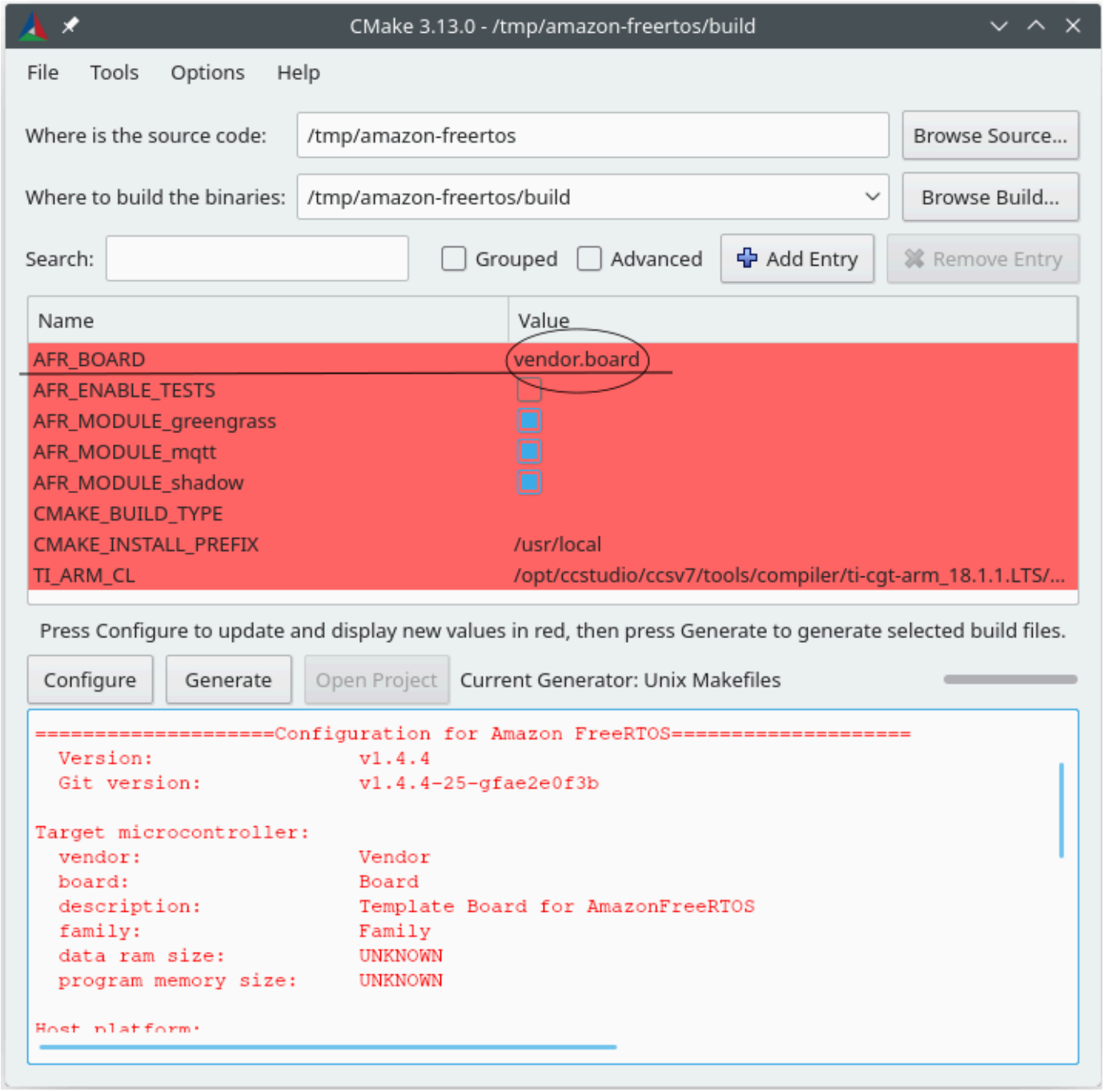
4. 選擇 Specify toolchain file for cross-compiling (指定跨編譯的工具鏈檔案)，然後選擇 Next (下一步)。
5. 選擇工具鏈檔案 (例如，*freertos*/tools/cmake/toolchains/arm-ti.cmake)，然後選擇 Finish (完成)。

   FreeRTOS 的預設組態為範本主機板，不提供任何可攜式圖層目標。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。

從生成的構建文件構建 FreeRTOS

使用原生建置系統進行建置

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

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

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

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**
>
> 此頁面指的是已棄用的亞馬遜 FreeRTOS 存儲庫。建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱亞馬遜自由 Github 存儲庫遷移指南.

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 AWS IoT 匯入私有金鑰

基於實驗室測試目的，如果您的裝置允許匯入私有金鑰，請遵循 設定 FreeRTOS 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，執行 設定 FreeRTOS 範 中的步驟，但略過最後一個步驟 (亦即不要進行格式化您的 AWS IoT 登入資料)。最終結果應是 `demos/include/aws_clientcredential.h` 檔案以您的設定更新，但 `demos/include/aws_clientcredential_keys.h` 文件則否。

示範專案組態

依照您主機板的指南中所述，開啟 CoremQtt 相互驗證示範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

由於裝置尚未佈建私密金鑰和用戶端憑證，示範將無法進行驗證AWS IoT。不過，CoremQtt 相互驗證示範會從執行開發人員模式金鑰佈建開始，如果私密金鑰尚未存在，就會建立私密金鑰。您應該會看到類似下列內容的序列主控台輸出的開頭。

```
7 910 [IP-task] Device public key, 91 bytes:
3059 3013 0607 2a86 48ce 3d02 0106 082a
8648 ce3d 0301 0703 4200 04cd 6569 ceb8
1bb9 1e72 339f e8cf 60ef 0f9f b473 33ac
6f19 1813 6999 3fa0 c293 5fae 08f1 1ad0
41b7 345c e746 1046 228e 5a5f d787 d571
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，裝置將不會簽署憑證要求 (亦即憑證服務要求或 CSR)，因為 FreeRTOS 示範專案已排除建立和簽署 CSR 所需的 X.509 編碼邏輯，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
```

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

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

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

憑證匯入

核發憑證後，下一步就是將其匯入裝置。您還需要匯入憑證授權單位 (CA) 憑證，因為必須如此，才能在使用 JITP 時成功完成第一次的 AWS IoT 驗證。在您專案的 `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

依使用您自己的憑證中所述匯入`deviceCert.pem`AWS IoT登錄。您必須建立新的 AWS IoT 物件，將 PENDING 憑證和政策連接到您的物件，然後將憑證標記為 ACTIVE (作用中)。所有這些步驟都可在 AWS IoT 主控台中手動執行。

一旦新的用戶端憑證為作用中並與物件和原則產生關聯，請再次執行 CoremQtt 相互驗證示範。現在，AWS IoT IoT MQTT 中介裝置的連線成功建立。

主機板特定的入門指南

> **Important**
>
> 此頁面指的是已棄用的亞馬遜 FreeRTOS 存儲庫。建議您使用新專案時使用中的來開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱亞馬遜自由 Github 存儲庫遷移指南.

完成之後首要步驟，請參閱您主機板的指南，瞭解如何開始使用 FreeRTOS 的特定主機板指示：

- Cypress CYW943907AEVAL1F 開發套件入門

- Cypress CYW954907AEVAL1F 開發套件入門
- 開始使用賽普拉斯 CY8CKIT-064S0S2-4343W 套件
- Infineon XMC4800 IoT Connectivity Kit 入門
- 開始使用 MW32 AWS IoT x 入門套件
- 開始使用 MediaTek MT7697Hx 開發套件
- Microchip Curiosity PIC32MZ EF 入門
- 開始使用新唐 NuMaker-IOT M487 系列產品
- 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
- 開始使用賽靈思安 MicroZed 富利工業 IoT 套件

Note

您不需要完成下首要步驟列 FreeRTOS 入門指南的自助式入門指南：

- 開始使用 Microchip ATECC608A 安全元素與 Windows 模擬器
- 開始使用意式濃縮咖啡 ESP32-DevKit C 和歐洲環保工具組
- 開始使用意式濃縮咖啡 ESP32-WROOM-32SE
- 開始使用濃縮咖啡 ESP32-S2
- 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

此參考集成託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱亞馬遜自由 Github 存儲庫遷移指南.

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**
>
> 此教學課程會逐步說明如何設定並執行 CoremQtt 相互驗證示範。此主機板的 FreeRTOS 連接埠目前不支援 TCP 伺服器和用戶端示範。

在開始之前，您必須先設定 FreeRTOSAWS IoT 並下載，才能將裝置連線到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。
> - 由於原始程式碼可能包含符號連結，因此如果您使用 Windows 來擷取歸檔，您可能必須：
>   - 啟用開發人員模式，或
>   - 使用以系統管理員身分提高權限的主控台。
>
>   如此一來，Windows 可以在擷取歸檔時正確建立符號連結。否則，符號鏈接將被寫入為普通文件，其中包含符號鏈接的路徑作為文本或為空。如需詳細資訊，請參閱部落格條目：Windows 10 中的符號鏈接！ 。
>
>   如果您在 Windows 下使用 Git，您必須啟用開發人員模式，或者您必須：
>   - 使用下列命令設定`core.symlinks`為 true：
>
>     ```
>     git config --global core.symlinks true
>     ```
>   - 每當您使用寫入系統的 git 命令時，請使用以管理員身份提升的主控台 (例如git pullgit clone、、和git submodule update --init --recursive)。
> - 如中所述下載 FreeRTOS，Cypress 的 FreeRTOS 連接埠目前僅適用於 GitHub。

概要

本教學課程包含以下入門步驟的指示：

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

在這份入門指南中，您可以使用賽普拉斯 WICED 工作室 SDK 來設計您的主機板與 FreeRTOS 示範。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。

監控雲端的 MQTT 訊息

在執行 FreeRTOS 示範專案之前，您可以在主控台中設定 MQTT 用戶端，以監AWS IoT控裝置傳送至AWS雲端的訊息。

使用 AWS IoT MQTT 用戶端訂閱 MQTT 主題

1. 登入 AWS IoT 主控台。
2. 在導覽窗格中，選擇 [測試]，然後選擇 [MQTT 測試用戶端] 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

在您設定主機板的序列連線之後，您可以建置 FreeRTOS 示範專案，將示範快閃到您的主機板，然後執行示範。

在 WICED 工作室中構建和運行 FreeRTOS 演示項目

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。

疑難排解

- 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**
>
> 此參考集成託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。建議您在建立新的專案時，從這裡開始建立新的專案。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱亞馬遜自由 Github 存儲庫遷移指南.

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**
>
> 此教學課程會帶您逐步說明如何設定並執行 CoremQtt 相互驗證的設定並執行 CoremQtt 此主機板的 FreeRTOS 連接埠目前不支援 TCP 伺服器和用戶端示範。

在開始之前，您必須先設定 FreeRTOSAWS IoT 並下載，才能將裝置連線到AWS雲端。如需說明，請參閱 首要步驟。在此教學課程中，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。
> - 由於原始程式碼可能包含符號連結，因此如果您使用 Windows 來擷取歸檔，您可能必須：
>   - 啟用開發人員模式，或
>   - 使用以系統管理員身分提高權限的主控台。
>
>   如此一來，Windows 可以在擷取歸檔時正確建立符號連結。否則，符號鏈接將被寫入為普通文件，其中包含符號鏈接的路徑作為文本或為空。如需詳細資訊，請參閱部落格條目部落格條目：部落格條目文章： 。
>
>   如果您在 Windows 下使用 Git，您必須啟用開發人員模式，或者您必須：
>   - 使用下列命令設定`core.symlinks`為 true：
>
>     ```
>     git config --global core.symlinks true
>     ```
>   - 每當您使用寫入系統的 git 命令時，請使用以管理員身份提升的主控台 (例如git pullgit clone、、和git submodule update --init --recursive)。
> - 如中所述下載 FreeRTOS，Cypress 的 FreeRTOS 連接埠目前僅適用於 GitHub。

概要

本教學課程包含以下入門步驟的指示：

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

在這份入門指南中，您可以使用賽普拉斯 WICED 工作室 SDK 來設計您的主機板與 FreeRTOS 示範。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。

監控雲端的 MQTT 訊息

在執行 FreeRTOS 示範專案之前，您可以在主控台中設定 MQTT 用戶端，以監AWS IoT控裝置傳送至AWS雲端的訊息。

使用 AWS IoT MQTT 用戶端訂閱 MQTT 主題

1. 登入 AWS IoT 主控台。
2. 在導覽窗格中，選擇 [測試]，然後選擇 [MQTT 測試用戶端] 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

在您設定主機板的序列連線之後，您可以建置 FreeRTOS 示範專案，將示範快閃到您的主機板，然後執行示範。

在 WICED 工作室中構建和運行 FreeRTOS 演示項目

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。

疑難排解

- 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。

開始使用賽普拉斯 CY8CKIT-064S0S2-4343W 套件

> **Important**
>
> 此參考集成託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。我們建議您從這裡開始當您創建一個新的項目。如果您已經有一個基於現在不推薦使用的亞馬遜-FreeRTOS 儲存庫的現有 FreeRTOS 專案，請參閱亞馬遜自由 Github 存儲庫遷移指南。

本教學課程提供開始使用的指示CY8CKIT-064S0S2-4343W套件。如果您還沒有，您可以使用上一個連結來購買套件。您也可以使用該連結存取套件使用者指南。

入門

在開始之前，您必須配置AWS IoT和 FreeRTOS 將您的設備連接到AWS雲。如需相關指示，請參閱首要步驟。在您完成的必要條件，您將有一個 FreeRTOS 的套件AWS IoT Core憑證。

> **Note**
>
> 在本教學課程中，在「第一個步驟」一節中建立的 FreeRTOS 下載目錄路徑稱為*freertos*。

設定開發環境

FreeRTOS 可以與 CMake 或製作構建流程一起使用。您可以使用 ModusToolbox 為您的製作構建流程。您可以使用提供的日食 IDE ModusToolbox 或合作夥伴 IDE，如 IAR 電子臂，手臂 MDK，或 Microsoft 視覺工作室代碼。Windows、macOS 和 Linux 都支援 ECLIPSE IDE。

在開始之前，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。

更新的工具 ModusToolbox 2.1 或更高版本

如果您使用的 ModusToolbox 2.1 Eclipse IDE 對此套件進行編程，您需要更新 OpenOCD 和固件加載器工具。

在下列步驟中，預設為*ModusToolbox*路徑：

- 窗戶是C:\Users*user_name*\ModusToolbox。
- Linux 是*user_home*/ModusToolbox或者您選擇提取存檔文件的位置。
- MacOS 位於您在精靈中所選磁碟區的「應用程式」資料夾下。

OpenOCD 新

此套件需要賽普拉斯 OpenOCD 4.0.0 或更新版本，才能成功擦除和編程芯片。

若要更新柏

1. 前往柏樹 OpenOCD 發布頁面。
2. 為您的操作系統（視窗/Mac /Linux）下載歸檔文件。
3. 刪除現有文件*ModusToolbox*/tools_2.x/openocd。
4. 取代中的檔案*ModusToolbox*/tools_2.x/openocd將替換為您前個步驟中下載的檔案的內容。

更新固件加載器

此套件需要賽普拉斯固件加載器 3.0.0 或更高版本。

更新賽普拉斯固件加載器

1. 前往絲柏固件加載器發布頁面。
2. 為您的操作系統（視窗/Mac /Linux）下載歸檔文件。
3. 刪除現有文件*ModusToolbox*/tools_2.x/fw-loader。
4. 取代中的檔案*ModusToolbox*/tools_2.x/fw-loader與您在上一步中下載的存檔的內容。

或者，您可以使用 CMake 從 FreeRTOS 應用程序源代碼生成項目構建文件，使用首選的構建工具構建項目，然後使用 OpenOCD 對套件進行編程。如果您偏好使用 GUI 工具與 CMake 流程進行程式設計，請從賽普拉斯編程解網頁。如需詳細資訊，請參閱搭配使用 FreeRTOS 體。

設定您的硬體

請按照下列步驟，設定套件的硬體。

1. 佈建您的套件

   按照CY8CKIT-064S0S2-4343W 套件的佈建指南安全佈建套件的說明AWS IoT。

   此套件需要 CySecureTools 3.1.0 或更高版本。

2. 設定序列連線

   a. 將套件 Connect 到主機電腦。

   b. 套件的 USB 序列埠會自動列舉在主機電腦上。查看通訊埠號碼。在視窗中，您可以使用裝置管理員下連接埠(COM & LPT).

   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，您將構建並運行演示。

1. 請務必按照中的步驟CY8CKIT-064S0S2-4343W 套件的佈建指南。
2. 建置免費使用者示範。

   a. 打開日食 IDE ModusToolbox 然後選擇或建立工作區。

   b. 從 File (檔案) 功能表中，選擇 Import (匯入)。

      展開將軍，選擇現有專案進入工作區，然後選擇下一步。

   c. 在根目錄，輸入*freertos*/projects/cypress/CY8CKIT-064S0S2-4343W/mtb/aws_demos然後選擇項目名稱aws_demos。它應該在默認情況下選擇。

   d. 選擇完成將專案匯入您的工作區。

   e. 使用下列步驟，建立應用程式：

      - 來自的。快速面板，選擇構建 aw_ 演示應用程序。
      - 選擇項目並選擇建立全部。

      確保項目編譯沒有錯誤。

3. 監控雲端的 MQTT 訊息

   在執行示範之前，您可以在中設定 MQTT 用戶端AWS IoT用於監視設備發送到的消息的控制台AWS雲。若要使用上一個步驟訂閱 MQTT 主題AWS IoTMQTT 客戶端，請按照下列步驟操作。

   a. 登入 AWS IoT 主控台。

   b. 在瀏覽窗格中，選擇測試，然後選擇MQTT 測試客戶端以開啟 MQTT 用戶端。

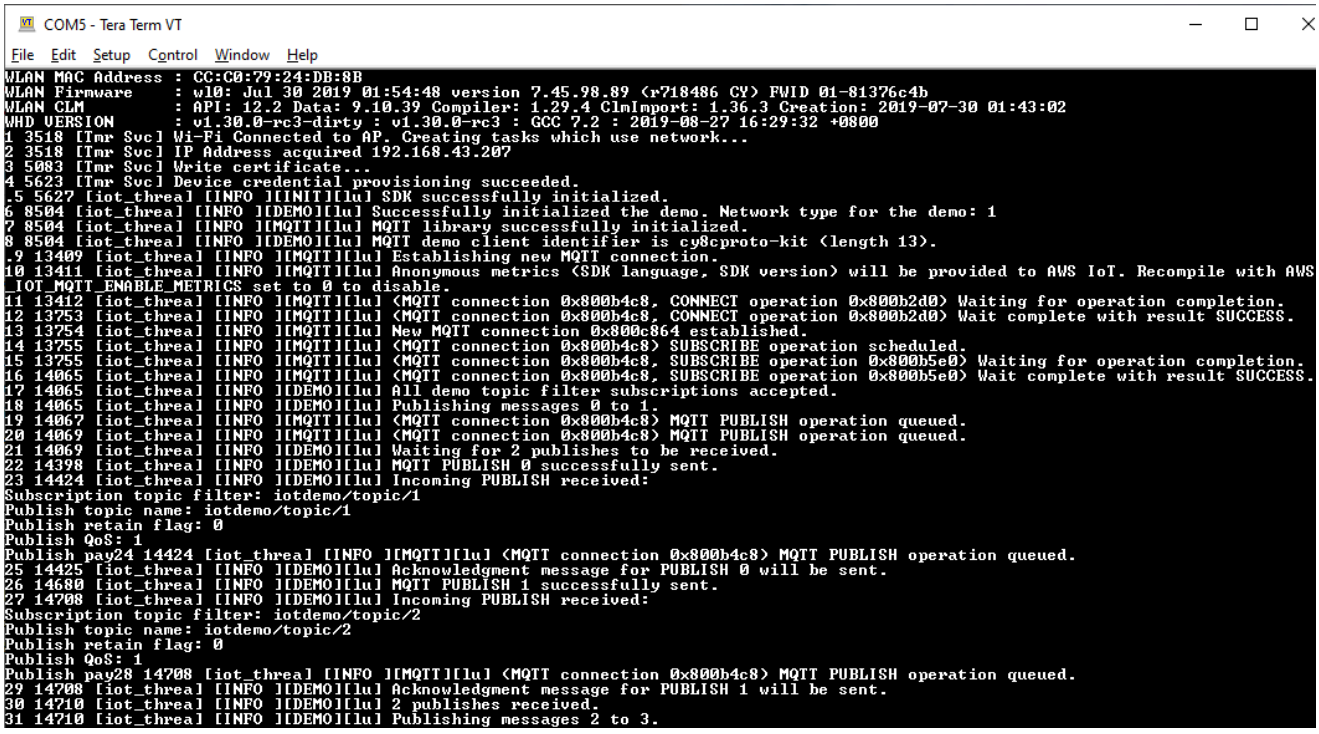
   c. 對於訂閱主題，輸入***your-thing-name*/example/topic**，然後選擇訂閱主題。

4. 執行 FreeRTOS 示範專案

   a. 選擇項目aws_demos在工作區中。

   b. 來自的。快速面板，選擇aw_ 演示計劃 (KitProg3)。該程序的電路板和演示應用程序在編程完成後開始運行。

   c. 您可以查看在串行終端中查看正在運行的應用程序的狀態。下圖顯示了端子輸出的一部分。



MQTT 示範會針對四個不同主題發佈訊息 (iotdemo/topic/*n*，其中 n = 1 至 4）並訂閱所有這些主題以接收相同的消息。當收到訊息時，示範會發佈關於該主題的確認訊息iotdemo/acknowledgements。下列清單描述出現在終端機輸出中的除錯訊息，以及訊息序號的參照。在輸出中，WICED 主機驅動程序（WHD）驅動程序的詳細信息首先打印，而不進行序列編號。

1. 1 至 4 — 裝置連線至已設定的存取點 (AP)，並透過連線至AWS使用設定的端點和憑證的伺服器。
2. 5 至 13 — 初始化 CoremQTT 程式庫，裝置會建立 MQTT 連線。
3. 14 到 17 — 設備訂閱所有主題以接收發布的消息。
4. 18 至 30 — 裝置會發佈兩則訊息，並等待接收回來。收到每則訊息後，裝置會傳送確認訊息。

發佈、接收和確認的相同週期會持續進行，直到發佈所有訊息為止。每個週期會發佈兩則訊息，直到配置的週期數完成為止。

5. 搭配 FreeRTOS 體使用 CMake

您也可以使用 CMake 來構建和運行演示應用程序。要設置 CMake 和本地構建系統，請參閱必要條件。

a. 使用下列命令，建立檔案。使用指定目標板-DBOARD選項。

```
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，因為 CMake 默認使用視覺工作室。

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. 使用下面的命令，建立使用 CMake 的項目。

```
cmake --build build_dir
```

c. 最後，編程cm0.hex和cm4.hex下產生的檔案*build_dir*使用賽普拉斯程序員。

執行其他示範

下列示範應用程式已經過測試和驗證，可與目前版本搭配使用。您可以在下面找到這些演示*freertos*/demos目錄。如需有關如何執行這些示範的資訊，請參閱FreeRTOS 示。

- 藍牙低功耗演示
- 空中更新演示
- 安全套接字迴聲客戶端演
- AWS IoT。Device Shadow 演示

除錯

該 KitProg套件上的 3 支援透過 SWD 通訊協定進行除錯。

- 若要偵錯 FreeRTOS 應用程式，請選取aw_ 演示項目在工作區中，然後選取aw_ 演示調試（KitProg3)來自。快速面板。

大田區更新

PSoC 64 MCU 已通過所有必要的 FreeRTOS 資格測試。但是，可選 over-the-air在 PSoC 64 標準安全中實現的（OTA）功能AWS韌體程式庫仍在等待評估中。實施的 OTA 功能目前通過了所有 OTA 資格測試，除了aws_ota_test_case_rollback_if_unable_to_connect_after_update.py。

當使用 PSOc64 標準安全將成功驗證的 OTA 映像應用於設備時 —AWSMCU 和設備無法通信AWS IoT Core，設備無法自動回滾到原始已知的良好圖像。這可能會導致無法訪問該設備AWS IoT Core以獲取進一步更新。賽普拉斯團隊仍在開發此功能。

如需詳細資訊，請參閱OTA 更新AWS和 CY8CKIT-064S0S2-4343W 套件。如果您有其他問題或需要技術支援，請聯絡柏樹開發者社區。

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

> **Important**
>
> 此參考集成託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。我們建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱. 亞馬遜自由 Github 存儲庫遷移指南

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

在開始之前，您必須先設定 FreeRTOS AWS IoT 並下載，才能將裝置連線到雲端 AWS 。如需說明，請參閱 首要步驟。*########FreeRTOS ######### 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為二進位映像檔。
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。

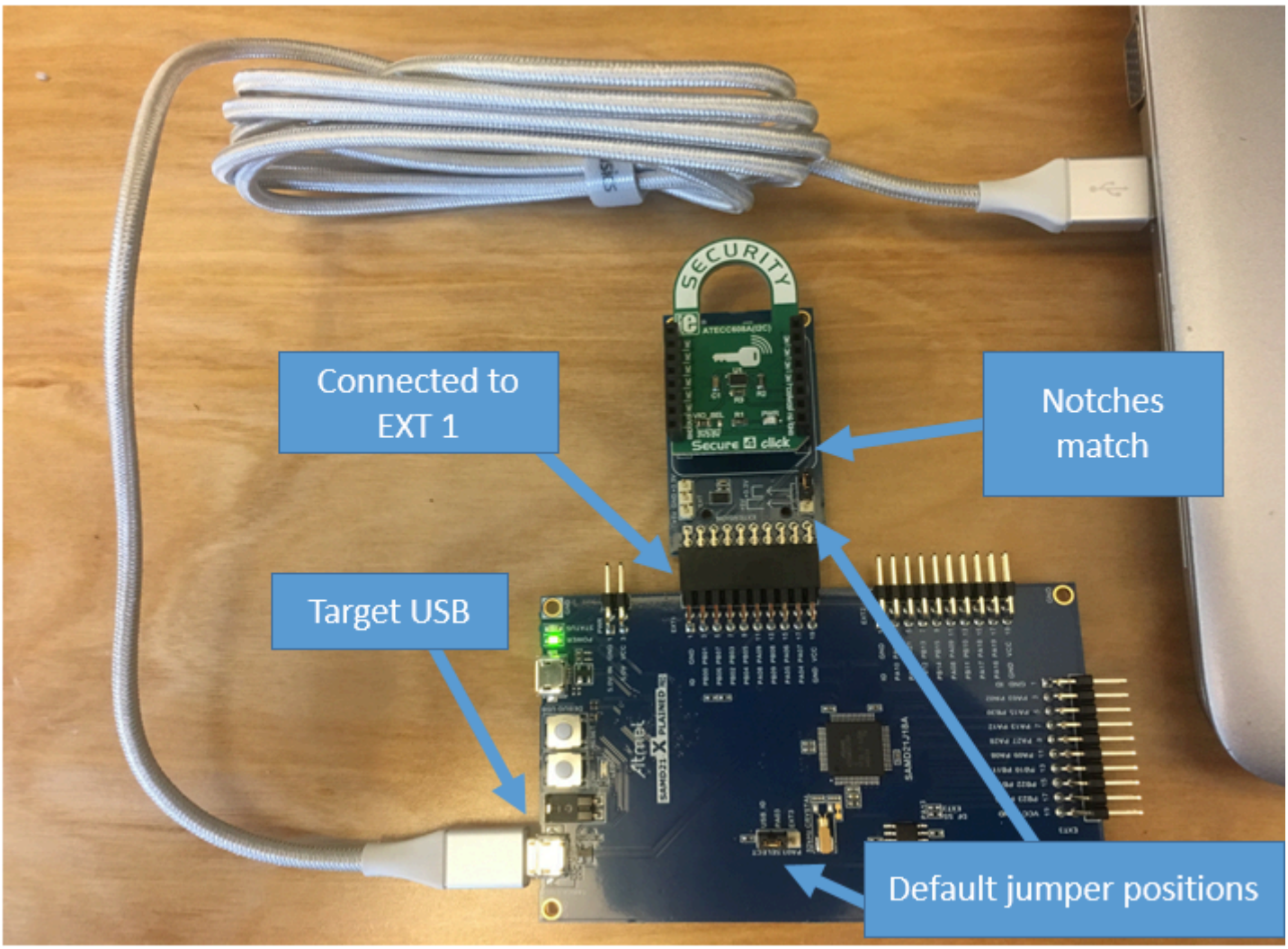
設定 SAMD21 XPlained Pro 面板

1. 請按照 CryptoAuthSSH-XSTK (DM320109)-最新韌體連結下載包含指令 (PDF) 的 .zip 檔案，以及可編程到 D21 上的二進位檔案。
2. 下載並安裝阿特梅爾工作室 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

註冊一個 AWS 帳戶

如果您沒有 AWS 帳戶，請完成以下步驟來建立一個。

若要註冊成為 AWS 帳戶

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

   部分註冊程序需接收來電，並在電話鍵盤輸入驗證碼。

   當您註冊一個時 AWS 帳戶，將創建AWS 帳戶根使用者一個。根使用者有權存取該帳戶中的所有 AWS 服務 和資源。安全性最佳做法是將管理存取權指派給使用者，並僅使用 root 使用者來執行需要 root 使用者存取權的工作。

AWS 註冊過程完成後，會向您發送確認電子郵件。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。

建立具有管理權限的使用者

註冊後，請保護 AWS 帳戶 AWS 帳戶根使用者、啟用和建立系統管理使用者 AWS IAM Identity Center，這樣您就不會將 root 使用者用於日常工作。

保護您的 AWS 帳戶根使用者

1. 選擇 Root 使用者並輸入您的 AWS 帳戶 電子郵件地址，以帳戶擁有者身分登入。AWS Management Console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)。

   如需指示，請參閱《IAM 使用者指南》中的為 AWS 帳戶 根使用者啟用虛擬 MFA 裝置 (主控台)。

建立具有管理權限的使用者

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 身分中心中，將管理存取權授予使用者。

   若要取得有關使用 IAM Identity Center 目錄 做為身分識別來源的自學課程，請參閱《使用指南》IAM Identity Center 目錄中的「以預設值設定使用AWS IAM Identity Center 者存取」。

以具有管理權限的使用者身分登入

- 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 身分中心使用者登入的說明，請參閱使用AWS 登入 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 身分中心中，建立遵循套用最低權限許可的最佳做法的權限集。

如需指示，請參閱《AWS IAM Identity Center 使用指南》中的「建立權限集」。

2. 將使用者指派給群組，然後將單一登入存取權指派給群組。

   如需指示，請參閱《AWS IAM Identity Center 使用指南》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：

- 使用者和群組位於 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。請按照 IAM 使用者指南 的 為第三方身分提供者 (聯合) 建立角色 中的指示進行操作。

- 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 中的指示進行操作。
  - (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. 從 FreeRTOS 軟體庫下載 FreeRTOS 軟體庫。 GitHub

   若要從下列位置下載 FreeRTOS： GitHub

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

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

      Important

      - 在本主題中，FreeRTOS 下載目錄的路徑稱為。*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。
- 由於原始程式碼可能包含符號連結，因此如果您使用 Windows 來擷取歸檔，您可能必須：
  - 啟用開發人員模式，或
  - 使用以系統管理員身分提高權限的主控台。

  如此一來，Windows 可以在擷取歸檔時正確建立符號連結。否則，符號鏈接將被寫入為普通文件，其中包含符號鏈接的路徑作為文本或為空。如需詳細資訊，請參閱 Windows 10 中的部落格條目符號連結！ 。

  如果您在 Windows 下使用 Git，您必須啟用開發人員模式，或者您必須：
  - 使用下列命令設定`core.symlinks`為 true：

    ```
    git config -\-global core.symlinks true
    ```
  - 每當您使用寫入系統的 git 命令時，請使用以管理員身份提升的主控台 (例如git pullgit clone、、和git submodule update -\-init -\-recursive)。

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。

若要使用 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 [建置方案]，並確定解決方案建置時沒有錯誤。選擇 Debug (偵錯)、Start Debugging (開始偵錯) 以執行專案。在第一次執行時，您需要設定您的裝置界面並重新編譯。如需詳細資訊，請參閱 設定網路界面。

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. 使用建立並註冊憑證授權單位 AWS。
   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，並使用註冊該憑證 AWS。
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。

開始使用意式濃縮咖啡 ESP32-DevKit C 和歐洲環保工具組

> **Important**
>
> 此參考集成託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。我們建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱. 亞馬遜自由 Github 存儲庫遷移指南

> **Note**
>
> 若要探索如何在您自己的 Expressif IDF 專案中整合 FreeRTOS 模組化程式庫和示範，請參閱我們針對 ESP32-C3 平台的精選參考整合。

按照本教程開始使用配備 ESP32-WROOM-32，ESP32-SOLO-1，或 ESP-勞弗模塊和 ESP-勞弗羅弗套件-VB 的意式濃縮咖啡 ESP32-DevKit C。若要在合作夥伴裝置目錄上向我們的 AWS 合作夥伴購買合作夥伴，請使用下列連結：

- ESP32-WROOM-32 DevKit C
- ESP32-SOLO-1
- ESP32-WROVER-KIT

FreeRTOS 支援這些版本的開發板。

如需有關這些主機板最新版本的詳細資訊，請參閱義式咖啡網站上的 ESP32-DevKit C V4 或 ESP-勞弗羅夫套件第 4.1 版。

> **Note**
>
> 目前 ESP32-WROVER-KIT 和 ESP DevKit C 的 FreeRTOS 連接埠不支援對稱式多重處理 (SMP) 功能。

概觀

本教學課程將指引您完成下列步驟：

1. 將主機板連線到主機機器。
2. 在主機機器上安裝軟體以對微控制器主機板的內嵌應用程式進行開發和除錯。
3. 將 FreeRTOS 示範應用程式交叉編譯為二進位映像檔。
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。

必要條件

在您開始使用 FreeRTOS 之前，您必須先設定您的帳戶和權限。 AWS

註冊一個 AWS 帳戶

如果您沒有 AWS 帳戶，請完成以下步驟來建立一個。

若要註冊成為 AWS 帳戶

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

   部分註冊程序需接收來電，並在電話鍵盤輸入驗證碼。

   當您註冊一個時 AWS 帳戶，將創建AWS 帳戶根使用者一個。根使用者有權存取該帳戶中的所有 AWS 服務 和資源。安全性最佳做法是將管理存取權指派給使用者，並僅使用 root 使用者來執行需要 root 使用者存取權的工作。

AWS 註冊過程完成後，會向您發送確認電子郵件。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。

建立具有管理權限的使用者

註冊後，請保護您的 AWS 帳戶 AWS 帳戶根使用者 AWS IAM Identity Center、啟用和建立系統管理使用者，這樣您就不會將 root 使用者用於日常工作。

保護您的 AWS 帳戶根使用者

1. 選擇 Root 使用者並輸入您的 AWS 帳戶 電子郵件地址，以帳戶擁有者身分登入。AWS Management Console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)。

   如需指示，請參閱《IAM 使用者指南》中的為 AWS 帳戶 根使用者啟用虛擬 MFA 裝置 (主控台)。

建立具有管理權限的使用者

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 身分中心中，將管理存取權授予使用者。

   若要取得有關使用 IAM Identity Center 目錄 做為身分識別來源的自學課程，請參閱《使用指南》IAM Identity Center 目錄中的「以預設值設定使用AWS IAM Identity Center 者存取」。

以具有管理權限的使用者身分登入

- 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 身分中心使用者登入的說明，請參閱使用AWS 登入 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 身分中心中，建立遵循套用最低權限許可的最佳做法的權限集。

   如需指示，請參閱《AWS IAM Identity Center 使用指南》中的「建立權限集」。
2. 將使用者指派給群組，然後將單一登入存取權指派給群組。

如需指示，請參閱《AWS IAM Identity Center 使用指南》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：

- 使用者和群組位於 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。請按照 IAM 使用者指南 的 為第三方身分提供者 (聯合) 建立角色 中的指示進行操作。
- 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 中的指示進行操作。
  - (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。

開始使用

> **Note**
>
> 本教學課程中的 Linux 指令需要您使用 Bash 外殼。

1. 設定濃縮咖啡硬體。

   - 如需有關設定 ESP32-DevKit C 開發主機板硬體的詳細資訊，請參閱 ESP32-DevKit C V4 入門指南。
   - 如需有關設定開發板硬體的詳細資訊，請參閱 ESP 開發板硬體的相關資訊，請參閱 ESP-WROVER-KIT 第 4.1 版入門指南。

   > **Important**
   >
   > 當您到達 Espressif 指南的 [開始使用] 區段時，請停止，然後返回此頁面上的指示。

2. 從下載 Amazon FreeRTOS。GitHub如需指示，請參閱 Readme.md 檔案。)

3. 設定您的開發環境。

   若要與主機板通訊，您必須安裝工具鏈。咖啡提供 ESP-IDF 為他們的主板開發軟件。由於 ESP-IDF 已將自己的 FreeRTOS 核心版本整合為一個元件，因此 Amazon FreeRTOS 內含 ESP-IDF v4.2 的自訂版本，該版本已移除了 FreeRTOS 核心。這修復了編譯時重複文件的問題。若要使用 Amazon FreeRTOS 隨附的 ESP-IDF v4.2 的自訂版本，請遵循以下適用於主機作業系統的指示。

   Windows

   1. 下載適用於視窗的 ESP-IDF 通用線上安裝程式。
   2. 執行通用線上安裝程式。
   3. 當您進入下載或使用 ESP-IDF 步驟時，請選取 [使用現有的 ESP-IDF 目錄] 並將 [選擇現有的 ESP-IDF 目錄] 設定為。 *freertos*/vendors/espressif/esp-idf
   4. 完成安裝。

   macOS

   1. 請依照適用於 macOS 的「工具鏈先決條件標準設定」(ESP-IDF 第 4.2 版) 中的指示進行。

      > **Important**
      >
      > 當您到達「後續步驟」下的「取得 ESP-IDF」指示時，請停止，然後返回此頁面上的指示。

   2. 開啟命令列視窗。
   3. 瀏覽至 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
      ```

   4. 使用下列指令，將 ESP-IDF 工具鏈工具新增至終端機的路徑。

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

   Linux

   1. 遵循適用於 Linux 的工具鏈先決條件標準設定 (ESP-IDF v 4.2) 中的指示。

> **Important**
>
> 當您到達「後續步驟」下的「取得 ESP-IDF」指示時，請停止，然後返回此頁面上的指示。

2. 開啟命令列視窗。
3. 瀏覽至 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
```

4. 使用下列指令，將 ESP-IDF 工具鏈工具新增至終端機的路徑。

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

4. 建立序列連線。

   a. 若要在主機與 ESP32-DevKit C 之間建立序列連線，您必須安裝 CP210 轉 UART 橋接器 VCP 驅動程式。您可以從 Silicon Labs 下載這些驅動程式。

      若要在主機與 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 下載此驅動程序。

   b. 請按照以下步驟使用 ESP32 建立串行連接。

   c. 在您建立序連接之後，請記下開發板連接的序列連接埠。你需要它來閃存演示。

設定 FreeRTOS 示範應用程式

在本教學課程中，FreeRTOS 組態檔案位於。*freertos*/vendors/espressif/boards/*board-name*/aws_demos/config_files/FreeRTOSConfig.h(例如，如果AFR_BOARD espressif.esp32_devkitc選擇，則組態檔案位於*freertos*/vendors/espressif/boards/esp32/aws_demos/config_files/FreeRTOSConfig.h。)

1. 如果您執行的是 macOS 或 Linux，請開啟終端機提示。如果您正在運行 Windows，請打開「ESP-IDF 4.x CMD」應用程序（如果您在安裝 ESP-IDF 工具鏈時包含此選項），否則打開「命令提示符」應用程序。
2. 要驗證您是否已安裝 Python3，請運行

```
python --version
```

安裝的版本即會顯示。如果您沒有安裝 Python 3.0.1 或更高版本，則可以從 Python 網站進行安裝。

3. 您需要命 AWS 令列介面 (CLI) 才能執行命 AWS IoT 令。如果您正在執行視窗，請使用`easy_install awscli`指令在「命令」或「ESP-IDF 4.x CMD」應用程式中安裝 AWS CLI。

   如果您執行的是 macOS 或 Linux，請參閱安裝 AWS CLI。

4. 執行

   ```
   aws configure
   ```

   並 AWS 使用您的訪問密鑰 ID，秘密訪問密鑰和默認 AWS 區域配置 AWS CLI。如需詳細資訊，請參閱設定 AWS CLI。

5. 使用下面的命令來安裝開 AWS 發套件的 Python（博托 3）：

   - 在視窗上，在「命令」或「ESP-IDF 4.x CMD」應用程式中執行

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

     > **Note**
     >
     > 有關詳細信息，請參閱 Boto3 文檔。

   - 在 macOS 或 Linux 上執行

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

     然後運行

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

FreeRTOS 包含`SetupAWS.py`腳本，可以更輕鬆地設置您的濃縮咖啡板以連接到。 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

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

thing_name

您要指派給代表您看板之 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。

以下是有效的安全性類型：

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

6. 執行組態指令碼。

a. 如果您執行的是 macOS 或 Linux，請開啟終端機提示。如果您正在執行視窗，請開啟「ESP-IDF 4.x CMD」或「命令」應用程式。

b. 導航到目*freertos*/tools/aws_config_quick_start錄並運行

```
python SetupAWS.py setup
```

指令碼會執行以下操作：

- 建立 IoT 物件、憑證和原則。
- 將 IoT 原則附加至憑證，並將憑證附加至物 AWS IoT 件。
- 使用您的 AWS IoT 端點、Wi-Fi SSID 和憑證填入`aws_clientcredential.h`檔案。
- 格式化您的憑證和私密金鑰，並將其寫入`aws_clientcredential_keys.h`標頭檔案。

> **Note**
>
> 憑證的硬式編碼僅供演示之用。生產層級應用程式必須將這些檔案存放在安全的位置。

如需有關的詳細資訊SetupAWS.py，請參閱*freertos*/tools/aws_config_quick_start目錄README.md中的。

監控雲端的 MQTT 訊息

在執行 FreeRTOS 示範專案之前，您可以在主控台中設定 MQTT 用戶端，以監 AWS IoT 控裝置傳送至雲端的訊息。 AWS

若要向 MQTT 用戶端訂閱 MQTT 主題 AWS IoT

1. 導覽至 AWS IoT 主控台。
2. 在導覽窗格中，選擇 [測試]，然後選擇 [MQTT 測試用戶端]。
3. 在訂閱主題中輸入 *your-thing-name*/example/topic，然後選擇訂閱主題。

當演示項目成功運行在您的設備上，你會看到「你好世界！」 多次發送到您訂閱的主題。

使用 idf.py 指令碼建置、快閃記憶體及執行 FreeRTOS 示範專案

您可以使用 Espressif 的 IDF 實用程序（ idf.py ）來構建項目並將二進製文件刷新到您的設備上。

> **Note**
>
> 某些設定可能需要使用連接埠選項"-p port-name"與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
> ```

在視窗、Linux 和蘋果系統上建置和更新免費軟體 (ESP-IDF 第 4.2 版)

1. 瀏覽至 FreeRTOS 下載目錄的根目錄。

2. 在命令行視窗中，輸入以下命令，以將 ESP-IDF 工具添加到終端機的 PATH。

   視窗 (「命令」應用程式)

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

   視窗 (「ESP-IDF 4.x CMD」應用程式)

   （當您打開應用程序時，這已經完成了。）

   macOS 系統

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

3. 在build目錄中配置 cmake 並使用以下命令構建固件映像。

   ```
   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，構建將生成固件二進製文件 .bin 文件。

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腳本將應用程序二進製文件刷新到您的主板。

```
idf.py flash
```

6. 使用下列指令監控主機板序列埠的輸出。

```
idf.py monitor
```

> **Note**
>
> 您可以結合這些指令，如下列範例所示。
>
> ```
> idf.py erase_flash flash monitor
> ```

對於某些主機設定，您必須在刷新主機板時指定連接埠，如以下範例所示。

```
idf.py erase_flash flash monitor -p /dev/ttyUSB1
```

使用 CMake 構建和閃存 FreeRTOS 務器

除了 IDF SDK 提供的idf.py腳本來構建和運行代碼之外，您還可以使用 CMake 構建項目。目前，它支持 Unix 生成文件或忍者構建系統。

要構建和刷新項目

1. 在命令列視窗中，瀏覽至 FreeRTOS 下載目錄的根目錄。
2. 執行下列指令碼，將 ESP-IDF 工具新增至殼層的路徑。

   Windows

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

macOS 系統

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

3. 輸入以下命令以生成構建文件。

   使用 Unix 生成文件

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

   與忍者

   ```
   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. 建置專案。

   使用 Unix 生成文件

   ```
   make -C ./YOUR_BUILD_DIRECTORY -j8
   ```

   與忍者

   ```
   ninja -C ./YOUR_BUILD_DIRECTORY -j8
   ```

5. 擦除閃光燈，然後閃爍板。

   使用 Unix 生成文件

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

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

   與忍者

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

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

執行低功耗藍牙示範

FreeRTOS 援低功耗藍牙程式庫連線能力。

若要跨藍牙低功耗執行 FreeRTOS 示範專案，您必須在 iOS 或 Android 行動裝置上執行 FreeRTOS 藍牙低功耗行動 SDK 示範應用程式。

若要設定 FreeRTOS 藍牙低功耗行動 SDK 示範應用程式

1. 按照 適用於 FreeRTOS 藍牙裝置的行動 SDK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 免費藍牙低功耗移動 SDK 演示應用程序 中的指示，來在您的行動裝置上設定示範行動應用程式。

如需如何在主機板上透過藍牙低功耗示範執行 MQTT 的指示，請參閱。透過低功耗藍牙執行的 MQTT

如需有關如何在主機板上執行 Wi-Fi 佈建示範的指示，請參閱Wi-Fi 佈建。

在您自己的 ESP32 的 CMake 項目中使用自由服務器

如果您想在自己的 CMake 項目中使用 FreeRTOS，則可以將其設置為子目錄並與應用程序一起構建。首先，從中獲取 FreeRTOS 的副本。GitHub您也可以使用以下命令將其設置為 Git 子模塊，以便 future 更容易更新。

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

如果發行了更新版本，您可以使用這些指令更新本端複本。

```
# 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"
```

如果您的項目具有以下目錄結構：

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

接下來是可用來與 FreeRTOS 一起建置應用程式的最上層CMakeLists.txt檔案範例。

```
cmake_minimum_required(VERSION 3.13)

project(freertos_examples)

# Tell IDF build to link against this target.
set(IDF_EXECUTABLE_SRCS "<complete_path>/src/main.c")
set(IDF_PROJECT_EXECUTABLE my_app)

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

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

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-directory
```

要將應用程序刷新到主板上，請運行以下命令。

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

使用免費伺服器中的元件

執行 CMake 之後，您可以在摘要輸出中找到所有可用的元件。它應該看起來像下面的例子。

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

```
  Version:                  202107.00
  Git version:              202107.00-g79ad6defb

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

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

FreeRTOS modules:
  Modules to build:         backoff_algorithm, common, common_io, core_http,
                            core_http_demo_dependencies, core_json, core_mqtt,
                            core_mqtt_agent, core_mqtt_agent_demo_dependencies,
                            core_mqtt_demo_dependencies, crypto, defender, dev_mode_key_
                            provisioning, device_defender, device_defender_demo_
                            dependencies, device_shadow,
 device_shadow_demo_dependencies,
                            freertos_cli_plus_uart, freertos_plus_cli, greengrass,
                            http_demo_helpers, https, jobs, jobs_demo_dependencies,
                            kernel, logging, mqtt, mqtt_agent_interface, mqtt_demo_
                            helpers, mqtt_subscription_manager, ota, ota_demo_
                            dependencies, ota_demo_version, pkcs11, pkcs11_helpers,
                            pkcs11_implementation, pkcs11_utils, platform,
 secure_sockets,
                            serializer, shadow, tls, transport_interface_secure_sockets,
                            wifi
  Enabled by user:          common_io, core_http_demo_dependencies, core_json,
                            core_mqtt_agent_demo_dependencies, core_mqtt_demo_
                            dependencies, defender, device_defender,
 device_defender_demo_
                            dependencies, device_shadow,
 device_shadow_demo_dependencies,
                            freertos_cli_plus_uart, freertos_plus_cli, greengrass,
 https,
```

```
                            jobs, jobs_demo_dependencies, logging,
 ota_demo_dependencies,
                            pkcs11, pkcs11_helpers, pkcs11_implementation, pkcs11_utils,
                            platform, secure_sockets, shadow, wifi
  Enabled by dependency:    backoff_algorithm, common, core_http, core_mqtt,
                            core_mqtt_agent, crypto, demo_base,
 dev_mode_key_provisioning,
                            freertos, http_demo_helpers, kernel, mqtt, mqtt_agent_
                            interface, mqtt_demo_helpers, mqtt_subscription_manager,
 ota,
                            ota_demo_version, pkcs11_mbedtls, serializer, tls,
                            transport_interface_secure_sockets, utils
  3rdparty dependencies:    jsmn, mbedtls, pkcs11, tinycbor
  Available demos:          demo_cli_uart, demo_core_http, demo_core_mqtt,
 demo_core_mqtt_
                            agent, demo_device_defender, demo_device_shadow,
                            demo_greengrass_connectivity, demo_jobs, demo_ota_core_http,
                            demo_ota_core_mqtt, demo_tcp
  Available tests:
=============================================================================
```

您可以參考清單中的任何元Modules to build件。若要將它們連結到您的應用程式中，請將AFR::命名空間放在名稱前面AFR::core_mqtt，例如AFR::ota、、等等。

使用 ESP-IDF 加入自訂元件

您可以在使用 ESP-IDF 時加入更多元件。例如，假設你想要新增一個名為 example_component 的元件，並且你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CMakeLists.txt檔案的範例。

```
add_library(example_component src/example_component.c)
target_include_directories(example_component PUBLIC include)
```

然後，在頂層CMakeLists.txt文件中，通過在後面插入以下行來添加組件add_subdirectory(freertos)。

```
add_subdirectory(component/example_component)
```

然後，修改target_link_libraries以包含元件。

```
target_link_libraries(my_app PRIVATE AFR::core_mqtt PRIVATE example_component)
```

根據預設，此元件現在會自動連結至您的應用程式碼。現在，您可以包含其頭文件並調用它定義的函數。

覆寫免費伺服器的組態

目前沒有明確定義的方法來重新定義 FreeRTOS 源代碼樹之外的配置。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，在您的頂級CMakeLists.txt文件中，添加此行之前，add_subdirectory(freertos)以便編譯器首先搜索此目錄。

```
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
```

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

如需詳細資訊，請參閱 Espressif API 參考中的專案組態，如果您在成功編譯之後遇到問題，請參閱該頁面上已停用的選項及其取代部分。

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。

```
cmake_minimum_required(VERSION 3.13)

project(freertos_examples)

set(IDF_PROJECT_EXECUTABLE my_app)
set(IDF_EXECUTABLE_SRCS "src/main.c")

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

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

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

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

```
add_subdirectory(freertos)

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

故障診斷

- 如果您執行的是 macOS，但作業系統無法辨識您的 ESP-WROVER-KIT，請確定您並未安裝 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 提供的監視器公用程式 (並使用 make 監視器叫用) 可協助您解碼位址。出於這個原因，它可以幫助您在應用程序停止工作時獲得一些有意義的回溯。如需詳細資訊，請參閱 Espressif 網站上的自動位址解碼。
- 也可以啟用 GDBStub 與 gdb 進行通信，而不需要任何特殊的 JTAG 硬件。如需詳細資訊，請參閱在濃縮咖啡網站上使用 GDBStub 啟動 GDB。
- 如需有關在需要 JTAG 硬體型偵錯時設定 OpenOCD 環境的資訊，請參閱 Espressif 網站上的 JTAG 偵錯。
- 如果`pyserial`無法`pip`在 macOS 上使用安裝，請從 pyserial 網站下載。
- 如果主機板持續重設，請在終端機上輸入以下指令來刪除閃光燈。

  ```
  make erase_flash
  ```

- 如果您在執行 `idf_monitor.py` 時看到錯誤，請使用 Python 2.7。
- 來自 ESP-IDF 的所需庫包含在 FreeRTOS 中，因此無需從外部下載它們。如果已設定`IDF_PATH`環境變數，建議您在建置 FreeRTOS 之前先清除它。
- 在 Windows 上，系統可能需要 3-4 分鐘來建置專案。為了減少構建時間，您可以使用 make 命令上的`-j4`開關。

  ```
  make flash monitor -j4
  ```

- 如果您的裝置無法連線到 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[]`應該看起來像`1234567890123-ats.iot.us-east-1.amazonaws.com`。
- 如果您遵循 在您自己的 ESP32 的 CMake 項目中使用自由服務器 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 版本 4.2 支持使用的特殊\-ESP32\-精靈\-gcc 8\ .2\ .0\。 工具鏈。如果您使用的是較早版本的 Xtensa 工具鏈，請下載所需的版本。
- 如果您看到類似以下有關 ESP-IDF v4.2 未滿足的 python 依賴關係的錯誤日誌：

```
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
```

使用以下 Python 命令在您的平台上安裝 python 依賴關係：

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

如需疑難排解詳細資訊，請參閱故障診斷入門。

除錯

調試代碼壓縮 ESP32-DevKit C 和 ESP-勞弗羅弗套件（ESP-IDF V4.2）

本節說明如何使用 ESP-IDF v4.2 來除錯壓縮式硬體。您需要 JTAG 對 USB 纜線。我們使用 USB 轉 MPSE 電纜（例如，FTDI C232HM-DDHSL-0）。

ESP-DevKit C 安裝

對於 FTDI C232HM-DDHSL-0 電纜，這些是與 ESP32 開發套件 C 的連接。

```
| C232HM-DDHSL-0 Wire Color | ESP32 GPIO Pin | JTAG Signal Name |
| ------------------------- | -------------- | ---------------- |
|  Brown (pin 5)            |  IO14          |  TMS             |
|  Yellow (pin 3)           |  IO12          |  TDI             |
|  Black (pin 10)           |  GND           |  GND             |
```

```
|  Orange (pin 2)           |  IO13          |  TCK             |
|  Green (pin 4)            |  IO15          |  TDO             |
```

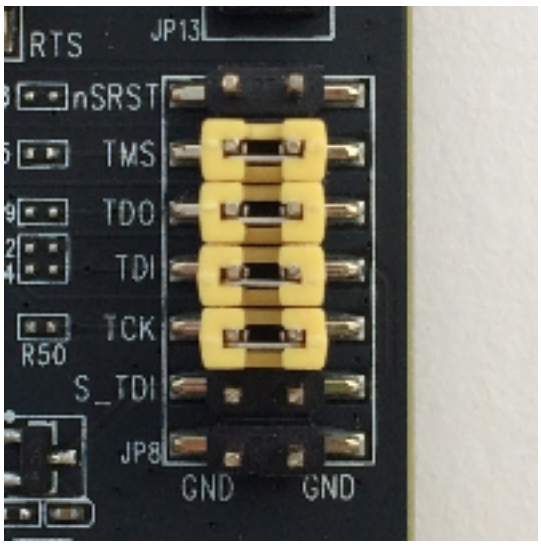
ESP-WROVER-KIT JTAG 設定

對於 FTDI C232HM-DDHSL-0 電纜，這些是與 ESP32-WROVER-KIT 的連接。

```
| C232HM-DDHSL-0 Wire Color | ESP32 GPIO Pin | JTAG Signal Name |
| ------------------------- | -------------- | ---------------- |
|  Brown (pin 5)            |  IO14          |  TMS             |
|  Yellow (pin 3)           |  IO12          |  TDI             |
|  Orange (pin 2)           |  IO13          |  TCK             |
|  Green (pin 4)            |  IO15          |  TDO             |
```

這些資料表是從 FTDI C232HM-DDHSL-0 datasheet 開發而來。如需詳細資訊，請參閱產品資料表中的「C232HM MPSE 纜線連接和機械詳細資料」一節。

若要在 ESP 套件上啟用 JTAG，請將跳接器放置在 TMS、TDO、TDI、TCK 和 S_TDI 接腳上，如下所示。



在視窗上偵錯 (ESP-IDF 第 4.2 版)

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，而另一端則按調試代碼壓縮 ESP32-DevKit C 和 ESP-勞弗羅弗套件（ESP-IDF V4.2）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，以滑鼠右鍵按一下 C232HM-DDHSL-0 裝置，然後選擇 [內容]。

   > **Note**
   >
   > 裝置可能會列為 USB Serial Port (USB 序列連接埠)。

   若要查看裝置的內容，請在內容視窗中選擇 [詳細資料] 索引標籤。如果未列出該裝置，請安裝適用於 FTDI C232HM-DDHSL-0 的視窗驅動程式。

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)。您應該在「值」欄位中看到類似的內容。

   ```
   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。ID 在一行中指定，開頭`ftdi_vid_pid`後跟供應商 ID 和產品 ID。

   ```
   ftdi_vid_pid 0x0403 0x6014
   ```

4. 下載 OpenOCD for Windows。
5. 將檔案解壓縮至 `C:\`，並新增 `C:\openocd-esp32\bin` 到您的系統路徑。
6. OpenOCD 需要 libusb，但預設不會在 Windows 中安裝。若要安裝 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. 從下拉式功能表中選擇 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. 對於目標驅動程式欄位下的清單，請選擇箭頭，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，瀏覽至 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，瀏覽至 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，瀏覽至 FreeRTOS 下載目錄的根目錄，然後等到示範開始在主機板上執行。當它這樣做時，運行

```
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 上進行除錯 (ESP-IDF 第 4.2 版)

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. 使用以下命令在 macOS `libusb` 上安裝。

```
brew install libusb
```

5. 使用下列指令卸載序列埠驅動程式。

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

6. 使用下列指令卸載序列埠驅動程式。

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

7. 如果您執行的 macOS 版本低於 10.9，請使用以下指令卸載 Apple FTDI 驅動程式。

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

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

   ```
   system_profiler SPUSBDataType
   ```

   來自的輸出`system_profiler`應如下所示。

   ```
   DEVICE:

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

9. 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。
10. 開啟終端機視窗，瀏覽至 FreeRTOS 下載目錄的根目錄，然後使用下列命令執行 OpenOCD。

    ```
    idf.py openocd
    ```

    保持此終端機視窗開啟。
11. 開啟新終端機，並使用以下指令載入 FTDI 序列埠驅動程式。

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

12. 瀏覽至 FreeRTOS 下載目錄的根目錄，然後執行

    ```
    idf.py flash monitor
    ```

13. 開啟另一個新的終端機，瀏覽至 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。

在 Linux 上進行除錯 (ESP-IDF 第 4.2 版)

1. 下載 OpenOCD。解壓縮 tarball 並遵循讀我檔中的安裝指示。

2. 使用以下命令在 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。這可協助您檢查主機板的 USB 連接埠是否已被作業系統辨識。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。請確定群組擁有者已從變更dialout為plugdev。

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

   數字小的 /dev/ttyUSBn 界面用於 JTAG 通訊。另一個介面會路由至 ESP32 的序列埠 (UART)，用於將程式碼上傳至 ESP32 的快閃記憶體。

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

```
idf.py openocd
```

6. 開啟另一個終端機，瀏覽至 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，瀏覽 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。

開始使用意式濃縮咖啡 ESP32-WROOM-32SE

> **Important**
>
> 此參考集成託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。我們建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱. 亞馬遜自由 Github 存儲庫遷移指南

> **Note**
>
> - 若要探索如何在您自己的 Expressif IDF 專案中整合 FreeRTOS 模組化程式庫和示範，請參閱我們針對 ESP32-C3 平台的精選參考整合。
> - 目前 ESP32-WROOM-32SE 的 FreeRTOS 連接埠不支援對稱式多重處理 (SMP) 功能。

本教程向您展示如何開始使用濃縮咖啡 ESP32-WROOM-32SE。若要在合作夥伴裝置目錄上向我們的 AWS 合作夥伴購買，請參閱 ESP32-WROOM-32SE。

概觀

本教學課程將指引您完成下列步驟：

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面板的內嵌應用程式。
3. 將 FreeRTOS 示範應用程式交叉編譯成二進位映像檔。
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. 使用序列連線來監控和偵錯執行中的應用程式。

必要條件

在您開始使用 FreeRTOS 之前，您必須先設定您的帳戶和權限。 AWS

註冊一個 AWS 帳戶

如果您沒有 AWS 帳戶，請完成以下步驟來建立一個。

若要註冊成為 AWS 帳戶

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

   部分註冊程序需接收來電，並在電話鍵盤輸入驗證碼。

   當您註冊一個時 AWS 帳戶，將創建AWS 帳戶根使用者一個。根使用者有權存取該帳戶中的所有 AWS 服務 和資源。安全性最佳做法是將管理存取權指派給使用者，並僅使用 root 使用者來執行需要 root 使用者存取權的工作。

AWS 註冊過程完成後，會向您發送確認電子郵件。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。

建立具有管理權限的使用者

註冊後，請保護您的 AWS 帳戶 AWS 帳戶根使用者 AWS IAM Identity Center、啟用和建立系統管理使用者，這樣您就不會將 root 使用者用於日常工作。

保護您的 AWS 帳戶根使用者

1. 選擇 Root 使用者並輸入您的 AWS 帳戶 電子郵件地址，以帳戶擁有者身分登入。AWS Management Console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)。

   如需指示，請參閱《IAM 使用者指南》中的為 AWS 帳戶 根使用者啟用虛擬 MFA 裝置 (主控台)。

建立具有管理權限的使用者

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 身分中心中，將管理存取權授予使用者。

   若要取得有關使用 IAM Identity Center 目錄 做為身分識別來源的自學課程，請參閱《使用指南》IAM Identity Center 目錄中的「以預設值設定使用AWS IAM Identity Center 者存取」。

以具有管理權限的使用者身分登入

- 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 身分中心使用者登入的說明，請參閱使用AWS 登入 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 身分中心中，建立遵循套用最低權限許可的最佳做法的權限集。

   如需指示，請參閱《AWS IAM Identity Center 使用指南》中的「建立權限集」。
2. 將使用者指派給群組，然後將單一登入存取權指派給群組。

   如需指示，請參閱《AWS IAM Identity Center 使用指南》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：

- 使用者和群組位於 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。請按照 IAM 使用者指南 的 為第三方身分提供者 (聯合) 建立角色 中的指示進行操作。
- 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 中的指示進行操作。
  - (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。

開始使用

> **Note**
>
> 本教學課程中的 Linux 指令需要您使用 Bash 外殼。

1. 設定濃縮咖啡硬體。

   如需有關設定 ESP32-WROOM-32SE 開發主機板硬體的詳細資訊，請參閱 ESP32-DevKit C V4 入門指南。

   > **Important**
   >
   > 當您到達指南的逐步安裝部分時，請按照步驟操作，直到完成步驟 4（設置環境變量）。完成步驟 4 後，請停止，然後按照此處的剩餘步驟進行操作。

2. 從下載 Amazon FreeRTOS。GitHub(如需相關指示，請參閱 Re adme.md 檔案。)
3. 設定您的開發環境。

   若要與主機板通訊，您必須安裝工具鏈。咖啡提供 ESP-IDF 為他們的主機板開發軟體。由於 ESP-IDF 將自己的 FreeRTOS 核心版本整合為一個元件，因此 Amazon FreeRTOS 包含 ESP-IDF v4.2 的自訂版本，該版本已移除了 FreeRTOS 核心。這修復了編譯時重複文件的問題。若要使用 Amazon FreeRTOS 隨附的 ESP-IDF v4.2 的自訂版本，請遵循以下適用於主機作業系統的指示。

   Windows

   1. 下載適用於視窗的 ESP-IDF 通用線上安裝程式。
   2. 執行通用線上安裝程式。
   3. 當您進入下載或使用 ESP-IDF 步驟時，請選取 [使用現有的 ESP-IDF 目錄] 並將 [選擇現有的 ESP-IDF 目錄] 設定為。 `freertos/vendors/espressif/esp-idf`
   4. 完成安裝。

   macOS

   1. 請依照適用於 macOS 的「工具鏈先決條件標準設定」(ESP-IDF 第 4.2 版) 中的指示進行。

      > **Important**
      >
      > 當您到達「後續步驟」下的「取得 ESP-IDF」指示時，請停止，然後返回此頁面上的指示。

   2. 開啟命令列視窗。

3. 瀏覽至 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
```

4. 使用下列指令，將 ESP-IDF 工具鏈工具新增至終端機的路徑。

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

Linux

1. 遵循適用於 Linux 的工具鏈先決條件標準設定 (ESP-IDF v 4.2) 中的指示。

> **Important**
>
> 當您到達「後續步驟」下的「取得 ESP-IDF」指示時，請停止，然後返回此頁面上的指示。

2. 開啟命令列視窗。
3. 瀏覽至 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
```

4. 使用下列指令，將 ESP-IDF 工具鏈工具新增至終端機的路徑。

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

4. 建立序列連線。

   a. 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。

   b. 請依照步驟來建立與 ESP32 的序列連線。

   c. 在您建立序連接之後，請記下開發板連接的序列連接埠。你需要它來閃存演示。

設定 FreeRTOS 示範應用程式

在本教學課程中，FreeRTOS 組態檔案位於。*freertos*/vendors/espressif/boards/*board-name*/aws_demos/config_files/FreeRTOSConfig.h(例如，如果AFR_BOARD espressif.esp32_devkitc選擇，則組態檔案位於*freertos*/vendors/espressif/boards/esp32/aws_demos/config_files/FreeRTOSConfig.h。)

> **Important**
>
> ATECC608A 設備具有一次性初始化，在第一次運行項目時（在呼叫期間C_InitToken）將其鎖定到設備上。不過，FreeRTOS 示範專案和測試專案有不同的組態。如果裝置在示範專案組態期間鎖定，並非測試專案中的所有測試都會成功。

1. 依照中的步驟設定 FreeRTOS 示範專案。設定 FreeRTOS 範當您進入最後一個步驟若要格式化您的 AWS IoT 認證，請停止並執行下列步驟。
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/example_trust_chain_tool 目錄並開啟 README.md 檔案。
3. 若要佈建您的裝置，請依照README.md檔案中的指示進行。這些步驟如下：
   1. 使用建立並註冊憑證授權單位 AWS。
   2. 在 ATECC608A 上產生您的金鑰，並匯出公有金鑰和裝置序號。
   3. 產生裝置的憑證，並使用註冊該憑證 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。

監控雲端上的 MQTT 訊息 AWS

在執行 FreeRTOS 示範專案之前，您可以在主控台中設定 MQTT 用戶端，以監 AWS IoT 控裝置傳送至雲端的訊息。 AWS

若要向 MQTT 用戶端訂閱 MQTT 主題 AWS IoT

1. 登入 AWS IoT 主控台。
2. 在導覽窗格中，選擇 [測試]，然後選擇 [MQTT 測試用戶端]。
3. 在訂閱主題中，輸入，*your-thing-name*/example/topic然後選擇訂閱主題。

使用 idf.py 指令碼建置、快閃記憶體及執行 FreeRTOS 示範專案

您可以使用 Espressif 的 IDF 實用程序（`idf.py`）生成構建文件，構建應用程序二進製文件，並將二進製文件刷新到您的設備上。

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

在視窗、Linux 和蘋果系統上建置和更新免費軟體 (ESP-IDF 第 4.2 版)

1. 瀏覽至 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：

   視窗 (「命令」應用程式)

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

   視窗 (「ESP-IDF 4.x CMD」應用程式)

   （當您打開應用程序時，這已經完成了。）

   macOS 系統

   ```
   source vendors/espressif/esp-idf/export.sh
   ```
3. 在`build`目錄中配置 cmake 並使用以下命令構建固件映像。

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

   你應該看到像這樣的輸出下面的例子。

   ```
     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，構建將生成固件二進製文件 .bin 文件。

4. 使用以下命令清除開發板的快閃記憶體。

```
idf.py erase_flash
```

5. 使用該`idf.py`腳本將應用程序二進製文件刷新到您的主板。

```
idf.py flash
```

6. 使用下列指令監控主機板序列埠的輸出。

```
idf.py monitor
```

> **Note**
>
> - 您可以結合這些指令，如下列範例所示。
>
>   ```
>   idf.py erase_flash flash monitor
>   ```
>
> - 對於某些主機設定，您必須在刷新主機板時指定連接埠，如下列範例所示。
>
>   ```
>   idf.py erase_flash flash monitor -p /dev/ttyUSB1
>   ```

使用 CMake 構建和閃存 FreeRTOS 務器

除了使用 IDF SDK 提供的`idf.py`腳本來構建和運行代碼之外，您還可以使用 CMake 構建項目。目前它支持 Unix 生成文件和忍者構建系統。

要構建和刷新項目

1. 在命令列視窗中，瀏覽至 FreeRTOS 下載目錄的根目錄。
2. 執行下列指令碼，將 ESP-IDF 工具新增至殼層的路徑中。

   Windows

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

   macOS 系統

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

3. 輸入以下命令以生成構建文件。

   使用 Unix 生成文件

   ```
   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
   ```

   與忍者

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

4. 擦除閃光燈，然後閃爍板。

   使用 Unix 生成文件

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

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

與忍者

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

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

其他資訊

如需使用及疑難排解 Espressif ESP32 主機板的相關資訊，請參閱下列主題：

- 在您自己的 ESP32 的 CMake 項目中使用自由服務器
- 故障診斷
- 除錯

開始使用濃縮咖啡 ESP32-S2

> **Important**
>
> 此參考集成託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。我們建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱. 亞馬遜自由 Github 存儲庫遷移指南

> **Note**
>
> 若要探索如何在您自己的 Expressif IDF 專案中整合 FreeRTOS 模組化程式庫和示範，請參閱我們針對 ESP32-C3 平台的精選參考整合。

本教程向您展示如何開始使用濃縮咖啡 ESP32-S2 SoC 和 ES P32-S2-索拉 -1 開發板。

概觀

本教學課程將指引您完成下列步驟：

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面板的內嵌應用程式。
3. 將 FreeRTOS 示範應用程式交叉編譯為二進位映像檔。
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. 使用序列連線監控執行中的應用程式並進行偵錯。

必要條件

在您開始使用 FreeRTOS 之前，您必須先設定您的帳戶和權限。 AWS

註冊一個 AWS 帳戶

如果您沒有 AWS 帳戶，請完成以下步驟來建立一個。

若要註冊成為 AWS 帳戶

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

   部分註冊程序需接收來電，並在電話鍵盤輸入驗證碼。

   當您註冊一個時 AWS 帳戶，將創建AWS 帳戶根使用者一個。根使用者有權存取該帳戶中的所有 AWS 服務 和資源。安全性最佳做法是將管理存取權指派給使用者，並僅使用 root 使用者來執行需要 root 使用者存取權的工作。

AWS 註冊過程完成後，會向您發送確認電子郵件。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。

建立具有管理權限的使用者

註冊後，請保護您的 AWS 帳戶 AWS 帳戶根使用者 AWS IAM Identity Center、啟用和建立系統管理使用者，這樣您就不會將 root 使用者用於日常工作。

保護您的 AWS 帳戶根使用者

1. 選擇 Root 使用者並輸入您的 AWS 帳戶 電子郵件地址，以帳戶擁有者身分登入。AWS Management Console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)。

如需指示，請參閱《IAM 使用者指南》中的為 AWS 帳戶 根使用者啟用虛擬 MFA 裝置 (主控台)。

建立具有管理權限的使用者

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 身分中心中，將管理存取權授予使用者。

   若要取得有關使用 IAM Identity Center 目錄 做為身分識別來源的自學課程，請參閱《使用指南》IAM Identity Center 目錄中的「以預設值設定使用AWS IAM Identity Center 者存取」。

以具有管理權限的使用者身分登入

- 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 身分中心使用者登入的說明，請參閱使用AWS 登入 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 身分中心中，建立遵循套用最低權限許可的最佳做法的權限集。

   如需指示，請參閱《AWS IAM Identity Center 使用指南》中的「建立權限集」。
2. 將使用者指派給群組，然後將單一登入存取權指派給群組。

   如需指示，請參閱《AWS IAM Identity Center 使用指南》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：

- 使用者和群組位於 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。請按照 IAM 使用者指南 的 為第三方身分提供者 (聯合) 建立角色 中的指示進行操作。

- 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 中的指示進行操作。
  - (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。

開始使用

> **Note**
>
> 本教學課程中的 Linux 指令需要您使用 Bash 外殼。

1. 設定濃縮咖啡硬體。

   如需有關設定 ESP32-S2 開發主機板硬體的詳細資訊，請參閱 ESP32-S2-S2-S A2-相關入門指南。

   > **Important**
   >
   > 當您到達 Espressif 指南的 [開始使用] 區段時，請停止，然後返回此頁面上的指示。

2. 從下載 Amazon FreeRTOS。GitHub如需指示，請參閱 Readme.md 檔案。)
3. 設定您的開發環境。

   若要與主機板通訊，您必須安裝工具鏈。咖啡提供 ESP-IDF 為他們的主板開發軟件。由於 ESP-IDF 將自己的 FreeRTOS 核心版本整合為一個元件，因此 Amazon FreeRTOS 包含 ESP-IDF v4.2 的自訂版本，該版本已移除了 FreeRTOS 核心。這修復了編譯時重複文件的問題。若要使用 Amazon FreeRTOS 隨附的 ESP-IDF v4.2 的自訂版本，請遵循以下適用於主機作業系統的指示。

   Windows

   1. 下載適用於視窗的 ESP-IDF 通用線上安裝程式。
   2. 執行通用線上安裝程式。
   3. 當您進入下載或使用 ESP-IDF 步驟時，請選取 [使用現有的 ESP-IDF 目錄] 並將 [選擇現有的 ESP-IDF 目錄] 設定為。 *freertos*/vendors/espressif/esp-idf
   4. 完成安裝。

macOS

1. 請依照適用於 macOS 的「工具鏈先決條件標準設定」(ESP-IDF 第 4.2 版) 中的指示進行。

   > **Important**
   >
   > 當您到達「後續步驟」下的「取得 ESP-IDF」指示時，請停止，然後返回此頁面上的指示。

2. 開啟命令列視窗。
3. 瀏覽至 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
   ```

4. 使用下列指令，將 ESP-IDF 工具鏈工具新增至終端機的路徑。

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

Linux

1. 遵循適用於 Linux 的工具鏈先決條件標準設定 (ESP-IDF v 4.2) 中的指示。

   > **Important**
   >
   > 當您到達「後續步驟」下的「取得 ESP-IDF」指示時，請停止，然後返回此頁面上的指示。

2. 開啟命令列視窗。
3. 瀏覽至 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
   ```

4. 使用下列指令，將 ESP-IDF 工具鏈工具新增至終端機的路徑。

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

4. 建立序列連線。

   a. 若要在主機與 ESP32-DevKit C 之間建立序列連線，請安裝 CP210 轉 UART 橋接器 VCP 驅動程式。您可以從 Silicon Labs 下載這些驅動程式。

   b. 請依照步驟來建立與 ESP32 的序列連線。

   c. 在您建立序連接之後，請記下開發板連接的序列連接埠。你需要它來閃存演示。

設定 FreeRTOS 示範應用程式

在本教學課程中，FreeRTOS 組態檔案位於。*freertos*/vendors/espressif/boards/*board-name*/aws_demos/config_files/FreeRTOSConfig.h(例如，如果AFR_BOARD espressif.esp32_devkitc選擇，則組態檔案位於*freertos*/vendors/espressif/boards/esp32/aws_demos/config_files/FreeRTOSConfig.h。)

1. 如果您執行的是 macOS 或 Linux，請開啟終端機提示。如果您正在運行 Windows，請打開「ESP-IDF 4.x CMD」應用程序（如果您在安裝 ESP-IDF 工具鏈時包含此選項），否則打開「命令提示符」應用程序。
2. 要驗證您是否已安裝 Python3，請運行以下命令：

   ```
   python --version
   ```

   安裝的版本即會顯示。如果您沒有安裝 Python 3.0.1 或更高版本，則可以從 Python 網站進行安裝。
3. 您需要命 AWS 令列介面 (CLI) 才能執行命 AWS IoT 令。如果您正在執行視窗，請使用easy_install awscli指令在「命令」或「ESP-IDF 4.x CMD」應用程式中安裝 AWS CLI。

   如果您執行的是 macOS 或 Linux，請參閱安裝 AWS CLI。
4. 執行

   ```
   aws configure
   ```

   並 AWS 使用您的訪問密鑰 ID，秘密訪問密鑰和默認 AWS 區域配置 AWS CLI。如需詳細資訊，請參閱設定 AWS CLI。
5. 使用下面的命令來安裝開 AWS 發套件的 Python（博托 3）：

- 在視窗上，在「命令」或「ESP-IDF 4.x CMD」應用程式中執行

```
easy_install boto3
```

- 在 macOS 或 Linux 上執行

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

然後運行

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

FreeRTOS 包含SetupAWS.py腳本，可以更輕鬆地設置您的濃縮咖啡板以連接到。 AWS IoT

執行組態指令碼

1. 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

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

   thing_name

   您要指派給代表您看板之 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。以下是有效的安全性類型：

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

- eWiFiSecurityWPA2 (WPA2 安全性)

2. 如果您執行的是 macOS 或 Linux，請開啟終端機提示。如果您正在執行視窗，請開啟「ESP-IDF 4.x CMD」或「命令」應用程式。
3. 導航到目*freertos*/tools/aws_config_quick_start錄並運行

```
python SetupAWS.py setup
```

指令碼會執行以下操作：

- 建立 AWS IoT 物件、憑證和原則。
- 將 AWS IoT 原則附加至憑證，並將憑證附加至 AWS IoT 物件。
- 使用您的 AWS IoT 端點、Wi-Fi SSID 和憑證填入aws_clientcredential.h檔案。
- 格式化您的憑證和私密金鑰，並將其寫入aws_clientcredential_keys.h標頭檔案。

> **Note**
> 憑證的硬式編碼僅供演示之用。生產層級應用程式必須將這些檔案存放在安全的位置。

如需有關的詳細資訊SetupAWS.py，請參閱*freertos*/tools/aws_config_quick_start目錄README.md中的。

在雲端監控 MQTT 訊息 AWS

在執行 FreeRTOS 示範專案之前，您可以在主控台中設定 MQTT 用戶端，以監 AWS IoT 控裝置傳送至雲端的訊息。 AWS

若要向 MQTT 用戶端訂閱 MQTT 主題 AWS IoT

1. 登入 AWS IoT 主控台。
2. 在導覽窗格中，選擇 [測試]，然後選擇 [MQTT 測試用戶端]。
3. 在訂閱主題中輸入 *your-thing-name*/example/topic，然後選擇訂閱主題。

當演示項目成功運行在您的設備上，你會看到「你好世界！」 多次發送到您訂閱的主題。

使用 idf.py 指令碼建置、快閃記憶體及執行 FreeRTOS 示範專案

您可以使用 Espressif 的 IDF 實用程序來生成構建文件，構建應用程序二進製文件，並刷新您的主板。

在視窗、Linux 和蘋果系統上建置和更新免費軟體 (ESP-IDF 第 4.2 版)

使用該`idf.py`腳本構建項目並將二進製文件刷新到設備上。

> **Note**
>
> 某些設定可能需要使用連接埠選項`-p port-name`與`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
> ```

要構建和刷新項目

1. 瀏覽至 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：

   視窗 (「命令」應用程式)

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

   視窗 (「ESP-IDF 4.x CMD」應用程式)

   （當您打開應用程序時，這已經完成了。）

   macOS 系統

   ```
   source vendors/espressif/esp-idf/export.sh
   ```
3. 在`build`目錄中配置 cmake 並使用以下命令構建固件映像。

   ```
   idf.py -DVENDOR=espressif -DBOARD=esp32s2_saola_1 -DCOMPILER=xtensa-esp32s2 build
   ```

   你應該看到像這樣的輸出下面的例子。

   ```
   Executing action: all (aliases: build)
   ```

```
   Running cmake in directory /path/to/hello_world/build
   Executing "cmake -G Ninja -DPYTHON_DEPS_CHECKED=1 -DESP_PLATFORM=1
 -DVENDOR=espressif -DBOARD=esp32s2_saola_1 -DCOMPILER=xtensa-esp32s2 -
DCCACHE_ENABLE=0 /path/to/hello_world"...
   -- The C compiler identification is GNU 8.4.0
   -- The CXX compiler identification is GNU 8.4.0
   -- The ASM compiler identification is GNU

   ... (more lines of build system output)

   [1628/1628] Generating binary image from built executable
   esptool.py v3.0
   Generated /path/to/hello_world/build/aws_demos.bin

   Project build complete. To flash, run this command:
   esptool.py -p (PORT) -b 460800 --before default_reset --after hard_reset --chip
 esp32s2  write_flash --flash_mode dio --flash_size detect --flash_freq 80m 0x1000
 build/bootloader/bootloader.bin 0x8000 build/partition_table/partition-table.bin
 0x16000 build/ota_data_initial.bin 0x20000 build/aws_demos.bin
   or run 'idf.py -p (PORT) flash'
```

如果沒有錯誤，則構建將生成固件二進製文件 .bin 文件。

4. 使用以下命令清除開發板的快閃記憶體。

```
idf.py erase_flash
```

5. 使用該`idf.py`腳本將應用程序二進製文件刷新到您的主板。

```
idf.py flash
```

6. 使用下列指令監控主機板序列埠的輸出。

```
idf.py monitor
```

> **Note**
>
> - 您可以結合這些指令，如下列範例所示。
>
>   ```
>   idf.py erase_flash flash monitor
>   ```
>
> - 對於某些主機設定，您必須在刷新主機板時指定連接埠，如下列範例所示。

```
idf.py erase_flash flash monitor -p /dev/ttyUSB1
```

使用 CMake 構建和閃存 FreeRTOS 務器

除了使用 IDF SDK 提供的`idf.py`腳本來構建和運行代碼之外，您還可以使用 CMake 構建項目。目前它支持 Unix 生成文件和忍者構建系統。

要構建和刷新項目

1. 在命令列視窗中，瀏覽至 FreeRTOS 下載目錄的根目錄。
2. 執行下列指令碼，將 ESP-IDF 工具新增至殼層的路徑。
   - Windows

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

   - macOS 系統

     ```
     source vendors/espressif/esp-idf/export.sh
     ```
3. 輸入以下命令以生成構建文件。
   - 使用 Unix 生成文件

     ```
     cmake -DVENDOR=espressif -DBOARD=esp32s2_saola_1 -DCOMPILER=xtensa-esp32s2 -S . -
     B ./YOUR_BUILD_DIRECTORY -DAFR_ENABLE_ALL_MODULES=1 -DAFR_ENABLE_TESTS=0
     ```

   - 與忍者

     ```
     cmake -DVENDOR=espressif -DBOARD=esp32s2_saola_1 -DCOMPILER=xtensa-esp32s2 -S . -
     B ./YOUR_BUILD_DIRECTORY -DAFR_ENABLE_ALL_MODULES=1 -DAFR_ENABLE_TESTS=0 -GNinja
     ```
4. 建置專案。
   - 使用 Unix 生成文件

     ```
     make -C ./YOUR_BUILD_DIRECTORY -j8
     ```

   - 與忍者

```
ninja -C ./YOUR_BUILD_DIRECTORY -j8
```

5. 擦除閃光燈，然後閃爍板。

   - 使用 Unix 生成文件

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

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

   - 與忍者

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

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

其他資訊

如需使用及疑難排解 Espressif ESP32 主機板的相關資訊，請參閱下列主題：

- 在您自己的 ESP32 的 CMake 項目中使用自由服務器
- 故障診斷
- 除錯

Infineon XMC4800 IoT Connectivity Kit 入門

> **Important**
>
> 此參考集成託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。我們建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱. 亞馬遜自由 Github 存儲庫遷移指南

本教學課程提供 Infineon XMC4800 IoT 連線套件入門指示。如果您沒有英飛凌 XMC4800 IoT 連接套件，請訪問 AWS 合作夥伴設備目錄，向我們的合作夥伴購買一個。

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。

在開始之前，您必須先設定 FreeRTOS AWS IoT 並下載，才能將裝置連線到雲端 AWS 。如需說明，請參閱 首要步驟。在此教學課程中，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. 將 FreeRTOS 示範應用程式交叉編譯為二進位映像檔。
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 使用英飛凌的 DAVE 開發環境對 XMC4800 進行程式設計。開始之前，您需要下載並安裝 DAVE 和一些 J-Link 驅動程式，以與內建除錯器通訊。

安裝 DAVE

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

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

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

如需詳細資訊，請參閱 DAVE Quick Start Guide。

安裝 Segger J-Link 驅動程式

若要與 XMC4800 Relax EtherCAT 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。

監控雲端的 MQTT 訊息

在執行 FreeRTOS 示範之前，您可以在主控台中設定 MQTT 用戶端，以監 AWS IoT 控裝置傳送至雲端的訊息。 AWS

若要向 MQTT 用戶端訂閱 MQTT 主題 AWS IoT

1. 登入 AWS IoT 主控台。
2. 在導覽窗格中，選擇 [測試]，然後選擇 [MQTT 測試用戶端] 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。

當演示項目成功運行在您的設備上，你會看到「你好世界！」 多次發送到您訂閱的主題。

建置並執行 FreeRTOS 示範專案

將 FreeRTOS 演示導入到 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 中的 MQTT 用戶端應顯示裝置傳送的 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 使用者示範

如果您不想使用 IDE 進行 FreeRTOS 開發，也可以使用 CMake 來構建和運行使用第三方代碼編輯器和調試工具開發的演示應用程序或應用程序。

> **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 搭配使用 FreeRTOS 體。(MinGW 是本機 Microsoft Windows 應用程式的簡約開發環境。)

若要使用 CMake 建置 FreeRTOS 體示範

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. 將目錄更改為 FreeRTOS 下載目錄 (*freertos*)，然後使用以下命令生成構建文件：

   ```
   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。

故障診斷

如果您還沒有，請務必進行配置 AWS IoT 並下載 FreeRTOS 以將您的設備連接到雲端 AWS 。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**
>
> 此參考集成託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱亞馬遜自由 Github 存儲庫遷移指南.

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-英飛凌 XMC4800 IoT 連接套件，請訪問AWS合作夥伴設備目錄，向我們的合作夥伴購買一個。
2. 安全性延伸套件：
   - 安全元素 - Infineon OPTIGA Trust X。

     請造訪合AWS作夥伴裝置目錄，向我們的合作夥伴購買。
   - 個人化面板 - Infineon OPTIGA 個人化面板。
   - 適配器板-英飛凌 MyIo T 適配器。

若要遵循此處的步驟，您必須使用面板開啟序列連線，才能檢視記錄和偵錯資訊。其中一個步驟要求您從主機板的序列除錯輸出中複製公開金鑰，並將其貼到檔案中。）為此，除了 XMC4800 IoT 連接套件之外，您還需要一個 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。您還需要三條 male-to-male 跳線（用於接收（RX），傳輸（TX）和接地（GND）），以將串行電纜連接到英飛凌 MyIo T 適配器板。

在開始之前，您必須先設定 FreeRTOSAWS IoT 並下載，才能將裝置連線到AWS雲端。如需相關指示，請參閱選項 #2：產生內建私有金鑰。在此教學課程中，FreeRTOS 下載目錄的路徑稱為*freertos*。

概要

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為二進位映像檔。
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 使用英飛凌的 DAVE 開發環境對 XMC4800 進行程式設計。開始之前，請下載並安裝 DAVE 和一些 J-Link 驅動程式，以與內建除錯器通訊。

安裝 DAVE

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

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

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

如需詳細資訊，請參閱 DAVE Quick Start Guide。

安裝 Segger J-Link 驅動程式

若要與 XMC4800 IoT Connectivity Kit 的內建偵錯探查通訊，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

在執行 FreeRTOS 示範專案之前，您可以在主控台中設定 MQTT 用戶端，以監AWS IoT控裝置傳送至AWS雲端的訊息。

使用 AWS IoT MQTT 用戶端訂閱 MQTT 主題

1. 登入 AWS IoT 主控台。
2. 在導覽窗格中，選擇 [測試]，然後選擇 [MQTT 測試用戶端] 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。

當演示項目成功運行在您的設備上，你會看到「你好世界！」 多次發送到您訂閱的主題。

建置並執行 FreeRTOS 示範專案

將 FreeRTOS 演示導入到 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，請移至 AWS IoT 主控台。您之前設定的 MQTT 用戶端應該會顯示您裝置傳送的 MQTT 訊息。透過裝置的序列連線，您應該會在 UART 輸出中看到與下雷同的內容：

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

```
26 80085 [Echoing] Message returned with ACK: 'Hello World 6 ACK'
.27 85019 [MQTTEcho] Echo successfully published 'Hello World 7'
.28 88086 [Echoing] Message returned with ACK: 'Hello World 7 ACK'
.29 93020 [MQTTEcho] Echo successfully published 'Hello World 8'
.30 96088 [Echoing] Message returned with ACK: 'Hello World 8 ACK'
.31 101021 [MQTTEcho] Echo successfully published 'Hello World 9'
32 104102 [Echoing] Message returned with ACK: 'Hello World 9 ACK'
.33 109022 [MQTTEcho] Echo successfully published 'Hello World 10'
.34 112047 [Echoing] Message returned with ACK: 'Hello World 10 ACK'
.35 117023 [MQTTEcho] Echo successfully published 'Hello World 11'
36 120089 [Echoing] Message returned with ACK: 'Hello World 11 ACK'
.37 122068 [MQTTEcho] MQTT echo demo finished.
38 122068 [MQTTEcho] ----Demo finished----
```

使用 CMake 建置 FreeRTOS 使用者示範

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 搭配使用 FreeRTOS 體。(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，可以使用 CMake 來構建和運行使用第三方代碼編輯器和調試工具開發的演示應用程序或應用程序。

若要使用 CMake 建置 FreeRTOS 體示範

1. 設定 GNU Arm 內嵌式工具鏈。

   a. 從 Arm Embedded Toolchain 下載頁面下載 Windows 版的工具鏈。

      > **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. 將目錄更改為 FreeRTOS 下載目錄 (*freertos*)，然後使用以下命令生成構建文件：

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

5. 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。完成所有步驟之後，請移至 AWS IoT 主控台。您之前設定的 MQTT 用戶端應該會顯示您裝置傳送的 MQTT 訊息。

疑難排解

如需一般疑難排解資訊，請參閱故障診斷入門。

開始使用 MW32 AWS IoT x 入門套件

> **Important**
>
> 此參考集成託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。我們建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱. 亞馬遜自由 Github 存儲庫遷移指南

入 AWS IoT 門套件是以 88MW320/88MW322 為基礎的開發套件，這是恩智浦的最新整合式皮質 M4 微控制器，可在單一微控制器晶片上整合 802.11b/g/n 無線網路。開發套件已通過 FCC 認證。如需詳細資訊，請參閱合AWS 作夥伴裝置目錄，向我們的合作夥伴購買。該模塊也通過 FCC 認證，可用於定制和批量銷售。

本入門指南說明如何將應用程式與主機電腦上的 SDK 交叉編譯，然後使用 SDK 隨附的工具將產生的二進位檔案載入主機板上。當應用程式開始在主機板上執行時，您可以從主機電腦上的序列主控台進行除錯或與之互動。

Ubuntu 16.04 是支持開發和調試的主機平台。您也許可以使用其他平台，但這些平台尚未受到官方支援。您必須擁有在主機平台上安裝軟體的權限。構建 SDK 所需的以下外部工具：

- 主機平台
- ARM 工具鏈第四版
- 日食

ARM 工具鏈需要交叉編譯您的應用程序和 SDK。SDK 利用最新版本的工具鏈來優化映像佔用空間，並將更多功能放入更小的空間中。本指南假設您使用的是工具鏈的 4_9_2015q3 版本。不建議使用較舊版本的工具鏈。該開發套件已預先刷新無線微控制器演示項目固件。

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 電纜將 MW32x 板 Connect 到筆記本電腦。將迷你 USB 電纜 Connect 到板上唯一存在的迷你 USB 連接器。你不需要一個跳線換。

如果主機板已連接至筆記型電腦或桌上型電腦，則不需要外接電源供應器。

此 USB 連接提供以下功能：

- 主控台存取主機板。虛擬 tty/com 連接埠會向開發主機註冊，可用來存取主控台。
- JTAG 訪問董事會。這可用於將韌體影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

出於開發目的，最低要求是 ARM 工具鏈和 SDK 捆綁的工具。以下幾節提供 ARM 工具鏈設定的詳細資訊。

GNU 工具鏈

SDK 正式支援 GCC 編譯器工具鏈。GNU ARM 的跨編譯器工具鏈可在 GNU Arm 嵌入式工具鏈 4.9-2015 年第 3 季更新中獲得。

構建系統被配置為默認使用 GNU 工具鏈。Makefile 會假設 GNU 編譯器工具鏈二進位檔案位於使用者的路徑上，而且可以從 Makefile 中叫用。Makefile 也會假設 GNU 工具鏈二進位檔案的檔案名稱前面加上。`arm-none-eabi-`

GCC 工具鏈可以與 GDB 一起使用，以使用 OpenOCD 進行調試（與 SDK 捆綁在一起）。這提供了與 JTAG 接口的軟件。

我們建議使用工具鏈的版本 4_9_2015q3。 gcc-arm-embedded

工具鏈設定程序

請按照以下步驟在 Linux 中設置 GCC 工具鏈。

1. 在 GNU Arm 嵌入式工具鏈 4.9-2015 年第 3 季更新處下載工具鏈壓縮包。檔案是`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. 使用下面的命令來解壓文件。

   ```
   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*目錄中的`.profile`文件末尾附加以下行。

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

> **Note**
>
> 較新的 Ubuntu 發行版可能會附帶 Debian 版本的海灣合作委員會交叉編譯器。如果是這樣，您必須刪除本機交叉編譯器並按照上述設置過程進行操作。

使用 Linux 開發主機

任何現代 Linux 桌面發行版，如 Ubuntu 或軟呢帽都可以使用。不過，我們建議您升級至最新版本。下列步驟已驗證可在 Ubuntu 16.04 上運作，並假設您使用的是該版本。

安裝套件

SDK 包含一個指令碼，可讓您在新設定的 Linux 機器上快速設定開發環境。該腳本嘗試 auto 檢測機器類型並安裝適當的軟件，包括 C 庫，USB 庫，FTDI 庫，安裝，蟒蛇和乳膠。在本節中，通用目錄名稱*amzsdk_bundle-x.y.z*表示 AWS SDK 根目錄。實際的目錄名稱可能不同。您必須具有根權限。

- 導航到目*amzsdk_bundle-x.y.z/*錄並運行此命令。

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

避免須藤

在本指南中，該`flashprog`操作使用`flashprog.py`腳本刷新板的 NAND，如下所述。同樣地，此`ramload`作業使用`ramload.py`指令碼將韌體影像從主機直接複製到微控制器的 RAM，而不會閃爍 NAND。

您可以將 Linux 開發主機設定為執行`flashprog`和`ramload`作業，而不需要每次`sudo`指令。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**
>
> 您必須以這種方式配置 Linux 開發主機權限，以確保順暢的 Eclipse IDE 體驗。

設定序列主控台

將 USB 連接線插入主機 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 是序列控制台。在上面的例子中，它被命名為「ttyusB1」。

在本指南中，我們使用 minicom 查看串行控制台輸出。您也可以使用其他串行程序，例如putty. 運行以下命令以在設置模式下執行 minicom。

```
minicom -s
```

在 minicom 中，導航到串行端口設置並捕獲以下設置。

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

您可以在 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。這在屏幕上顯示一個哈希（ # ）。

> **Note**
>
> 該開發板包含 FTDI 矽元件。FTDI 裝置為主機公開了兩個 USB 介面。第一介面與 MCU 的 JTAG 功能相關聯，第二介面與 MCU 的實體 Uartx 連接埠相關聯。

安裝 OpenOCD

OpenOCD 是為嵌入式目標裝置提供除錯、系統內程式設計和邊界掃描測試的軟體。

需有 OpenOCD 0.9 版。它也是 Eclipse 功能所必需的。如果您的 Linux 主機上已安裝舊版 (例如 0.7 版)，請使用您目前使用之 Linux 發行版本的適當指令移除該儲存庫。

執行標準的 Linux 指令來安裝

```
apt-get install openocd
```

如果上述指令未安裝 0.9 版或更新版本，請使用下列程序來下載並編譯 openocd 原始程式碼。

若要安裝 OpenOCD

1. 執行下列命令來安裝 libusb-1.0。

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

2. 請從以下網址下載 0.9.0 原始碼。 http://openocd.org/
3. 解壓縮 openocd 並導航到您解壓縮它的目錄。
4. 請使用下列指令來設定 openocd。

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

5. 執行製作公用程式來編譯 opencd。

   ```
   make install
   ```

設置日食

> **Note**
>
> 本節假設您已完成中的步驟 避免須藤

Eclipse 是應用程序開發和調試的首選 IDE。它提供了一個豐富的，用戶友好的 IDE 與集成的調試支持，包括線程感知調試。本節介紹了所有支持的開發主機的常見 Eclipse 設置。

設定 Eclipse

1. 下載並安裝 Java 執行階段環境 (JRE)。

日食要求您安裝 JRE。我們建議您先安裝此安裝，儘管它可以在安裝 Eclipse 之後進行安裝。JRE 版本 (32/64 位元) 必須與日蝕版本 (32/64 位元) 相符。您可以從 Java SE 運行時環境 8 下載在甲骨文網站上下載 JRE。

2. 從以下位置下載並安裝「適用於 C/C ++ 開發人員的日食 IDE」。 http://www.eclipse.org 支持日食版 4.9.0 或更高版本。安裝只需要您解壓縮下載的歸檔。您執行平台特定的 Eclipse 可執行檔來啟動應用程式。

建置並執行 FreeRTOS 示範專案

有兩種方式可以執行 FreeRTOS 示範專案：

- 使用命令行。
- 使用日食 IDE。

本主題涵蓋了這兩個選項。

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 安全性值：
>
> - `eWiFiSecurityOpen`
> - `eWiFiSecurityWEP`
> - `eWiFiSecurityWPA`

- eWiFiSecurityWPA2

SSID 和密碼應該用雙引號括起來。

使用命令列建置並執行 FreeRTOS 示範

1. 使用下面的命令開始構建演示應用程序。

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

確保你得到相同的輸出，如下面的例子所示。

```
marvell@pe-lt586:amzsdk$
marvell@pe-lt586:amzsdk$ cmake -DVENDOR=marvell -DBOARD=mw300_rd -DCOMPILER=arm-gcc -S . -
Bbuild -DAFR_ENABLE_TESTS=0

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

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

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

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

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

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

2. 導航到構建目錄。

```
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：

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

4. 使用下列命令建置測試應用程式。

```
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。韌體會在開發板重設後執行。您必須先建立 SDK，然後再將映像快閃記錄到微控制器。

   a. 在刷新固件映像之前，請使用通用組件佈局和 Boot2 準備開發板的閃存。請使用下列指令。

```
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
```

該flashprog命令啟動以下內容：

- 版面配置 — Flashprog 公用程式會先指示將版面編寫至閃光燈。佈局類似於閃存的分區信息。預設配置位於/lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/OpenOCD/mw300/layout.txt。
- 引導 2-這是由 WMSDK 使用的啟動加載程序。該flashprog命令還將引導加載程序寫入閃存。引導加載程序的工作是在閃爍後加載微控制器的固件映像。確保您獲得與下圖所示相同的輸出。

```
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. 固件使用 Wi-Fi 芯片組的功能，並且 Wi-Fi 芯片組具有自己的固件，該固件也必須存在於閃存中。您可以使用flashprog.py公用程式來更新 Wi-Fi 韌體的方式，就如同您刷新 Boot2 開機載入程式和 MCU 韌體一樣。使用以下命令刷新 Wi-Fi 固件。

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

確保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. 使用下列指令來刷新 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記錄。

e. 要運行測試應用程序，請刷新位於同一目錄中的`aws_tests.bin`二進製文件。

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

您的命令輸出應與下圖所示類似。

```
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 0.245499s (115.708 KiB/s)
verified 29088 bytes in 0.350231s (81.107 KiB/s)
semihosting is enabled

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


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

6. 刷新固件並重置板後，演示應用程序應該如下圖所示啟動。

```
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. (選擇性) 作為測試影像的替代方法，請使用 flashprog 公用程式將微控制器影像從主機直接複製到微控制器 RAM 中。影像不會在快閃記憶體中複製，因此在重新啟動微控制器後會遺失影像。

   將韌體映像載入 SRAM 是較快的作業，因為它會立即啟動執行檔案。此方法主要用於迭代開發。

   使用下列指令將韌體載入 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
```

命令執行完成後，您應該會看到演示應用程序的日誌。

使用 Eclipse IDE 構建和運行 FreeRTOS 務器演示

1. 在設定 Eclipse 工作區之前，您必須執行cmake命令。

   運行下面的命令與 aws_demos Eclipse 項目的工作。

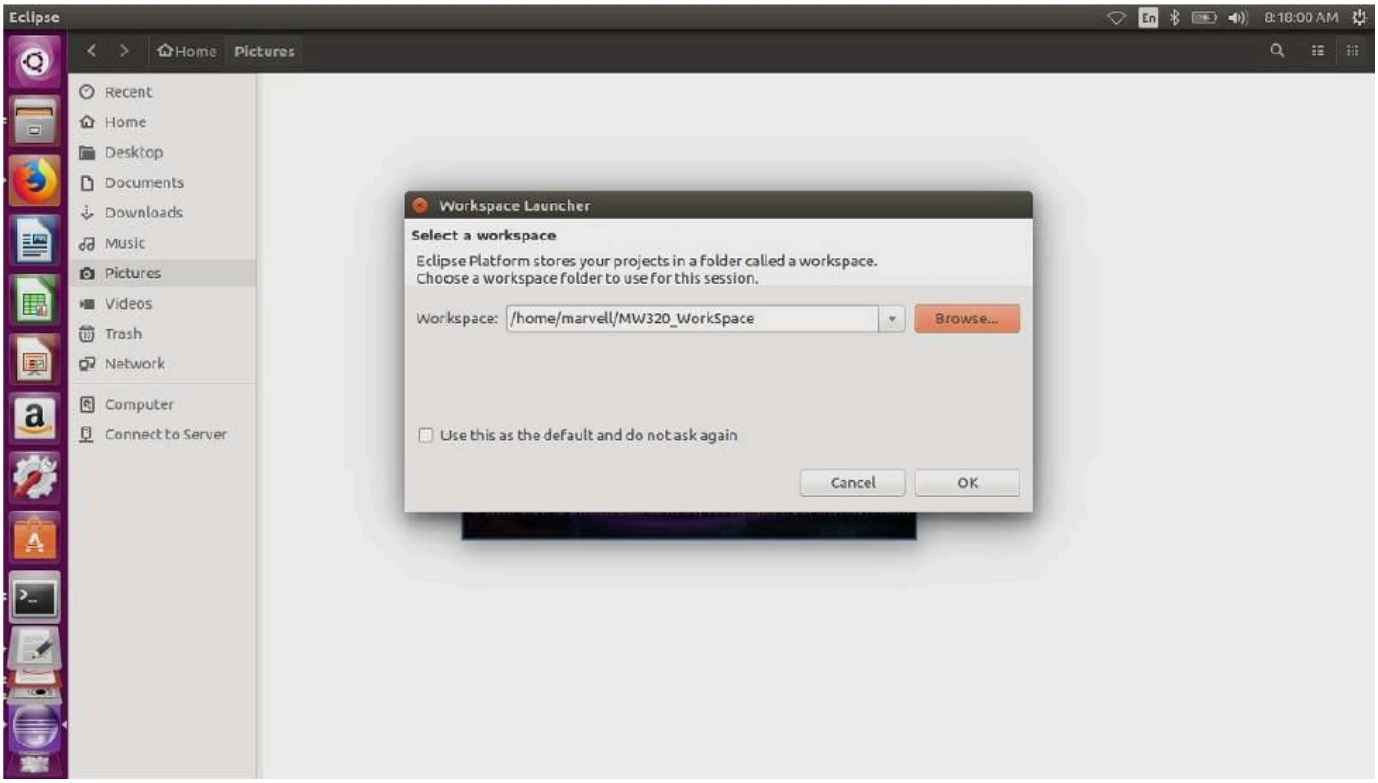
   ```
   cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S .  -Bbuild -
   DAFR_ENABLE_TESTS=0
   ```

   運行下面的命令與 aws_tests Eclipse 項目的工作。

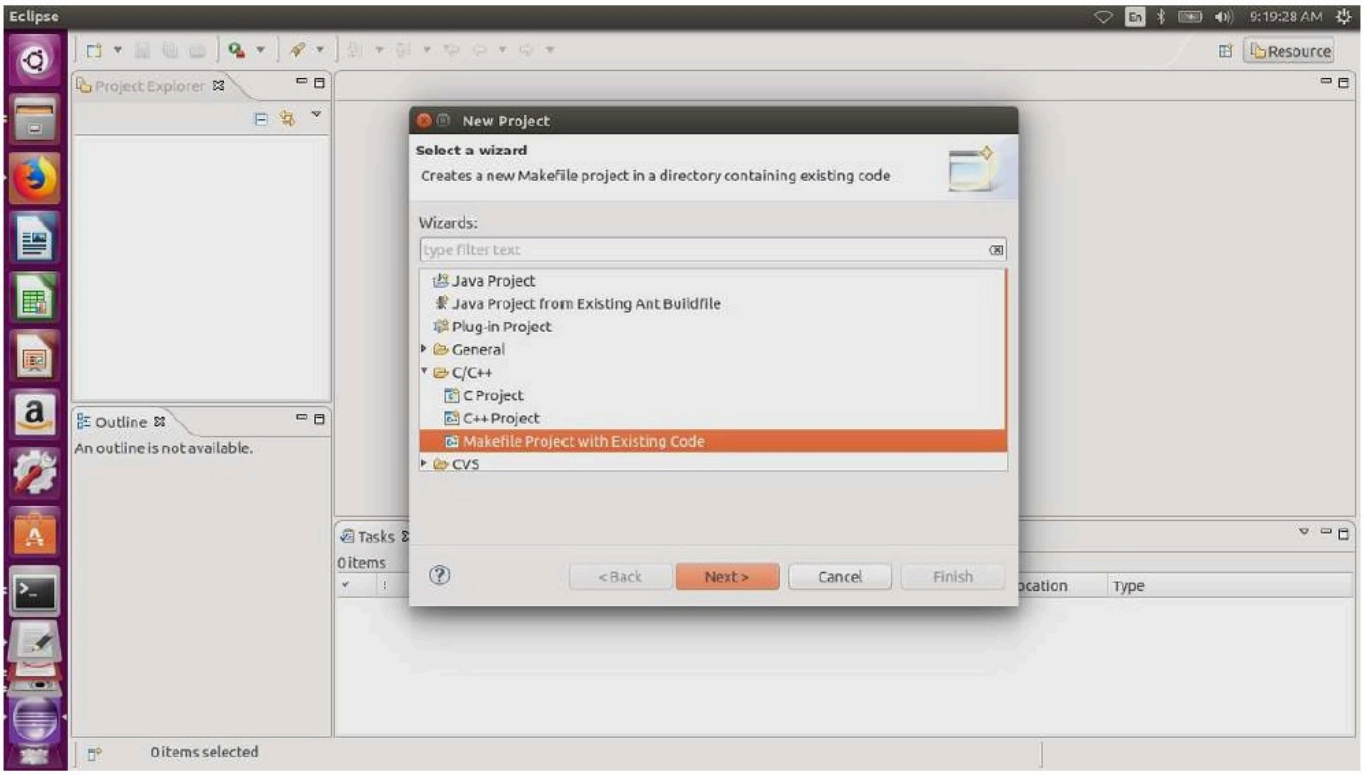
   ```
   cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S .  -Bbuild -
   DAFR_ENABLE_TESTS=1
   ```

> **Tip**
>
> 每次在aws_demos專案和專案之間切換時執行cmakeaws_tests命令。

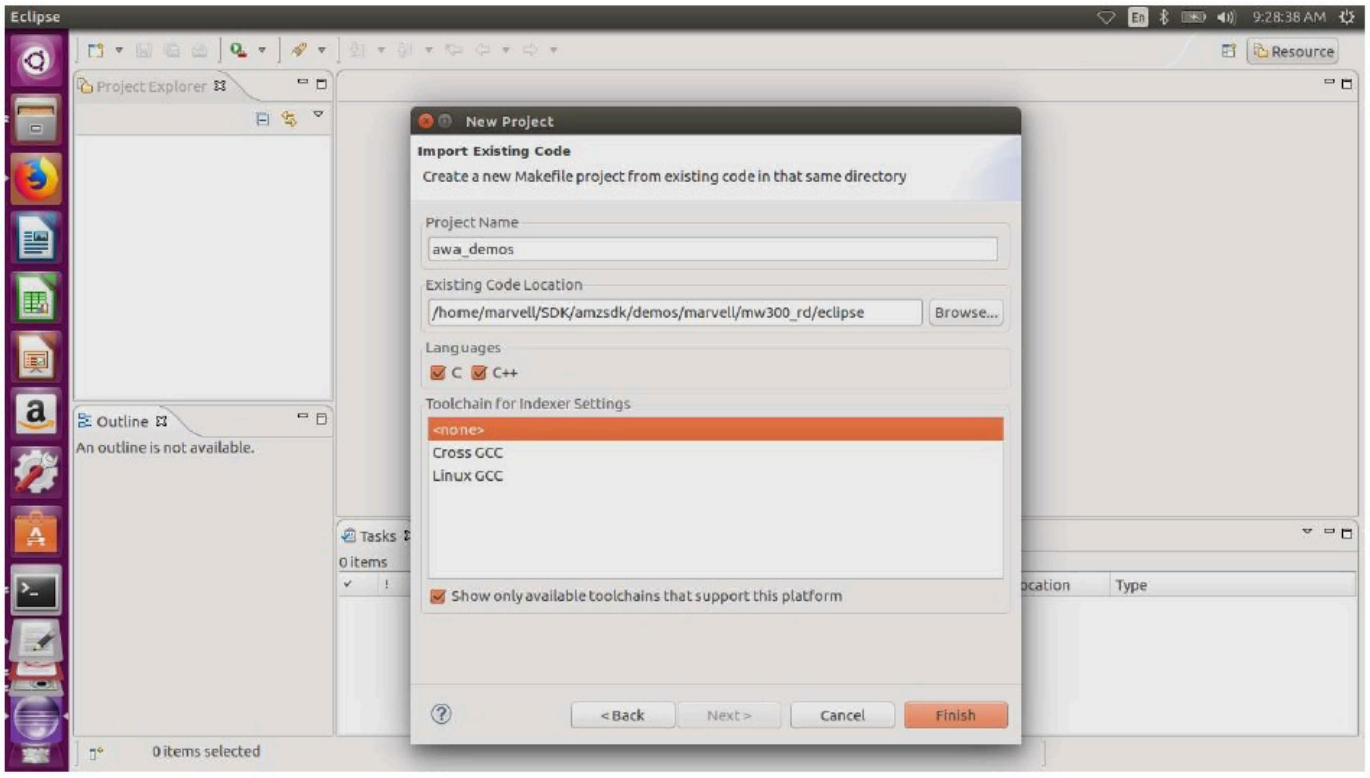
2. 打開 Eclipse，並在出現提示時，選擇您的 Eclipse 工作區，如下圖所示。



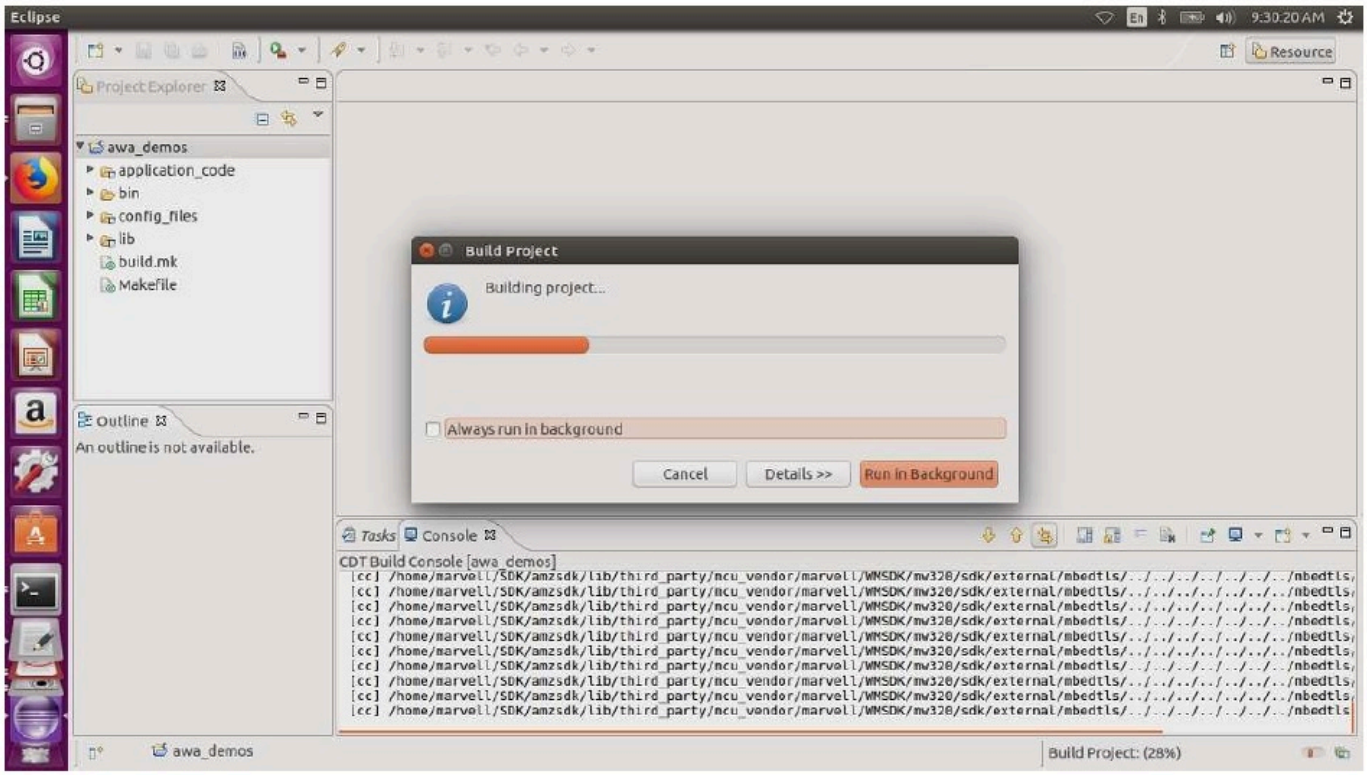
3. 選擇創建 Makefile 項目的選項：與現有代碼，如下圖所示。



4. 選擇 [瀏覽]，指定現有程式碼的目錄，然後選擇 [完成]。



5. 在瀏覽窗格中，選擇專案總管中的 aws_demo。以滑鼠右鍵按一下 aws_demo 開啟功能表，然後選擇建置。



如果構建成功，它會生成build/cmake/vendors/marvell/mw300_rd/aws_demos.bin文件。

6. 使用命令行工具來刷新佈局文件（layout.txt），Boot2 二進製文件（boot2.bin），MCU 固件二進製文件（aws_demos.bin）和 Wi-Fi 固件。

   a. 在刷新固件映像之前，請使用常見的組件，佈局和 Boot2 準備開發板的閃存。請使用下列指令。

   ```
   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
   ```

   該flashprog命令啟動以下內容：

   - 版面配置 — Flashprog 公用程式會先指示將版面編寫至閃光燈。佈局類似於閃存的分區信息。預設配置位於/lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/OpenOCD/mw300/layout.txt。

- 引導 2-這是由 WMSDK 使用的啟動加載程序。flashprog 命令也會將開機載入程式寫入快閃記憶體。引導加載程序的工作是在刷新微控制器的固件映像後加載它。確保您獲得與下圖所示相同的輸出。

```
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. 固件使用 Wi-Fi 芯片組的功能，並且 Wi-Fi 芯片組具有自己的固件，該固件也必須存在於閃存中。您可以使用`flashprog.py`公用程式來更新 Wi-Fi 韌體的方式，就如同您刷新 boot2 開機載入程式和 MCU 韌體一樣。使用以下命令刷新 Wi-Fi 固件。

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

確保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. 使用下列指令來刷新 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記錄。

e. 要運行測試應用程序，請刷新位於同一目錄中的`aws_tests.bin`二進製文件。

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

您的命令輸出應與下圖所示類似。

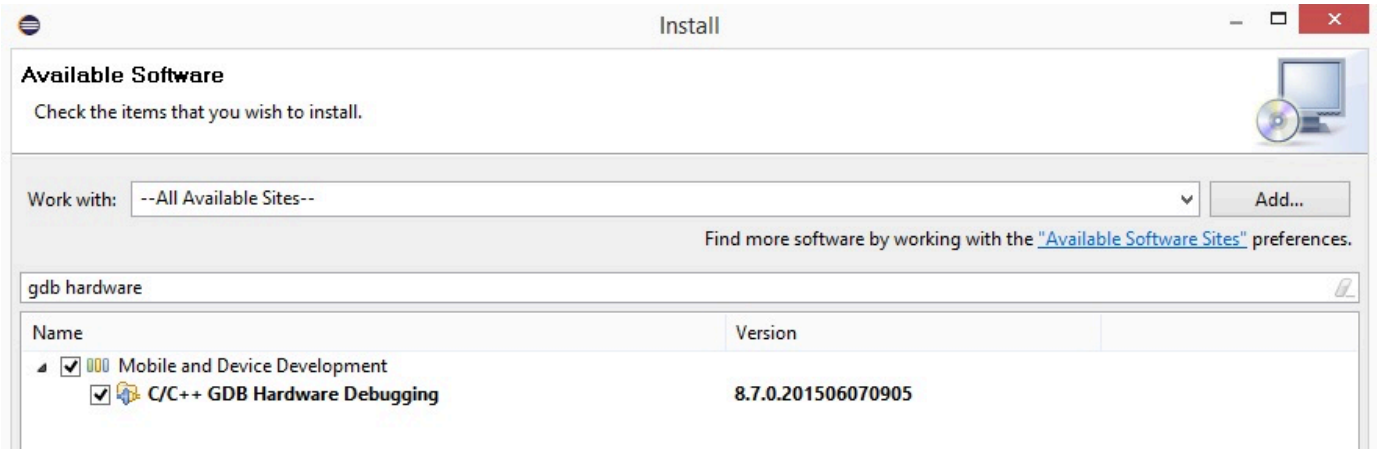
```
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00007f14 msp: 0x20001000
29088 bytes written at address 0x00100000
downloaded 29088 bytes in 0.245499s (115.708 KiB/s)
verified 29088 bytes in 0.350231s (81.107 KiB/s)
semihosting is enabled

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 並選擇「幫助」，然後選擇「安裝新軟件」。在 [使用] 功能表中，選擇 [所有可用的網站]。輸入篩選文字`GDB Hardware`。選擇 C/C ++ GDB 硬件調試選項並安裝插件。



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。

啟用其他記錄

- 啟用主機板特定記錄。

  在main.c檔案wmstdio_init(UART0_ID, 0)中的函式prvMiscInitialization中啟用對測試或示範的呼叫。

- 啟用 Wi-Fi 記錄

  啟用*freertos*/vendors/marvell/WMSDK/mw320/sdk/src/incl/autoconf.h檔案CONFIG_WLCMGR_DEBUG中的巨集。

使用 GDB

我們建議您使用隨 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
```

運行以下命令（在單行上）連接到 GDB。

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

開始使用 MediaTek MT7697Hx 開發套件

> **Important**
>
> 此參考集成託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。我們建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱. 亞馬遜自由 Github 存儲庫遷移指南

本教學課程提供如何開始使用 MediaTek MT7697hx 開發套件的指示。如果您沒有 MediaTek MT7697hx 開發套件，請造訪 AWS 合作夥伴裝置目錄，向我們的合作夥伴購買。

在開始之前，您必須先設定 FreeRTOS AWS IoT 並下載，才能將裝置連線到雲端 AWS 。如需說明，請參閱 首要步驟。在此教學課程中，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. 將 FreeRTOS 示範應用程式交叉編譯為二進位映像檔。
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

在設定環境之前，請先將電腦連接至 MediaTek MT7697hx 開發套件上的 USB 連接埠。

下載並安裝 Keil MDK

您可以使用以 GUI 為基礎的 Keil 微控制器開發套件 (MDK) 在主機板上設定、建置和執行 FreeRTOS 專案。凱爾 MDK 包括微視覺 IDE 和微視覺除錯器。

> **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。確保您安裝了 MediaTek 設備包（MT76x7 系列）。

建立序列連線

使用 USB 電纜將主機板 Connect 到主機。COM 連接埠會出現在視窗裝置管理員中。若要進行偵錯，您可以使用終端機公用程式工具 (例如 HyperTerminal 或) 開啟連接埠的工作階段 TeraTerm。

監控雲端的 MQTT 訊息

在執行 FreeRTOS 示範專案之前，您可以在主控台中設定 MQTT 用戶端，以監 AWS IoT 控裝置傳送至雲端的訊息。 AWS

若要向 MQTT 用戶端訂閱 MQTT 主題 AWS IoT

1. 登入 AWS IoT 主控台。
2. 在導覽窗格中，選擇 [測試]，然後選擇 [MQTT 測試用戶端] 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。

當演示項目成功運行在您的設備上，你會看到「你好世界！」 多次發送到您訂閱的主題。

使用 Keil MDK 建置並執行 FreeRTOS 示範專案

要在凱爾微視野中構建 FreeRTOS 由演示項目

1. 從「開始」功能表中，打開「凱爾微視覺 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。

若要執行 FreeRTOS 範專案

1. 將 MediaTek MT7697Hx 開發套件設置為「程序」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，選擇除錯索引標籤。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. 在您的 MediaTek MT7697Hx 開發套件上選擇重設按鈕。

故障診斷

在基爾微視野中調試 FreeRTOS 項目

目前，您必須先編輯包含在 Keil µVision 中的 MediaTek 套件，才能使用 Keil µVision 對 MediaTek FreeRTOS 示範專案進行除錯。

若要編輯用於偵錯 FreeRTOS 專 MediaTek 案的套件

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. 重建示範專案。
6. 將 MediaTek MT7697Hx 開發套件設置為「程序」模式。

   若要將套件設定為 PROGRAM 模式，請按住 PROG 按鈕。在 PROG 按鈕仍然按住的情況下，按下並放開 RESET 按鈕，然後放開 PROG 按鈕。
7. 從功能表中，選擇 Flash，然後選擇 Download (下載)。

   下載完成時 µVision 會通知您。
8. 按下 MediaTek MT7697Hx 開發套件上的「重設」按鈕。
9. 從 µVision 功能表中選擇除錯，然後選擇啟動/停止除錯工作階段。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：

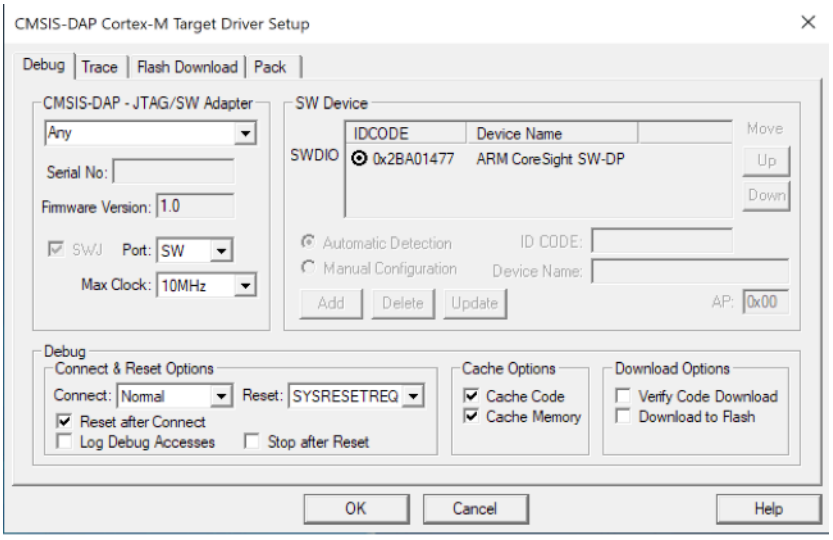
    ```
    { volatile int wait_ice = 1 ; while ( wait_ice ) ; }
    ```
11. 在 Call Stack + Locals (呼叫堆疊 + 本機) 視窗中，將 `wait_ice` 的值變更為 `0`。
12. 在專案的原始程式碼中設定中斷點，然後執行程式碼。

故障診斷 IDE 除錯器設定

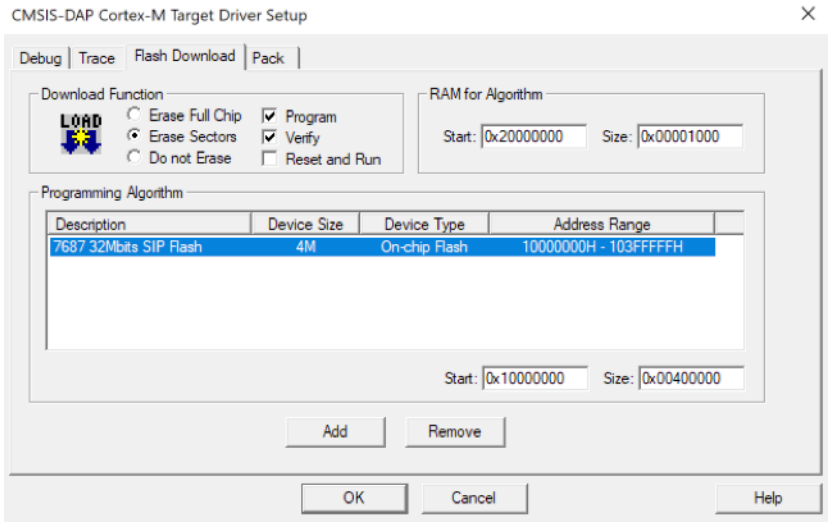
如果您無法除錯應用程式，您的除錯器設定可能不正確。

驗證您的除錯器設定是否正確

1. 開啟 Keil µVision。
2. 右鍵單擊項aws_demos目，選擇選項，然後在「實用程序」選項卡下選擇「設置」，在 「--使用調試驅動程序-」 旁邊。
3. 驗證 Debug (除錯) 索引標籤中的設定如下所示：



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



如需 FreeRTOS 入門的一般疑難排解資訊，請參閱。故障診斷入門

Microchip Curiosity PIC32MZ EF 入門

> **Important**
>
> 此參考集成託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱亞馬遜自由 Github 存儲庫遷移指南.

> **Note**
>
> 在與微晶協議下，我們將從 FreeRTOS 參考整合儲存庫主要分支移除好奇心 PIC32MZEF (DM320104)，並且將不再於新版本中攜帶。微晶已發出正式通知，表示不再建議將 PIC32MZEF (DM320104) 用於新設計。PIC32MZEF 專案和原始程式碼仍可透過舊版標籤存取。微晶片建議客戶在新設計上使用好奇心 PIC32MZ-EF-2.0 開發板 (DM320209)。Pic32MZv1 平台仍然可以在 FreeRTOS 參考整合儲存庫的 v202012.00 中找到。不過，《自由服務參考》的 v202107.00 已不再支援該平台。

本教學課程提供 Microchip Curiosity PIC32MZ EF 入門指示。如果您沒有微晶片好奇心 PIC32MZ EF 套裝軟體，請造訪AWS合作夥伴裝置目錄，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 點擊板
- MikroElectronika WiFi 7 點擊「看板」
- 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

在開始之前，您必須先設定 FreeRTOSAWS IoT 並下載，才能將裝置連線到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。
> - 由於原始程式碼可能包含符號連結，因此如果您使用 Windows 來擷取歸檔，您可能必須：

- 啟用開發人員模式或，
- 使用以系統管理員身分提高權限的主控台。

如此一來，Windows 可以在擷取歸檔時正確建立符號連結。否則，符號鏈接將被寫入為普通文件，其中包含符號鏈接的路徑作為文本或為空。如需更多資訊，請參閱部落格文章：Windows 10 中的符號連結！ 。

如果您在 Windows 下使用 Git，您必須啟用開發人員模式，或者您必須：

- 使用下列命令設定`core.symlinks`為 true：

```
git config --global core.symlinks true
```

- 每當您使用寫入系統的 git 命令時，請使用以管理員身份提升的主控台 (例如git pullgit clone、、和git submodule update --init --recursive)。

概要

本教學課程包含以下入門步驟的指示：

1. 將主機板連線到主機機器。
2. 在主機機器上安裝軟體以對微控制器主機板的內嵌應用程式進行開發和除錯。
3. 將 FreeRTOS 示範應用程式交叉編譯為二進位映像檔。
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 點擊板 Connect 到 microBUS 線 1 連接器上的微型芯片好奇心 PIC32MZ EF。
2. 將 PIC32 LAN8720 PHY 子板接到 Microchip Curiosity PIC32MZ EF 的 J18 接頭。
3. 使用 MikroElectronika USB A 到 USB 迷你 B 電纜將 USB 單擊板 Connect 到您的計算機。
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：
   - 要使用無線網絡連接，無線 MikroElectronika 網絡連接 7 點擊板 microBUS 線 2 連接器上微芯片好奇心 PIC32MZ EF。請參閱 設定 FreeRTOS 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。您還必須定義預處理器宏`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。

   如需有關 MPLAB 鎖點除錯器的詳細資訊，請參閱 MPLAB 鎖點在線除錯器資訊表。

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 主機版 (PKOB) 程式設計工具/偵錯工具來評估和執行 FreeRTOS 示範，並進行基本偵錯。

1. 將 MikroElectronika USB UART 點擊板 Connect 到 microBUS 線 1 連接器上的微型芯片好奇心 PIC32MZ EF。
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：

   - 要使用無線網絡連接，無線 MikroElectronika 網絡連接 7 點擊板 microBUS 線 2 連接器上微芯片好奇心 PIC32MZ EF。(按照設定 FreeRTOS 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。您還必須定義預處理器宏`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 MikroElectronika USB A 到 USB 迷你 B 電纜將 USB 單擊板 Connect 到您的計算機。

設定開發環境

> **Note**
>
> 這個裝置的 FreeRTOS 專案是以 MPLAB 和諧 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**
   >
   > 目前僅支援 FreeRTOS 軟體AWS參考整合功能。先前版本的 FreeRTOSAWS 參考整合功能在 MPLabv5.40 上受到支援。

   電腦下載

   - 視窗視窗整合式開發環境
   - 適用於 macOS 的 MPLAB X 整合式開發環境
   - 適用於 Linux 的 MPLAB X 整合式開發環境

   最新 MPLAB 下載 (英文版)

   - 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

在雲端中監控 MQTT 訊息

在執行 FreeRTOS 示範專案之前，您可以在主控台中設定 MQTT 用戶端，以監AWS IoT控裝置傳送至AWS雲端的訊息。

使用 AWS IoT MQTT 用戶端訂閱 MQTT 主題

1. 登入 AWS IoT 主控台。
2. 在導覽窗格中，選擇 [測試]，然後選擇 [MQTT 測試用戶端] 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。

當演示項目成功運行在您的設備上，你會看到「你好世界！」 多次發送到您訂閱的主題。

建置並執行 FreeRTOS 示範專案

在 MPLAB IDE 中開啟 FreeRTOS 使用者示範

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。

若要使用乙太網路連線，您必須定義預處理器巨集`PIC32_USE_ETHERNET`。

若要使用乙太網路透過 MPLAB IDE 進行連線

1. 在 MPLAB IDE 中，以滑鼠右鍵按一下專案，然後選擇「內容」。
2. 在 [專案屬性] 對話方塊中，選擇 [***#####*** (全域選項)] 以展開它，然後選取 [***#####-*** gcc]。
3. 在 [選項] 類別中，選擇 [預處理] 和 [訊息]，然後將字`PIC32_USE_ETHERNET`串新增至預處理器巨集。

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 使用者示範

如果您不想使用 IDE 進行 FreeRTOS 開發，也可以使用 CMake 來構建和運行使用第三方代碼編輯器和調試工具開發的演示應用程序或應用程序。

若要使用 CMake 建置 FreeRTOS 體示範

1. 創建一個目錄以包含生成的構建文件，例如構*###*。
2. 使用下列命令從源代碼中產生文件。

```
cmake -DVENDOR=microchip -DBOARD=curiosity_pic32mzef -DCOMPILER=xc32 -
DMCHP_HEXMATE_PATH=path/microchip/mplabx/version/mplab_platform/bin  -
DAFR_TOOLCHAIN_PATH=path/microchip/xc32/version/bin -S freertos -B build-folder
```

> **Note**
>
> 您必須指定 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. 將目錄更改為構建目錄（*####*），然後make從該目錄運行。

如需詳細資訊，請參閱搭配使用 FreeRTOS 體。

若要使用乙太網路連線，您必須定義預處理器巨集PIC32_USE_ETHERNET。

疑難排解

如需故障診斷資訊，請參閱故障診斷入門。

Nordic nRF52840-DK 入門

> **Important**
>
> 此參考集成託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱亞馬遜自由 Github 存儲庫遷移指南.

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合作夥伴裝置目錄，向我們的合作夥伴購買。

開始之前，您需要為 FreeRTOS 低功耗藍牙設置 AWS IoT 和 Amazon Cognito。

若要執行 FreeRTOS 低功耗藍牙示範，您還需要具備藍牙和 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。

概要

本教學課程包含以下入門步驟的指示：

1. 將主機板連線到主機機器。
2. 在主機機器上安裝軟體以對微控制器主機板的內嵌應用程式進行開發和除錯。
3. 將 FreeRTOS 示範應用程式交叉編譯為二進位映像檔。
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 支持塞格嵌入式工作室作為北歐 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 SDK 示範應用程式

若要跨藍牙低功耗執行 FreeRTOS 示範專案，您需要在行動裝置上執行 FreeRTOS 藍牙低功耗行動 SDK 示範應用程式。

若要設定 FreeRTOS 藍牙低功耗行動 SDK 示範應用程式

1. 按照 適用於 FreeRTOS 藍牙裝置的行動 SDK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 免費藍牙低功耗移動 SDK 演示應用程序 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. 從頂部菜單中，選擇工具，終端仿真器，Connect 端**#**（ 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。

下載和設定 FreeRTOS

設定好硬體和環境之後，您可以下載 FreeRTOS。

下載 FreeRTOS

若要下載適用於北歐 NRF52840-dk 的 FreeRTOS，請移至 FreeRTOS GitHub 頁面並複製存放庫。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。
> - 由於原始程式碼可能包含符號連結，因此如果您使用 Windows 來擷取歸檔，您可能必須：
>   - 啟用開發人員模式，或
>   - 使用以系統管理員身分提高權限的主控台。
>
>   如此一來，Windows 可以在擷取歸檔時正確建立符號連結。否則，符號鏈接將被寫入為普通文件，其中包含符號鏈接的路徑作為文本或為空。如需更多資訊，請參閱部落格文章：Windows 10！ 。
>
>   如果您在 Windows 下使用 Git，您必須啟用開發人員模式，或者您必須：
>   - 使用下列命令設定`core.symlinks`為 true：

```
git config --global core.symlinks true
```

- 每當您使用寫入系統的 git 命令時，請使用以管理員身份提升的主控台 (例如git pullgit clone、、和git submodule update --init --recursive)。

設定專案

若要啟用示範，您需要設定要使用的專案AWS IoT。若要將專案設定為與 AWS IoT 搭配使用，必須將您的裝置註冊為 AWS IoT 實物。您應該會在 為 FreeRTOS 低功耗藍牙設置 AWS IoT 和 Amazon Cognito 時註冊您的裝置。

設定 AWS IoT 端點

1. 登入 AWS IoT 主控台。
2. 在導覽窗格中，選擇 Settings (設定)。

   您的AWS IoT端點會顯示在裝置資料端點文字方塊中。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)。記下您的裝置的 AWS IoT 實物名稱。
4. 備妥您的 AWS IoT 端點和您的 AWS IoT 物件名稱，在您的 IDE 中開啟*freertos*/demos/include/aws_clientcredential.h，並指定下列 #define 常數的值：

   - clientcredentialMQTT_BROKER_ENDPOINT *## 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_demo_config.h，並按照此範CONFIG_OTA_HTTP_BLE_TRANSPORT_DEMO_ENABLED例中的方式定義CONFIG_OTA_MQTT_BLE_TRANSPORT_DEMO_ENABLED或。

```
/* To run a particular demo you need to define one of these.
```

```
 * Only one demo can be configured at a time
 *
 *          CONFIG_BLE_GATT_SERVER_DEMO_ENABLED
 *          CONFIG_MQTT_BLE_TRANSPORT_DEMO_ENABLED
 *          CONFIG_SHADOW_BLE_TRANSPORT_DEMO_ENABLED
 *          CONFIG_OTA_MQTT_BLE_TRANSPORT_DEMO_ENABLED
 *          CONFIG_OTA_HTTP_BLE_TRANSPORT_DEMO_ENABLED
 *          CONFIG_POSIX_DEMO_ENABLED
 *
 *  These defines are used in iot_demo_runner.h for demo selection */

#define CONFIG_OTA_MQTT_BLE_TRANSPORT_DEMO_ENABLED
```

3. 由於 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。當您新增特定於供應商的 UUID 時，將此計數增加 1。

   - 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 此值表示屬性表格專用的記憶體設定數量 (包括特徵大小)，因此這會因專案而異。如果超過屬性表的大小，您將得到一個 NRF_ERROR 錯誤 _ NO_MEM 錯誤。如果您修改了 NRF_SDH_BLE_GATT_ATTR_ 標籤大小，通常您也必須重新設定記憶體設定。

   (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 並設定您的示範專案之後，您就可以在主機板上建置並執行示範專案了。

> **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 嵌入式工作室構建和運行 FreeRTOS 藍牙低功耗演示

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. 以滑鼠右鍵按一下專案總管中的**aws_demos**示範專案，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。請求免費許可證然後，在安裝過程中選擇激活您的免費許可證，然後按照說明進行操作。
4. 選擇 Debug (除錯)，然後選擇 Go (前往)。

   該示範開始執行後，即會等待以低功耗藍牙與行動裝置配對。
5. 按照 MQTT 藍牙低功耗示範應用程式的說明，以 FreeRTOS 藍牙低功耗行動 SDK 示範應用程式做為行動 MQTT 代理完成示範。

疑難排解

如需 FreeRTOS 入門的一般疑難排解資訊，請參閱故障診斷入門。

開始使用新唐 NuMaker-IOT M487 系列產品

> **Important**
>
> 此參考集成託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。我們建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱. 亞馬遜自由 Github 存儲庫遷移指南

本教學提供如何開始使用新唐 NuMaker-IOT-M487 開發板的操作說明。該系列微控制器，並包括內置 RJ45 以太網和 Wi-Fi 模塊。如果您沒有新唐 NuMaker-IOT-M487，請造訪AWS 合作夥伴裝置目錄，向我們的合作夥伴購買。

在開始之前，您必須先設定 AWS IoT FreeRTOS 軟體，才能將開發板連線到雲端 AWS 。如需說明，請參閱首要步驟。在此教學課程中，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為二進位映像檔。
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。

若要安裝 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. 安裝新唐開發工具頁面上的 NUL 連結驅動程式 (或最新版本)。

建置並執行 FreeRTOS 示範專案

若要建置 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)。

您可以使用主控台中的 MQTT 用戶端監 AWS IoT 控裝置傳送至雲端的訊息。 AWS

若要向 MQTT 用戶端訂閱 MQTT 主題 AWS IoT

1. 登入 AWS IoT 主控台。
2. 在導覽窗格中，選擇 [測試]，然後選擇 [MQTT 測試用戶端] 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。

若要執行 FreeRTOS 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))。

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

搭配 FreeRTOS 體使用 CMake

您也可以使用 CMake 來建置和執行 FreeRTOS 示範應用程式或使用協力廠商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 搭配使用 FreeRTOS 體 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`。

您也可以使用主控台中的 MQTT 用戶端監 AWS IoT 控裝置傳送至雲端的訊息。 AWS

若要向 MQTT 用戶端訂閱 MQTT 主題 AWS IoT

1. 登入 AWS IoT 主控台。
2. 在導覽窗格中，選擇 [測試]，然後選擇 [MQTT 測試用戶端] 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. 在主機上，開啟命令提示字元並導覽至 *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：

     瀏覽至包含 FreeRTOS 示範專案之來源檔案的目錄。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：

  瀏覽至包含 FreeRTOS 示範專案之來源檔案的目錄。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
```

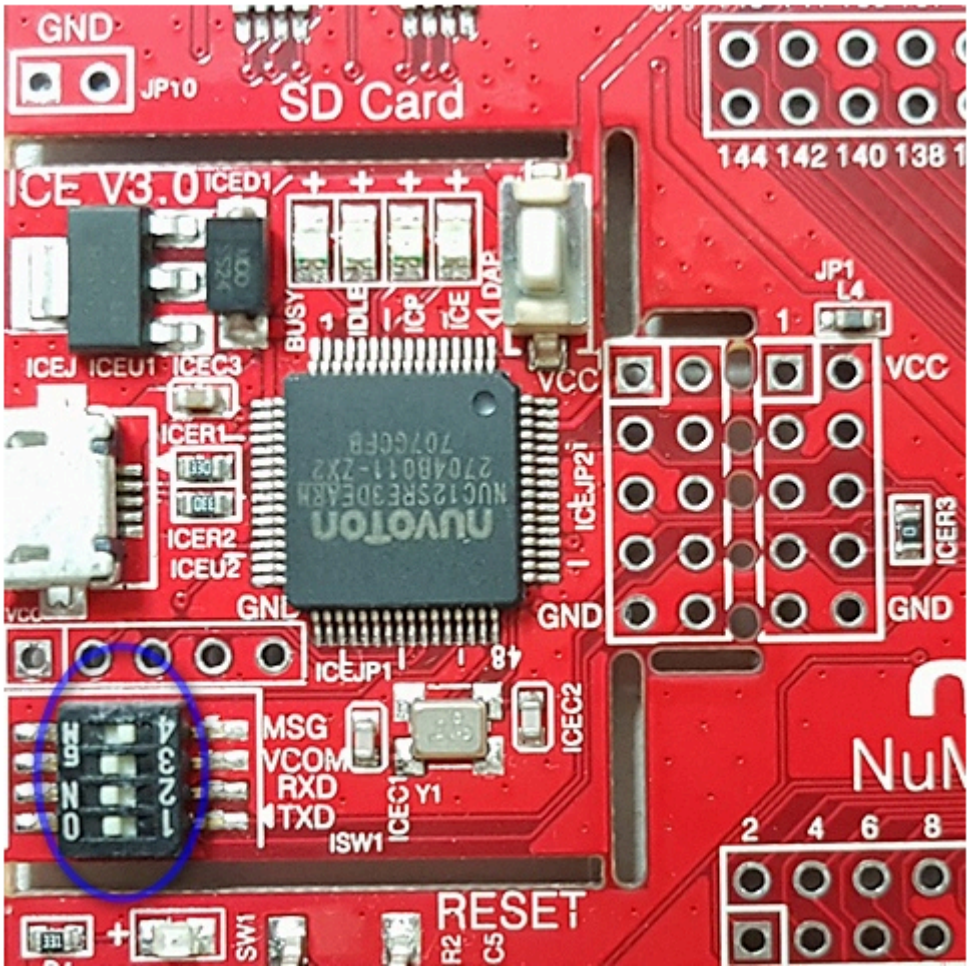
   此時，二進位檔案 aws_demos.bin 應該位於 *BUILD_FOLDER*/vendors/Nuvoton/boards/numaker_iot_m487_wifi 資料夾。

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。

   如果您透過 MQTT 用戶端訂閱 MQTT 主題，您應該會在主 AWS IoT 控台中看到裝置傳送的 MQTT 訊息。 AWS IoT

故障診斷

- 如果您的窗口無法識別設備VCOM，請從鏈接 NU-Link USB 驅動程序 v1.6 安裝 NuMaker Windows 串行端口驅動程序。
- 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。



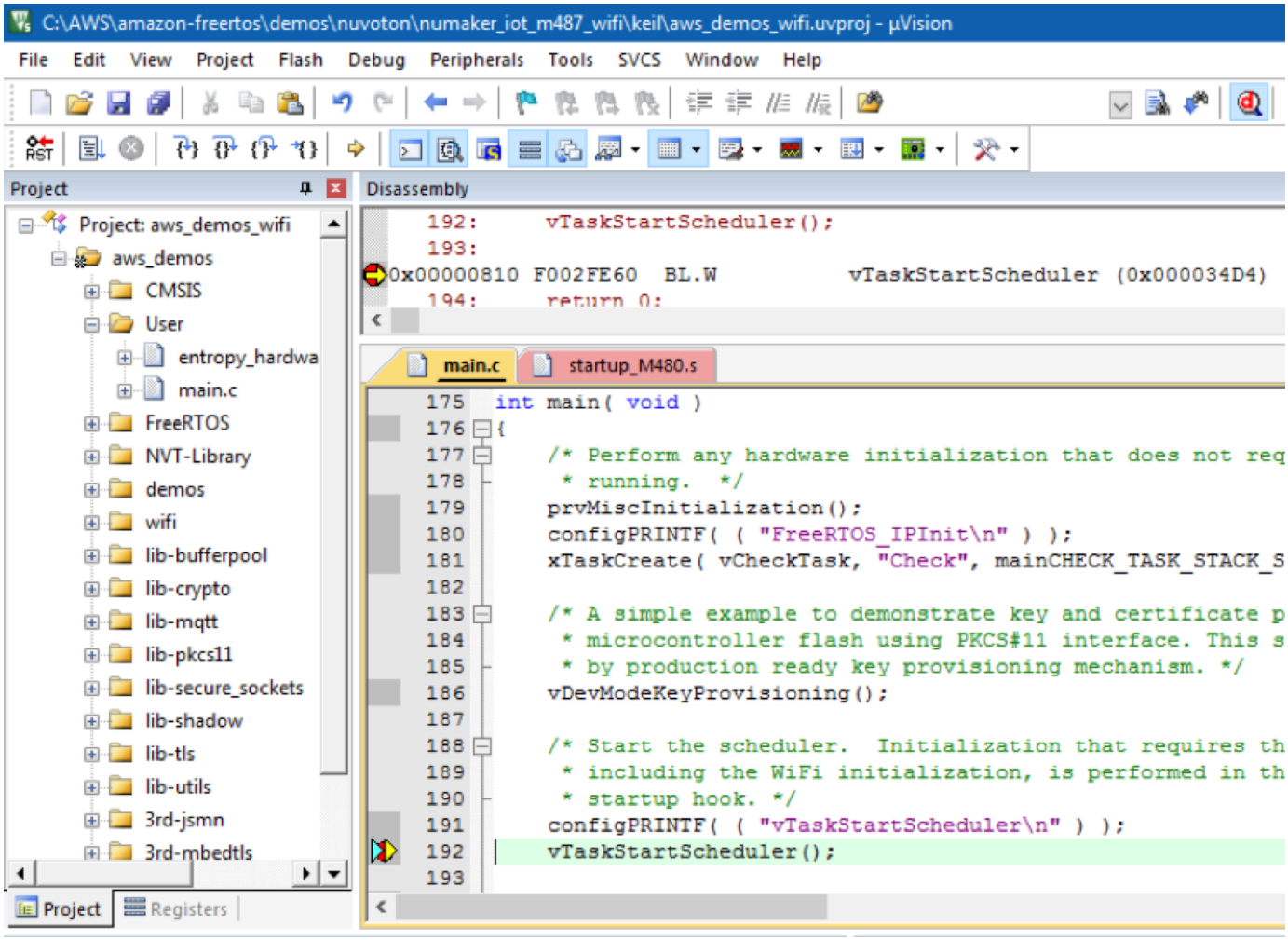
如果您在設定開發環境或連接到主機板時遇到問題，請聯絡 Nuvoton。

在基爾微視野中調試 FreeRTOS 項目

在 Keil µVision 中啟動偵錯工作階段

1. 開啟 Keil µVision。
2. 請依照下列步驟在中建置 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：

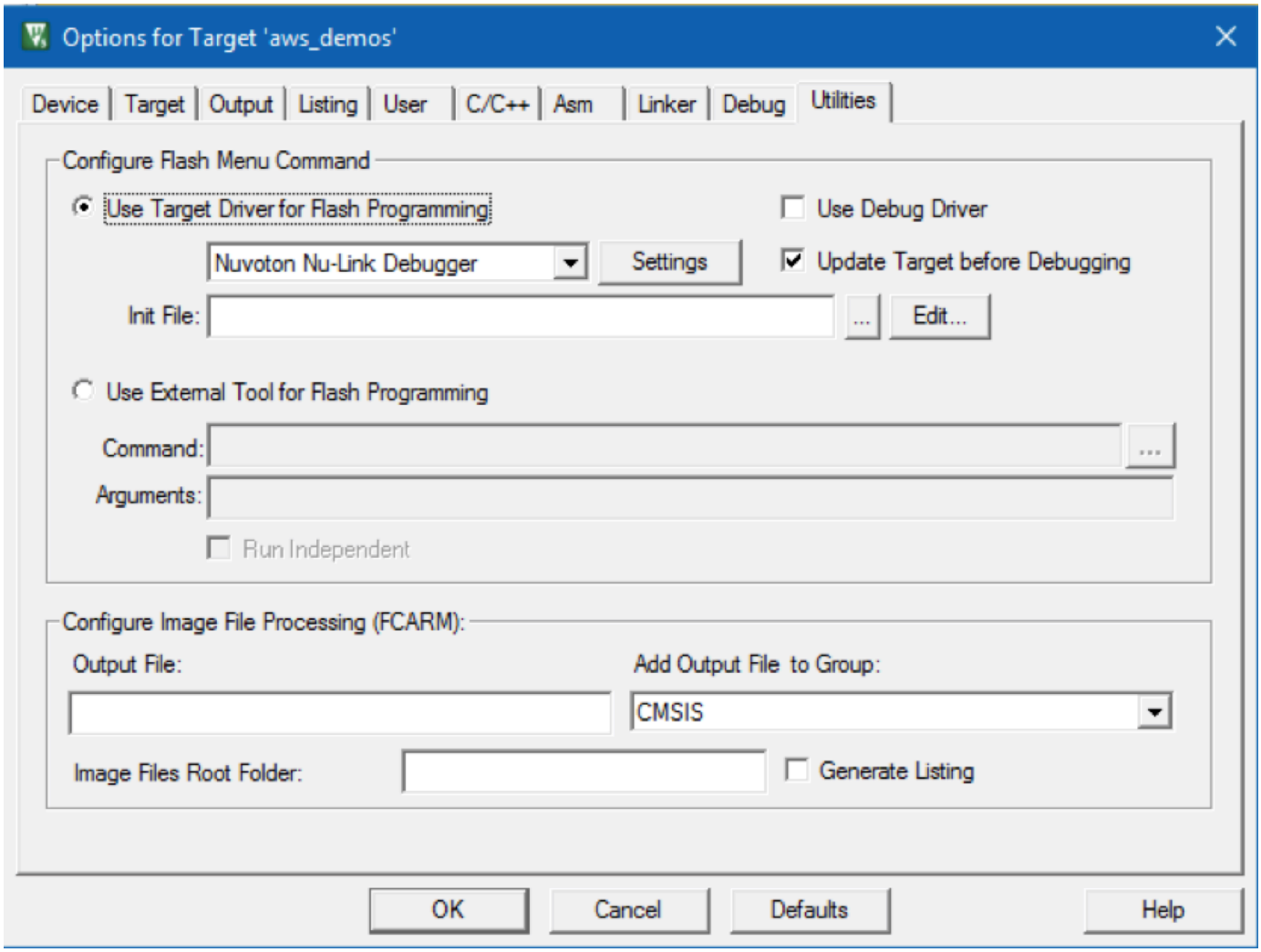


故障診斷 µVision 偵錯設定

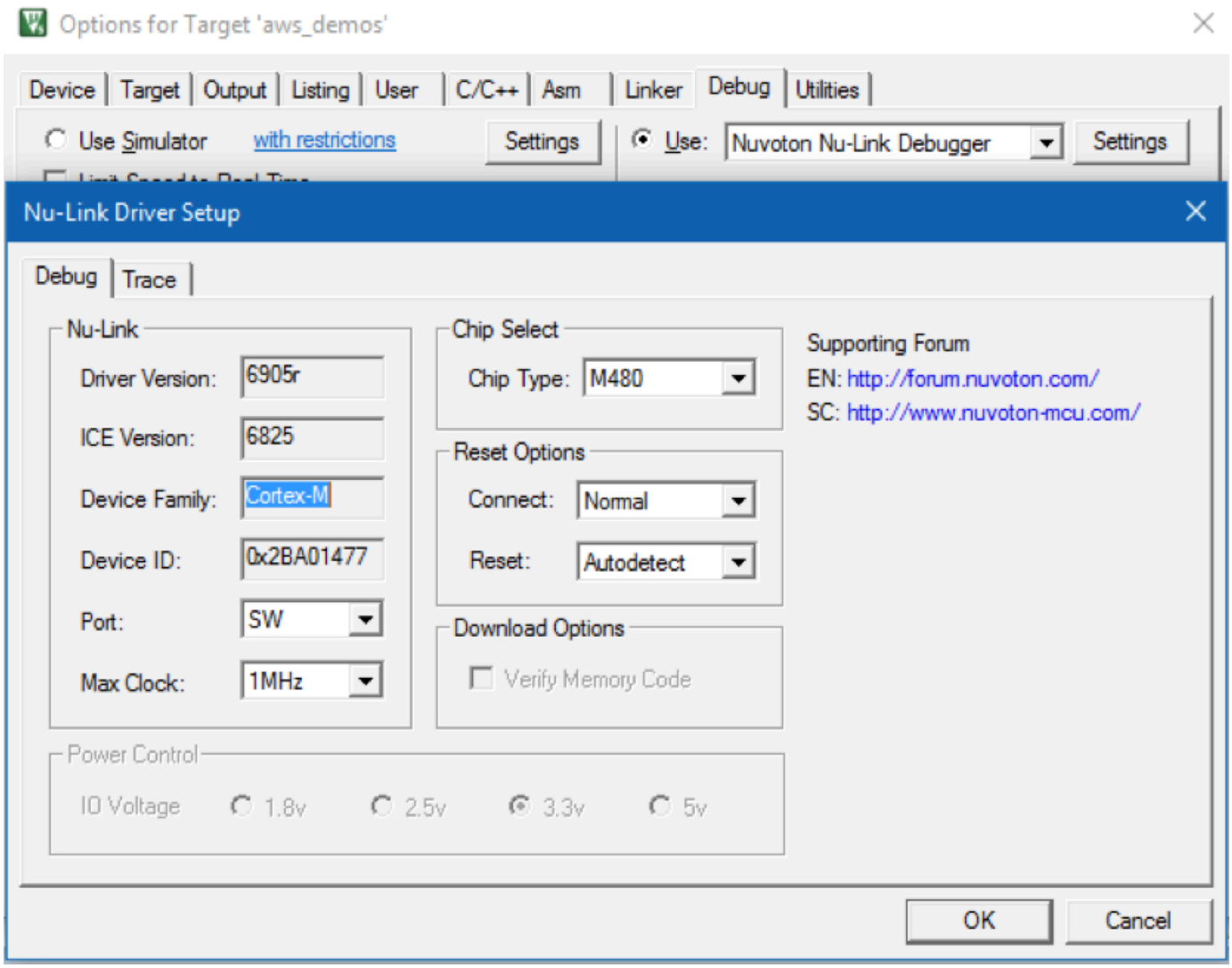
如果您在偵錯應用程式時遇到問題，請檢查您的偵錯設定是否在 Keil µVision 中正確設定。

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**
>
> 此參考集成託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。我們建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱. 亞馬遜自由 Github 存儲庫遷移指南

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 合作夥伴裝置目錄，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。

在開始之前，您必須先設定 FreeRTOS AWS IoT 並下載，才能將裝置連線到雲端 AWS 。如需說明，請參閱 首要步驟。在此教學課程中，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. 將 FreeRTOS 示範應用程式交叉編譯為二進位映像檔。
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 除錯程式來啟動並除錯在恩智浦 LPC54018 主機板上執行的程式碼。FreeRTOS 使用 OM40006 IoT 模組進行測試。如需有關支援除錯器的詳細資訊，請參閱可從 OM40007 LPC54018 IoT 模組產品頁面取得的恩智浦 LPC54018 IoT 模組使用者手冊。

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 支援兩個適用於恩智浦 LPC54018 IoT 模組的 IDE：IAR 嵌入式工作台和麥卡派索。

開始之前，請先安裝其中一種 IDE。

安裝 IAR Embedded Workbench for ARM

1. 瀏覽至適用於 ARM 的 IAR 嵌入式工作台並下載軟體。

   > **Note**
   >
   > IAR Embedded Workbench for ARM 必須使用 Microsoft Windows。

2. 運行安裝程序並按照提示進行操作。
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。

監控雲端的 MQTT 訊息

在執行 FreeRTOS 示範專案之前，您可以在主控台中設定 MQTT 用戶端，以監 AWS IoT 控裝置傳送至雲端的訊息。 AWS

若要向 MQTT 用戶端訂閱 MQTT 主題 AWS IoT

1. 登入 AWS IoT 主控台。
2. 在導覽窗格中，選擇 [測試]，然後選擇 [MQTT 測試用戶端] 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。

當演示項目成功運行在您的設備上，你會看到「你好世界！」 多次發送到您訂閱的主題。

建置並執行 FreeRTOS 示範專案

將 FreeRTOS 導入您的 IDE

若要將 FreeRTOS 範例程式碼匯入 IAR 嵌入式工作台 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)。

若要將 FreeRTOS 範例程式碼匯入至麥卡派索 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 嵌入式工作台 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。

使用麥卡派索 IDE 執行 FreeRTOS 師示範專案

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)。

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**
>
> 此參考集成託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。我們建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱. 亞馬遜自由 Github 存儲庫遷移指南

本教學課程提供 RX65N-2MB 的 Renesas Starter Kit+ 入門指示。如果您沒有 RX65N-2MB 適用的瑞薩 RSK+，請瀏覽 AWS 合作夥伴裝置目錄，並向我們的合作夥伴購買。

在開始之前，您必須先設定 FreeRTOS AWS IoT 並下載，才能將裝置連線到雲端 AWS 。如需說明，請參閱 首要步驟。在此教學課程中，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. 將 FreeRTOS 示範應用程式交叉編譯為二進位映像檔。
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

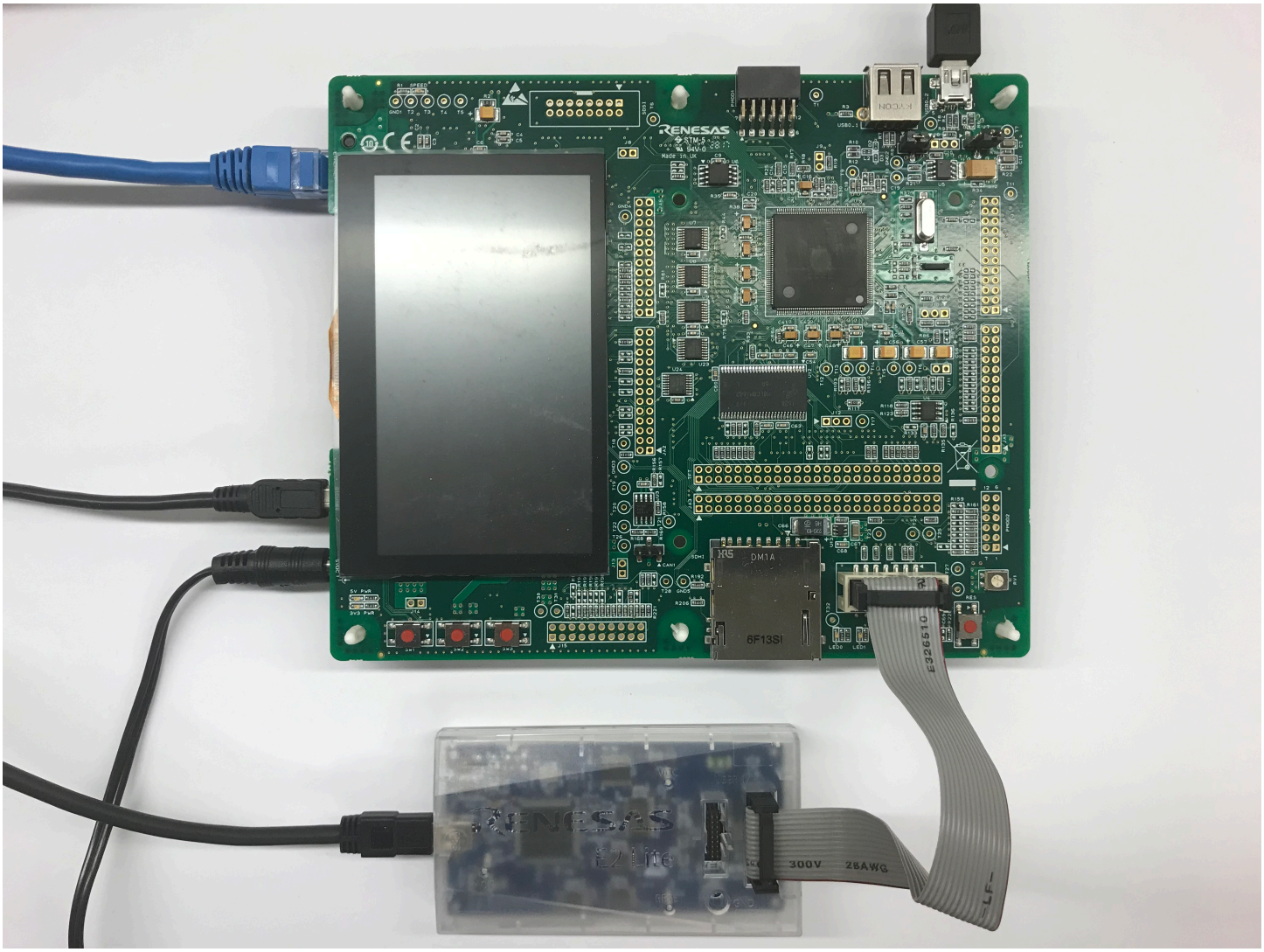
1. 將正極 +5V 電源整流器連接到 RSK+RX65N-2MB 的 PWR 連接器。
2. 將您的電腦連線至 RSK+ for RX65N-2MB 上的 USB 2.0 FS 連接埠。
3. 將您的電腦連線至 RSK+ for RX65N-2MB 上的 USB 序列連接埠。
4. 將路由器或連接網際網路的乙太網路連接埠連接至 RSK+ for RX65N-2MB 上的乙太網路連接埠。

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

若要為 RX65N-2MB 專用的 RSK+ 設定 FreeRTOS 務器組態，請使用瑞薩工作室 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. 前往瑞薩 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。

   如果您擁有瑞薩帳戶，請輸入您的登入認證，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，請參閱瑞薩網站上的 e ^{2} 工作室。

下載並安裝 RX Family C/C++ 編譯器套件

1. 轉到 RX 系列 C/C ++ 編譯器 Package 下載頁面，然後下載 V3.00.00 軟件包。
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。

建置及執行 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。

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 以使用代碼生成器」下，選擇 「確定」。
6. 出現重新啟動 e ^{2} 工作室窗口。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} 工作室歡迎屏幕上，選擇轉到 e ^{2} 工作室工作台箭頭圖標。
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。

監控雲端的 MQTT 訊息

在執行 FreeRTOS 示範專案之前，您可以在主控台中設定 MQTT 用戶端，以監 AWS IoT 控裝置傳送至雲端的訊息。 AWS

若要向 MQTT 用戶端訂閱 MQTT 主題 AWS IoT

1. 登入 AWS IoT 主控台。

2. 在導覽窗格中，選擇 [測試]，然後選擇 [MQTT 測試用戶端] 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。

當演示項目成功運行在您的設備上，你會看到「你好世界！」 多次發送到您訂閱的主題。

執行 FreeRTOS 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. 展開瑞薩 GDB 硬體除錯，然後選擇 aws_示範。 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。

有關瑞薩發行的最新專案，請參閱上的存放`amazon-freertos`庫`renesas-rx`分支。GitHub

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**
>
> 此參考集成託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。我們建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱. 亞馬遜自由 Github 存儲庫遷移指南

本教學課程提供 STMicroelectronics STM32L4 Discovery Kit IoT Node 的入門指示。如果您尚未擁有意法半導體 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))。

在開始之前 AWS IoT，您必須配置 FreeRTOS 下載和 Wi-Fi 以將設備連接到雲端 AWS 。如需說明，請參閱 首要步驟。在此教學課程中，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. 將 FreeRTOS 示範應用程式交叉編譯為二進位映像檔。
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 系統工作台

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。

監控雲端的 MQTT 訊息

在執行 FreeRTOS 示範專案之前，您可以在主控台中設定 MQTT 用戶端，以監 AWS IoT 控裝置傳送至雲端的訊息。 AWS

若要向 MQTT 用戶端訂閱 MQTT 主題 AWS IoT

1. 登入 AWS IoT 主控台。
2. 在導覽窗格中，選擇 [測試]，然後選擇 [MQTT 測試用戶端] 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。

當演示項目成功運行在您的設備上，你會看到「你好世界！」 多次發送到您訂閱的主題。

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。(請參閱主機板隨附的製造商說明文件，找出要使用的正確 USB 連接埠。)
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 體使用 CMake

如果您不想使用 IDE 進行 FreeRTOS 開發，也可以使用 CMake 來構建和運行使用第三方代碼編輯器和調試工具開發的演示應用程序或應用程序。

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
```

如需將 CMake 與 FreeRTOS 搭配使用的詳細資訊，請參閱。搭配使用 FreeRTOS 體

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**
>
> 此參考集成託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。我們建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱. 亞馬遜自由 Github 存儲庫遷移指南

本教學課程提供 Texas Instruments CC3220SF-LAUNCHXL 入門指示。如果您沒有德州儀器 (TI) CC3220SF-LAUNCHXL 開發套件，請造訪 AWS 合作夥伴裝置目錄，向我們的合作夥伴購買。

在開始之前，您必須先設定 FreeRTOS AWS IoT 並下載，才能將裝置連線到雲端 AWS 。如需說明，請參閱 首要步驟。在此教學課程中，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. 將 FreeRTOS 示範應用程式交叉編譯為二進位映像檔。
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 IDE：程式碼撰寫工作室和 IAR 嵌入式工作台 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. 若要安裝的產品系列，請選擇 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 無線網路 CC3220 SDK 包含 CC3220SF 可程式化 MCU 的驅動程式、超過 40 個範例應用程式，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 主機板出現在偵測到的裝置下，請選擇 [開始]。如果未偵測到您的主機板，請從 [新增組態] 下的主機板清單中選擇 CC3220SF-LAUNCHXL，然後選擇 [啟動映像建立工具]。
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 [瀏覽]，然後瀏覽至您安裝 CC3220SF SimpleLink SDK 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 範
- SmartConfig從德州儀器公司使用。

建置並執行 FreeRTOS 示範專案

在 TI 程式碼撰寫器中建置並執行 FreeRTOS 示範專案

若要將 FreeRTOS 示範匯入 TI 程式碼撰寫器

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。

若要在 TI 程式碼撰寫器中執行 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 無線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)。

監控雲端的 MQTT 訊息

在執行 FreeRTOS 示範專案之前，您可以在主控台中設定 MQTT 用戶端，以監 AWS IoT 控裝置傳送至雲端的訊息。 AWS

若要向 MQTT 用戶端訂閱 MQTT 主題 AWS IoT

1. 登入 AWS IoT 主控台。
2. 在導覽窗格中，選擇 [測試]，然後選擇 [MQTT 測試用戶端] 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。

當演示項目成功運行在您的設備上，你會看到「你好世界！」 多次發送到您訂閱的主題。

在 IAR 嵌入式工作台中建置並執行 FreeRTOS 示範專案

若要將 FreeRTOS 示範匯入 IAR 嵌入式工作台

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)。

若要在 IAR 嵌入式工作台中執行 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 無線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)。如果顯示的話 EnergyTrace，您可以忽略「警告：無法初始化」。如需相關資訊 EnergyTrace，請參閱 MSP EnergyTrace 技術。
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 體使用 CMake

如果您不想使用 IDE 進行 FreeRTOS 開發，也可以使用 CMake 來構建和運行使用第三方代碼編輯器和調試工具開發的演示應用程序或應用程序。

若要使用 CMake 建置 FreeRTOS 體示範

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 搭配使用 FreeRTOS 體。

故障診斷

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

若要為 TI 主機板進行偵錯設定

1. 在 Code Composer 的 Project Explorer (專案瀏覽器) 中，選擇 aws_demos。
2. 在 Run (執行) 選單中，選擇 Debug Configurations (組態除錯)。
3. 在導覽窗格中，選擇 aws_demos。
4. 在 Target (目標) 標籤中，選擇 Connection Options (連線選項) 下方的 Reset the target on a connect (重設連線目標)。
5. 選擇 Apply (套用)，然後選擇 Close (關閉)。

如果無法使用這些步驟，請查看序列終端機中的程式輸出。您應該會看到一些文字，指出問題來源。

如需 FreeRTOS 入門的一般疑難排解資訊，請參閱。故障診斷入門

Windows 裝置模擬器入門

本教學課程提供如何開始使用 FreeRTOS 視窗裝置模擬器的指示。

在開始之前，您必須先設定 FreeRTOSAWS IoT 並下載，才能將裝置連線到AWS雲端。如需說明，請參閱 首要步驟。在此教學課程中，FreeRTOS 下載目錄的路徑稱為*freertos*。

FreeRTOS 會以壓縮檔的形式發行，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 Npcap。在安裝過程中選擇「WinPcap API 兼容模式」。
2. 安裝 Microsoft Visual Studio。

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

   除 IDE 外，安裝使用 C++ 的桌面開發元件。

   安裝最新的 Windows 10 軟體開發套件。您可以在使用 C ++ 組件的桌面開發的可選部分下選擇此選項。
3. 請確認您的有線乙太網路連線為作用中。
4. （可選）如果您想使用基於 CMake 的構建系統來構建 FreeRTOS 項目，請安裝最新版本的 CMake。FreeRTOS 需要 CMake 版本 3.13 或更新版本。

監控雲端的 MQTT 訊息

在執行 FreeRTOS 示範專案之前，您可以在主控台中設定 MQTT 用戶端，以監AWS IoT控裝置傳送至AWS雲端的訊息。

使用 AWS IoT MQTT 用戶端訂閱 MQTT 主題

1. 登入 AWS IoT 主控台。
2. 在導覽窗格中，選擇 [測試]，然後選擇 [MQTT 測試用戶端] 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。

當演示項目成功運行在您的設備上，你會看到「你好世界！」 多次發送到您訂閱的主題。

建置並執行 FreeRTOS 示範專案

您可以使用視覺工作室或 CMake 來構建 FreeRTOS 項目。

使用 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。選擇電腦上存在的 Windows SDK 版本 (下拉式清單中的初始值很好)，然後選擇 [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。第一次執行時，您必須選取網路介面。

使用 CMake 構建和運 FreeRTOS 演示項目

我們建議您使用 CMake 圖形用戶界面而不是 CMake 命令行工具來構建 Windows 模擬器的演示項目。

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 的原始程式碼目錄。

   在 GUI 中，將 FreeRTOS 原始程式碼目錄 (*`freertos`*) 設定為原始程式碼在哪裡。

   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

類型

STRING

值

pc.windows

描述

(選用)

3. 選擇 Configure (設定)。如果 CMake 提示您建立建置目錄，請選擇 Yes (是)，然後選取 Specify the generator for this project (為此專案指定產生器) 下的產生器。我們建議您使用 Visual Studio 作為產生器，但也支援使用 Ninja。(請注意，使用 Visual Studio 2019 時，版本應設為 Win32，而非使用其預設設定。) 保持其他產生器選項不變，然後選擇「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。針對此教學的目的，您可以將此選項保留為預設值。

   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aws_demos專案上按一下滑鼠右鍵，然後選擇 [設定為 StartUp 專案]。這可讓您建置並執行該專案。第一次執行時，您必須選取網路介面。

如需有關使用 CMake 搭配 FreeRTOS 搭配的詳細資訊，請參閱搭配使用 FreeRTOS 體。

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。該程序計算您的網絡接口。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。在上一個範例中，要使用的數字為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。

疑難排解

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。

如需有關 FreeRTOS 入門的一般疑難排解資訊，請參閱故障診斷入門。

開始使用賽靈思安 MicroZed 富利工業 IoT 套件

> **Important**
>
> 此參考集成託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。我們建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱. 亞馬遜自由 Github 存儲庫遷移指南

本教學提供如何開始使用賽靈思安富利工業 MicroZed IoT 套件的指示。如果您沒有 Xilinx Avnet MicroZed 工業 IoT 套件，請造訪 AWS 合作夥伴裝置目錄，向我們的合作夥伴購買。

在開始之前，您必須先設定 FreeRTOS AWS IoT 並下載，才能將裝置連線到雲端 AWS 。如需說明，請參閱 首要步驟。在此教學課程中，FreeRTOS 下載目錄的路徑稱為。*freertos*

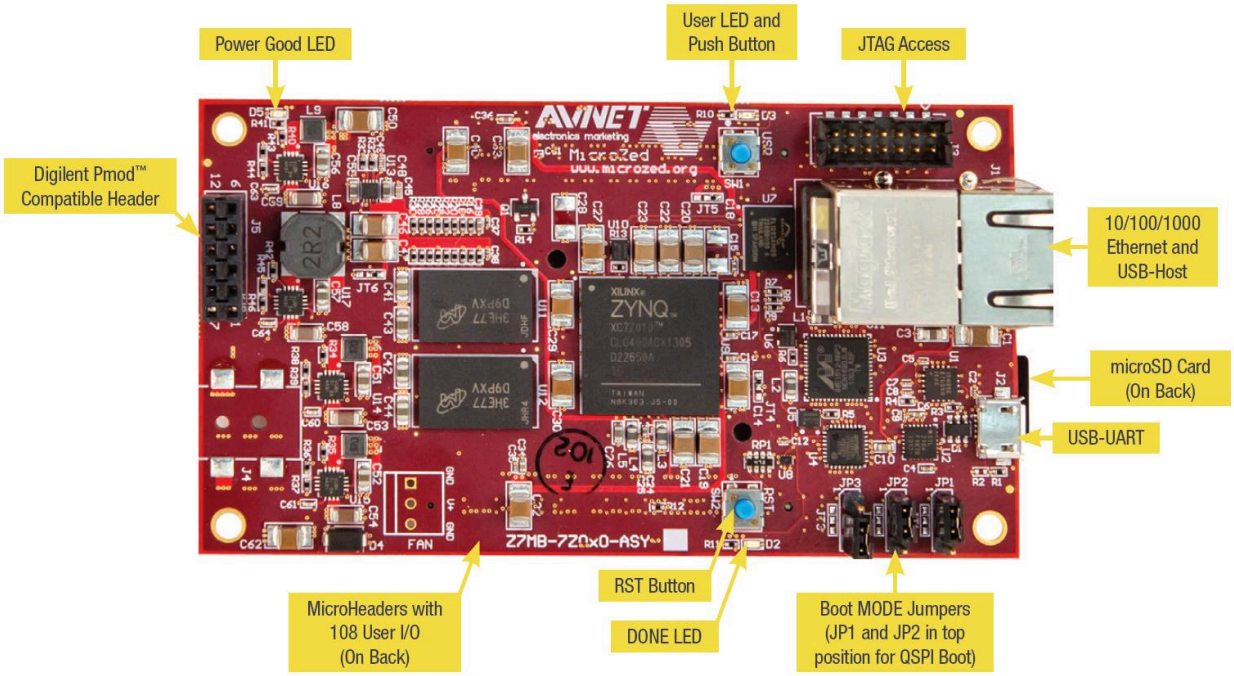
概觀

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. 將 FreeRTOS 示範應用程式交叉編譯為二進位映像檔。
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，下圖可能會有所幫助：



若要設定主 MicroZed 機板

1. 將電腦 Connect 至主機板上的 USB-UART 連接 MicroZed 埠。
2. 將電腦 Connect 至主機 MicroZed 板上的 JTAG 存取連接埠。
3. 將路由器或 Connect 網際網路的乙太網路連接埠連接至主機板上的乙太網路和 USB 主機連接埠。MicroZed

設定開發環境

若要為 MicroZed 套件設定 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. 前往軟體開發套件獨立式 WebInstall 用戶端下載頁面。
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。

   如果您擁有 Xilinx 的帳戶，請輸入您的登錄憑據，然後選擇「登錄」。

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 MicroZed的電腦未偵測到 USB UART 連線，請手動安裝 CP210 USB 對橋接器 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。

監控雲端的 MQTT 訊息

在執行 FreeRTOS 示範專案之前，您可以在主控台中設定 MQTT 用戶端，以監 AWS IoT 控裝置傳送至雲端的訊息。 AWS

若要向 MQTT 用戶端訂閱 MQTT 主題 AWS IoT

1. 登入 AWS IoT 主控台。
2. 在導覽窗格中，選擇 [測試]，然後選擇 [MQTT 測試用戶端] 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

在 XSDK IDE 中開啟 FreeRTOS 使用者示範

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。

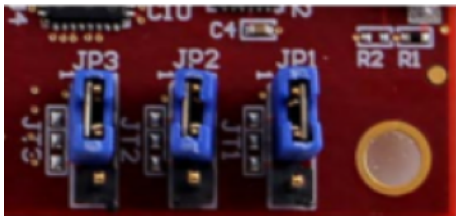
> **Note**
>
> 若要建立所有專案，而不清除它們，請選擇 Project (專案)，然後選擇 Build All (全部建立)。
> 若要建立個別專案，請選取您要建置的專案、選擇 Project (專案)，然後選擇 Build Project (建置專案)。

為 FreeRTOS 示範專案產生開機映像

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

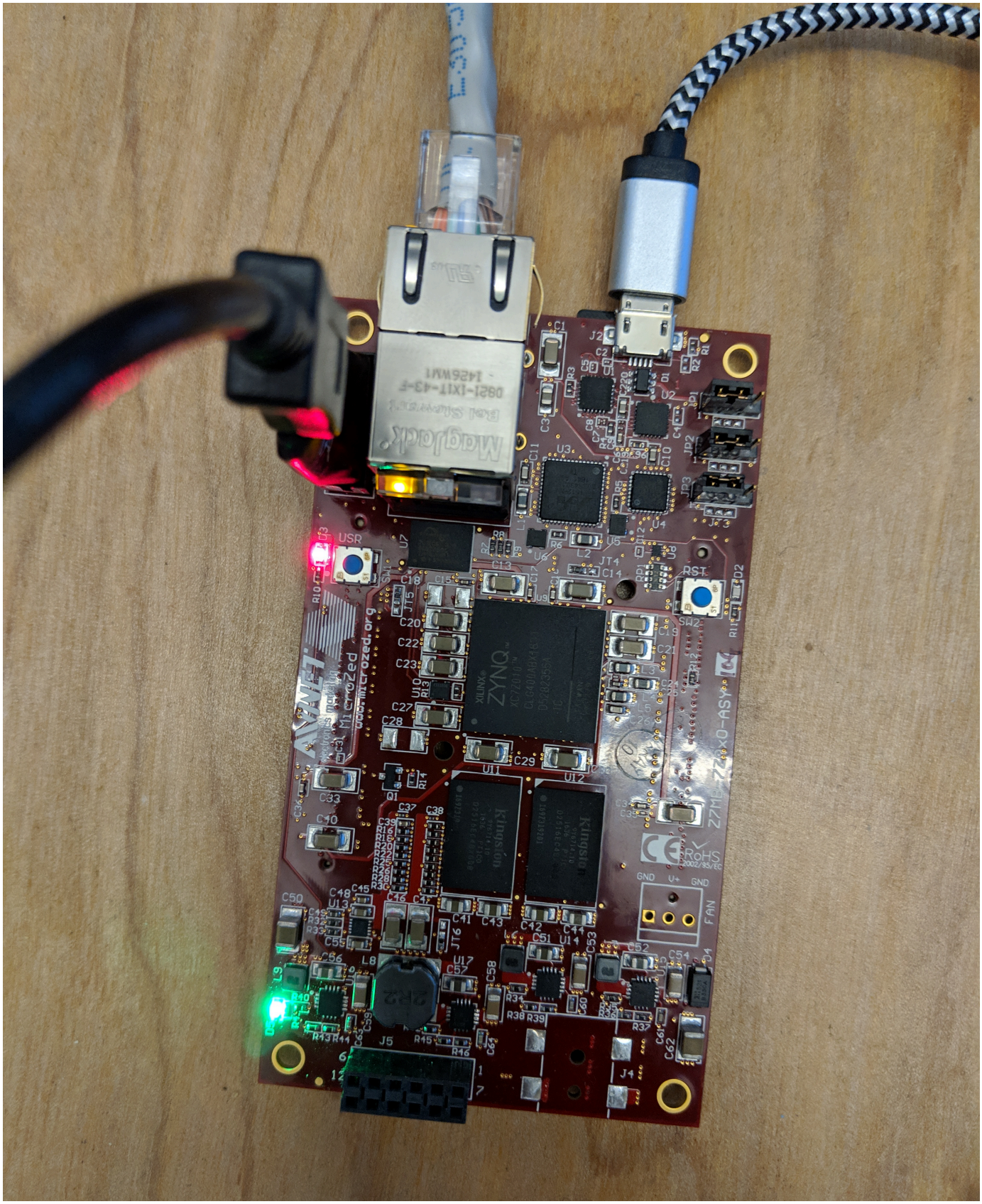
1. 將主機 MicroZed 板的開機模式跳接器設定為 JTAG 開機模式。



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
GND
V+
GND
FAN

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 記憶卡或 QSPI 啟動主機 MicroZed 板，以執行 FreeRTOS 示範專案。如需說明，請參閱 為 FreeRTOS 示範專案產生開機映像 與 執行 FreeRTOS 範專案。

執行 FreeRTOS 範專案

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

當您設定 MicroZed 主機板來執行 FreeRTOS 示範專案時，請參閱中的圖表。設定 MicroZed 硬體請確定您已將 MicroZed 主機板連接到電腦。

從 MicroSD 卡啟動自由攝影機專案

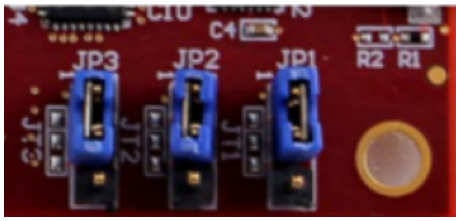
格式化賽靈思工業 IoT 套件隨附 MicroZed的 microSD 卡。

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 開機模式跳線設定為 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. 將主機 MicroZed 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。針對「連線」，請選擇您的 MicroZed 硬體伺服器，將主機板與主機電腦連接。

> **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. 將主機 MicroZed 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。

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 務的後續步驟

> **Important**
>
> 此頁面指的是已棄用的亞馬遜 FreeRTOS 存儲庫。建議您在建立新專案時入 如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱亞馬遜自由 Github 存儲庫遷移指南.

在您為主機板建置、快閃執行 FreeRTOS 示範專案之後，您可以造訪 FreeRTOS 網站，進一步瞭解如何建立新的 FreeRTOS 專案。此外，還有許多 FreeRTOS 程式庫的示範，說明如何執行重要工作、與AWS IoT服務互動，以及程式化主機板特定功能 (例如行動數據機)。如需詳細資訊，請參閱 FreeRTOS 程式庫類別頁面。

FreerTos.org 網站也有關於 FreeRTOS 核心的深入資訊，以及基本的即時作業系統概念。如需詳細資訊，請參閱 FreeRTOS 核心開發人員文件和 FreeRTOS 核心次要文件頁面。

空中免費更新

> **Note**
>
> 有關執行 O（OTA）更新的最新信息，請參閱 FreeRTOS 網站上的AWS IoT無線ver-the-air（OTA）。

O ver-the-air (OTA) 更新可讓您將韌體更新部署到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 演示應用程序在您的設備上運行，並創建一個 FreeRTOS 任務，該任務訂閱AWS IoT 作業的通知主題並監聽更新消息。有更新可用時，OTA 代理程式會根據您選擇的設定，使用 HTTP 或 MQTT 通訊協定發佈要求AWS IoT並接收更新。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，則必須將其整合AWS IoT空中 (OTA) 圖書館到您自己的應用程式中，才能取得韌體更新功能。

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 (例如依據目的、擁有者或環境) 將 AWS IoT 資源分類。這在您擁有許多相同類型的資源時很有用。您可以根據指派給資源的標籤來快速識別資源。

如需詳細資訊，請參閱標記您的 AWS IoT 資源。

OTA 更新先決條件

要使用over-the-air（OTA）更新，請執行以下操作：

- 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 儲存貯體來存放您的更新.
- 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.
- 如果您使用適用於 AWS IoT 的程式碼簽章，將存取權限授予適用於 AWS IoT 的程式碼簽署。
- 使用 OTA 圖書館下載免費服務.

建立 Amazon S3 儲存貯體來存放您的更新

OTA 更新文件存儲在亞馬遜 S3 存儲桶中。

如果您是使用適用於 AWS IoT 的程式碼簽署，您用來建立程式碼簽署任務的命令會接收一個來源儲存貯體 (未簽署韌體映像所在的位置) 以及一個目的地儲存貯體 (簽署韌體映像的寫入位置)。您可以為來源及目標指定相同的儲存貯體。檔案名稱會變更為 GUID，因此原始檔案不會遭到覆寫。

建立 Amazon S3 儲存貯體

1. 登入亞馬遜 S3 主控台，網址為 https://console.aws.amazon.com/s3/。
2. 選擇 建立儲存貯體 。
3. 輸入值區名稱。
4. 在「封鎖公用存取」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
5. 在「值區版本控制」下，選取「啟用」，將所有版本保留在同一個值區
6. 選擇 建立儲存貯體 。

如需 Amazon S3 的詳細資訊，請參閱 Amazon 簡易儲存服務使用者指南。

建立 OTA 更新服務角色

OTA 更新服務會取得此角色，代您建立及管理 OTA 更新任務。

建立 OTA 服務角色

1. 請登入以下網址：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 Select type of trusted entity (選取可信任執行個體類型) 下，選取 AWS service (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。

> **Important**
>
> 要解決混淆的副安全問題，您必須按照AWS IoT Core指南中的說明進行操作。

將 OTA 更新許可新增到您的 OTA 服務角色

1. 在 IAM 主控台頁面的搜尋方塊中，輸入角色的名稱，然後從清單中選擇。
2. 選擇 Attach policies (連接政策)。
3. 在 [搜尋] 方塊中，輸入「AmazonFreeRTOsotaUpdate」，從篩選策略清單中選取 AmazonFreeRTOsotaUpdate，然後選擇 [附加原則]，將原則附加至您的服務角色。

將所需的 IAM 許可添加到您的 OTA 服務角色

1. 在 IAM 主控台頁面的搜尋方塊中，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. 請選擇 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"
      }
    ]
}
```

   確保您將 *<your_account_id>* 取代為您的 AWS 帳戶 ID，並將 *<your_role_name>* 取代為 OTA 服務角色的名稱。

5. 選擇 Review policy (檢閱政策)。

6. 輸入政策名稱，然後選擇 Create policy (建立政策)。

> **Note**
>
> 如果您的 Amazon S3 儲存貯體名稱以「afr-ota」開頭，則不需要執行下列程序。若是如此，則AWS 受管政策 `AmazonFreeRTOSOTAUpdate` 已包含必要許可。

將所需的 Amazon S3 許可新增至您的 OTA 服務角色

1. 在 IAM 主控台頁面的搜尋方塊中，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. 請選擇 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 物件的權限。確保您將 *example-bucket* 取代為儲存貯體的名稱。

5. 選擇 Review policy (檢閱政策)。
6. 輸入政策名稱，然後選擇 Create policy (建立政策)。

建立 OTA 使用者政策

您必須授與使用者執行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 AWS Certificate Manager 中的憑證。
- 存取以 AWS IoT MQTT 為基礎的檔案傳遞功能。
- 存取免費使用者 OTA 更新。
- 存取 AWS IoT 任務。
- 存取 IAM。
- 存取適用於 AWS IoT 的程式碼簽署。請參閱 將存取權限授予適用於 AWS IoT 的程式碼簽署。
- 列出免費伺服器硬體平台。
- 標記和取消標記AWS IoT資源。

若要授與使用者所需的許可，請參閱 IAM 政策。另請參閱授權使用者和雲端服務以使用AWS IoT工作。

若要提供存取權，請新增許可到您的使用者、群組或角色：

- AWS IAM Identity Center 中的使用者和群組：

  建立許可集合。請遵循《AWS IAM Identity Center 使用者指南》的建立許可集合中的指示。
- 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。請遵循《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。請遵循《IAM 使用者指南》的為 IAM 使用者建立角色中的指示。
  - (不建議) 將政策直接連接至使用者，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

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密金鑰。對於生產環境，請透過知名的憑證授權單位 (CA) 購買憑證。

不同平台需要不同類型的程式碼簽署憑證。以下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 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 視窗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**
>
> 此參考集成託管在亞馬遜的 Freertos 存儲庫上，該存儲庫已被棄用。我們建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱. 亞馬遜自由 Github 存儲庫遷移指南

SimpleLinkWi-Fi CC3220SF 無線微控制器啟動板開發套件支援兩個用於韌體程式碼簽章的憑證鏈：

- 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)

  操場證書鏈允許您使用自簽代碼簽名證書嘗試 OTA 更新。

使用 AWS Command Line Interface 來將程式碼簽署憑證、私密金鑰及憑證鏈匯入至 AWS Certificate Manager。如需詳細資訊，請參閱《使用指南》AWS CLI中的AWS Command Line Interface〈安裝〉。

下載並安裝最新版本的 SimpleLinkCC3220 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)

SimpleLinkCC3220 開發套件中的憑證採用 DER 格式。若要建立自我簽署的程式碼簽署憑證，您必須將它們轉換為 PEM 格式。

遵循這些步驟來建立連結到 Texas Instruments 遊樂場憑證階層的程式碼簽署憑證，並符合 AWS Certificate Manager 及適用於 AWS IoT 程式碼簽署的條件。

> **Note**
>
> 若要建立程式碼簽署的憑證，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、私有金鑰及憑證鏈匯入 AWS Certificate Manager：

   ```
   aws acm import-certificate --certificate fileb://tisigner.crt.pem --private-key
    fileb://tisigner.key --certificate-chain fileb://dummy-root-ca-cert.pem
   ```

此命令會顯示您憑證的 ARN。在建立 OTA 更新任務時，您將需要此 ARN。

> **Note**
>
> 在此撰寫的步驟假設您要使用適用於 AWS IoT 的程式碼簽署來簽署您的韌體映像。雖然建議使用適用於 AWS IoT 的程式碼簽署，但您也可以手動簽署您的韌體映像。

為 Espressif ESP32 建立程式碼簽署憑證

> **Important**
>
> 此參考集成託管在亞馬遜的 Freertos 存儲庫上，該存儲庫已被棄用。我們建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱. 亞馬遜自由 Github 存儲庫遷移指南

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 來將程式碼簽署憑證、私密金鑰及憑證鏈匯入至 AWS Certificate Manager。如需安裝 AWS CLI 的相關資訊，請參閱安裝 AWS CLI。

1. 在您的工作目錄中，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、私有金鑰及憑證鏈匯入 AWS Certificate Manager：

```
aws acm import-certificate --certificate fileb://ecdsasigner.crt --private-key
 fileb://ecdsasigner.key
```

此命令會顯示您憑證的 ARN。在建立 OTA 更新任務時，您將需要此 ARN。

> **Note**
>
> 在此撰寫的步驟假設您要使用適用於 AWS IoT 的程式碼簽署來簽署您的韌體映像。雖然建議使用適用於 AWS IoT 的程式碼簽署，但您也可以手動簽署您的韌體映像。

為 Nordic nrf52840-dk 建立程式碼簽署憑證

> **Important**
>
> 此參考集成託管在亞馬遜的 Freertos 存儲庫上，該存儲庫已被棄用。我們建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱. 亞馬遜自由 Github 存儲庫遷移指南

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 來將程式碼簽署憑證、私密金鑰及憑證鏈匯入至 AWS Certificate Manager。如需安裝 AWS CLI 的相關資訊，請參閱安裝 AWS CLI。

1. 在您的工作目錄中，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、私有金鑰及憑證鏈匯入 AWS Certificate Manager：

```
aws acm import-certificate --certificate fileb://ecdsasigner.crt --private-key
 fileb://ecdsasigner.key
```

此命令會顯示您憑證的 ARN。在建立 OTA 更新任務時，您將需要此 ARN。

> **Note**
>
> 在此撰寫的步驟假設您要使用適用於 AWS IoT 的程式碼簽署來簽署您的韌體映像。雖然建議使用適用於 AWS IoT 的程式碼簽署，但您也可以手動簽署您的韌體映像。

為 FreeRTOS 視窗模擬器建立程式碼簽署憑證

虛擬伺服器視窗模擬器需要具有 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 來將程式碼簽署憑證、私密金鑰及憑證鏈匯入至 AWS Certificate Manager。如需安裝 AWS CLI 的相關資訊，請參閱安裝 AWS CLI。

1. 在您的工作目錄中，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、私有金鑰及憑證鏈匯入 AWS Certificate Manager：

```
aws acm import-certificate --certificate fileb://ecdsasigner.crt --private-key
 fileb://ecdsasigner.key
```

此命令會顯示您憑證的 ARN。在建立 OTA 更新任務時，您將需要此 ARN。

> **Note**
>
> 在此撰寫的步驟假設您要使用適用於 AWS IoT 的程式碼簽署來簽署您的韌體映像。雖然建議使用適用於 AWS IoT 的程式碼簽署，但您也可以手動簽署您的韌體映像。

為自訂硬體建立程式碼簽署憑證

使用適當的工具組，為您的硬體建立自我簽署憑證及私有金鑰。

使用 AWS Command Line Interface 來將程式碼簽署憑證、私密金鑰及憑證鏈匯入至 AWS Certificate Manager。如需安裝 AWS CLI 的相關資訊，請參閱安裝 AWS CLI。

建立程式碼簽署憑證之後，您可以使用將其匯AWS CLI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AWS Certificate Manager憑證匯入。

您必須將程式碼簽署憑證的內容複製、貼上並格式化到稍後下載 FreeRTOS 程式碼的`vendors/vendor/boards/board/aws_demos/config_files/ota_demo_config.h`檔案中。

將存取權限授予適用於 AWS IoT 的程式碼簽署

若要提供存取權，請新增許可到您的使用者、群組或角色：

- AWS IAM Identity Center 中的使用者和群組：

  建立許可集合。請遵循《AWS IAM Identity Center 使用者指南》的建立許可集合中的指示。
- 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。請遵循《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。請遵循《IAM 使用者指南》的為 IAM 使用者建立角色中的指示。
  - (不建議) 將政策直接連接至使用者，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 圖書館下載免費服務

您可以從中克隆或下載 FreeRTOS。GitHub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Over-the-air 更新演示應用。

⚠ 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。
- 由於原始程式碼可能包含符號連結，因此如果您使用 Windows 來擷取歸檔，您可能必須：
  - 啟用開發人員模式或，
  - 使用以系統管理員身分提高權限的主控台。

  如此一來，Windows 可以在擷取歸檔時正確建立符號連結。否則，符號鏈接將被寫入為普通文件，其中包含符號鏈接的路徑作為文本或為空。如需詳細資訊，請參閱 Windows 10 中的部落格條目符號連結！ 。

  如果您在 Windows 下使用 Git，您必須啟用開發人員模式，或者您必須：
  - 使用下列命令設定`core.symlinks`為 true：

    ```
    git config --global core.symlinks true
    ```
  - 每當您使用寫入系統的 git 命令時，請使用以管理員身份提升的主控台 (例如git pullgit clone、、和git submodule update --init --recursive)。

使用 MQTT 進行 OTA 更新的先決條件

本節介紹使用 MQTT 執行over-the-air（OTA 更新）的一般要求。

最低需求

- 裝置韌體必須包含必要的 FreeRTOS 程式庫 (CoremQtt 代理程式、OTA 更新及其相依性)。
- 需要 1.4.0 或更新版本。不過，我們建議您儘可能使用最新版本。

組態

自 2019 年 12 月 00 日起，自由軟件 OTA 可以使用 HTTP 或 MQTT 協議將固件更新映像從設備傳輸到設備。AWS IoT如果您在 FreeRTOS 中建立 OTA 更新時同時指定這兩種通訊協定，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。

默認情況下，中的 OTA 協議的配置`ota_config.h`是使用 MQTT 協議。

裝置特定的組態

無。

記憶體用量

當 MQTT 用於資料傳輸時，MQTT 連線不需要額外的記憶體，因為它是在控制操作與資料操作之間共用。

裝置政策

每部使用 MQTT 接收 OTA 更新的裝置都必須註冊為 AWS IoT 中的物件，且該物件必須有連接的政策，如此處列出的政策所示。如需 `"Action"` 和 `"Resource"` 物件中的項目的詳細資訊，請參閱 AWS IoT 核心政策動作和 AWS IoT 核心動作資源。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:Connect",
            "Resource": "arn:partition:iot:region:account:client/
${iot:Connection.Thing.ThingName}"
        },
```

```
        {
            "Effect": "Allow",
            "Action": "iot:Subscribe",
            "Resource": [
                "arn:partition:iot:region:account:topicfilter/$aws/things/
${iot:Connection.Thing.ThingName}/streams/*",
                "arn:partition:iot:region:account:topicfilter/$aws/things/
${iot:Connection.Thing.ThingName}/jobs/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish",
                "iot:Receive"
            ],
            "Resource": [
                "arn:partition:iot:region:account:topic/$aws/things/
${iot:Connection.Thing.ThingName}/streams/*",
                "arn:partition:iot:region:account:topic/$aws/things/
${iot:Connection.Thing.ThingName}/jobs/*"
            ]
        }
    ]
}
```

備註

- `iot:Connect` 許可允許您的裝置透過 MQTT 連接至 AWS IoT。
- AWS IoT 工作 (`.../jobs/*`) 主題的 `iot:Subscribe` 和 `iot:Publish` 許可允許連接的裝置接收工作通知和工作文件，並發佈工作執行的完成狀態。
- AWS IoT OTA 串流 (`.../streams/*`) 主題的 `iot:Subscribe` 和 `iot:Publish` 許可允許連接的裝置從 AWS IoT 中擷取 OTA 更新資料。需要這些許可才能透過 MQTT 執行韌體更新。
- `iot:Receive` 許可允許 AWS IoT Core 將這些主題上的訊息發佈到連接的裝置。每次交付 MQTT 訊息時，都會檢查此許可。您可以使用此許可來撤銷目前訂閱主題之用戶端的存取權。

使用 HTTP 進行 OTA 更新的先決條件

本節說明使用 HTTP 執行 over-the-air (OTA) 更新的一般需求。自 2019 年 12 月 00 日起，自由軟件 OTA 可以使用 HTTP 或 MQTT 協議將固件更新映像從設備傳輸到設備。AWS IoT

Note

- 雖然可能會使用 HTTP 通訊協定來傳輸韌體影像，但 CoremQtt Agent 程式庫仍然需要使用 CoremQtt Agent 程式庫，因為與其他互動會AWS IoT Core使用 CoremQtt Agent 程式庫，包括傳送或接收工作執行通知、工作文件以及執行狀態更新。
- 當您為 OTA 更新任務同時指定 MQTT 和 HTTP 通訊協定時，每個個別裝置上 OTA 代理程式軟體的設定都會決定用來傳輸韌體映像的通訊協定。若要將 OTA 代理程式從預設 MQTT 通訊協定方法變更為 HTTP 通訊協定，您可以修改用來編譯裝置的 FreeRTOS 原始程式碼的標頭檔案。

最低需求

- 裝置韌體必須包含必要的 FreeRTOS 程式庫 (CoremQtt 代理程式、HTTP、OTA 代理程式及其相依性)。
- 需要 FreeRTOS 版本 2019 年 12 月 00 日或更新版本才能更改 OTA 協議的配置以啟用通過 HTTP 進行 OTA 數據傳輸。

組態

請參閱 \vendors\boards*board*\aws_demos\config_files\ota_config.h 檔案中 OTA 通訊協定的下列組態。

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使用量資料，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 工作 (`.../jobs/*`) 主題的 `iot:Subscribe` 和 `iot:Publish` 許可允許連接的裝置接收工作通知和工作文件，並發佈工作執行的完成狀態。
- `iot:Receive` 許可允許 AWS IoT Core 將這些主題上的訊息發佈到目前連接的裝置。每次交付 MQTT 訊息時，都會檢查此許可。您可以使用此許可來撤銷目前訂閱主題之用戶端的存取權。

OTA 教學

本節包含使用 OTA 更新在運行 FreeRTOS 的設備上更新固件的教程。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。當您要自動執行 OTA 更新工作、處理大量裝置或正在使用未符合

FreeRTOS 資格的裝置時，此功能非常有用。AWS CLI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，然後將其上傳到 Amazon S3 儲存貯體。如果您使用程式碼簽章AWS IoT，請將未簽署的韌體映像上傳到 Amazon S3 儲存貯體。
5. 建立 OTA 更新。

當您建立 OTA 更新時，請指定映像傳遞通訊協定 (MQTT 或 HTTP)，或指定兩者以允許裝置選擇。裝置上的 FreeRTOS OTA 代理程式會接收更新的韌體映像，並驗證新映像的數位簽章、總和檢查碼和版本號碼。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。如果您的設備沒有運行 FreeRTOS，則必須實現在設備上運行的 OTA 代理程序。

安裝初始韌體

若要更新韌體，您必須安裝使用 OTA 代理程式程式庫的初始版本韌體，接聽 OTA 更新任務。如果您沒有執行 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 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**
>
> 此參考集成託管在亞馬遜的 Freertos 存儲庫上，該存儲庫已被棄用。我們建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱. 亞馬遜自由 Github 存儲庫遷移指南

這些步驟的撰寫是假設您已完成建置 aws_demos 專案，如在德州儀器 CC3220SF-LAUNCHXL 上下載、建置、快閃記憶體和執行 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 New Project (新專案)。
5. 在 Start new project (開始新專案) 頁面上，輸入您專案的名稱。針對 Device Type (裝置類型)，選擇 CC3220SF。針對 Device Mode (裝置模式)，選擇 Develop (開發)。選擇 Create Project (建立專案)。
6. 中斷您終端機模擬器的連線。
7. 在 Uniflash 應用程式視窗的右側，選擇 Connect (連線)。
8. 在 Advanced (進階) 下的 Files (檔案)，選擇 User Files (使用者檔案)。
9. 在 File (檔案) 選擇器窗格中，選擇 Add File (新增檔案) 圖示  。
10. 瀏覽至 /Applications/Ti/simplelink_cc32xx_sdk_*version*/tools/cc32xx_tools/certificate-playground 目錄，選取 dummy-root-ca-cert，選擇 Open (開啟)，然後選擇 Write (寫入)。
11. 在 File (檔案) 選擇器窗格中，選擇 Add File (新增檔案) 圖示  。
12. 瀏覽至您建立程式碼簽署憑證及私有金鑰的工作目錄，選擇 tisigner.crt.der，選擇 Open (開啟)，然後選擇 Write (寫入)。
13. 從 Action (動作) 下拉式清單中，選擇 Select MCU Image (選取 MCU 映像)，然後選擇 Browse (瀏覽) 來選擇要用於寫入您裝置的韌體映像 (aws_demos.bin)。這個檔案位於 *freertos*/vendors/ti/boards/cc3220_launchpad/aws_demos/ccs/Debug 目錄中。選擇 Open (開啟)。
    a. 在檔案對話方塊中，確認檔案名稱已設為 mcuflashimg.bin。
    b. 選取 Vendor (廠商) 核取方塊。
    c. 在 File Token (檔案字符) 下方，輸入 **1952007250**。
    d. 在 Private Key File Name (私有金鑰檔案名稱) 下方，選擇 Browse (瀏覽)，然後從您建立程式碼簽署憑證及私有金鑰的工作目錄中選擇 tisigner.key。

e. 在 Certification File Name (認證檔案名稱) 下方，選擇 `tisigner.crt.der`。

f. 選擇 Write (寫入)。

14. 在左側窗格中，於 Files (檔案) 下方，選擇 Service Pack (服務套件)。
15. 在 Service Pack File Name (服務套件檔案名稱) 下方，選擇 Browse (瀏覽)，瀏覽至 `simplelink_cc32x_sdk_`*`version`*`/tools/cc32xx_tools/servicepack-cc3x20`，選擇 `sp_3.7.0.1_2.0.0.0_2.2.0.6.bin`，然後選擇 Open (開啟)。
16. 在左側窗格中，於 Files (檔案) 下方，選擇 Trusted Root-Certificate Catalog (受信任根憑證目錄)。
17. 清除 Use default Trusted Root-Certificate Catalog (使用預設受信任根憑證目錄) 核取方塊。
18. ***######################## _cc32xx_sdk_ ##/## /cc32xx_ ##/#####/ 20160911.lst###########certcatalogPlayGround***
19. ***#########################/###/####/############ #####certcatalogPlayGround***
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 Espressif ESP32 上安裝初始版本韌體

> **Important**
>
> 此參考集成託管在亞馬遜的 Freertos 存儲庫上，該存儲庫已被棄用。我們建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱. 亞馬遜自由 Github 存儲庫遷移指南

本指南的撰寫方式是假設您已經執行了 [開始使用濃縮咖啡 ESP32-DevKit C] 和 [ESP-WROVER-KIT] 和 [無線更新先決條件] 中的步驟。在嘗試 OTA 更新之前，您可能需要執行 FreeRTOS 入門中所述的 MQTT 示範專案，以確保您的主機板和工具鏈已正確設定。

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、註解掉#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED，以及定義CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED或CONFIG_OTA_HTTP_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 vendors/*vendor*/boards/*board*/aws_demos/config_files/ota_demo_config.h。它應以下列方式進行格式化。

```
#define otapalconfigCODE_SIGNING_CERTIFICATE  \
"-----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**
>
> 此參考集成託管在亞馬遜的 Freertos 存儲庫上，該存儲庫已被棄用。我們建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱. 亞馬遜自由 Github 存儲庫遷移指南

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。在嘗試 OTA 更新之前，您可能需要執行 FreeRTOS 入門中所述的 MQTT 示範專案，以確保您的主機板和工具鏈已正確設定。

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. 將 `#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED` 取代為 `CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` 或 `CONFIG_OTA_HTTP_UPDATE_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

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

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

請務必在 FreeRTOS 專案或您的裝置上包含您的程式碼簽署憑證、私密金鑰和憑證信任鏈。

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

FreeRTOS 隨附的 OTA 代理程式會檢查任何更新的版本，並僅在比現有韌體版本更新時進行安裝。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. 找到檔案 /vendors/*vendor*/boards/*board*/aws_demos/config_files/ota_demo_config.h並遞增應用程式版本的值。
3. 若要排程更新至檔案類型非 0 (非韌體檔案) 的瑞薩 rx65n 平台，您必須使用「瑞薩安全快閃記憶體程式設計師」工具簽署檔案，否則裝置上的簽章檢查將會失敗。此工具會建立具有副檔名的已簽署檔案套件.rsu，副檔名為瑞薩專屬的檔案類型。該工具可以在 Github 上找到。您可以使用下列範例指令來產生影像：

```
"Renesas Secure Flash Programmer.exe" CUI Update "RX65N(ROM 2MB)/Secure
 Bootloader=256KB" "sig-sha256-ecdsa" 1 "file_name" "output_file_name.rsu"
```

4. 重新建置專案。

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 儲存貯體來存放您的更新。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`
- Espressif ESP32：`vendors/espressif/boards/esp32/aws_demos/make/build/aws_demos.bin`

建立 OTA 更新 (AWS IoT 主控台)

1. 在AWS IoT主控台的導覽窗格中，於 [管理] 下選取 [遠端動作]，然後選擇 [工作]。
2. 選擇 Create job (建立任務)。
3. 在 [工作類型] 下選取 [建立 FreeRTOS OTA 更新工作]，然後選擇 [下一步
4. 在工作屬性中，輸入工作名稱並 (選擇性) 輸入工作的說明，然後選擇下一步。
5. 您可以將 OTA 更新部署到單一裝置或一組裝置。在「要更新的裝置」下，從下拉式清單中選擇一或多個物件或物件群組。
6. 在 [選取檔案傳輸的通訊協定] 底下，選取 [HTTP] 或 [MQTT]，或選取兩者以允許每個裝置決定要使用的通訊協定。
7. 在「簽署」下方，選擇您的檔案，選取「為我簽署新檔案」。
8. 在 [程式碼簽章設定檔] 下，選擇 [建立新設定
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 資格的硬體平台才會顯示在此清單中。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。如果您使用的是不符資格的平台，而且您使用 ECDSA P-256 SHA-256 以外的密碼套件來進行簽署，您可以使用適用於 AWS IoT 的程式碼簽署，也可以自行簽署韌體更新。如需詳細資訊，請參閱數位簽署您的韌體更新。

b. 在 [程式碼簽署憑證] 底下，選擇 [選取現有憑證]，然後選取先前匯入的憑證，或選擇 [匯入新的程式碼簽署憑證]，選擇您的檔案並選取 [匯入] 以匯入新憑證。

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。對於大多數設備，您可以將此字段留空。對於 Windows 模擬器以及確實將憑證放在特定檔案位置的裝置，請在此處輸入路徑名稱。

> **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。

d. 選取 Create (建立)。

10. 在 [檔案] 下選取 [選取現有檔案]，然後選擇 [瀏覽 S3 隨即顯示您的亞馬遜 S3 儲存貯體清單。選擇包含您韌體更新的儲存貯體，然後在儲存貯體中選擇您的韌體更新。

> **Note**
>
> Microchip Curiosity PIC32MZEF 示範專案會以預設名稱 `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. 在設備上文件的路徑名下，輸入設備上 OTA 作業將復制固件映像的位置的完整路徑名稱。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. 開啟「檔案類型」並輸入 0-255 範圍內的整數值。您輸入的檔案類型將新增至傳送至 MCU 的工作文件中。MCU 韌體/軟體開發人員擁有如何處理此值的完整擁有權。可能的情況包括具有次要處理器的 MCU，其韌體可獨立於主要處理器進行更新。當設備收到 OTA 更新作業時，它可以使用文件類型來識別更新的處理器。

13. 在「IAM 角色」下，根據中的指示選擇角色建立 OTA 更新服務角色。

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。對於大多數設備，您可以將此字段留空。對於 Windows 模擬器以及確實將憑證放在特定檔案位置的裝置，請在此處輸入路徑名稱。
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 Amazon S3 中選取您的韌體映像下，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，且狀態為 IN PROGRESS (進行中)。

> **Note**
>
> - AWS IoT 主控台不會自動更新任務狀態。請重新整理您的瀏覽器，以查看更新。

將您的序列 UART 終端機連線到您的裝置。您應該會看到輸出，指出裝置正在下載更新韌體。

在裝置下載完更新韌體後，它便會重新啟動並安裝韌體。您可以在 UART 終端機中查看實際發生情況。

如需示範如何使用主控台建立 OTA 更新的教學課程，請參閱Over-the-air 更新演示應用。

使用 AWS CLI 建立 OTA 更新

當您使用 AWS CLI 建立 OTA 更新時，您必須執行以下操作：

1. 數位簽署您的韌體映像。
2. 建立您數位簽署韌體映像的串流。

3. 啟動 OTA 更新任務。

數位簽署您的韌體更新

當您使用 AWS CLI 執行 OTA 更新時，您可以使用適用於 AWS IoT 的程式碼簽署，或是可以自行簽署您的韌體更新。如需程式碼簽章所支援的加密簽章與雜湊演算法清單AWS IoT，請參閱。SigningConfigurationOverrides如果您想要使用程式碼簽章不支援的加密演算法AWS IoT，則必須先簽署韌體二進位檔，然後再將其上傳到 Amazon S3。

使用適用於 AWS IoT 的程式碼簽署簽署您的韌體映像

若要使用程式碼簽章來簽署韌體映像AWS IoT，您可以使用其中一個 AWSSDK 或命令列工具。如需有關程式碼簽章的詳細資訊AWS IoT，請參閱AWS IoT.

安裝並設定程式碼簽署工具後，請將未簽署的韌體映像複製到 Amazon S3 儲存貯體，然後使用下列命令啟動程式碼簽署任務。AWS CLIput-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*並且*your-destination-bucket-name*可以是相同的亞馬遜 S3 存儲桶。

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 Amazon S3 主控台、選擇儲存貯體，然後在頁面頂端的「版本」旁邊選擇「顯示」來找到它。

destination

將 S3 儲存貯體中已簽署韌體複製到裝置上的目的地。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。

簽署任務會啟動並將已簽署的韌體映像寫入目的地 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 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。如需適用於 AWS IoT 程式碼簽署的詳細資訊，請參閱適用於 AWS IoT 的程式碼簽署。

手動簽署您的韌體映像

對韌體映像進行數位簽名，並將已簽署的韌體映像上傳到 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 更新管理員服務可讓您使用存放在 Amazon S3 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，您也可以建立一個包含已簽署韌體更新的串流。建立可識別已簽署韌體映像的 JSON 檔案 (`stream.json`)。JSON 檔案應包含以下內容。

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

這些是 JSON 檔案中的屬性：

`fileId`

介於 0—255 之間的任意整數，可識別您的韌體影像。

`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檔的檔案名稱。您可以查看儲存貯體的內容，在 Amazon S3 主控台中找到此值。

若您使用適用於 AWS IoT 的程式碼簽署，檔案名稱為適用於 AWS IoT 程式碼簽署所產生的 GUID。

使用create-streamAWS CLI指令建立串流。

```
aws iot create-stream \
    --stream-id your_stream_id \
    --description your_description \
    --files file://stream.json \
    --role-arn your_role_arn
```

這些是命create-streamAWS CLI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)。

當您使用適用於 AWS IoT 的程式碼簽署時，此鍵為 GUID。

以下是範例 `stream.json` 檔案。

```
[
    {
        "fileId":123,
```

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

role-arn

OTA 服務角色，也授予存取韌體映像所在之 Amazon S3 儲存貯體的存取權。

若要尋找已簽署韌體映像的 Amazon S3 物件金鑰，請使用aws signer describe-signing-job --job-id ***my-job-id***指令，其中my-job-id是指create-signing-jobAWS CLI令顯示的工作 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

使用指create-ota-updateAWS CLI令建立 OTA 更新工作。

> **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	類型	描述
otaUpdateId	字串 (上限：128；下限：1)	欲建立的 OTA 更新 ID。
description	字串 (上限：2028)	OTA 更新的描述。

名稱	類型	描述
targets	list	將目標鎖定於接收 OTA 更新的裝置。
protocols	list	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	字串	指定更新是否要持續運作 (CONTINUOUS)，或在指定為目標的所有物件完成更新後視為完成 (SNAPSHOT)。如果是 CONTINUOUS，則在目標中偵測到變更時，更新也可能會運作於物件上。例如，當物件新增至目標群組，更新就會在物件上運作，即使該更新已被原本就在群組中的所有物件完成。有效值：CONTINUOUS | SNAPSHOT。 列舉：CONTINUOUS | SNAPSHOT
awsJobExecutionsRolloutConfig		OTA 更新的發行組態。
maximumPerMinute	integer (上限：1000；下限：1)	每分鐘可啟動的 OTA 更新任務執行數上限。
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。

名稱	類型	描述
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 最多支援小數點後一位數 (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。
awsJobPresignedUrlConfig		預先簽章的 URL 之組態資訊。
expiresInSec	長整數	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		決定工作停止發生的時間和方式的準則。
abortCriteriaList	list	決定何時以及如何停止工作的準則清單。

名稱	類型	描述
failureType	字串	可起始工作停止的工作執行失敗類型。 enum: FAILED | REJECTED | TIMED_OUT | ALL
action	字串	啟動工作停止時所要採取的工作動作類型。 enum: CANCEL
minNumberOfExecutedThings	integer (下限：1)	工作停止之前必須接收工作執行通知的最小數目。
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 。
inProgressTimeoutInMinutes	長整數	指定此裝置必須完成這項任務執行的時間 (以分鐘為單位)。逾時間隔可介於 1 分鐘到 7 天之間 (1 到 10080 分鐘)。進行中的計時器無法更新，並會套用到任務的所有任務執行。每當任務執行維持在 IN_PROGRESS 狀態超過此間隔時，任務執行就會失敗，並切換到結束 TIMED_OUT 狀態。
files	list	OTA 更新所串流的檔案。

名稱	類型	描述
fileName	字串	檔案名稱。
fileType	integer 範圍 - 上限：255；下限：0	您可以在工作文件中包含的整數值，讓您的裝置識別從雲端接收的檔案類型。
fileVersion	字串	檔案版本。
fileLocation		已更新韌體的位置。
stream		包含 OTA 更新的串流。
streamId	字串 (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。
bucket	字串 (下限：1)	S3 儲存貯體。
key	字串 (下限：1)	S3 金鑰。
version	字串	S3 儲存貯體版本。
codeSigning		檔案的代碼簽署方式。
awsSignerJobId	字串	為了簽署檔案AWSSignerJob而建立的 ID。
startSigningJobParameter		描述程式碼簽署任務。

名稱	類型	描述
signingProfilePara meter		描述程式碼簽署描述檔。
certificateArn	字串	憑證 ARN。
platform	字串	您裝置的硬體平台。
certificatePathOnD evice	字串	程式碼簽署憑證在您裝置上的位置。
signingProfileName	字串	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	字串 (下限：1)	內含已更新韌體的 S3 儲存貯體。
prefix	字串	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	字串	憑證名稱。
inlineDocument	字串	以 base64 編碼呈現的二進位代碼簽署憑證鏈。
hashAlgorithm	字串	用於以代碼簽署檔案的雜湊演算法。

名稱	類型	描述
signatureAlgorithm	字串	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	字串 (上限：2048；下限：20)	IAM 角色，可授AWS IoT予對 Amazon S3、任AWS IoT務和 AWS程式碼簽章資源的存取權，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	list	用於管理更新的中繼資料。
Key	字串 (上限：128；下限：1)	標籤的金鑰。
Value	字串 (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	類型	描述
otaUpdateId	字串	OTA 更新 ID。

名稱	類型	描述
	(上限：128；下限：1)	
awsIotJobId	字串	與 OTA 更新相關聯的AWS IoT 作業 ID。
otaUpdateArn	字串	OTA 更新 ARN。
awsIotJobArn	字串	與 OTA 更新相關聯的AWS IoT 作業 ARN。
otaUpdateStatus	字串	OTA 更新狀態。 列舉：CREATE_PENDING | CREATE_IN_PROGRESS | CREATE_COMPLETE | CREATE_FAILED

以下是 JSON 檔案的範例。該檔案會傳遞到使用適用於 AWS IoT 程式碼簽署的 create-ota-update 命令。

```
[
  {
    "fileName": "firmware.bin",
    "fileType": 1,
    "fileLocation": {
      "stream": {
        "streamId": "004",
        "fileId":123
      }
    },
    "codeSigning": {
      "awsSignerJobId": "48c67f3c-63bb-4f92-a98a-4ee0fbc2bef6"
    }
  }
]
```

以下是傳遞至create-ota-updateAWS CLI命令的 JSON 檔案範例，該檔案使用內嵌檔案提供自訂程式碼簽署材料。

```
[
  {
    "fileName": "firmware.bin",
    "fileType": 1,
    "fileLocation": {
      "stream": {
        "streamId": "004",
        "fileId": 123
      }
    },
    "codeSigning": {
      "customCodeSigning":{
        "signature":{
          "inlineDocument":"your_signature"
        },
        "certificateChain": {
          "certificateName": "your_certificate_name",
          "inlineDocument":"your_certificate_chain"
        },
        "hashAlgorithm":"your_hash_algorithm",
        "signatureAlgorithm":"your_signature_algorithm"
      }
    }
  }
]
```

下面是傳遞到命令的 JSON 文件的示例，該create-ota-updateAWS CLI命令允許 FreeRTOS OTA 啟動代碼簽名作業並創建代碼簽名配置文件和流。

```
[
  {
    "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"
          }
        }
      }
    }
  }
]
```

以下是傳遞到命令的 JSON 文件的示例，該create-ota-updateAWS CLI命令創建 OTA 更新，該更新使用現有配置文件啟動代碼簽名作業並使用指定的流。

```
[
  {
    "fileName": "your_firmware_path_on_device",
    "fileType": 1,
    "fileVersion": "1",
    "fileLocation": {
      "s3Location": {
        "bucket": "your_s3_bucket_name",
        "key": "your_object_key",
        "version": "your_S3_object_version"
      }
    },
    "codeSigning":{
      "startSigningJobParameter":{
        "signingProfileName": "your_unique_profile_name",
        "destination": {
          "s3Destination": {
            "bucket": "your_destination_bucket"
          }
        }
      }
    }
  }
]
```

以下是傳遞至create-ota-updateAWS CLI命令的 JSON 檔案範例，該檔案允許 FreeRTOS OTA 建立具有現有程式碼簽署工作 ID 的串流。

```
[
  {
    "fileName": "your_firmware_path_on_device",
    "fileType": 1,
    "fileVersion": "1",
    "codeSigning":{
      "awsSignerJobId": "your_signer_job_id"
    }
  }
]
```

以下是傳遞到創建 OTA 更新create-ota-updateAWS CLI命令的 JSON 文件的示例。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

您可以使用該list-ota-updatesAWS CLI命令獲取所有 OTA 更新的列表。

```
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

您可以使用該get-ota-updateAWS CLI命令獲取 OTA 更新的創建或刪除狀態。

```
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，請參閱描述工作執行。

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 更新期間建立的 AWS IoT 任務。

刪除 OTA 串流

建立一個使用 MQTT 的 OTA 更新時，您可以使用命令列或 AWS IoT 主控台建立串流，將韌體分成區塊，以便透過 MQTT 傳送。您可以使用delete-streamAWS 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。
- 執行更新的 AWS IoT 任務。
- 每個更新裝置的 AWS IoT 任務執行。

delete-ota-update 命令只會刪除 OTA 更新任務資料庫中的項目。您必須使用 delete-job 命令來刪除 AWS IoT 任務。

使用 delete-ota-update 命令刪除 OTA 更新。

```
aws iot delete-ota-update --ota-update-id your_ota_update_id
```

ota-update-id

欲刪除的 OTA 更新 ID。

delete-stream

刪除與 OTA 更新相關聯的串流。

force-delete-aws-job

刪除與 OTA 更新相關聯的 AWS IoT 任務。若未設定此標記，並且任務正處於 `In_Progress` 狀態，便不會刪除任務。

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工作。也會為每個裝置建立任務執行，處理任務。您可以使用指delete-jobAWS CLI令刪除工作及其相關聯的工作執行項目。

```
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
```

和delete-jobAWS CLI命令一樣，您可以將`--force`參數傳遞給，以強制刪除工作執行。delete-job-execution如需詳細資訊，請參閱 DeleteJobExecutionAPI。

> **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Over-the-air 更新演示應用。

OTA Update Manager 服務

over-the-air（OTA）更新管理器服務提供了一種方法：

- 建立 OTA 更新及其使用的資源，包括 AWS IoT 任務、AWS IoT 串流，以及程式碼簽署。
- 取得 OTA 更新的相關資訊。
- 列出與您AWS帳戶關聯的所有 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。如需您所需更新類型的討論，請參閱AWS IoT 開發人員指南的「工作」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。
- IAM 角色，可授AWS IoT予對 Amazon S3、任AWS IoT務和AWS程式碼簽章資源的存取權，以建立 OTA 更新任務。
- 選擇性的使用者定義名稱/值對清單。

OTA 更新的設計旨在更新裝置韌體，但您可以使用它們來將任何您想傳送的檔案傳送到向 AWS IoT 註冊的一或多個裝置。當您無線傳送韌體更新時，我們建議您以數位方式簽署它們，以便接收它們的裝置可以確認它們在途中未遭到竄改。

您可以使用 HTTP 或 MQTT 通訊協定來傳送更新的韌體映像，取決於您選擇的設定。您可以使用 FreeRTOS 的程式碼簽章來簽署韌體更新，也可以使用自己的程式碼簽署工具。

若要進一步控制流程，您可以在透過 MQTT 傳送更新時，使用 CreateStreamAPI 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之區塊的大小。

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，並偵聽更新訊息。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。如果您不使用 FreeRTOS，則必須實現自己的 OTA 代理程式來監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

over-the-air（OTA）代理程序旨在簡化您為產品添加 OTA 更新功能所必須編寫的代碼量。該集成負擔主要包括 OTA 代理的初始化以及創建用於響應 OTA Agent 事件消息的自定義回調函數。在初始化操作系統期間，MQTT，HTTP（如果 HTTP 用於文件下載）和平台特定實現（PAL）接口傳遞給 OTA 代理。緩衝區也可以初始化並傳遞給 OTA 代理。

> **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。若要熟悉如何使用項目、認證、程式碼簽署憑證、佈建裝置和 OTA 更新工作來設定AWS帳戶，請參閱 FreeRTOS 先決條AWS IoT件。

連線管理

OTA 代理程式會將 MQTT 通訊協定用於所有涉及 AWS IoT 服務的控制通訊操作，但它不會管理 MQTT 連線。為了確保 OTA 代理程式不會影響您應用程式的連線管理政策，MQTT 連線 (包含中斷連線及任何重新連線功能) 必須由主要使用者應用程式處理。檔案可透過 MQTT 或 HTTP 通訊協定下載。您可以選擇建立 OTA 任務時的通訊協定。如果您選擇 MQTT，OTA 代理程式會使用相同的連線來進行控制作業和下載檔案。

簡單的 OTA 演示

以下是簡易 OTA 示範的摘要，向您展示代理程式連線到 MQTT 中介裝置及初始化 OTA 代理程式的方式。在此示例中，我們將演示配置為使用默認的 OTA 應用程序回調，並每秒返回一些統計信息。為求簡化，我們會從此示範刪掉一下詳細資訊。

OTA 演示還通過監視斷開回調並重新建立連接來演示 MQTT 連接管理。發生中斷連線時，示範會先暫停 OTA 代理程式作業，然後嘗試重新建立 MQTT 連線。MQTT 重新連接嘗試會延遲一段時間，該時間將指數增加到最大值，並且還添加了抖動。如果重新建立連接，則 OTA 代理程序將繼續其操作。

如需使用 AWS IoT MQTT 中介裝置的運作範例，請參閱 `demos/ota` 目錄中的 OTA 示範程式碼。

因為 OTA 代理程式位於自己的任務中，此範例中刻意造成的一秒鐘延遲只會影響此應用程式。它不會影響代理程式的效能。

```
static BaseType_t prvRunOTADemo( void )
{
    /* Status indicating a successful demo or not. */
    BaseType_t xStatus = pdFAIL;

    /* OTA library return status. */
    OtaErr_t xOtaError = OtaErrUninitialized;

    /* OTA event message used for sending event to OTA Agent.*/
    OtaEventMsg_t xEventMsg = { 0 };
```

```
    /* OTA interface context required for library interface functions.*/
    OtaInterfaces_t xOtaInterfaces;

    /* OTA library packet statistics per job.*/
    OtaAgentStatistics_t xOtaStatistics = { 0 };

    /* OTA Agent state returned from calling OTA_GetState.*/
    OtaState_t xOtaState = OtaAgentStateStopped;

    /* Set OTA Library interfaces.*/
    prvSetOtaInterfaces( &xOtaInterfaces );

    /************************** Init OTA Library. **************************/

    if( ( xOtaError = OTA_Init( &xOtaBuffer,
                                &xOtaInterfaces,
                                ( const uint8_t * ) ( democonfigCLIENT_IDENTIFIER ),
                                prvOtaAppCallback ) ) != OtaErrNone )
    {
        LogError( ( "Failed to initialize OTA Agent, exiting = %u.",
                    xOtaError ) );
    }
    else
    {
        xStatus = pdPASS;
    }

    /************************ Create OTA Agent Task. ************************/

    if( xStatus == pdPASS )
    {
        xStatus = xTaskCreate( prvOTAAgentTask,
                               "OTA Agent Task",
                               otaexampleAGENT_TASK_STACK_SIZE,
                               NULL,
                               otaexampleAGENT_TASK_PRIORITY,
                               NULL );

        if( xStatus != pdPASS )
        {
            LogError( ( "Failed to create OTA agent task:" ) );
        }
    }
```

```
    /****************************** Start OTA ******************************/

    if( xStatus == pdPASS )
    {
        /* Send start event to OTA Agent.*/
        xEventMsg.eventId = OtaAgentEventStart;
        OTA_SignalEvent( &xEventMsg );
    }

    /******************** Loop and display OTA statistics ********************/

    if( xStatus == pdPASS )
    {
        while( ( xOtaState = OTA_GetState() ) != OtaAgentStateStopped )
        {
            /* Get OTA statistics for currently executing job. */
            if( xOtaState != OtaAgentStateSuspended )
            {
                OTA_GetStatistics( &xOtaStatistics );

                LogInfo( ( " Received: %u   Queued: %u   Processed: %u   Dropped: %u",
                           xOtaStatistics.otaPacketsReceived,
                           xOtaStatistics.otaPacketsQueued,
                           xOtaStatistics.otaPacketsProcessed,
                           xOtaStatistics.otaPacketsDropped ) );
            }

            vTaskDelay( pdMS_TO_TICKS( otaexampleEXAMPLE_TASK_DELAY_MS ) );
        }
    }

    return xStatus;
}
```

以下是此示範應用程式的高層級流程：

- 建立 MQTT 代理程式內容。
- 連線到您的 AWS IoT 端點。
- 初始化 OTA 代理程式。
- 允許 OTA 更新作業並每秒輸出統計信息的循環。
- 如果 MQTT 中斷連線，請暫停 OTA 代理程式作業。

- 嘗試使用指數延遲和抖動再次連接。
- 如果重新連線，請繼續 OTA 代理程式作業。
- 如果代理程式停止，請延遲一秒鐘，然後嘗試重新連線。

對 OTA 代理事件使用應用程序回調

前面的示例用prvOtaAppCallback作 OTA 代理程序事件的回調處理程序。（請參閱 OTA_Init API 調用的第四個參數）。如果要實現完成事件的自定義處理，則必須更改 OTA 演示/應用程序中的默認處理。在 OTA 過程中，OTA 代理可以將以下事件枚舉之一發送到回調處理程序。如何及何時處理這些事件，則由應用程式開發人員決定。

```
/**
 * @ingroup ota_enum_types
 * @brief OTA Job callback events.
 *
 * After an OTA update image is received and authenticated, the agent calls the user
 * callback (set with the @ref OTA_Init API) with the value OtaJobEventActivate to
 * signal that the device must be rebooted to activate the new image. When the device
 * boots, if the OTA job status is in self test mode, the agent calls the user callback
 * with the value OtaJobEventStartTest, signaling that any additional self tests
 * should be performed.
 *
 * If the OTA receive fails for any reason, the agent calls the user callback with
 * the value OtaJobEventFail instead to allow the user to log the failure and take
 * any action deemed appropriate by the user code.
 *
 * See the OtaImageState_t type for more information.
 */
typedef enum OtaJobEvent
{
    OtaJobEventActivate = 0,       /*!< @brief OTA receive is authenticated and ready
 to activate. */
    OtaJobEventFail = 1,           /*!< @brief OTA receive failed. Unable to use this
 update. */
    OtaJobEventStartTest = 2,      /*!< @brief OTA job is now in self test, perform
 user tests. */
    OtaJobEventProcessed = 3,      /*!< @brief OTA event queued by OTA_SignalEvent is
 processed. */
    OtaJobEventSelfTestFailed = 4, /*!< @brief OTA self-test failed for current job. */
    OtaJobEventParseCustomJob = 5, /*!< @brief OTA event for parsing custom job
 document. */
```

```
    OtaJobEventReceivedJob = 6,    /*!< @brief OTA event when a new valid AFT-OTA job
 is received. */
    OtaJobEventUpdateComplete = 7, /*!< @brief OTA event when the update is completed.
 */
    OtaLastJobEvent = OtaJobEventStartTest
} OtaJobEvent_t;
```

OTA 代理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：

OtaJobEventActivate

當回調處理程序收到此事件時，您可以立即重置設備，也可以安排呼叫以稍後重置設備。這可讓您延期裝置重設及自我測試階段 (若需要的話)。

OtaJobEventFail

當回呼處理常式收到此事件時，更新失敗。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。

OtaJobEventStartTest

自我測試階段的目的在於允許新更新的韌體在判斷其是否正常運作並認可為最新的永久應用程式映像之前執行並自行測試。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 OtaJobEventStartTest 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(OtaImageStateAccepted) 函數來將韌體遞交為新的永久映像。

OtaJobEventProcessed

處理排隊的 OTA 事件，因此可以完成諸如釋放 OTA 緩衝區之類的清理操作。OTA_SignalEvent

OtaJobEventSelfTestFailed

目前工作的 OTA 自我測試失敗。此事件的默認處理方法是關閉 OTA 代理並重新啟動它，以便設備回滾到上一個圖像。

OtaJobEventUpdateComplete

OTA 工作更新完成的通知事件。

OTA 安全性

以下是over-the-air（OTA）安全性的三個方面：

連線安全

OTA Update Manager 服務倚賴 AWS IoT 使用的現有安全機制，例如 Transport Layer Security (TLS) 交互身分驗證。OTA 更新流量會透過 AWS IoT 裝置閘道傳遞，並使用 AWS IoT 安全機制。每個透過裝置閘道傳入及傳出的 HTTP 或 MQTT 訊息都會經過身分驗證及授權。

OTA 更新的真確性及完整性

韌體可在 OTA 更新之前進行數位簽署，確保其來自可靠的來源並且並未遭到竄改。

FreeRTOS OTA 更新管理員服務使用程式碼簽章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 呼叫都會經過標準的 IAM 簽名版本 4 身份驗證和授權。若要建立部署，您必須擁有呼叫 `CreateDeployment`、`CreateJob` 及 `CreateStream` API 的許可。此外，在 Amazon S3 儲存貯體政策或 ACL 中，您必須授與AWS IoT服務主體的讀取許可，以便在串流期間存取存放在 Amazon S3 中的韌體更新。

適用於 AWS IoT 的程式碼簽署

AWS IoT 主控台會使用 適用於 AWS IoT 的程式碼簽署為任何 AWS IoT 支援的裝置自動簽署您的韌體映像。

程式碼簽章AWS IoT使用您匯入 ACM 的憑證和私密金鑰。您可以使用自我簽署的憑證進行測試，但建議您從知名的商業憑證授權單位 (CA) 取得憑證。

程式碼 — 簽署憑證使用 X.509 版本 3 `Key Usage` 和延伸功能。`Extended Key UsageKey Usage` 延伸模組設為 `Digital Signature`，`Extended Key Usage` 延伸模組設為 `Code Signing`。如需簽署程式碼映像的詳細資訊，請參閱 適用於 AWS IoT 的程式碼簽署開發人員指南 和 適用於 AWS IoT API w的程式碼簽署。

> **Note**
>
> 您可以從適用於 Amazon 網路服務的工具下載 AWS IoT SDK 的程式碼簽章。

OTA 故障診斷

以下章節包含有助於您故障診斷 OTA 更新問題的資訊。

主題

- 設置 OTA 更新的CloudWatch日誌
- 使用 AWS CloudTrail 記錄 AWS IoT OTA API 呼叫
- 取得建立輸出更新失敗詳細資料，使用 AWS CLI
- 使用 AWS CLI 取得 OTA 失敗代碼
- 故障診斷多個裝置的 OTA 更新
- 故障診斷 Texas Instruments CC3220SF Launchpad 的 OTA 更新

設置 OTA 更新的CloudWatch日誌

OTA 更新服務支持使用亞馬遜日誌記錄CloudWatch。您可以使用AWS IoT主控台啟用和設定用於 OTA 更新的 Amazon CloudWatch 日誌記錄。如需詳細資訊，請參閱 Cloudwatch Logs。

若要啟用記錄功能，您必須建立 IAM 角色並設定 OTA 更新記錄。

> **Note**
>
> 在啟用 OTA 更新日誌記錄之前，請確保您了解CloudWatch日誌訪問權限。具有CloudWatch記錄存取權的使用者可以看到您的偵錯資訊。如需詳細資訊，請參閱 Amazon CloudWatch 日誌的身份驗證和存取控制。

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「新建」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 Update (更新)。

OTA 更新日誌

OTA 更新服務會在發生下列其中一項情況時，將日誌發佈到您的帳戶：

- 已建立 OTA 更新。
- 已完成 OTA 更新。
- 已建立程式碼簽署任務。
- 已完成程式碼簽署任務。
- 已建立 AWS IoT 任務。
- 已完成 AWS IoT 任務。
- 已建立串流。

您可以在 CloudWatch 主控台中檢視您的日誌。

在CloudWatch日誌中查看 OTA 更新

1. 從導覽窗格中，選擇 Logs (日誌)。
2. 在記錄群組中，選擇AWSIoTLogsV2。

OTA 更新日誌可包含以下屬性：

accountId

產生記錄檔時所使用的帳AWS號 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`：串流的要求已傳送至 AWS IoT MQTT 型檔案傳送功能。

- `DescribeStream`：串流相關資訊的要求已傳送至 AWS IoT MQTT 型檔案傳送功能。

awsJobId

產生日誌的 AWS IoT 任務 ID。

clientId

發出產生日誌請求的 MQTT 用戶端 ID。

clientToken

與產生日誌請求相關聯的用戶端字符。

details

用於產生日誌之操作的詳細資訊。

logLevel

日誌的記錄層級。針對 OTA 更新日誌，這一律設為 `DEBUG`。

otaUpdateId

產生日誌的 OTA 更新 ID。

protocol

用於發出產生日誌請求的通訊協定。

status

產生日誌操作的狀態。有效值為：

- Success (成功)
- 失敗

streamId

產生日誌的 AWS IoT 串流 ID。

timestamp

日誌產生的時間。

topicName

用於發出產生日誌請求的 MQTT 主題。

範例日誌

以下是啟動程式碼簽署任務時產生的範例日誌：

```
{
    "timestamp": "2018-07-23 22:59:44.955",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "actionType": "StartCodeSigning",
    "otaUpdateId": "08957b03-eea3-448a-87fe-743e6891ca3a",
    "details": "Start code signing job. The request status is SUCCESS."
}
```

以下是建立 AWS IoT 任務時產生的範例日誌：

```
{
    "timestamp": "2018-07-23 22:59:45.363",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "actionType": "CreateAWSJob",
    "otaUpdateId": "08957b03-eea3-448a-87fe-743e6891ca3a",
    "awsJobId": "08957b03-eea3-448a-87fe-743e6891ca3a",
    "details": "Create AWS Job The request status is SUCCESS."
}
```

以下是建立 OTA 更新時產生的範例日誌：

```
{
    "timestamp": "2018-07-23 22:59:45.413",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "actionType": "CreateOTAUpdate",
    "otaUpdateId": "08957b03-eea3-448a-87fe-743e6891ca3a",
    "details": "OTAUpdate creation complete. The request status is SUCCESS."
}
```

以下是建立串流時產生的範例日誌：

```
{
```

```
    "timestamp": "2018-07-23 23:00:26.391",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "actionType": "CreateStream",
    "otaUpdateId": "3d3dc5f7-3d6d-47ac-9252-45821ac7cfb0",
    "streamId": "6be2303d-3637-48f0-ace9-0b87b1b9a824",
    "details": "Create stream. The request status is SUCCESS."
}
```

以下是刪除 OTA 更新時產生的範例日誌：

```
{
    "timestamp": "2018-07-23 23:03:09.505",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "actionType": "DeleteOTAUpdate",
    "otaUpdateId": "9bdd78fb-f113-4001-9675-1b595982292f",
    "details": "Delete OTA Update. The request status is SUCCESS."
}
```

以下是裝置向 MQTT 型檔案傳遞功能要求串流時所產生的範例記錄檔：

```
{
    "timestamp": "2018-07-25 22:09:02.678",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "actionType": "GetStream",
    "protocol": "MQTT",
    "clientId": "b9d2e49c-94fe-4ed1-9b07-286afed7e4c8",
    "topicName": "$aws/things/b9d2e49c-94fe-4ed1-9b07-286afed7e4c8/
streams/1e51e9a8-9a4c-4c50-b005-d38452a956af/get/json",
    "streamId": "1e51e9a8-9a4c-4c50-b005-d38452a956af",
    "details": "The request status is SUCCESS."
}
```

以下是裝置呼叫 `DescribeStream` API 時產生的範例日誌：

```
{
    "timestamp": "2018-07-25 22:10:12.690",
```

```
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "actionType": "DescribeStream",
    "protocol": "MQTT",
    "clientId": "581075e0-4639-48ee-8b94-2cf304168e43",
    "topicName": "$aws/things/581075e0-4639-48ee-8b94-2cf304168e43/streams/71c101a8-
bcc5-4929-9fe2-af563af0c139/describe/json",
    "streamId": "71c101a8-bcc5-4929-9fe2-af563af0c139",
    "clientToken": "clientToken",
    "details": "The request status is SUCCESS."
}
```

使用 AWS CloudTrail 記錄 AWS IoT OTA API 呼叫

FreeRTOS 與這項服務整合在一起CloudTrail，可擷取 AWS IoT OTA API 呼叫，並將日誌檔案傳送到您指定的 Amazon S3 儲存貯體。CloudTrail捕獲從您的代碼到 AWS IoT OTA API 的 API 調用。您可以使用 CloudTrail 所收集的資訊來判斷對 AWS IoT OTA 提出的請求、提出請求的來源 IP 地址、提出請求的對象，以及提出請求的時間等等。

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。

免費伺服器資訊 CloudTrail

在您的AWS帳戶中啟用日CloudTrail誌記錄時，對 AWS IoT OTA 操作進行的 API 調用將在CloudTrail日誌文件中跟踪，該文件將與其他AWS服務記錄一起編寫。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**
>
> CloudTrail 不會記錄 AWS IoT OTA 資料平面動作 (裝置端)，因此您需使用 CloudWatch 來監控這些動作。

每個日誌項目都會包含產生要求之人員的資訊。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。
- 該請求是否由另一項 AWS 服務提出。

如需詳細資訊，請參閱CloudTrail使用者識別元素。 AWS IoTOTA API 參考中記錄了 AWS IoT OTA 操作。

您可以視需要將日誌檔存放在 Amazon S3 儲存貯體中，但也可以定義 Amazon S3 生命週期規則以自動存檔或刪除日誌檔。根據預設，您的日誌檔會使用 Amazon S3 伺服器端加密 (SSE) 加密。

如果您希望在日誌檔交付時收到通知，可以設定CloudTrail為發佈 Amazon SNS 通知。如需詳細資訊，請參閱為 CloudTrail.

您也可以將來自多個AWS區域和多個AWS帳戶的 AWS IoT OTA 日誌檔彙總到單一 Amazon S3 儲存貯體中。

如需詳細資訊，請參閱從多個區域接CloudTrail收CloudTrail記錄檔和從多個帳戶接收記錄檔。

瞭解 FreeRTOS 記錄檔項目

CloudTrail 日誌檔案可包含一個或多個日誌項目。每一項目均列出多個 JSON 格式的事件。一個日誌項目為任何來源提出的單一請求，並包含請求動作、請求的日期和時間、請求參數等資訊。日誌項目並非公有 API 呼叫的有序堆疊追蹤，因此不會以任何特定順序顯示。

以下範例顯示 CloudTrail 日誌項目，示範呼叫 `CreateOTAUpdate` 動作的日誌。

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EXAMPLE",
        "arn": "arn:aws:iam::your_aws_account:user/your_user_id",
```

```
        "accountId": "your_aws_account",
        "accessKeyId": "your_access_key_id",
        "userName": "your_username",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2018-08-23T17:27:08Z"
            }
        },
        "invokedBy": "apigateway.amazonaws.com"
    },
    "eventTime": "2018-08-23T17:27:19Z",
    "eventSource": "iot.amazonaws.com",
    "eventName": "CreateOTAUpdate",
    "awsRegion": "your_aws_region",
    "sourceIPAddress": "apigateway.amazonaws.com",
    "userAgent": "apigateway.amazonaws.com",
    "requestParameters": {
        "targets": [
            "arn:aws:iot:your_aws_region:your_aws_account:thing/Thing_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"
}
```

取得建立輸出更新失敗詳細資料，使用 AWS CLI

如果創建 OTA 更新作業的過程失敗，則可以採取一些措施來解決問題。當您建立 OTA 更新任務時，OTA 管理員服務會建立 IoT 任務並針對目標裝置排程，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，而不建立AWS IoT工作。在此故障排除部分中，我們提供有關如何檢索故障詳細信息的說明。

1. 安裝及設定 AWS CLI。
2. 運行aws configure並輸入以下信息。

   ```
   $ aws configure
   AWS Access Key ID [None]: AccessID
   AWS Secret Access Key [None]: AccessKey
   Default region name [None]: Region
   Default output format [None]: json
   ```

   如需詳細資訊，請參閱使用快速組態aws configure。
3. 執行：

   ```
   aws iot get-ota-update --ota-update-id ota_update_job_001
   ```

   其中，#### OTA ##時提供的識別碼。
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)"
        }
    }
}
```

如果創建失敗，命令輸出中的otaUpdateStatus字段將包含，CREATE_FAILED並且該errorInfo字段將包含失敗的詳細信息。

使用 AWS CLI 取得 OTA 失敗代碼

1. 安裝及設定 AWS CLI。
2. 運行aws configure並輸入以下信息。

```
$ aws configure
AWS Access Key ID [None]: AccessID
AWS Secret Access Key [None]: AccessKey
Default region name [None]: Region
Default output format [None]: json
```

如需詳細資訊，請參閱使用快速組態aws configure。

3. 執行：

```
aws iot describe-job-execution --job-id JobID --thing-name ThingName
```

其中 *joB* id 是我們想要獲得其狀態的作業的完整作業 ID 字符串（創建時與 OTA 更新作業相關聯），並且*ThingName*是設備註冊為的AWS IoT事物名稱 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

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

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。要恢復設備，您可以對其進行重新編程或使用該restore-to-factory過程恢復為出廠映像。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 式庫

FreeRTOS 程式庫為 FreeRTOS 核心及其內部程式庫提供額外的功能。您可以在嵌入式應用程式中使用 FreeRTOS 程式庫來實現網路和安全性。FreeRTOS 程式庫也可讓您的應用程式與AWS IoT服務互動。FreeRTOS 包含的程式庫可讓您：

- 使用 MQTT 和裝置影子安全地將裝置連線至 AWS IoT 雲端。
- 搜索並連線到 AWS IoT Greengrass 核心。
- 管理 Wi-Fi 連線。
- 聆聽和處理 空中免費更新。

此目libraries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

FreeRTOS 的組態中包含下列移植程式庫，可在 FreeRTOS 主控台上下載。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相關資訊，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 代理程式進行通訊。如需詳細資訊，請參閱低功耗藍牙程式庫。
無線更新	AWS IoTOver-the-air 更新 API 參考資料	FreeRTOSAWS IoT Over-the-air (OTA) 更新程式庫可讓您管理更新通知、下載更新，以及執行 FreeRTOS 裝置上韌體更新的密碼編譯驗證。 如需詳細資訊，請參閱AWS IoT空中 (OTA) 圖書館。
FreeRTOS+POSIX	FreeRTOS+POSIX API 參考	您可以使用 FreeRTOS 程式庫將符合 POSIX 標準的應用程式移植到 FreeRTOS 生態系統。 如需詳細資訊，請參閱 FreeRTOS+POSIX。
Secure Sockets	安全通訊端 API 參考	如需詳細資訊，請參閱Secure Sockets 程式庫。
FreeRTOS+TCP	免費連線 + TCP API 參考資料	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 程式庫可讓您與微控制器的較低階無線堆疊連接。 如需詳細資訊，請參閱 Wi-Fi 程式庫。

程式庫	API 參考	描述
海洋公園 11		CorepkCS11 程式庫是公開金鑰加密標準 #11 的參考實作，可支援佈建和 TLS 用戶端驗證。 如需詳細資訊，請參閱 Corep11 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。
蜂窩界面	行動網路 API 參考	蜂窩接口庫通過統一的 API 公開了一些流行的蜂窩調製解調器的功能。如需詳細資訊，請參閱 行動網路界面。

FreeRTOS 應用程式庫

您可以選擇性地在 FreeRTOS 組態中包含下列獨立應用程式程式庫，以便與雲端上的AWS IoT服務互動。

> **Note**
>
> 某些應用程式程式庫與嵌入式 CAWS IoT 裝置 SDK 中的程式庫具有相同的 API。如需這些程式庫，請參閱AWS IoT裝置 SDK C API 參考。如需嵌入式 C 的AWS IoT詳細資訊，請參閱適用於 Embedded C 的 AWS IoT 裝置 SDK。

FreeRTOS 應用程式庫

程式庫	API 參考	描述
AWS IoT Device Defender	設備防禦者 C SDK API 參考	FreeRTOSAWS IoT Device Defender 程式庫會將您的 FreeRTOS 裝置連接到AWS IoT Device Defender。

程式庫	API 參考	描述
		如需詳細資訊，請參閱AWS IoT Device Defender 程式庫。
AWS IoT Greengrass	Greengrass API 參考	FreeRTOSAWS 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 Discovery 程式庫。
MQTT	MQTT (v1.x.x) 圖書館應用程式介面參考 MQTT (v1) 代理程式 API 參考 MQTT (v2.x.x) C 開發套件應用程式介面參考	CoreMQTT 程式庫提供您的 FreeRTOS 裝置的用者來發佈和訂閱 MQTT 主題。MQTT 是裝置用來與 AWS IoT 互動的通訊協定。 如需有關 CoremQtt 程式庫 3.0.0 版的詳細資訊，請參閱通訊協定圖書館。
CoreMTT 代理程式	核心代理程式程式庫 API 參考	CoremQtt 代理程式庫是一種高階 API，可將執行緒安全性新增至 CoremQtt 程式庫。它可讓您建立專用的 MQTT 代理程式工作，以便在背景管理 MQTT 連線，而且不需要其他工作的任何介入。此程式庫提供與 CoremQtt API 的執行緒安全等效項目，因此可在多執行緒環境中使用。 如需 CoremQTT 代理程式庫的詳細資訊，請參閱CoremQTT 代理器程式庫。

程式庫	API 參考	描述
AWS IoT Device Shadow	Device Shadow C SDK API 參考	AWS IoTDevice Shadow 程式庫可讓 FreeRTOS 裝置與AWS IoT裝置陰影互動。 如需詳細資訊，請參閱AWS IoT 裝置影子程式庫。

設定 FreeRTOS 式庫

FreeRTOS 的組態設定和嵌入式 C 的AWS IoT裝置 SDK 定義為 C 預處理器常數。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。

退回演算法程式庫

> **Note**
>
> 此頁面上的內容可能不是 up-to-date。請參閱免費圖書館頁面以獲取最新更新。

簡介

該退款算法庫是一個實用程序庫，用於將同一數據塊的重複重傳輸空間，以避免網絡擁塞。此程式庫會計算重試網路作業的輪詢期間 (例如與伺服器的網路連線失敗)具有抖動的指數輪詢演算法。

當對伺服器的網路擁塞或伺服器上的高負載造成的伺服器重試失敗的連線或網路要求時，通常會使用含抖動的指數輪詢。它用於分散由嘗試同時進行網絡連接的多個設備創建的重試請求的時間。在連線不良的環境中，用戶端可能會隨時中斷連線；因此，輪詢策略也可協助用戶端在不太可能成功時不會重複嘗試重新連線，以節省電池電力。

該庫是用 C 編寫的，設計符合ISO和米什拉 C: 2012。除了標準 C 庫之外，該庫沒有依賴於任何其他庫，並且沒有堆分配，因此適用於 IoT 微控制器，但也可以完全移植到其他平台。

該庫可以自由使用，並在MIT 開放原始碼授權。

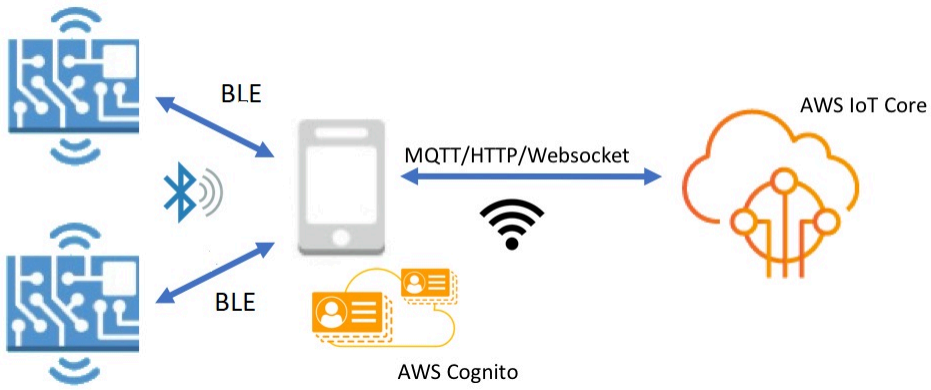
備用算法的代碼大小（用於 ARM Cortex-M 的 GCC 生成的示例）		
檔案	使用-O1 最佳化	使用-Os 優化
退出算法	0.1K	0.1K
估計總數	0.1K	0.1K

低功耗藍牙程式庫

> **Important**
>
> 該庫託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。我們建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱. 亞馬遜自由 Github 存儲庫遷移指南

概觀

FreeRTOS 支援透過藍牙低功耗透過代理裝置 (例如行動電話) 發佈及訂閱訊息佇列遙測傳輸 (MQTT) 主題。使用 FreeRTOS 低功耗藍牙 (BLE) 程式庫，您的微控制器可以安全地與 MQTT 代理程式通訊。AWS IoT

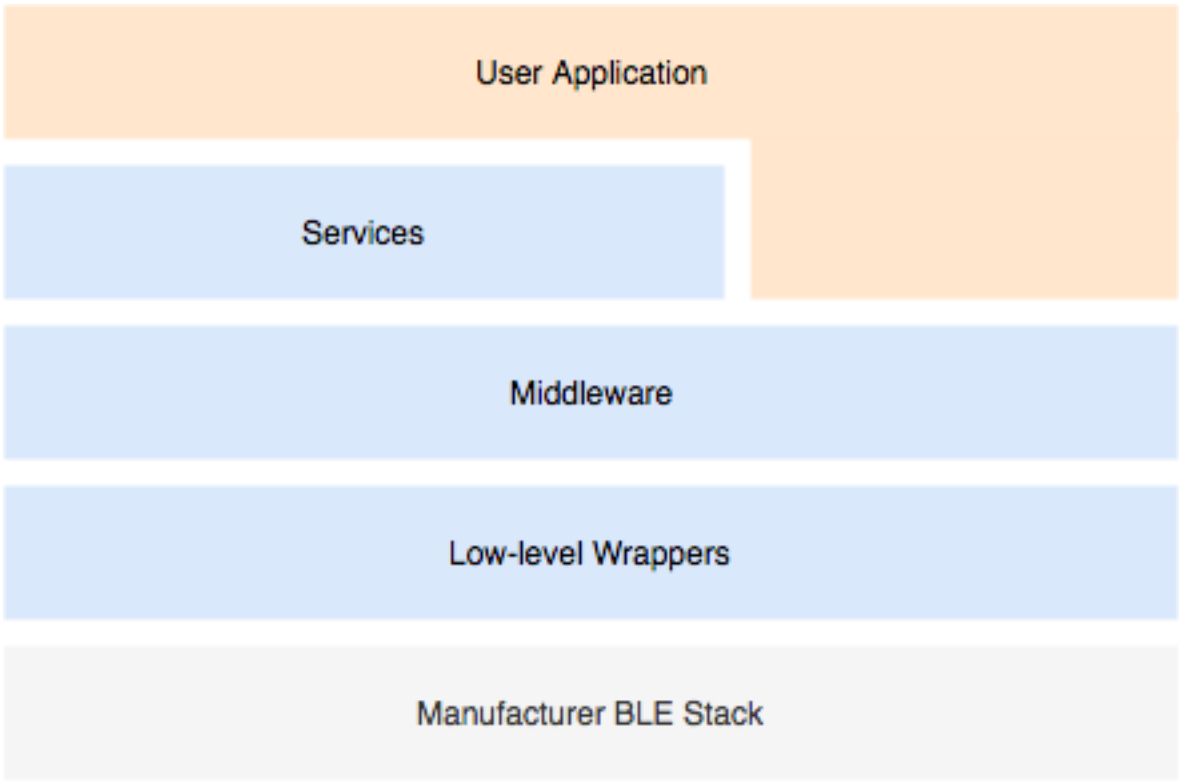


使用 FreeRTOS 藍牙裝置的行動 SDK，您可以撰寫原生行動應用程式，透過 BLE 與微控制器上的嵌入式應用程式進行通訊。如需行動軟體開發套件的詳細資訊，請參閱 適用於 FreeRTOS 藍牙裝置的行動 SDK。

FreeRTOS BLE 程式庫包含設定 Wi-Fi 網路、傳輸大量資料，以及透過 BLE 提供網路抽象的服務。FreeRTOS BLE 程式庫也包含中介軟體和較低層級的 API，可讓您更直接地控制 BLE 堆疊。

架構

FreeRTOS BLE 程式庫有三層組成：服務、中介軟體和低層級包裝函式。



服務

FreeRTOS BLE 服務層由四個通用屬性 (GATT) 服務組成，這些服務會利用中介軟體 API：

- 裝置資訊
- Wi-Fi 佈建
- 網路抽象
- 大型物件傳輸

裝置資訊

裝置資訊服務會收集微控制器的詳細資料，包括：

- 您的裝置正在使用的 FreeRTOS 版本。

- 為其註冊裝置的帳戶 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」服務會將資料傳送至用戶端，並從用戶端接收資料。其他服務 (例如 Wi-Fi 佈建和網路抽象) 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。

透過 BLE 的 MQTT

通過 BLE 的 MQTT 包含用於通過 BLE 創建 MQTT 代理服務的關貿總協定文件。MQTT 代理服務允許 MQTT 客戶端通過閘道設備與 AWS MQTT 代理商進行通信。例如，您可以使用代理伺服器，透過智慧型手機應用程式將執行 FreeRTOS 的裝置連接到 AWS MQTT。BLE 設備是 GATT 服務器，並公開了網關設備的服務和特性。GATT 伺服器使用這些公開的服務和特性，透過該裝置的雲端執行 MQTT 作業。如需更多詳細資訊，請參閱 附錄 A：關於 BLE 關貿總協定檔的 MQTT。

中介軟體

FreeRTOS 藍牙低功耗中介軟體是從較低層級的 API 抽象化。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

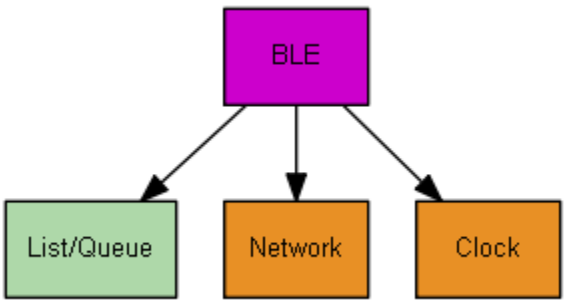
低階 FreeRTOS 低功耗藍牙包裝器是製造商藍牙低功耗堆疊的抽象。低階包裝函式提供一組常用的 API，以直接控制硬體。低階 API 會最佳化 RAM 使用量，但限於功能方面。

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：

- 線性容器庫
- 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

要與 AWS IoT MQTT 經紀人進行通信，您必須擁有一個 AWS 帳戶，並且必須將設備註冊為 AWS IoT 東西。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份驗證。若要使用 MQTT 代理服務，您必須建立 Amazon Cognito 身分識別和使用者集區。每個 Amazon Cognito 身分都必須附加適當的政策。如需詳細資訊，請參閱 Amazon Cognito 開發人員指南。

程式庫組態檔案

透過藍牙低功耗服務使用 FreeRTOS MQTT 的應用程式必須提供`iot_ble_config.h`標頭檔案，並在其中定義組態參數。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 藍牙低功耗程式庫會將內`eBTpropertySecureConnectionOnly`容設定為 TRUE，讓裝置處於僅限安全連線模式。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 藍牙低功耗 GATT 服務，您可以略過中介軟體並直接與低階 API 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 藍牙低功耗示範應用程式，請參閱藍牙低功耗示範應用程式。

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. AddCustomServicesCb 在您的應用程序中創建 IotBle _：

```
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 移植到新的目標，您必須為 Wi-Fi 佈建服務和低功耗藍牙功能實作一些 API。

低功耗藍牙 API

若要使用 FreeRTOS 藍牙低功耗中介軟體，您必須實作一些 API。

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 低功耗藍牙元件庫移植到您的平台的詳細資訊，請參閱 FreeRTOS 移植指南中的移植低功耗藍牙程式庫。

適用於 FreeRTOS 藍牙裝置的行動 SDK

> **Important**
>
> 該庫託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。我們建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱. 亞馬遜自由 Github 存儲庫遷移指南

您可以使用 FreeRTOS 藍牙裝置的行動 SDK，建立透過藍牙低功耗與微控制器互動的行動應用程式。行動開發套件也可以使用 Amazon Cognito 進行使用者身份驗證，與 AWS 服務進行通訊。

適用於免費藍牙設備的安卓 SDK

使用適用於 FreeRTOS 藍牙設備的 Android SDK 來構建 Android 移動應用程序，該應用程序可通過藍牙低功耗與微控制器進行交互。您可以在上使用 SDK GitHub。

若要安裝適用於 FreeRTOS 軟體的安卓軟體開發套件藍牙裝置，請依照專案的 Re adme.md 檔案中「設定 SDK」的指示進行。

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，請參閱必要條件和免費藍牙低功耗移動 SDK 演示應用程序。

適用於免費藍牙裝置的 iOS 開發套件

使用適用於 FreeRTOS 藍牙裝置的 iOS SDK，建立 iOS 行動應用程式，透過低功耗藍牙與微控制器互動。您可以在上使用 SDK GitHub。

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（將其添加到您的 podfile 中）安裝 SDK：

```
$ 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，請參閱必要條件和免費藍牙低功耗移動 SDK 演示應用程序。

附錄 A：關於 BLE 關貿總協定檔的 MQTT

關貿總協定服務詳情

透過 BLE 的 MQTT 使用資料傳輸 GATT 服務的執行個體，在 FreeRTOS 裝置與代理裝置之間傳送 MQTT 簡明二進位物件表示 (CBOR) 訊息。資料傳輸服務公開某些特性，可協助透過 BLE GATT 通訊協定傳送和接收原始資料。它還處理大於 BLE 最大傳輸單元（MTU）大小的有效負載的分段和組件。

服務 UUID

A9D7-166A-D72E-40A9-A002-4804-4CC3-FF00

服務實例

GATT 服務的一個實例是為與代理商的每個 MQTT 會話創建的。每個服務都有一個唯一的 UUID（兩個字節），用於識別其類型。每個單獨的例證由實例 ID 區分開來。

每個服務都被實例化為每個 BLE 服務器設備上的主要服務。您可以在指定裝置上建立服務的多個執行個體。MQTT 代理伺服器服務類型具有唯一的 UUID。

特性

特徵內容格式：CBOR

最大特性值大小：512 位元組

特性	需求	強制性質	可選屬性	安全性權限	簡要說明	UUID
控制項	M	寫入	無	寫入需要加密	用於啟動和停止 MQTT 代理。	A9D7-166A-D72E-40A9-A002-4804-4CC3-FF01
TX留言	M	讀取、通知	無	讀取需要加密	用於通過代理將包含消息的通知發送給代理人。	A9D7-166A-D72E-40A9-A002-4804-4CC3-FF02
Rx 訊息	M	讀取、寫入無回應	無	讀取、寫入需要加密	用於通過代理接收來自經紀	A9D7-166A-D72E-40A9-

特性	需求	強制性質	可選屬性	安全性權限	簡要說明	UUID
					人的消息。	A002-4804-4CC3-FF03
TX LargeMessage	M	讀取、通知	無	讀取需要加密	用於透過代理伺服器傳送大型訊息 (訊息 > BLE MTU 大小) 給代理程式。	A9D7-166A-D72E-40A9-A002-4804-4CC3-FF04
接收 LargeMessage	M	讀取、寫入無回應	無	讀取、寫入需要加密	用於透過代理伺服器從代理程式接收大型訊息 (訊息 > BLE MTU 大小)。	A9D7-166A-D72E-40A9-A002-4804-4CC3-FF05

關貿總協定程序要求

讀取特性值	強制性
讀取長特性值	強制性
寫入特性值	強制性
寫入長特性值	強制性
讀取特性描述元	強制性
寫入特徵描述元	強制性

通知	強制性
適應症	強制性

訊息類型

會交換下列訊息類型。

訊息類型	訊息	使用這些鍵/值對對映
0x01	CONNECT	• 鍵 =「W」，值 = 類型 0 整數，消息類型（1） • 鍵 =「d」，值 = 類型 3，文本字符串，會話的客戶端標識符 • 索引鍵 =「a」，值 = 類型 3、文字字串、工作階段的代理人端點 • 鍵 =「C」，值 = 簡單值類型真/假
0X02	CONNACK	• 鍵 =「W，值 = 類型 0 整數，消息類型（2） • 鍵 =「s」，值 = 類型 0 整數，狀態碼
0X03	發佈	• 鍵 =「W」，值 = 類型 0 整數，消息類型（3） • 鍵 =「u」，值 = 類型 3，文本字符串，發布主題 • 鍵 =「n」，值 = 類型 0，整數，用於發佈的 QoS • 鍵 =「i」，值 = 類型 0，整數，消息標識符，僅適用於 QoS 1 發布

訊息類型	訊息	使用這些鍵/值對對映
		• 鍵 =「k」，值 = 類型 2，字節字符串，有效負載發布
0X04	恥骨	• 僅針對 QoS 1 訊息傳送。 • 鍵 =「W」，值 = 類型 0 整數，消息類型（4） • 鍵 =「i」，值 = 類型 0，整數，消息標識符
0X08	訂閱	• 鍵 =「W」，值 = 類型 0 整數，消息類型（8） • 鍵 =「v」，值 = 類型 4，文本字符串數組，主題訂閱 • 鍵 =「o」，值 = 類型 4，整數數組，用於訂閱的 QoS • 鍵 =「i」，值 = 類型 0，整數，消息標識符
0X09	副本	• 鍵 =「W」，值 = 類型 0 整數，消息類型（9） • 鍵 =「i」，值 = 類型 0，整數，消息標識符 • 鍵 =「s」，值 = 類型 0，整數，訂閱的狀態碼
0A	取消訂閱	• 鍵 =「W」，值 = 類型 0 整數，消息類型（10） • 鍵 =「v」，值 = 類型 4，文本字符串數組，取消訂閱的主題 • 鍵 =「i」，值 = 類型 0，整數，消息標識符

訊息類型	訊息	使用這些鍵/值對對映
0B	解開	• 鍵 =「W」，值 = 類型 0 整數，消息類型（11） • 鍵 =「i」，值 = 類型 0，整數，消息標識符 • 鍵 =「s」，值 = 類型 0，整數，狀態碼 UnSubscription
0℃	平格雷克	• 鍵 =「W」，值 = 類型 0 整數，消息類型（12）
0D	平格雷斯普	• 鍵 =「W」，值 = 類型 0 整數，消息類型（13）
0E	不承認	• 鍵 =「W」，值 = 類型 0 整數，消息類型（14）

大有效載荷傳輸特性

TX LargeMessage

裝置會使用 TX LargeMessage 來傳送大於針對 BLE 連線協商的 MTU 大小的大型裝載。

- 該設備通過該特徵發送有效負載的第一個 MTU 字節作為通知。
- Proxy 會針對剩餘位元組傳送此特性的讀取要求。
- 設備發送到 MTU 大小或有效負載的剩餘字節，以較少者為準。每次，它都會增加發送的有效負載大小讀取的偏移量。
- 代理將繼續讀取特徵，直到獲得零長度有效負載或小於 MTU 大小的有效負載為止。
- 如果裝置在指定的逾時內未收到讀取要求，則傳輸會失敗，而 Proxy 和閘道會釋放緩衝區。
- 如果 Proxy 在指定的逾時內未取得讀取回應，則傳輸會失敗，而 Proxy 會釋放緩衝區。

接收 LargeMessage

裝置會使LargeMessage 用 RX 來接收大於針對 BLE 連線協商的 MTU 大小的大型裝載。

- 代理伺服器會使用寫入並回應此特性，一個接一個地寫入訊息 (最大為 MTU 大小)。
- 裝置會緩衝訊息，直到收到長度為零或長度小於 MTU 大小的寫入要求為止。

- 如果設備在指定的超時內沒有收到寫入請求，則傳輸會失敗，並且設備釋放緩衝區。
- 如果 Proxy 在指定的逾時內未取得寫入回應，則傳輸會失敗，而 Proxy 會釋放緩衝區。

行動網路界面

> **Note**
>
> 此頁面上的內容可能不是 up-to-date。如需最新的更新，請參閱 FreerToss.org 程式庫頁面。

簡介

蜂窩接口庫實現了一個簡單的統一 API，該 API 隱藏了蜂窩調製解調器特定 AT 命令的複雜性，並向 C 程序員公開類似插座的接口。

大多數蜂窩調製解調器實現或多或少由 3GPP TS v27.007 標準定義的 AT 命令。該項目在可重複使用的通用組件中提供了這種標準 AT 命令的實現。該項目中的三個蜂窩接口庫都利用了該通用代碼。每個調製解調器的庫僅實現特定於供應商的 AT 命令，然後公開完整的蜂窩接口庫 API。

實作 3GPP TS v27.007 標準的通用元件已撰寫符合下列程式碼品質標準：

- GNU 複雜度分數未超過 8
- 米斯拉 C：2012 編碼標準。與標準的任何偏差都記錄在源代碼註釋中，標有「隱蔽性」。

相依性和要求

行動網路介面程式庫沒有直接相依性。但是，以太網，Wi-Fi 和蜂窩無法在 FreeRTOS 網絡堆棧中共存。開發人員必須選擇其中一個網路介面，以與安全通訊端程式庫整合。

移植

如需將行動網路介面程式庫移植到您平台的相關資訊，請參閱《FreeRTOS 移植指南》中的移植行動網路介面程式庫。

記憶體使用

蜂窩接口庫的代碼大小（用於 ARM Cortex-M 的 GCC 生成的示例）		
檔案	使用-O1 最佳化	使用-Os 優化
西班牙蜂窩	6.3K	5.7 公里
手机网络处理器	0.9K	0.8 萬
蜂窩核心	1.4K	1.2 千
蜂窩共同體	0.5 公里	0.5 公里
蜂窩常用	1.6 千	1.4K
蜂窩電話 .c	1.4K	1.2 千
蜂窩行動	1.8 千	1.6 千
估計總數	13.9 千	12.4K

入門

下載原始程式

原始程式碼可以下載為 FreeRTOS 程式庫的一部分，也可以自行下載。

要使用 HTTPS 從 Github 克隆庫：

```
git clone https://github.com/FreeRTOS/FreeRTOS-Cellular-Interface.git
```

使用 SSH：

```
git clone git@github.com:FreeRTOS/FreeRTOS-Cellular-Interface.git
```

資料夾結構

在此儲存庫的根目錄，您將看到以下資料夾：

- `source`：實現由 3GPP TS v27.007 定義的標準 AT 命令的可重複使用的通用代碼

- doc: 文件
- test：單元測試和 cbmc
- tools：隱蔽性靜態分析和 cMock 的工具

設定和建置程式庫

行動網路界面程式庫應建立為應用程式的一部分。若要這樣做，您必須提供某些組態。自由行動組織介面視窗模擬器專案提供如何設定組建的範例。在行動網路 API 參考中可以找到更多的詳細資訊。

如需詳細資訊，請參閱行動網路介面頁面。

將行動網路介面程式庫與 MCU 平台整合

蜂窩接口庫使用抽象的接口，即通信接口在 MCU 上運行，以便與蜂窩調製解調器進行通信。一個通信接口也必須在 MCU 平台上實現。通訊接口的最常見實現通過 UART 硬件進行通信，但也可以通過其他物理接口（例如 SPI）實現。通訊接口的文檔可以在蜂窩圖書館 API 參考中找到。可以使用以下的 Com 接口示例實現：

- 自由視窗模擬器通訊介面
- FreeRTOS 通用 IO 通訊介面
- STM32 探索板通訊介面
- 塞拉利昂傳感器集線器板通信

通用 I/O

> **Important**
>
> 該庫託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱亞馬遜自由 Github 存儲庫遷移指南.

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 I/O 充當這個硬體抽象層。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 — 在傳輸要求的資料量之前，不會傳回函數。
- 非同步讀取/寫入 — 函數會立即傳回，且資料傳輸非同步發生。當動作完成時，即會叫用已註冊的用戶回呼。

周邊裝置特定

- I2C — 將多個操作合併為一個交易。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，這意味著寫入和讀取同時發生。

移植

如需詳細資訊，請參閱 FreeRTOS 移植指南。

AWS IoT Device Defender 程式庫

> **Note**
>
> 此頁面上的內容可能不是 up-to-date。請參閱免費圖書館頁面以獲取最新更新。

簡介

您可以使用AWS IoT Device Defender將安全指標從 IoT 裝置傳送至的程式庫AWS IoT Device Defender。您可以使用AWS IoT Device Defender持續監控裝置上的這些安全指標是否與您定義為每個裝置的適當行為有所偏差。如果有什麼東西看起來不對AWS IoT Device Defender傳送警示，方便您採

取動作，方便您採取動作。與之互動AWS IoT Device Defender使用MQTT，一個輕量級的發布-訂閱協議。該庫提供了一個 API 來組成和識別使用的 MQTT 主題字符串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。

該庫是用 C 編寫的，設計符合ISO和米什拉 C: 2012。除了標準 C 庫之外，該庫沒有任何其他庫的依賴關係。它也沒有任何平台依賴關係，例如線程或同步。它可以與任何 MQTT 庫和任何一個使用JSON或者CBR圖書館。圖書館有樣張顯示安全的內存使用和沒有堆分配，使其適用於 IoT 微控制器，但也完全可移植到其他平台。

該AWS IoT Device Defender圖書館可以自由使用，並在MIT 開放原始碼授權。

的代碼大小AWS IoT設備後衛（使用 GCC 為 ARM 皮質-M 生成的示例）		
檔案	使用-O1 最佳化	使用-Os 優化
防禦者 .c	1.1K	0.6 公里
估計總數	1.1K	0.6 公里

AWS IoT Greengrass Discovery 程式庫

> **Note**
>
> 此頁面上的內容可能不是 up-to-date。如需最新的更新，請參閱 FreerTos.org 程式庫頁面。

概要

您的微控制器裝置會使用AWS IoT Greengrass探索程式庫來探索網路上的 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 Discovery 程式庫，您必須在 AWS IoT 中建立物件，包含憑證與政策。如需詳細資訊，請參閱 AWS IoT 入門。

您必須在 *freertos*/demos/include/aws_clientcredential.h 檔案中設定以下常數的值：

clientcredentialMQTT_BROKER_ENDPOINT

您的 AWS IoT 端點。

clientcredentialIOT_THING_NAME

您的 IoT 實物名稱。

clientcredentialWIFI_SSID

您的 Wi-Fi 網路 SSID。

clientcredentialWIFI_PASSWORD

您的 Wi-Fi 密碼。

clientcredentialWIFI_SECURITY

您 Wi-Fi 網路所使用的安全類型。

您必須同時在 *freertos*/demos/include/aws_clientcredential_keys.h 檔案中設定以下常數的值：

keyCLIENT_CERTIFICATE_PEM

與您物件相關聯的憑證 PEM。

keyCLIENT_PRIVATE_KEY_PEM

與您物件相關聯的私有金鑰 PEM。

您必須在主控台中設定一個 Greengrass 群組及核心裝置。如需詳細資訊，請參閱 AWS IoT Greengrass 入門。

雖然 CoremQtt 程式庫不需要 Greengrass 連線，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 裝置會透過從 AWS IoT 請求包含 Greengrass 核心連線能力參數的 JSON 檔案來初始化搜索程序。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 AWS IoT 發送 HTTP GET 請求，初始化搜索 Greengrass 核心的搜索請求。`GD_SecureConnect_Send` 會用於向 AWS IoT 傳送請求。

`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 函式庫

> **Note**
>
> 此頁面上的內容可能不是 up-to-date。請參閱圖書館頁面以獲取最新更新。

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 程式庫是一個子集的用戶端實作12.11.0 版標準。HTTP 標準提供在 TCP/IP 之上執行的無狀態通訊協定，通常用於分散式、協同作業的超文字資訊系統中。

CoreHTTP 程式庫會實作下列項目的子集12.11.0 版協議標準。此程式庫已針對低記憶體佔用量進行了最佳化。該庫提供了一個完全同步的 API，因此應用程序可以完全管理它們的並發。它僅使用固定緩衝區，因此應用程序可以完全控制其內存分配策略。

該庫是用 C 編寫的，設計符合ISO和米什拉 C: 2012。庫唯一的依賴關係是標準的 C 庫和http-分析器的 LTS 12.19.1 版是從 Node.js 版本。圖書館有樣張顯示安全的內存使用和沒有堆分配，使其適用於 IoT 微控制器，但也完全可移植到其他平台。

在 IoT 應用程式中使用 HTTP 連線時，建議您使用安全的傳輸介面，例如使用 TLS 通訊協定的介面，如核心 HTTP 相互驗證示範。

該庫可以自由使用，並在MIT 開放原始碼授權。

CoreHTTP 的代碼大小（使用 GCC 為 ARM 皮質-M 生成的示例）		
檔案	使用-O1 最佳化	使用-Os 優化
客戶端管理系統	3.2K	2.6 公里

CoreHTTP 的代碼大小（使用 GCC 為 ARM 皮質-M 生成的示例）		
檔案	使用-O1 最佳化	使用-Os 優化
管理局 (有限公司)	2.6 公里	2.0 公里
网址管理局 (有限公司)	0.3K	0.3K
有限公司網站 (有限公司)	17.9	15.9
估計總數	23.9K	20.7 公里

CoreJSON 程式庫

> **Note**
>
> 此頁面上的內容可能不是 up-to-date。如需最新的更新，請參閱 FreerToss.org 程式庫頁面。

簡介

JSON（JavaScript 對象符號）是一種人類可讀的數據序列化格式。它被廣泛用於交換數據，例如與 AWS IoTDevice Shadow 服務，並且是許多 API 的一部分，例如 GitHub REST API。JSON 是由埃克馬國際維護作為標準。

CoreJSON 程式庫提供支援金鑰查詢的剖析器，同時嚴格執行 ECMA-404 標準 JSON 資料交換語法。該圖書館是用 C 編寫的，旨在符合 ISO C90 和米斯拉 C：2012 的規定。它具有證明顯示安全的內存使用和沒有堆分配，使其適用於 IoT 微控制器，但也完全可移植到其他平台。

記憶體使用

CoreJSON 程式庫會使用內部堆疊來追蹤 JSON 文件中的巢狀結構。堆棧在單個函數調用期間存在; 它不會被保留。堆棧大小可以通過定義宏`JSON_MAX_DEPTH`，默認為 32 級來指定。每個級別消耗一個字節。

CoreJSON 的代碼大小（使用 GCC 為 ARM 皮質-M 生成的示例）		
檔案	使用-O1 最佳化	使用-Os 優化
核心股份有限公司	2.9 公里	2.4K
估計總數	2.9 千	2.4K

通訊協定圖書館

> **Note**
>
> 此頁面上的內容可能不是 up-to-date。請參閱圖書館頁面以獲取最新更新。

簡介

CoremQTT 程式庫是一個用戶端實作MQTT(訊息佇列遙測傳輸) 標準。MQTT 標準提供輕量級發布/訂閱（或PubSub) 在 TCP/IP 之上執行的訊息通訊協定，通常用於機器對機器 (M2M) 和物聯網 (IoT) 使用案例。

CoremQTT 程式庫符合毫克特特通訊協定標準。該庫已針對低內存佔用進行了優化。此程式庫的設計包含不同的使用案例，從僅使用 QoS 0 MQTT PUBLISH 訊息的資源受限平台，到使用 QoS 2 MQTT PUBLISH 透過 TLS (傳輸層安全性) 連線的資源豐富平台。該庫提供組合函數的菜單，可以選擇和組合以精確地滿足特定用例的需求。

圖書館是寫在C並且設計符合ISO和米什拉 C: 2012 年。除了以下內容之外，此 MQTT 庫沒有任何其他庫的依賴關係：

- C 標準函式庫
- 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介面規格，將程式庫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，或視其應用程式實作自己的傳輸介面。

該庫提供了一個高級 API，用於連接到 MQTT 代理，訂閱/取消訂閱主題，向主題發布消息以及接收傳入消息。此 API 會將上述傳輸介面作為參數，並使用該介面來傳送和接收訊息至 MQTT 代理程式。

該庫還公開了低級序列化程序/解串器 API。此 API 可用來建置只包含所需 MQTT 功能子集的簡單 IoT 應用程式，而不會產生任何其他額外負荷。序列化程式/還原序列化程式 API 可以與任何可用的傳輸層 API（如套接字）結合使用，以便向代理程式傳送和接收訊息。

在 IoT 應用程式中使用 MQTT 連線時，建議您使用安全的傳輸介面，例如使用 TLS 通訊協定的介面。

這個 MQTT 庫沒有平台依賴關係，例如線程或同步。這個庫確實有樣張這表明了安全的內存使用和沒有堆分配，這使得它適用於 IoT 微控制器，但也完全可移植到其他平台。它可以自由使用，並在MIT 開放原始碼授權。

CoremQtt 的代碼大小（使用 GCC 為 ARM 皮質-M 生成的示例）		
檔案	使用-O1 最佳化功能	使用-Os 優化
核心 _mqtt.c	4.0 千	3.4K
核心 _ 國家 .c	1.7 萬	1.3K
核心 mqtt_ 序列化器 .c	2.8 千	2.2 千
估計總數	8.5 千	6.9 公里

CoremQTT 代理器程式庫

> **Note**
>
> 此頁面上的內容可能不是 up-to-date。請參閱免費圖書館頁面以獲取最新更新。

簡介

CoremQtt 代理程式程式庫是一種高階 API，可將執行緒安全性新增至通訊協定圖書館。它可讓您建立專用的 MQTT 代理程式工作，以便在背景管理 MQTT 連線，而且不需要其他工作的任何介入。此程式庫提供與 CoremQtt API 的執行緒安全等效項目，因此可在多執行緒環境中使用。

MQTT 代理是一個獨立的任務（或執行線程）。它通過成為唯一允許訪問 MQTT 庫 API 的任務來實現線程安全性。它通過將所有 MQTT API 調用隔離到單個任務序列化訪問，並且不需要信號量或任何其他同步原語。

該庫使用線程安全消息傳遞隊列（或其他進程間通信機制）序列化調用 MQTT API 的所有請求。訊息傳遞實作透過訊息傳送介面與程式庫分離，可讓程式庫移植到其他作業系統。訊息傳送介面是由傳送和接收代理程式命令結構指標的函數組成，以及配置這些命令物件的函數，可讓應用程式寫入器決定適合其應用程式的記憶體配置策略。

該庫是用 C 編寫的，設計符合ISO和米什拉 C: 2012。除了圖書館之外，該庫沒有任何其他庫的依賴關係通訊協定圖書館和標準的 C 庫。程式庫有樣張顯示安全的內存使用和沒有堆分配，因此它可以用於 IoT 微控制器，但也可以完全移植到其他平台。

該庫可以自由使用，並在MIT 開放原始碼授權。

CoremQtt 代理程式的程式碼大小 (使用 GCC 為 ARM 皮質-M 產生的範例)		
檔案	使用-O1 最佳化	使用-Os 優化
核心 _mqtt_ 代理程式	1.7 萬	1.5 公里
核心 _ 代理程式 _ 命令 _ 函數	0.3K	0.2K
核心 _ mqttc (中央网站)	4.0 千	3.4K
核心 _ 國家 .c (中央核心)	1.7 萬	1.3K
核心 _mqtt_ 序列化程式. c (核心 QTT)	2.8 千	2.2 千
估計總數	10.5 公里	8.6 千

AWS IoT空中 (OTA) 圖書館

> **Note**
>
> 此頁面上的內容可能不是 up-to-date。如需最新的更新，請參閱 FreerTos.org 程式庫頁面。

簡介

AWS IoTOver-the-air (OTA) 更新程式庫可讓您使用 HTTP 或 MQTT 作為通訊協定來管理 FreeRTOS 裝置韌體更新的通知、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。

物聯網（IoT）將互聯網連接擴展到傳統上未連接的嵌入式設備。這些設備可以被編程為通過互聯網傳輸可用數據，並且可以遠程監控和控制。隨著技術的進步，這些傳統嵌入式裝置正以快速的速度在消費性、工業和企業領域獲得網際網路功能。

IoT 設備通常大量部署，通常部署在人類操作員難以訪問或不切實際的地方。想像一下發現可能暴露資料的安全性弱點的案例。在這種情況下，快速可靠地使用安全性修復程序更新受影響的設備非常重要。如果沒有執行 OTA 更新的能力，也可能很難更新分散在地理位置的設備。讓技術人員更新這些設備將是昂貴的，耗時，並且通常是不切實際的。更新這些裝置所需的時間會讓這些裝置長時間暴露在安全性弱點中。召回這些設備進行更新也將成本高昂，並且可能會因停機而對消費者造成重大干擾。

空中 (OTA) 更新可讓您更新裝置韌體，而無需昂貴的召回或技術人員拜訪。此方法增加以下優點：

- 安全性-快速回應在現場部署裝置後發現的安全漏洞和軟體錯誤的能力。
- 創新-產品可以隨著新功能的開發頻繁更新，推動創新週期。與傳統的更新方法相比，更新可以快速生效，並將停機時間降至最低。
- 成本-與傳統用於更新這些設備的方法相比，OTA 更新可以顯著降低維護成本。

提供 OTA 功能需要以下設計考量：

- 安全通訊-更新必須使用加密的通訊管道，以防止下載在傳輸過程中遭到竄改。
- 復原-更新可能會因為間歇性的網路連線或收到無效的更新而失敗。在這些情況下，設備需要能夠恢復到穩定狀態並避免變磚。
- 作者驗證-必須驗證更新來自受信任的來源，以及其他驗證，例如檢查版本和相容性。

如需設定 FreeRTOS 的 OTA 更新新新新新新新新新新新新新新新新新新新新空中免費更新

AWS IoT空中 (OTA) 圖書館

AWS IoTOTA 程式庫可讓您管理新可用更新的通知、下載並執行韌體更新的加密驗證。使用 over-the-air （OTA）客戶端庫，您可以從邏輯上將固件更新機制與設備上運行的應用程序分開。 over-the-air （OTA）客戶端庫可以與應用程序共享網絡連接，從而在資源受限設備中節省內存。此外， over-the-air (OTA) 用戶端程式庫可讓您定義應用程式特定的邏輯，以測試、認可或復原韌體更新。此程式庫支援不同的應用程式通訊協定，例如訊息佇列遙測傳輸 (MQTT) 和超文字傳輸通訊協定 (HTTP)，並提供各種設定選項，讓您可以針對網路類型和條件進行微調。

該庫的 API 提供以下主要功能：

- 註冊通知或輪詢可用的新更新要求。
- 接收、剖析及驗證更新要求。
- 根據更新要求中的資訊下載並驗證檔案。
- 在啟動接收的更新之前執行自我測試，以確保更新的功能有效性。
- 更新新新聞

此程式庫使用AWS服務來管理各種雲端相關功能，例如傳送韌體更新、監控跨多個區域的大量裝置、減少故障部署的爆炸半徑，以及驗證更新的安全性。此程式庫可與任何 MQTT 或 HTTP 程式庫搭配使用。

此程式庫的示範會示範在 FreeRTOS 裝置上使用 CoremQtt 程式庫和AWS服務的完整 over-the-air 更新。

功能

以下是完整的 OTA 代理程式界面：

`OTA_Init`

通過啟動系統中的 OTA 代理（「OTA 任務」）初始化 OTA 引擎。只有一個 OTA 代理可能存在。

`OTA_Shutdown`

向 OTA 代理商發出信號以關閉。OTA 代理程式將選擇性地取消訂閱所有 MQTT 作業通知主題，停止進行中的 OTA 作業（如果有），並清除所有資源。

`OTA_GetState`

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 代理程式作業。

OTA_Resume

繼續 OTA 代理程式作業。

OTA_SignalEvent

向 OTA 代理程式工作發出事件訊號。

OTA_EventProcessingTask

OTA 代理事件處理循環。

OTA_GetStatistics

獲取 OTA 消息數據包的統計信息，其中包括接收，排隊，處理和丟棄的數據包數量。

OTA_Err_strerror

OTA 錯誤的錯誤代碼到字符串轉換。

OTA_JobParse_strerror

將 OTA Job 解析錯誤代碼轉換為字符串。

OTA_PalStatus_strerror

OTA PAL 狀態的狀態代碼到字符串轉換。

OTA_OsStatus_strerror

OTA OS 狀態的狀態代碼到字符串轉換。

API 參考

如需詳細資訊，請參閱 AWS IoTOver-the-air 更新：函數。

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. Connect 至AWS IoT CoremQtt 代理程式。如需詳細資訊，請參閱CoremQTT 代理器程式庫。

2. 通過調用（包括緩衝區`OTA_Init`，所需的 OTA 接口，事物名稱和應用程序回調）初始化 OTA 代理。回撥會實作應用程式限定邏輯，在完成 OTA 更新任務後執行。
3. OTA 更新完成時，FreeRTOS 會使用下列事件之一呼叫作業完成回呼：`accepted`、`rejected`、或`self test`。
4. 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。
5. 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

有關將 OTA 功能移植到您的平台的信息，請參閱 FreeRTOS 移植指南中的移植 OTA 庫。

記憶體使用

AWS IoTOTA 的代碼大小（使用 GCC 為 ARM 皮質-M 生成的示例）		
檔案	使用-O1 最佳化功能	使用-Os 優化
大田 .c	8.3K	7.5 公里
網路介面. c	0.1K	0.1K
基地 64.c	0.6 公里	0.6 公里
俄羅斯電子股份有限公司	2.4	2.2 千
俄羅斯聯邦	0.8 萬	0.6 公里
網站	0.3K	0.3K
估計總數	12.5 公里	11.3K

Corep11 程式庫

> **Note**
>
> 此頁面上的內容可能不是up-to-date。如需最新的更新，請參閱 FreerTos.org 程式庫頁面。

概要

公開金鑰加密標準 #11 定義了一個獨立於平台的 API 來管理和使用密碼編譯權杖。PKCS #11 是指由標準和標準本身定義的 API。PKCS #11 密碼編譯 API 會抽象化金鑰儲存、取得/設定密碼編譯物件的屬性，以及工作階段語意。它被廣泛用於操作常見的密碼編譯對象，這很重要，因為它指定的功能允許應用程序軟件使用，創建，修改和刪除加密對象，而不會將這些對象暴露到應用程序的內存。例如，FreeRTOS AWS 參考整合使用 PKCS #11 API 的一小部分來存取建立由傳輸層安全性 (TLS) 通訊協定驗證和保護的網路連線所需的秘密 (私密) 金鑰，而不需要應用程式「看到」金鑰。

CorepkCS11 程式庫包含 PKCS #11 介面 (API) 的以軟體為基礎的模擬實作，該實作使用 MBed TLS 提供的加密功能。使用軟件模擬可以實現快速開發和靈活性，但是預計您將用特定於生產設備中使用的安全密鑰存儲的實現替換模擬。一般而言，安全加密處理器的廠商，例如信任平台模組 (TPM)、硬體安全模組 (HSM)、安全元件或任何其他類型的安全硬體隔離區，會隨硬體散佈 PKCS #11 實作。因此，CorepkCS11 軟體僅模擬程式庫的目的在於提供非硬體特定的 PKCS #11 實作，允許在生產裝置中切換到密碼處理器特定 PKCS #11 實作之前，進行快速原型製作和開發。

僅實作 PKCS #11 標準的子集，重點放在涉及非對稱金鑰、隨機數產生和雜湊的作業上。目標使用案例包括 TLS 驗證的憑證和金鑰管理，以及小型嵌入式裝置上的程式碼簽章驗證。請參閱 FreeRTOS 原始程式碼儲存庫中的檔案 `pkcs11.h` (從標準主體 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 用戶端驗證。`SOCKETS_Connect`PKCS #11 API 呼叫也是由我們的一次性開發人員佈建工作流程進行，以匯入 TLS 用戶端憑證和用於驗證的私密金鑰至 AWS IoT MQTT 代理程式。這兩個使用案例 (佈建和 TLS 用戶端驗證) 只需要實作 PKCS #11 介面標準的一小部分。

功能

使用 PKCS #11 的下列子集。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。有關功能的詳細說明，請參閱標準委員會提供的 PKCS #11 文檔。

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。以下說明解釋如何建立基於 P-256 用戶端憑證的 AWS IoT 物件。

請確定您使用的以下版本 (或更新版本) 的 AWS CLI 及 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指令來配置AWS CLI. 若要取得更多資訊，請參閱《使用指南》aws configure中的AWS Command Line Interface〈快速配置〉。

若要根據 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給 AWS IoT。

   ```
   aws iot create-certificate-from-csr  \
     --certificate-signing-request file://thing-name.req --set-as-active  \
     --certificate-pem-outfile thing-name.crt
   ```

5. 將憑證 (由先前命令的 ARN 輸出參考) 連接到物件。

   ```
   aws iot attach-thing-principal --thing-name thing-name \
     --principal "arn:aws:iot:us-
   east-1:123456789012:cert/
   86e41339a6d1bbc67abf31faf455092cdebf8f21ffbc67c4d238d1326c7de729"
   ```

6. 建立政策（這項政策太寬鬆了。 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。如果您不想針對 Greengrass 連線和探索執行 FreeRTOS 示範，可以省略此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

如需將 CorepkCS11 程式庫移植到您的平台的相關資訊，請參閱《免費伺服器移植指南》中的「移植 CorepkCS 11 程式庫」。

記憶體使用

CorepkCS11 的代碼大小（使用 GCC 為 ARM 皮質-M 生成的示例）		
檔案	使用-O1 最佳化	使用-Os 優化
核心 _pkcs11.c	0.8 萬	0.8 萬
核心公用程式 c	0.5 公里	0.3K
核心 _pkcs11 _ 中英尺	8.9K	7.5 公里
估計總數	10.2K	8.6 千

Secure Sockets 程式庫

> **Important**
>
> 該庫託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。我們建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱. 亞馬遜自由 Github 存儲庫遷移指南

概觀

您可以使用 FreeRTOS 安全通訊端程式庫來建立安全通訊的嵌入式應用程式。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 安全通訊端程式庫是以柏克萊通訊端介面為基礎，並提供 TLS 通訊協定的額外安全通訊選項。如需 FreeRTOS 安全通訊端程式庫與伯克利通訊端介面之間差異的相關資訊，請參閱安全通訊端 API 參考`SOCKETS_SetSockOpt`中的。

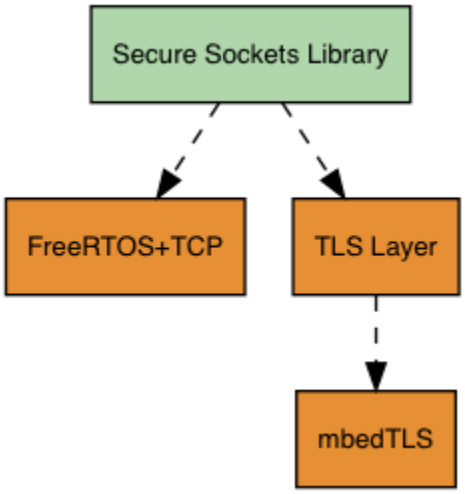
> **Note**
>
> 目前，FreeRTOS 安全通訊端只支援用戶端 API，以及伺服器端 `Bind` API 的輕量型 IP (LWIP) 實作。

相依性和要求

FreeRTOS 安全通訊端程式庫取決於 TCP/IP 堆疊和 TLS 實作。FreeRTOS 的連接埠以下列三種方式之一符合這些相依性：

- 同時自訂實作 TCP/IP 及 TLS
- TCP/IP 的自訂實作，以及含有 mBedTLS 的自 FreeRTOS TLS 層
- 免費伺服器 + TCP 和帶有 MDTLS 的自由傳輸層

下面的依賴關係圖顯示了 FreeRTOS 安全套接字庫中包含的參考實現。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。如需有關 FreeRTOS TLS 層的詳細資訊，請參閱。Transport Layer Security



功能

FreeRTOS 安全通訊端程式庫功能包括：

- 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
- 電子 asy-to-enable TLS

故障診斷

錯誤代碼

FreeRTOS 安全通訊端程式庫傳回的錯誤碼為負值。如需每個錯誤碼的詳細資訊，請參閱 Secure Sockets API 參考中的 Secure Sockets 錯誤碼。

> **Note**
>
> 如果 FreeRTOS 安全通訊端 API 傳回錯誤碼，則視 FreeRTOS 安全通訊端程式庫而定，會傳回錯誤碼。通訊協定圖書館 `AWS_IOT_MQTT_SEND_ERROR`

開發人員支援

FreeRTOS 安全通訊端程式庫包含兩個用於處理 IP 位址的輔助巨集：

`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 安全通訊端程式庫僅支援 TCP 通訊端。不支援 UDP 通訊端。

除了伺服器端 API 的輕量型 IP (LWIP) 實作外，FreeRTOS 安全通訊端程式庫不支援伺服器 API。`Bind`支援用戶端 API。

初始化

若要使用 FreeRTOS 安全通訊端程式庫，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 API 參考資料，請參閱安全通訊端 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 的安全通訊端取決於 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
- Corep11 程式庫
- Transport Layer Security

如需有關移植的詳細資訊，請參閱 FreeRTOS 移植指南中的移植安全通訊端程式庫。

AWS IoT 裝置影子程式庫

> **Note**
>
> 此頁面上的內容可能不是 up-to-date。如需最新的更新，請參閱 FreerTos.org 程式庫頁面。

簡介

您可以使用AWS IoT Device Shadow 程式庫來儲存和擷取每個已註冊裝置的目前狀態 (陰影)。裝置的陰影是裝置的持續性虛擬表示，即使裝置離線，您也可以在 Web 應用程式中與之互動。裝置狀態會擷取為其在 JSON 文件中的陰影。您可以透過 MQTT 或 HTTP 傳送命令至AWS IoT Device Shadow 服務，以查詢最新的已知裝置狀態或變更狀態。每個裝置的陰影都會由對應物件的名稱、特定裝置或AWS雲端上邏輯實體的表示法來唯一識別。如需詳細資訊，請參閱使用AWS IoT。有關陰影的更多詳細信息可以在AWS IoT文檔中找到。

除了標準 C 庫之外，AWS IoTDevice Shadow 庫沒有依賴於其他庫。它也沒有任何平台依賴關係，例如線程或同步。它可以與任何 MQTT 庫和任何 JSON 庫一起使用。

該庫可以自由使用，並根據 MIT 開源許可證進行分發。

AWS IoTDevice Shadow 的代碼大小（用於 ARM Cortex-M 的 GCC 生成的示例）		
檔案	使用-O1 最佳化	使用-Os 優化
陰影 .c	1.2 千	0.9K
估計總數	1.2 千	0.9K

AWS IoT工作庫

> **Note**
>
> 此頁面上的內容可能不是 up-to-date。請參閱圖書館頁面以獲取最新更新。

簡介

AWS IoTJobs 是一種服務，通知一個或多個連接的設備有待處理工作。您可以使用任務來管理裝置群組、更新裝置上的韌體和安全憑證，或執行系統管理工作，例如重新啟動裝置和執行診斷。如需詳細資訊，請參閱工作在AWS IoT開發者指南。與AWS IoT工作服務使用MQTT，輕量級發布-訂閱協議。此庫提供了一個 API 來組成和識別 MQTT 主題字符串由AWS IoT工作服務。

該AWS IoT作業庫是用 C 編寫的，設計符合ISO和米什拉 C: 2012 年。除了標準 C 庫之外，該庫沒有依賴於任何其他庫。它可以與任何 MQTT 庫和任何 JSON 庫一起使用。圖書館有樣張顯示安全的內存使用和沒有堆分配，使其適用於 IoT 微控制器，但也完全可移植到其他平台。

該庫可以自由使用，並在MIT 開放原始碼授權。

的代碼大小AWS IoT工作（使用 GCC 為 ARM 皮質-M 生成的示例）		
檔案	使用-O1 最佳化功能	使用-Os 優化
工作 .c	1.9K	1.6 千
估計總數	1.9K	1.6 千

Transport Layer Security

> **Important**
>
> 該庫託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。建議您在建立新的專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱亞馬遜自由 Github 存儲庫遷移指南.

FreeRTOS 傳輸層安全性 (TLS) 介面是一種精簡的選用包裝函式，用來從通訊協定堆疊中的安全通訊端層 (SSL) 介面擷取加密實作詳細資訊。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`iot_tls.h` FreeRTOS 的來源程式碼儲存庫。

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。

如需有關移植 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**
>
> 該庫託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。我們建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱. 亞馬遜自由 Github 存儲庫遷移指南

概觀

FreeRTOS Wi-Fi 程式庫將連接埠特定的 Wi-Fi 實作摘要為一個通用的 API，簡化應用程式開發和移植所有符合免費使用 Wi-Fi 功能的主機板。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 函式庫需要免費使用者 + TCP 核心。

功能

Wi-Fi 程式庫包含下列功能：

- Support WEP、WPA、WPA2 和 WPA3 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 程式庫不會在 AP 模式下提供 Wi-Fi 佈建。您必須提供額外的功能，包括 DHCP 和 HTTP 伺服器功能，才能完全支援 AP 模式。

P2P 模式

將您的裝置設定為 P2P 模式，以允許多個裝置無需存取點即可彼此直接連接。

安全

無線網路應用程式介面支援 WEP、WPA、WPA2 和 WPA3 安全性類型。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 媒體櫃的詳細資訊，請參閱《FreeRTOS 移植指南》中的移植 Wi-Fi 媒體櫃。

FreeRTOS 示

FreeRTOS 在主要 FreeRTOS 目錄下的demos資料夾中包含一些示範應用程式。FreeRTOS 可執行的所有範例都會顯示在資common料夾的下demos。在資料夾下方，每個符合免費使用者資格的平台都有一個資demos料夾。

嘗試使用示範應用程式前，我們建議您先完成 FreeRTOS 入中的教學。它將示範如何設定和執行 Coremtt 代理程式示範。

執行 FreeRTOS 示範

下列主題將示範如何設定和執行 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 演示
- AWS IoT GreengrassV1 探索示範應用程
- AWS IoT Greengrass V2
- 核心 HTTP 演示
- AWS IoT工作庫演示
- 演示
- Over-the-air 更新演示應用
- Secure Sockets echo 用戶端示範
- AWS IoT 裝置影子示範應用程式

該DEMO_RUNNER_RunDemos函數位於*freertos*/demos/demo_runner/iot_demo_runner.c文件中，初始化單個演示應用程序運行的分離線程。依預設，DEMO_RUNNER_RunDemos只會呼叫並啟動 CoremQtt 代理程式示範。根據您下載 FreeRTOS 時選取的組態，以及您下載 FreeRTOS 的位置，其他範例執行程式功能可能會依預設啟動。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能一起運作。視組合而定，由於記憶體限制，軟體可能無法在選取的目標上執行。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**
>
> 此演示託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。我們建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱. 亞馬遜自由 Github 存儲庫遷移指南

概觀

FreeRTOS 低功耗藍牙包括三個示範應用程式：

- 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

此應用程序演示如何使用 FreeRTOS 藍牙低功耗中間件 API 來創建一個簡單的 GATT 服務器。

> **Note**
>
> 若要設定並執行 FreeRTOS 示範，請遵循中的步驟。FreeRTOS 入

必要條件

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 適用於免費藍牙裝置的 iOS 開發套件或 適用於免費藍牙設備的安卓 SDK。

為 FreeRTOS 低功耗藍牙設置 AWS IoT 和 Amazon Cognito

要將您的設備連接到 AWS IoT 跨 MQTT，您需要設置 AWS IoT 和 Amazon Cognito。

若要設定 AWS IoT

1. 在 https://aws.amazon.com/ 上設置一個 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)。由於行動開發套件使用 Amazon Cognito 進行裝置身份驗證，因此您不需要為使用藍牙低功耗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 其餘 API 端點。

如需有關設定的詳細資訊，請參閱入門 AWS IoT。

若要建立 Amazon Cognito 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。

若要建立 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`。

如需有關設定 Amazon Cognito 的詳細資訊，請參閱 Amazon Cognito 入門。

建立 IAM 政策並將其附加至已驗證身分

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)。

保持您 AWS IoT 和 Amazon Cognito 信息在手。您需要端點和 ID，才能透過 AWS 雲端驗證您的行動應用程式。

為低功耗藍牙設定 FreeRTOS 環境

若要設定您的環境，您需要在您的微控制器低功耗藍牙程式庫上下載 FreeRTOS，並在您的行動裝置上下載並設定 FreeRTOS 藍牙裝置的行動 SDK。

使用 FreeRTOS 低功耗藍牙設定微控制器的環境

1. 從下載或克隆 FreeRTOS 軟件。GitHub如需說明，請參閱 README.md 檔案。
2. 在微控制器上設定 FreeRTOS。

   如需在 FreeRTOS 合格微控制器上開始使用 FreeRTOS 的相關資訊，請參閱 FreeRTOS 入門中適用的主機板指南。

   > **Note**
   >
   > 您可以在任何支援 FreeRTOS 的藍牙低功耗微控制器上執行示範，以及已移植的 FreeRTOS 低功耗藍牙程式庫。FreeRTOS 透過低功耗藍牙執行的 MQTT 示範專案目前已完全移植到下列啟用藍牙低功耗的裝置：

- 濃縮咖啡 ESP32-DevKit C 和歐洲普羅弗羅弗套件
- Nordic nRF52840-DK

常見元件

FreeRTOS 示範應用程式有兩個常見的元件：

- 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。它位於您的 FreeRTOS 目錄中，位於。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。

免費藍牙低功耗移動 SDK 演示應用程序

該 FreeRTOS 藍牙低功耗移動 SDK 演示應用程序位 GitHub 於下 FreeRTOS 藍牙設備的 Android SDK amazon-freertos-ble-android-sdk/app 和下的 iOS SDK FreeRTOS 藍牙設備。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 適用於免費藍牙裝置的 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. 在您所在地區建立 AWS IoT 政策 (如果您尚未建立)。

> **Note**
>
> 此政策與針對 Amazon Cognito 驗證身分建立的 IAM 政策不同。

   a. 開啟 AWS IoT 主控台。
   b. 在瀏覽窗格中，選擇 [安全]，選擇 [原則]，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 帳戶替換為您的 AWS 地區和## 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 地區。
   - iotPolicyName：您的 AWS IoT 策略名稱。
   - 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 身份集區 ID。
   - Region：您的 AWS 地區。

   在 CognitoUserPool 下，重新定義以下變數：

   - PoolId：您的 Amazon Cognito 戶集區 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 地區。

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 適用於免費藍牙設備的安卓 SDK。
2. 在您所在地區建立 AWS IoT 政策 (如果您尚未建立)。

> **Note**
>
> 此政策與針對 Amazon Cognito 驗證身分建立的 IAM 政策不同。

   a. 開啟 AWS IoT 主控台。

   b. 在瀏覽窗格中，選擇 [安全]，選擇 [原則]，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 帳戶替換為您的 AWS 地區和## 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. 選擇建立。

3. 開啟DemoConstants以下變數：https://github.com/aws/amazon-freertos-ble-android-sdk/blob/master/app/src/main/java/software/amazon/freertos/demo/

   - `AWS_IOT_POLICY_NAME`：您的 AWS IoT 策略名稱。
   - `AWS_IOT_REGION`：您的 AWS 地區。

4. 打開以下位置。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 身份集區 ID。
   - `Region`：您的 AWS 地區。

   在 `CognitoUserPool` 下，重新定義以下變數：

   - `PoolId`：您的 Amazon Cognito 戶集區 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 地區。

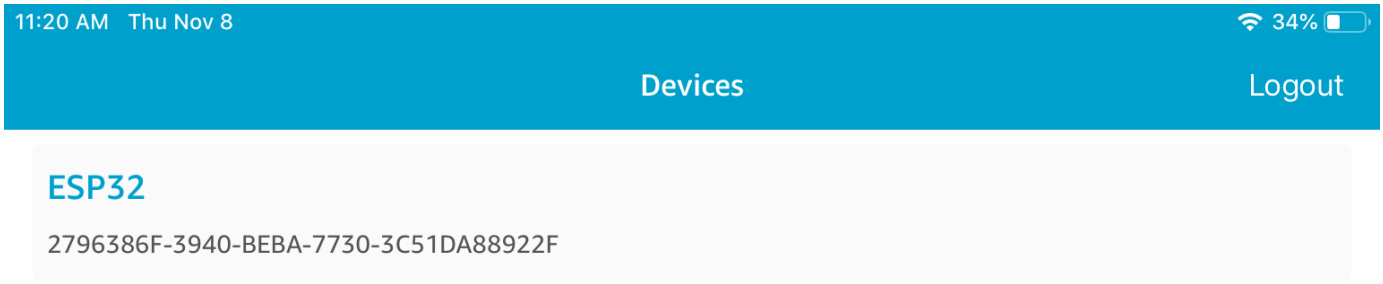
透過低功耗藍牙探索並建立微型控制器的安全連線

1. 為了安全地配對您的微控制器和移動設備（步驟 6），您需要一個具有輸入和輸出功能（例如 TeraTerm）的串行終端仿真器。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。
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：

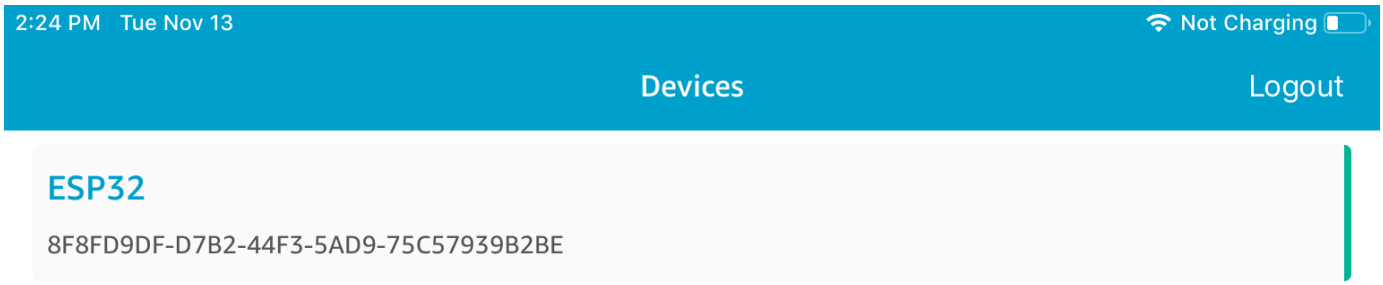
   ```
   $ ./gradlew installDebug
   ```

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。

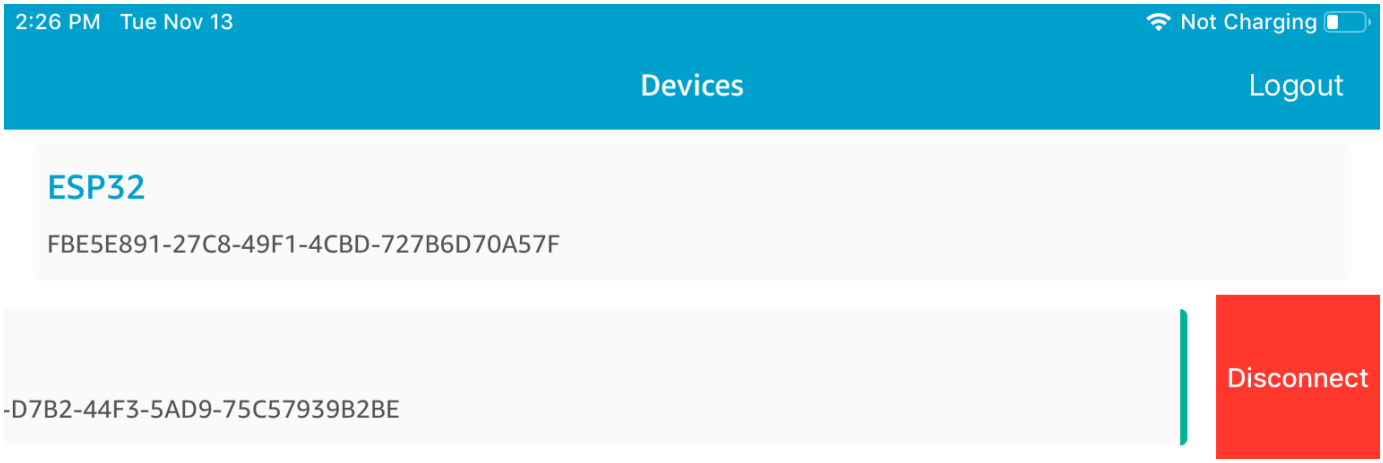


> **Note**
>
> 所有具有 FreeRTOS 的裝置和位於範圍內的裝置資訊服務 (*freertos*/.../device_information) 都會出現在清單中。

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。

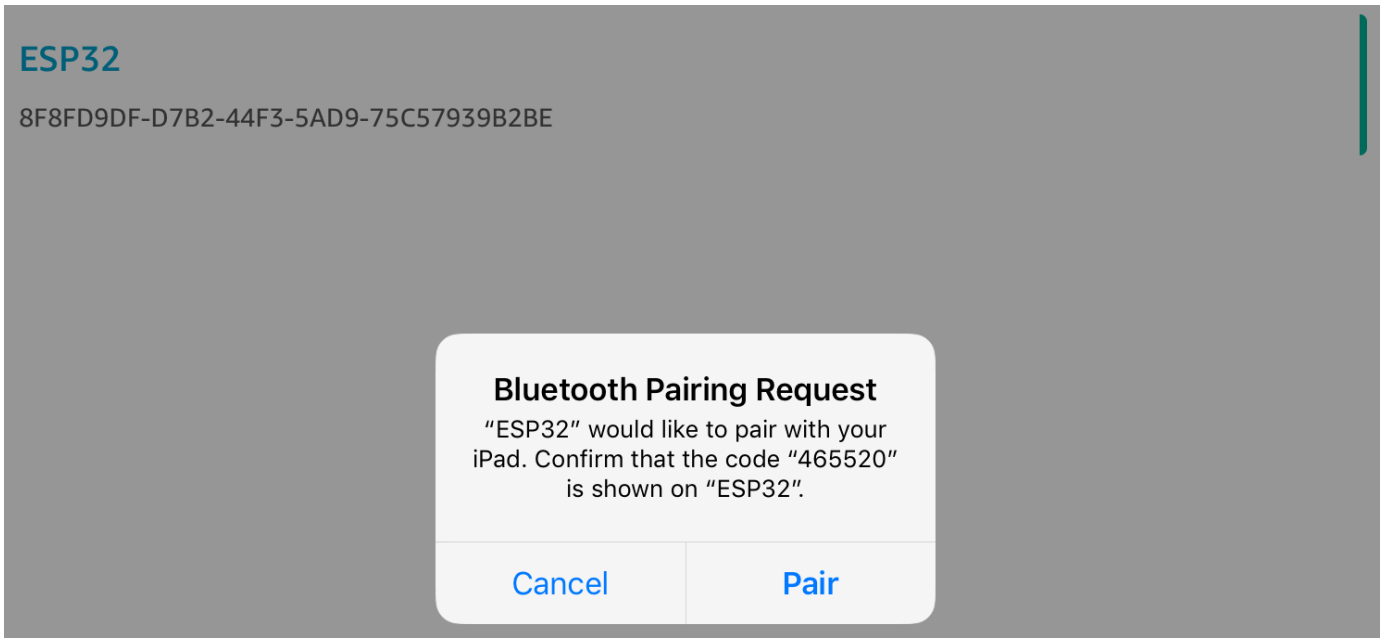


您可以通過將線向左拖動來斷開與微控制器的連接。



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。

```
24 261107 [Btc_task]  Disconnected from BLE device. Stopping the counter update
25 261108 [Btc_task] Disconnect received for MQTT service instance 0
26 261108 [Btc_task] BLE disconnected with remote device, start advertisement
27 261108 [Btc_task] Started advertisement. Listening for a BLE Connection.
28 261289 [Btc_task] BLE Connected to remote device, connId = 0
E (2614110) BT_GATT: gatts_write_attr_perm_check - GATT_INSUF_AUTHENTICATION: MITM required
E (2614420) BT_SMP: Value for numeric comparison = 465520
29 261412 [uTask] Numeric comparison:465520
30 261412 [uTask] Press 'y' to confirm
```



如果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**
>
> 藍牙低功耗行動 SDK 示範應用程式使用 Amazon Cognito 進行使用者身份驗證。請確定您已設定 Amazon Cognito 使用者和身分集區，並且已將 IAM 政策附加到身分驗證身分。

透過低功耗藍牙執行的 MQTT

在透過藍牙低功耗的 MQTT 示範中，您的微控制器會透過 MQTT 代理將訊息發佈到 AWS 雲端。

訂閱示範 MQTT 主題

1. 登入 AWS IoT 主控台。
2. 在導覽窗格中，選擇 [測試]，然後選擇 [MQTT 測試用戶端] 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。

透過藍牙低功耗啟用示範

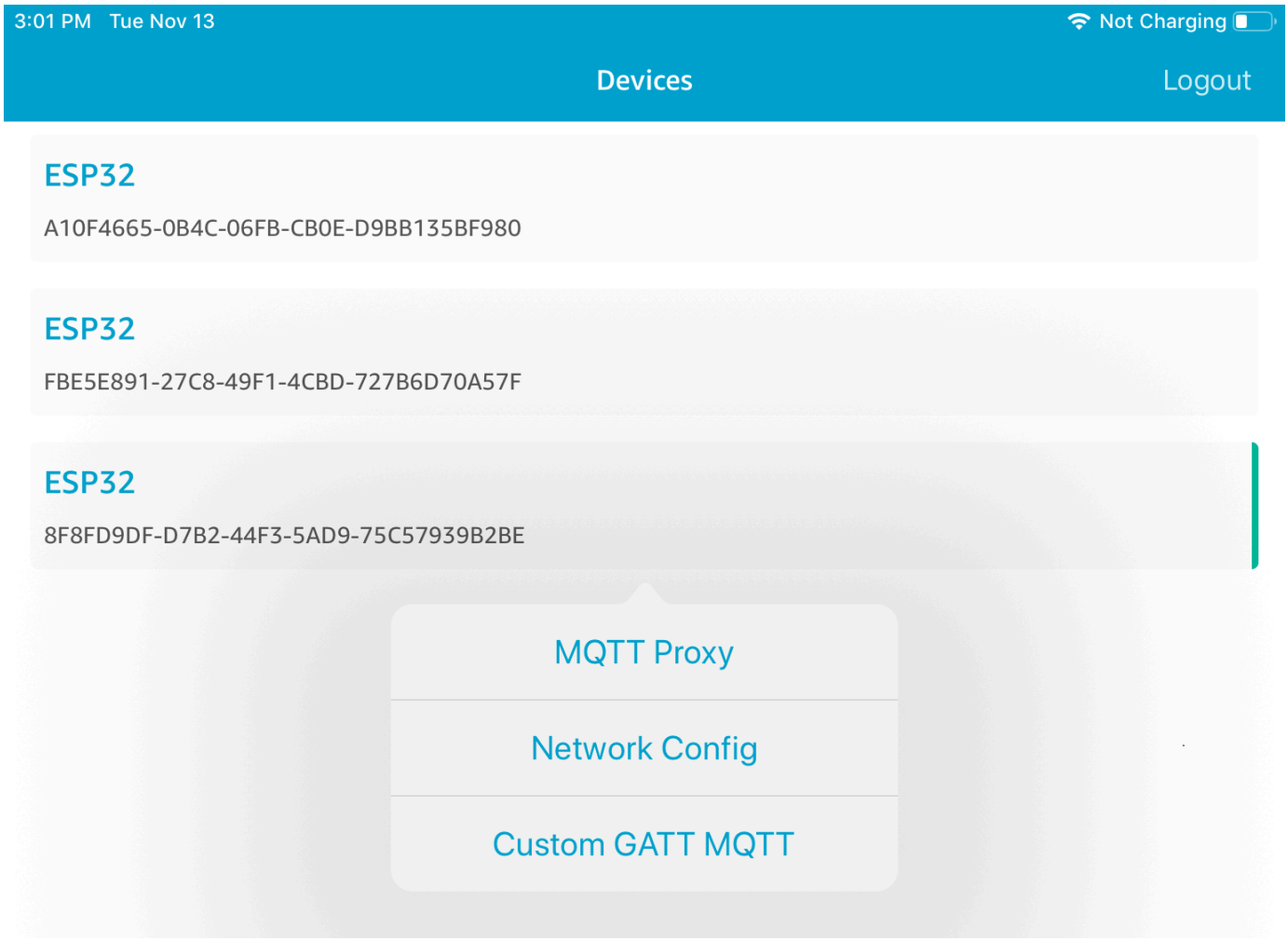
1. 開啟 `vendors/`*vendor*`/boards/`*board*`/aws_demos/config_files/aws_demo_config.h`，然後定義 `CONFIG_MQTT_BLE_TRANSPORT_DEMO_ENABLED`。
2. 開啟`demos/include/aws_clientcredential.h`並`clientcredentialMQTT_BROKER_ENDPOINT`使用 AWS IoT Broker 端點進行設定。`clientcredentialIOT_THING_NAME`使用 BLE 微控制器裝置的物件名稱進行設定。您可以從 AWS IoT 主控台取得 AWS IoT Broker 端點，方法是選擇左側導覽窗格中的 [設定]，或透過 CLI 執行命令：`aws iot describe-endpoint --endpoint-type=iot:Data-ATS`。

   > **Note**
   >
   > AWS IoT Broker 端點和物件名稱必須位於設定認知識識別和使用者集區的相同區域中。

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 免費藍牙低功耗移動 SDK 演示應用程序 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 FreeRTOS 低功耗藍牙服務，可讓您透過低功耗藍牙將 Wi-Fi 網路憑證從行動裝置安全地傳送至微控制器。您可以在 *freertos*/.../wifi_provisioning 中找到 Wi-Fi 佈建服務的原始程式碼。

> **Note**
>
> 無線網絡佈建演示目前在壓縮咖啡 ESP32-C 上支持。DevKit

啟用示範

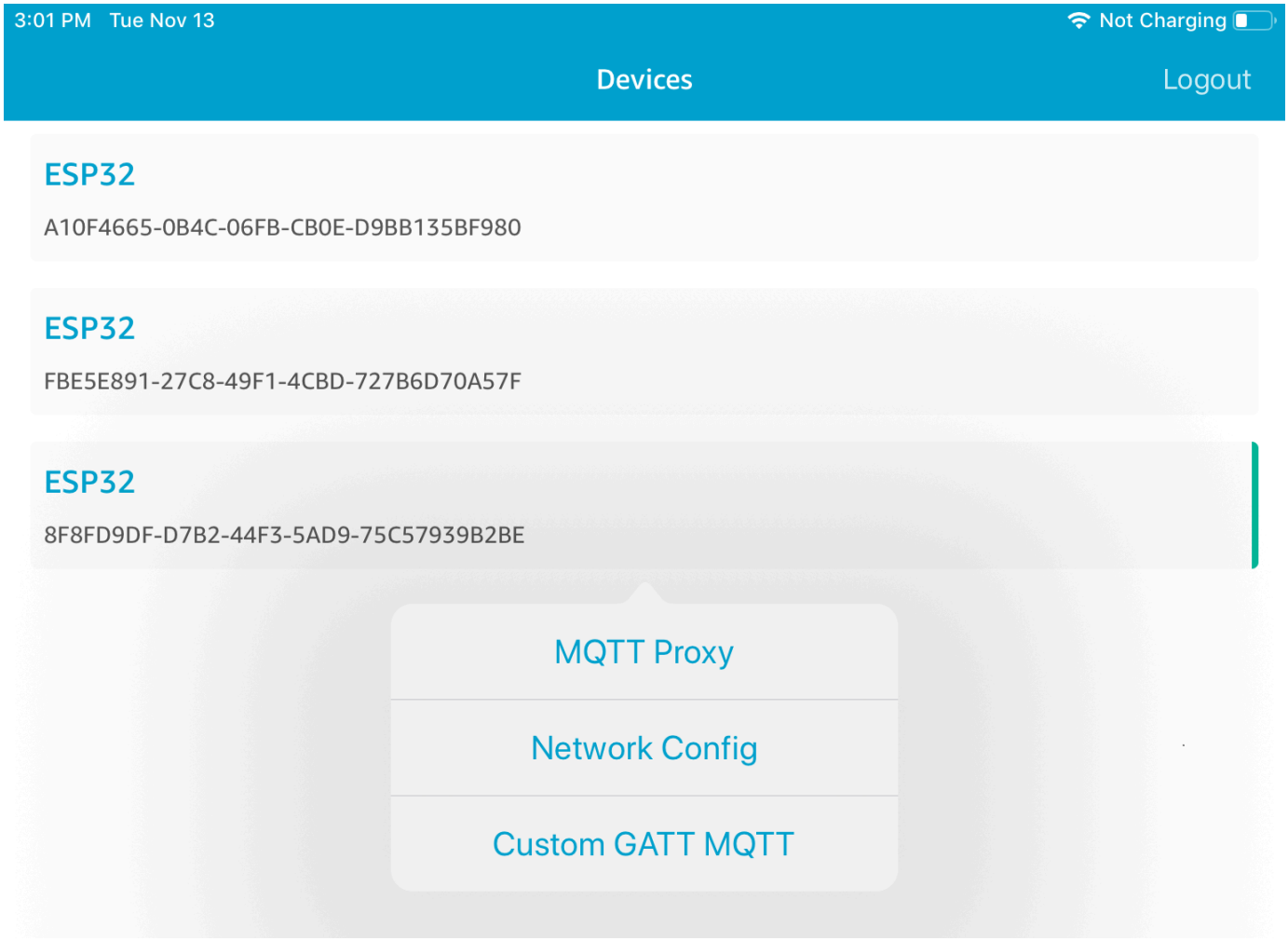
1. 啟用 Wi-Fi 佈建服務。開啟 `vendors/vendor/boards/board/aws_demos/config_files/iot_ble_config.h`，並將 `#define IOT_BLE_ENABLE_WIFI_PROVISIONING` 設定為 `1` (其中 *vendor* 是廠商名稱，而 *board* 是您用來執行示範的主機板名稱)。

   > **Note**
   >
   > 在預設情況下，Wi-Fi 佈建服務會處於停用狀態。

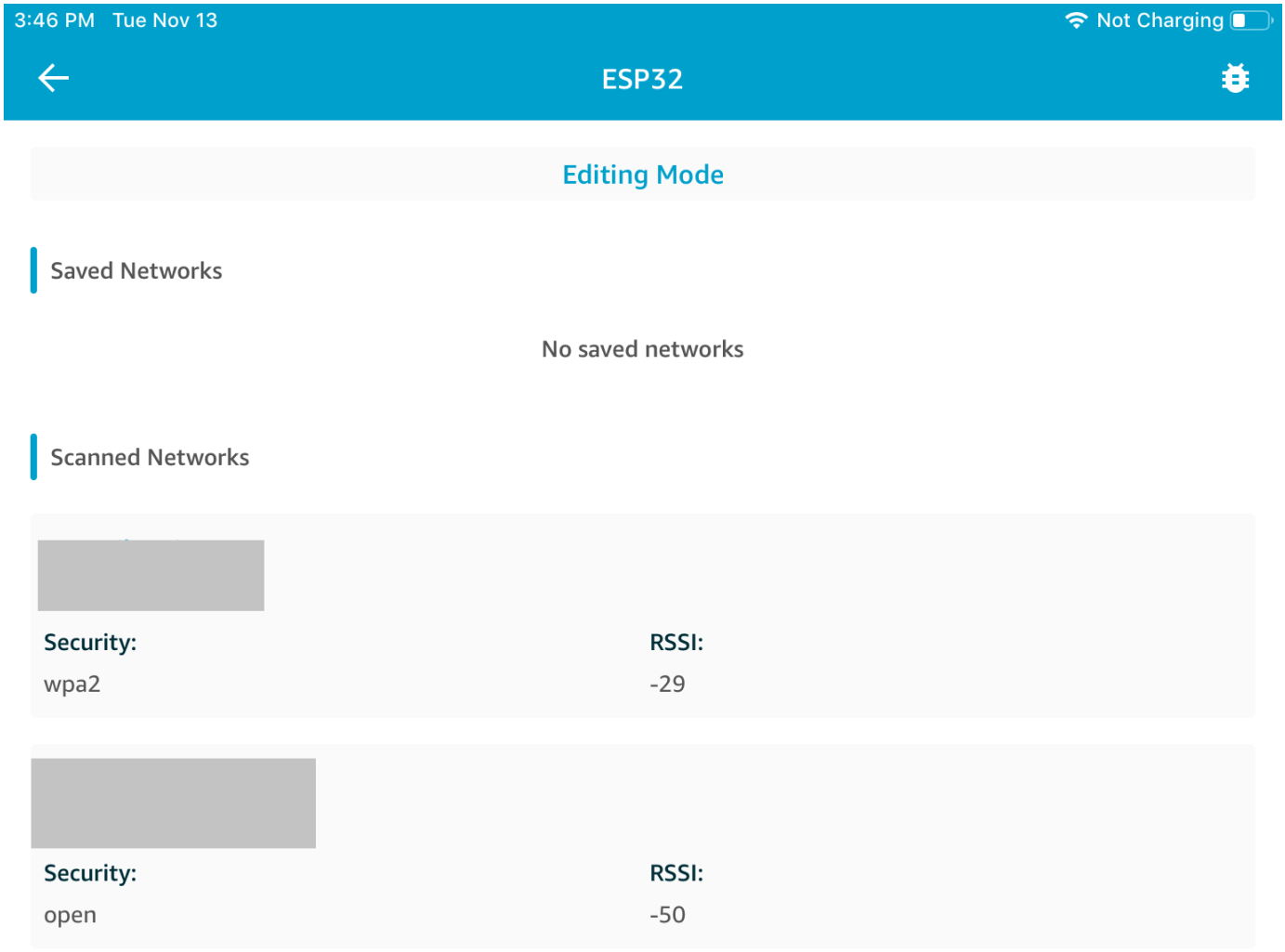
2. 設定 Network Manager，即可同時啟用低功耗藍牙和 Wi-Fi。

執行示範

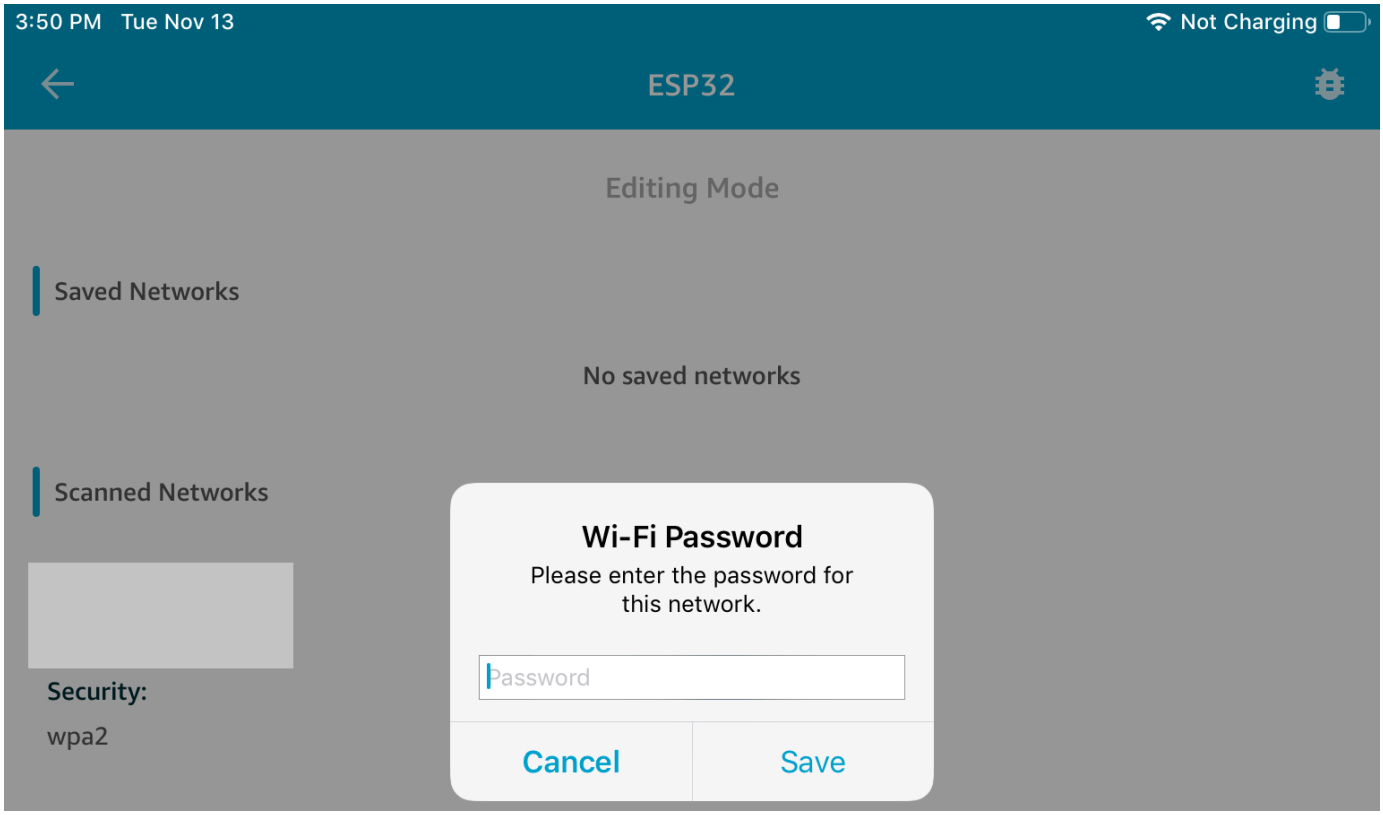
1. 在微型控制器上建置和執行示範專案。
2. 請確定您已使用微控制器和行動裝置配對。免費藍牙低功耗移動 SDK 演示應用程序
3. 從示範行動應用程式的 Devices (裝置) 清單中選擇您的微型控制器，然後選擇 Network Config (網路組態) 以開啟網路組態設定。



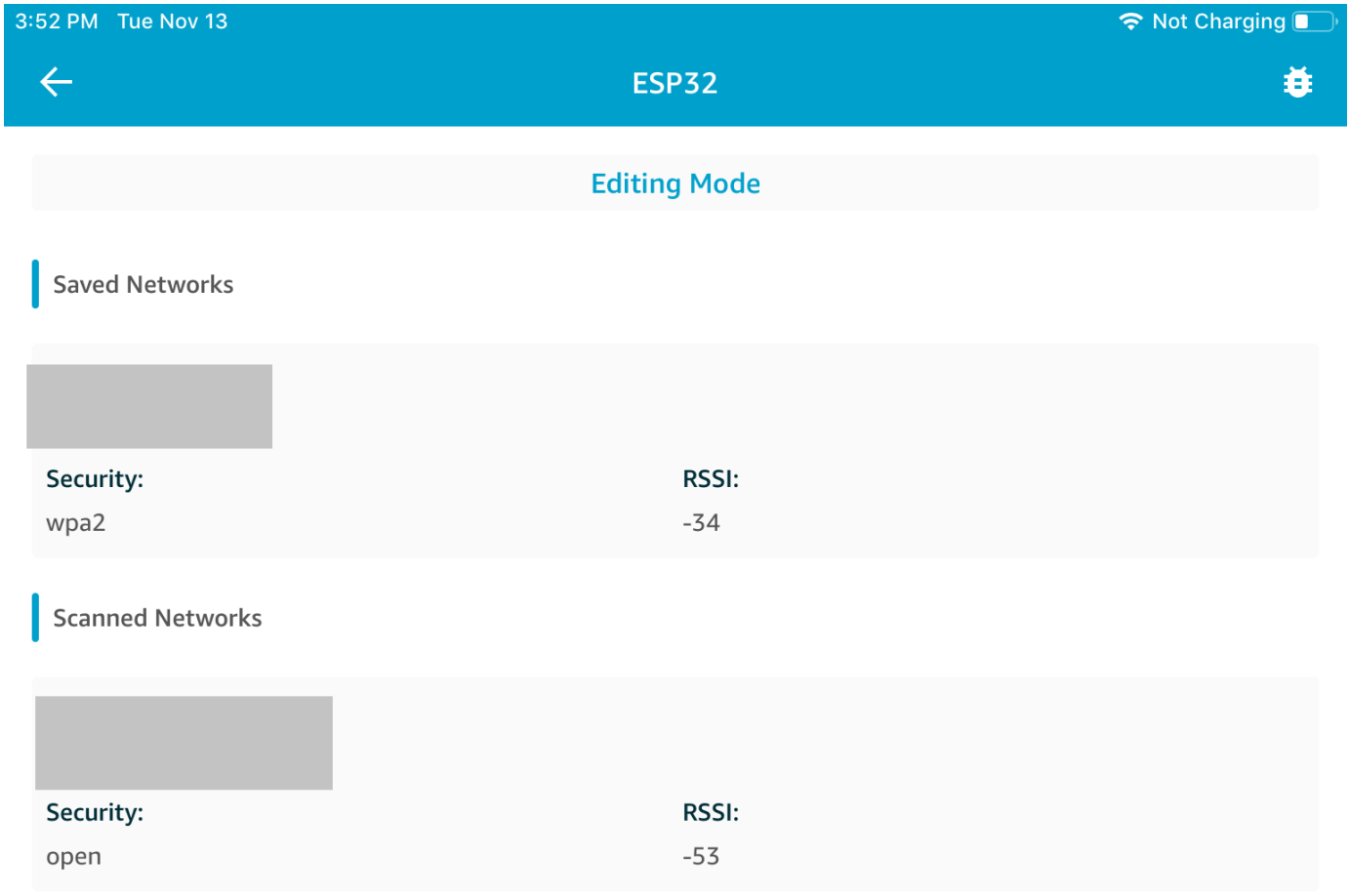
4. 選擇主機板的 Network Config (網路組態) 後，微型控制器會傳送鄰近的網路清單至行動裝置。可用的 Wi-Fi 網路會出現在 Scanned Networks (掃描的網路) 下的清單中。



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)。

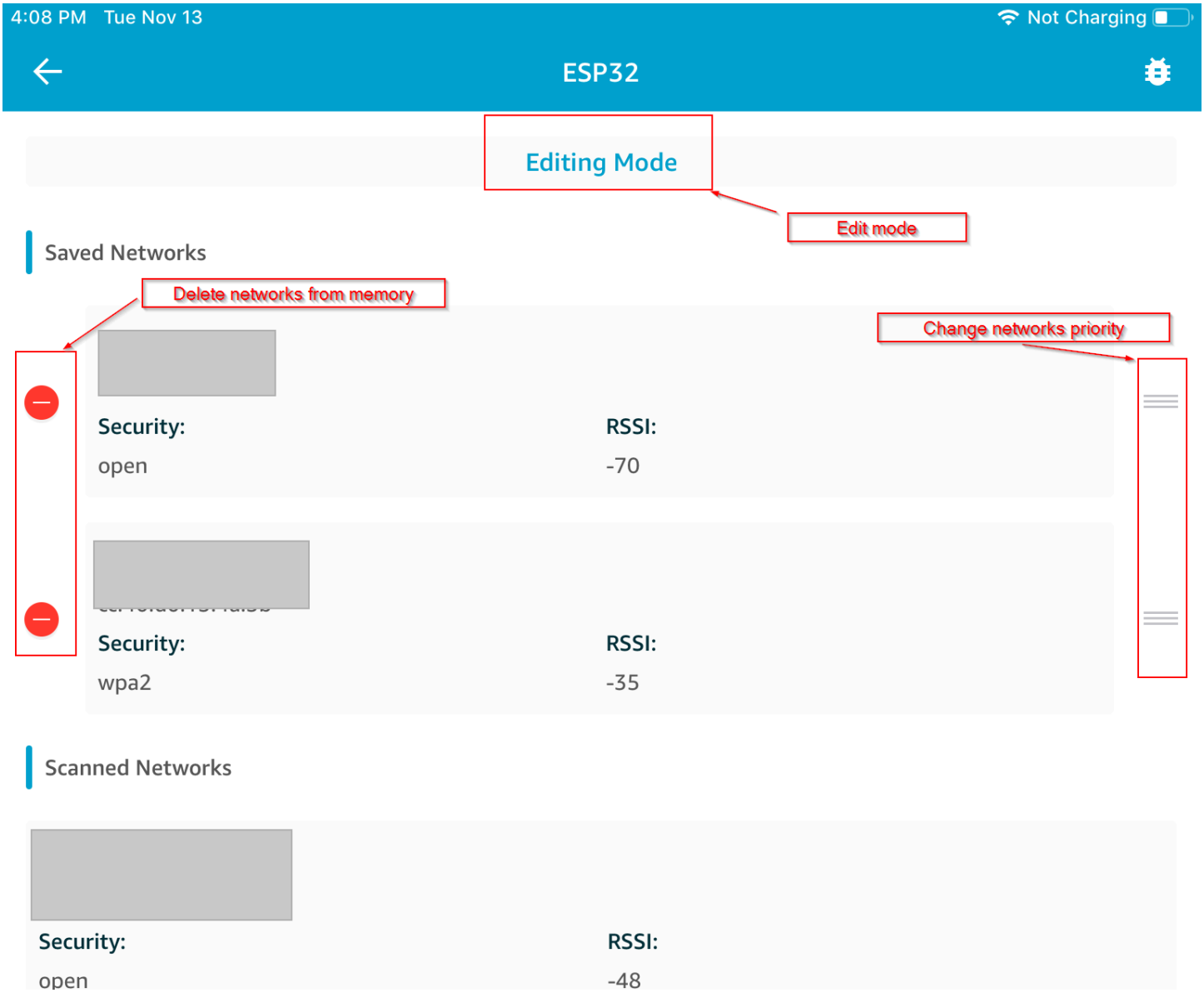


微控制器連接並保存網絡。網路會出現在 Saved Networks (儲存的網路) 下。



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 免費藍牙低功耗移動 SDK 演示應用程序。

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

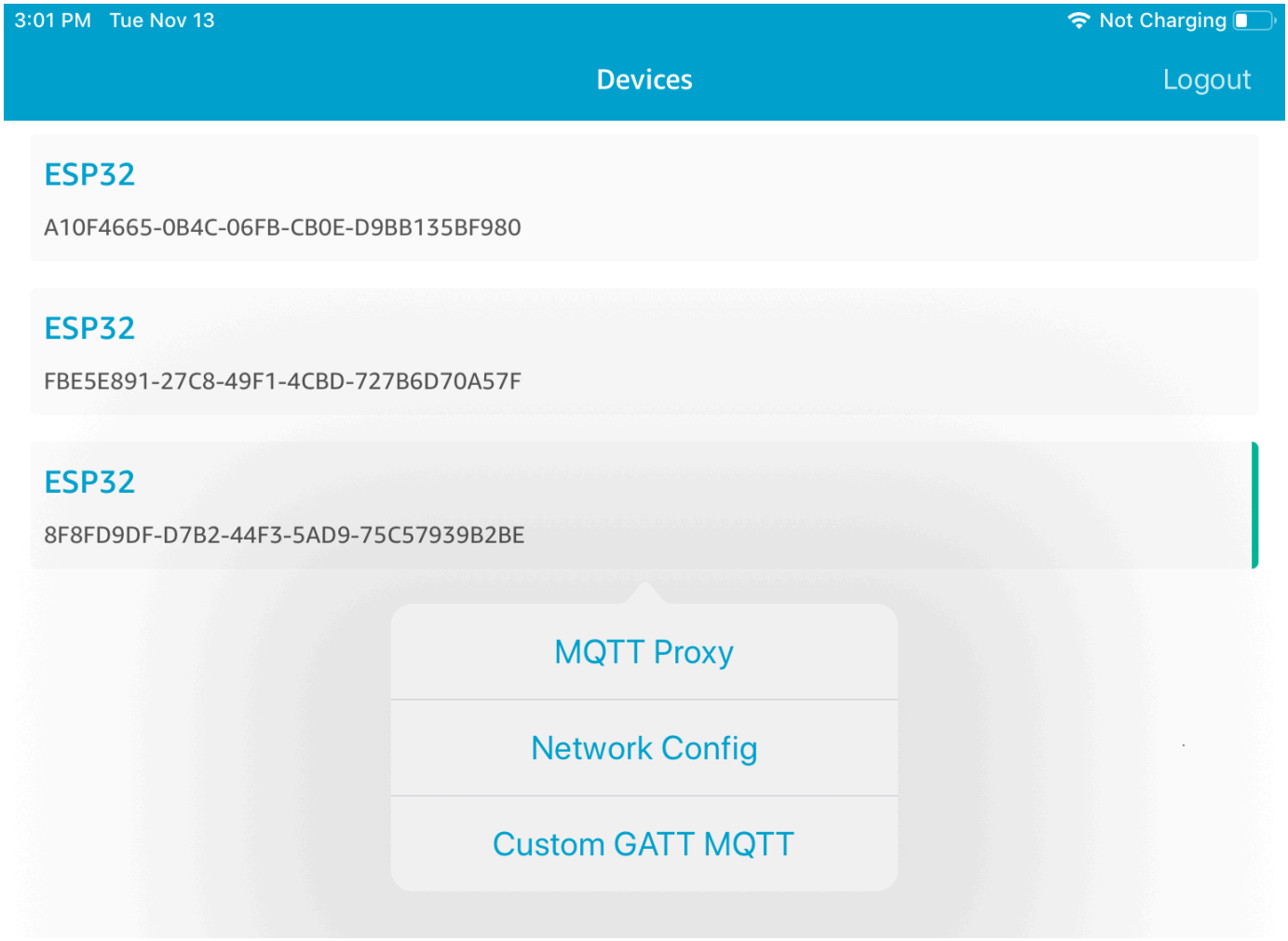
的主機板名稱) 中，將 `#define IOT_BLE_ADD_CUSTOM_SERVICES ( 1 )` 新增至 define 陳述式的清單。

> **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_CORE_MQTT_MUTUAL_AUTH_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 免費藍牙低功耗移動 SDK 演示應用程序 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

此演示託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。我們建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱. 亞馬遜自由 Github 存儲庫遷移指南

Note

在與微晶協議下，我們將從 FreeRTOS 參考整合儲存庫主要分支移除好奇心 PIC32MZEF (DM320104)，並且將不再於新版本中攜帶。微晶已發出正式通知，表示不再建議將 PIC32MZEF (DM320104) 用於新設計。PIC32MZEF 專案和原始程式碼仍可透過舊版標籤存取。微晶片建議客戶在新設計上使用好奇心 PIC32MZ-EF-2.0 開發板 (DM320209)。Pic32MZv1 平台仍然可以在 FreeRTOS 參考整合儲存庫的 v202012. 00 中找到。不過，《自由服務參考》的 v202107.00 已不再支援該平台。

此示範開機載入器實作韌體版本檢查、加密簽章驗證和應用程式自主測試。這些功能支援 FreeRTOS 的 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：

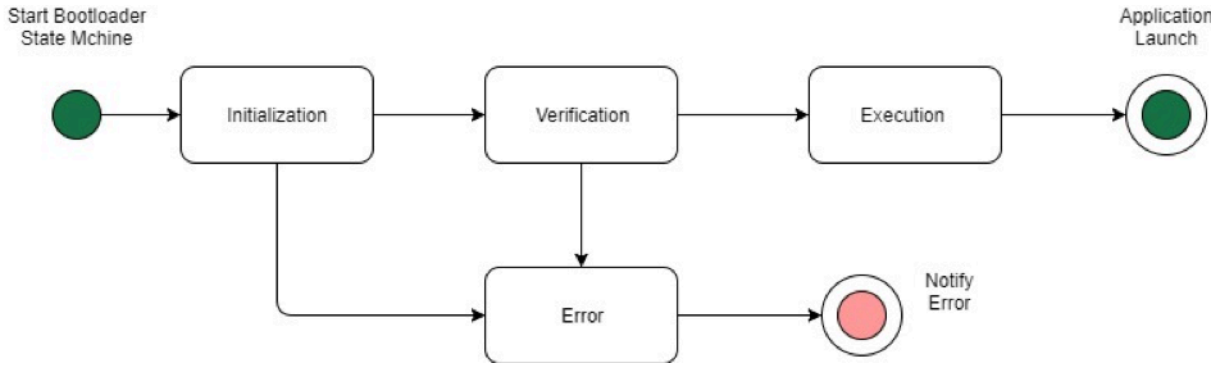
- 在裝置上維護應用程式映像，並在其間切換。
- 允許自主測試執行收到的 OTA 映像並在失敗時復原。
- 檢查 OTA 更新映像的簽章和版本。

Note

若要設定並執行 FreeRTOS 示範，請遵循中的步驟。FreeRTOS 入

開機載入器狀態

以下狀態機器中會顯示開機載入器程序。



下表描述開機載入器狀態。

開機載入器狀態	描述
初始化	開機載入器處於初始化狀態。
驗證	開機載入器正在驗證裝置上出現的映像。
執行映像	開機載入器正在啟動選取的映像。
執行預設	開機載入器正在啟動預設映像。
錯誤	開機載入器處於錯誤狀態。

在先前的圖表中，`Execute Image` 和 `Execute Default` 皆會顯示為 `Execution` 狀態。

開機載入器執行狀態

開機載入器處於 `Execution` 狀態，並準備好啟動所選的已驗證映像。如果要啟動的映像位於上層區塊中，則在執行映像之前會切換區塊，因為應用程式一律針對較低層區塊建置。

開機載入器預設執行狀態

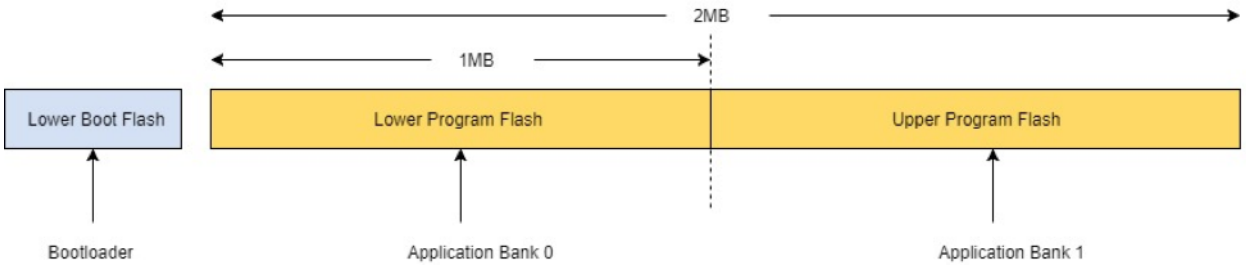
如果啟用了啟動預設映像的組態選項，則開機載入器將從預設執行地址啟動應用程式。除了偵錯外，此選項必須停用。

開機載入器錯誤狀態

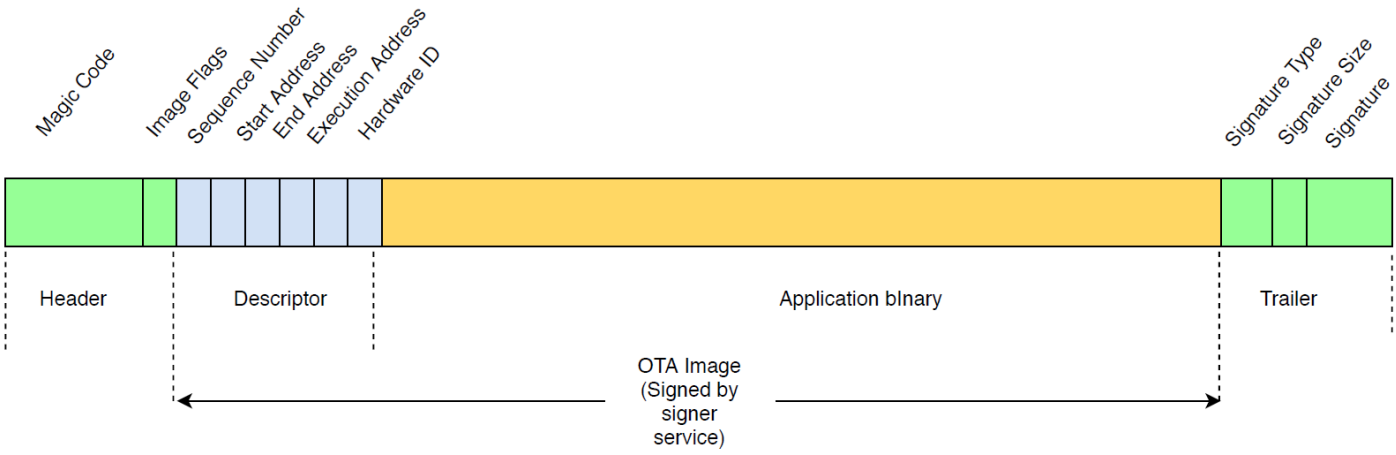
開機載入器處於錯誤狀態，且裝置上不存在有效映像。開機載入器必須通知使用者。預設實作會向主控台傳送日誌訊息，並無限期快速閃爍電路板上的 LED。

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)

元件	大小 (位元組)
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。

Status	Value	描述
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 4 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

示範開機載入程式會包含在 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 演示

> **Important**
>
> 此演示託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。我們建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱. 亞馬遜自由 Github 存儲庫遷移指南

簡介

此示範將示範如何使用 AWS IoT 裝置 Defender 程式庫連線到AWS IoT Device Defender。此示範使用 CoremQtt 程式庫，透過 TLS 建立 MQTT 連線 (相互驗證) 至 AWS IoT MQTT 代理程式和 CoreJSON 程式庫，以驗證及剖析從服務接收到的回應。 AWS IoT Device Defender 此示範將示範如何使用從裝置收集的量度建構 JSON 格式的報表，以及如何將建構的報表提交給 AWS IoT Device Defender 服務。此示範也會示範如何向 CoremQtt 程式庫註冊回呼函式，以處理來自 AWS IoT Device Defender 服務的回應，以確認傳送的報告是否已接受或拒絕。

> **Note**
>
> 若要設定並執行 FreeRTOS 示範，請遵循中的步驟。FreeRTOS 入

功能

此示範會建立單一應用程式工作，示範如何收集指標、建構 JSON 格式的裝置防禦者報告，並透過與 MQTT Broker 的安全 MQTT 連線將其提交給 AWS IoT Device Defender 服務。 AWS IoT 此示範包括標準網路指標以及自訂指標。對於自訂指標，示範包括：

- 名為 "`task_numbers`" 的度量，是 FreeRTOS 工作 ID 的清單。此度量的類型是「數字列表」。
- 名為 "`stack_high_water_mark`" 的度量，是示範應用程式工作的堆疊高浮水印。此量度的類型為「數字」。

我們如何收集網路度量取決於使用中的 TCP/IP 堆疊。對於 FreeRTO+TCP 和支援的 LWIP 組態，我們提供指標收集實作，以收集裝置的實際指標，並在報告中提交。 AWS IoT Device Defender 您可以在上找到自由式 + TCP 和 L WIP 的實作方式。 GitHub

對於使用任何其他 TCP/IP 堆疊的主機板，我們提供指標收集函數的 Stub 定義，這些函數會針對所有網路度量傳回零。*freertos*/demos/device_defender_for_aws/metrics_collector/

stub/metrics_collector.c為您的網路堆疊實作中的函數，以傳送實際指標。該文件也可以在GitHub網站上找到。

對於 ESP32，預設 LWIP 配置不使用核心鎖定，因此演示將使用存根的指標。如果您要使用參考 LWIP 度量集合實作，請在中lwiopts.h定義下列巨集：

```
#define LINK_SPEED_OF_YOUR_NETIF_IN_BPS 0
#define LWIP_TCPIP_CORE_LOCKING         1
#define LWIP_STATS                      1
#define MIB2_STATS                      1
```

以下是執行示範時的範例輸出。

```
24 1682 [iot_thread] [INFO] MQTT connection successfully established with broker.


25 1682 [iot_thread] [INFO] A clean MQTT connection is established. Cleaning up all the stored outgoing publishes.


26 1682 [iot_thread] [INFO] Attempt to subscribe to the MQTT topic $aws/things/DemoThing/defender/metrics/json/accepted.27 1682 [
iot_thread] [INFO] SUBSCRIBE sent for topic $aws/things/DemoThing/defender/metrics/json/accepted to broker.
28 1722 [iot_thread] [INFO] Packet received. ReceivedBytes=3.
29 1722 [iot_thread] [INFO] MQTT_PACKET_TYPE_SUBACK.


30 1722 [iot_thread] [INFO] Subscribed to the topic $aws/things/DemoThing/defender/metrics/json/accepted with maximum QoS 1.
31 2322 [iot_thread] [INFO] Attempt to subscribe to the MQTT topic $aws/things/DemoThing/defender/metrics/json/rejected.32 2322 [
iot_thread] [INFO] SUBSCRIBE sent for topic $aws/things/DemoThing/defender/metrics/json/rejected to broker.
33 2382 [iot_thread] [INFO] Packet received. ReceivedBytes=3.
34 2382 [iot_thread] [INFO] MQTT_PACKET_TYPE_SUBACK.


35 2382 [iot_thread] [INFO] Subscribed to the topic $aws/things/DemoThing/defender/metrics/json/rejected with maximum QoS 1.
36 2982 [iot_thread] [INFO] the published payload:{"hed": {"rid": 1109,"v": "1.0"},"met": {"tp": {"pts": [{"pt": 33251}],"t": 1},
"up": {"pts": [],"t": 0},"ns": {"bi": 0,"bo": 0,"pi": 0,"po": 0},"tc": {"ec": {"cs": [{"lp": 33251,"rad": "44.236.152.27:888337 2
982 [iot_thread] [INFO] PUBLISH sent for topic $aws/things/DemoThing/defender/metrics/json to broker with packet ID 3.


38 3102 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
39 3102 [iot_thread] [INFO] Ack packet deserialized with result: MQTTSuccess.
40 3102 [iot_thread] [INFO] State record updated. New state=MQTTPublishDone.
41 3102 [iot_thread] [INFO] PUBACK received for packet id 3.


42 3102 [iot_thread] [INFO] Cleaned up outgoing publish packet with packet id 3.


43 3102 [iot_thread] [INFO] Packet received. ReceivedBytes=141.
44 3102 [iot_thread] [INFO] De-serialized incoming PUBLISH packet: DeserializerResult=MQTTSuccess.
45 3102 [iot_thread] [INFO] State record updated. New state=MQTTPublishDone.
46 3502 [iot_thread] [INFO] UNSUBSCRIBE sent topic $aws/things/DemoThing/defender/metrics/json/accepted to broker.


47 3542 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
48 3542 [iot_thread] [INFO] MQTT_PACKET_TYPE_UNSUBACK.


49 4142 [iot_thread] [INFO] UNSUBSCRIBE sent topic $aws/things/DemoThing/defender/metrics/json/rejected to broker.


50 4202 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
51 4202 [iot_thread] [INFO] MQTT_PACKET_TYPE_UNSUBACK.


52 4802 [iot_thread] [INFO] Disconnected from the broker.
53 4802 [iot_thread] [INFO ][DEMO][4802] memory_metrics::freertos_heap::before::bytes::2088152

54 4802 [iot_thread] [INFO ][DEMO][4802] memory_metrics::freertos_heap::after::bytes::1985556

55 4802 [iot_thread] [INFO ][DEMO][4802] memory_metrics::demo_task_stack::before::bytes::1908

56 4802 [iot_thread] [INFO ][DEMO][4802] memory_metrics::demo_task_stack::after::bytes::1908

57 5802 [iot_thread] [INFO ][DEMO][5802] Demo completed successfully.

58 5804 [iot_thread] [INFO ][INIT][5804] SDK cleanup done.

59 5804 [iot_thread] [INFO ][DEMO][5804] -------DEMO FINISHED-------
```

如果您的主機板未使用 FreeRTO+TCP 或支援的 LWIP 組態，輸出將如下所示。

```
24 1716 [iot_thread] [INFO] MQTT connection successfully established with broker.


25 1716 [iot_thread] [INFO] A clean MQTT connection is established. Cleaning up all the stored outgoing publishes.


26 1716 [iot_thread] [INFO] Attempt to subscribe to the MQTT topic $aws/things/DemoThing/defender/metrics/json/accepted.27 1716
 [iot_thread] [INFO] SUBSCRIBE sent for topic $aws/things/DemoThing/defender/metrics/json/accepted to broker.
28 1756 [iot_thread] [INFO] Packet received. ReceivedBytes=3.
29 1756 [iot_thread] [INFO] MQTT_PACKET_TYPE_SUBACK.


30 1756 [iot_thread] [INFO] Subscribed to the topic $aws/things/DemoThing/defender/metrics/json/accepted with maximum QoS 1.
31 2356 [iot_thread] [INFO] Attempt to subscribe to the MQTT topic $aws/things/DemoThing/defender/metrics/json/rejected.32 2356
 [iot_thread] [INFO] SUBSCRIBE sent for topic $aws/things/DemoThing/defender/metrics/json/rejected to broker.
33 2436 [iot_thread] [INFO] Packet received. ReceivedBytes=3.
34 2436 [iot_thread] [INFO] MQTT_PACKET_TYPE_SUBACK.


35 2436 [iot_thread] [INFO] Subscribed to the topic $aws/things/DemoThing/defender/metrics/json/rejected with maximum QoS 1.
36 3036 [iot_thread] [ERROR] Using stub definition of GetNetworkStats! Please implement for your network stack to get correct m
etrics.
37 3036 [iot_thread] [ERROR] Using stub definition of GetOpenTcpPorts! Please implement for your network stack to get correct m
etrics.
38 3036 [iot_thread] [ERROR] Using stub definition of GetOpenUdpPorts! Please implement for your network stack to get correct m
etrics.
39 3036 [iot_thread] [ERROR] Using stub definition of GetEstablishedConnections! Please implement for your network stack to get
 correct metrics.
40 3036 [iot_thread] [INFO] the published payload:{"hed": {"rid": 1079,"v": "1.0"},"met": {"tp": {"pts": [],"t": 0},"up": {"pts
": [],"t": 0},"ns": {"bi": 0,"bo": 0,"pi": 0,"po": 0},"tc": {"ec": {"cs": [],"t": 0}}},"cmet": {"stack_high_water_mark": [{"num
41 3036 [iot_thread] [INFO] PUBLISH sent for topic $aws/things/DemoThing/defender/metrics/json to broker with packet ID 3.


42 3196 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
43 3196 [iot_thread] [INFO] Ack packet deserialized with result: MQTTSuccess.
44 3196 [iot_thread] [INFO] State record updated. New state=MQTTPublishDone.
45 3196 [iot_thread] [INFO] PUBACK received for packet id 3.


46 3196 [iot_thread] [INFO] Cleaned up outgoing publish packet with packet id 3.


47 3196 [iot_thread] [INFO] Packet received. ReceivedBytes=141.
48 3196 [iot_thread] [INFO] De-serialized incoming PUBLISH packet: DeserializerResult=MQTTSuccess.
49 3196 [iot_thread] [INFO] State record updated. New state=MQTTPublishDone.
50 3596 [iot_thread] [INFO] UNSUBSCRIBE sent topic $aws/things/DemoThing/defender/metrics/json/accepted to broker.


51 3656 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
52 3656 [iot_thread] [INFO] MQTT_PACKET_TYPE_UNSUBACK.


53 4256 [iot_thread] [INFO] UNSUBSCRIBE sent topic $aws/things/DemoThing/defender/metrics/json/rejected to broker.


54 4336 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
55 4336 [iot_thread] [INFO] MQTT_PACKET_TYPE_UNSUBACK.


56 4936 [iot_thread] [INFO] Disconnected from the broker.
57 4936 [iot_thread] [INFO ][DEMO][4936] memory_metrics::freertos_heap::before::bytes::2088152

58 4936 [iot_thread] [INFO ][DEMO][4936] memory_metrics::freertos_heap::after::bytes::1985556

59 4936 [iot_thread] [INFO ][DEMO][4936] memory_metrics::demo_task_stack::before::bytes::1908

60 4936 [iot_thread] [INFO ][DEMO][4936] memory_metrics::demo_task_stack::after::bytes::1908

61 5936 [iot_thread] [INFO ][DEMO][5936] Demo completed successfully.

62 5938 [iot_thread] [INFO ][INIT][5938] SDK cleanup done.

63 5938 [iot_thread] [INFO ][DEMO][5938] -------DEMO FINISHED-------
```

演示的源代碼是在您下載的*freertos*/demos/device_defender_for_aws/目錄或GitHub網站上。

訂閱主題 AWS IoT Device Defender

「 subscribeToDefender主題」功能會訂閱 MQTT 主題，以收到已發佈裝置防禦者報告的回應。它會使用巨集DEFENDER_API_JSON_ACCEPTED來建構主題字串，以便接收已接受裝置防禦者報告的回應。它會使用巨集DEFENDER_API_JSON_REJECTED來建構主題字串，以便接收已拒絕裝置防禦者報告的回應。

收集裝置指標

此 collectDeviceMetrics函數會使用中metrics_collector.h定義的函數來收集網路測量結果。收集的指標包括傳送和接收的位元組和封包數、開啟的 TCP 連接埠、開啟的 UDP 連接埠，以及建立的 TCP 連線。

產生 AWS IoT Device Defender 報告

Re generateDeviceMetricsport 函數會使用中定義的函數產生裝置防禦者報告report_builder.h。該函數採用網絡指標和緩衝區，按預期的格式創建 JSON 文檔， AWS IoT Device Defender 並將其寫入提供的緩衝區。在AWS IoT 開發人員指南的裝置端量度中指定預期的 JSON 文件格式。 AWS IoT Device Defender

發佈 AWS IoT Device Defender 報表

該 AWS IoT Device Defender 報告會在 MQTT 主題上發佈，以便發佈 JSON AWS IoT Device Defender 報告。該報告是使用宏構建的DEFENDER_API_JSON_PUBLISH，如 GitHub 網站上的此代碼片段所示。

處理響應的回調

發佈回呼函式會處理傳入 MQTT 發佈訊息。它使用來自 AWS IoT Device Defender 庫的 Defender_MatchTopic API 來檢查傳入的 MQTT 消息是否來自服務。 AWS IoT Device Defender 如果消息來自 AWS IoT Device Defender 服務，則會解析接收到的 JSON 響應並在響應中提取報告 ID。然後，系統會驗證報告 ID 與報告中傳送的 ID 相同。

AWS IoT GreengrassV1 探索示範應用程

> **Important**
>
> 此演示託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱亞馬遜自由 Github 存儲庫遷移指南.

執行 FreeRTOS 的AWS IoT Greengrass探索示範之前，您必須先設定AWSAWS IoT Greengrass、和AWS IoT。若要設定 AWS，請依照 設定您的 AWS 帳戶和權限 中的指示。若要設定 AWS IoT Greengrass，您需要建立 Greengrass 群組，然後新增 Greengrass 核心。如需設定 AWS IoT Greengrass 的詳細資訊，請參閱 AWS IoT Greengrass 入門。

在您設定 AWS 和 AWS IoT Greengrass 之後，您需要為 AWS IoT Greengrass 設定一些額外的許可。

若要設定 AWS IoT Greengrass 許可

1. 瀏覽至 IAM 主控台。
2. 在導覽窗格中，選擇 [角色]，然後尋找並選擇 [Greengrass_]ServiceRole。
3. 選擇 [附加政策]，選取 AmazonS3FullAccess 和 AWSIoTFullAccess，然後選擇 [附加原則]。
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. 選擇綠色 _ServiceRole，然後選擇 [儲存]。

將您的主機板 Connect 到 FreeRTOS 示範，AWS IoT並進行設定。

1. 註冊您的 MCU 主機板 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 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，選擇策略，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": "*"
      ```

```
}
```

此政策授予 AWS IoT Greengrass 對所有資源的許可。

5. 選擇 建立 。

將 AWS IoT Greengrass 政策連接至裝置憑證

1. 瀏覽至 AWS IoT 主控台。
2. 在導覽窗格中，依序選擇 Manage (管理)、Things (實物)，然後選擇您之前建立的實物。
3. 選擇 Security (安全)，然後選擇您裝置所連接的憑證。
4. 依序選擇 Policies (政策)、Actions (動作)，然後選擇 Attach Policy (連接政策)。
5. 尋找並選擇您之前建立的 Greengrass 政策，然後選擇 Attach (連接)。

2. 下載 FreeRTOS

> **Note**
>
> 如果您要從 FreeRTOS 主控台下載 FreeRTOS，請選擇 [Connect 至]AWS IoT Greengrass-[*##*] 而不是 [Connect 至]AWS IoT-[*##*]。

3. 設定 FreeRTOS 範.

開啟 *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h、註解 #define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED，然後定義 CONFIG_GREENGRASS_DISCOVERY_DEMO_ENABLED。

在您設定AWS IoT和AWS IoT Greengrass下載並設定 FreeRTOS 之後，您可以在裝置上建置、快閃記憶體和執行 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 示範會將一系列的訊息發佈到 Greengrass 核心以及 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]，選擇 [MQTT 測試用戶端]，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

如果您正在使用亞 Amazon EC2 實例

1. 尋找與您的 Amazon EC2 執行個體相關聯的公有 DNS (IPv4)，請前往 Amazon EC2 主控台，然後在左側導覽面板中選擇執行個體。選擇 Amazon EC2 執行個體，然後選擇說明面板。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，然後選擇連接到 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 Groups (群組)，然後選擇您先前建立的 Greengrass 群組。選擇 Settings (設定)。將 Local connection detection (區域連線偵測) 變更為 Manually manage connection information (手動管理連線資訊)。
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. 選擇向後箭頭以返回 AWS IoT Greengrass 群組頁面。在導覽窗格中，選擇 Devices (裝置)，然後選擇 Add Device (新增裝置)。

   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. 新增必要的訂閱 — 在 Greengrass 群組頁面中，選擇訂閱，然後選擇新增訂閱並輸入如下所示的資訊。

   訂閱

來源	Target	主題
TIGG1	IoT Cloud (IoT 雲端)	freertos/demos/ggd

   其中「來源」是指當您註冊板時在AWS IoT控制台中創建的AWS IoT東西的名稱-在此處給出的示例中「TIGG1」。
9. 啟動 AWS IoT Greengrass 群組的部署，並確定部署成功。您現在應該可以成功地執行 AWS IoT Greengrass 探索示範。

AWS IoT Greengrass V2

與AWS IoT Greengrass V2設備的兼容性

AWS IoT Greengrass V2客戶端設備的支持向後兼容AWS IoT Greengrass V1。您可以將 FreeRTOS 用戶端裝置連接到 V2 核心裝置，而無需變更應用程式程式碼。若要讓用戶端裝置連線到 V2 核心裝置，請執行下列動作。

- 將 Greengrass 軟體部署到核心裝置。請參閱將用戶端裝置連線到核心裝置以 Connect 裝置AWS IoT Greengrass V2。

- 若要在用戶端裝置、AWS IoT Core雲端服務和 Greengrass 元件之間轉送訊息（包括 Lambda 函數），請部署和設定 MQTT 橋接器元件。
- 部署 IP 偵測器元件以自動偵測連線資訊，或手動管理端點。
- 如需詳細資訊，請參閱與本機AWS IoT裝置互動。

如需詳細資訊，請參AWS閱上執行AWS IoT Greengrass V1應用程式的相關文件AWS IoT Greengrass V2。

核心 HTTP 演示

> **Important**
>
> 此演示託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。我們建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱. 亞馬遜自由 Github 存儲庫遷移指南

這些示範可協助您學習如何使用 CoreHTTP 程式庫。

主題

- 核心 HTTP 相互驗證示範
- Amazon S3 基本上傳演示
- 核心 HTTP 基本 S3 下載演示
- 核心 HTTP 基本多執行緒示範

核心 HTTP 相互驗證示範

> **Important**
>
> 此演示託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。我們建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱. 亞馬遜自由 Github 存儲庫遷移指南

簡介

CoreHTTP (相互驗證) 示範專案示範專案示範如何使用 TLS 與用戶端與伺服器之間的相互驗證來建立與 HTTP 伺服器的連線。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 HTTP 中的要求回應工作流程。

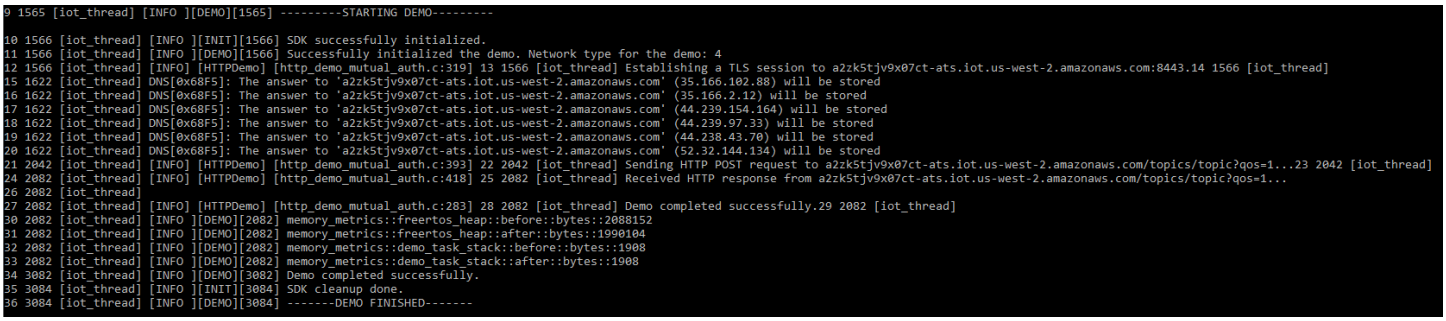
> **Note**
>
> 若要設定並執行 FreeRTOS 示範，請遵循中的步驟。FreeRTOS 入

功能

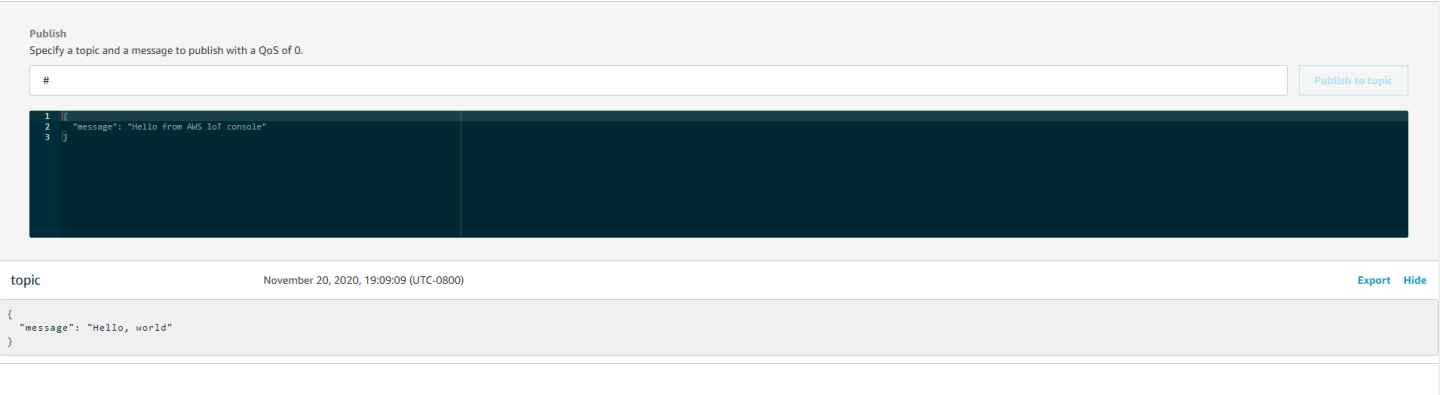
此示範會建立單一應用程式工作，其中包含示範如何完成下列項目的範例：

- Connect 至 AWS IoT 端點上的 HTTP 伺服器。
- 發送 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。



AWS IoT 控制台生成類似於下面的屏幕截圖輸出。



源代碼組織

演示源文件被命名`http_demo_mutual_auth.c`，可以在目*`freertos`*`/demos/coreHTTP/`錄和GitHub網站上找到。

連線至 AWS IoT HTTP 伺服器

connectToServerWithBackoff重試功能會嘗試與 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。逾時值會以指數方式增加，直到達到嘗試次數上限或達到最大逾時值為止。該`RetryUtils_BackoffAndSleep`函數提供指數增加的超`RetryUtilsRetriesExhausted`時值，並在達到最大嘗試次數時返回。如果在設定的嘗試次數之後無法建立與 Broker 的 TLS 連線，`connectToServerWithBackoffRetries`函數會傳回失敗狀態。

發送 HTTP 請求並接收響應

請 prvSendHttp求函數演示如何發送 POST 請求到 AWS IoT HTTP 服務器。如需有關向中的 REST API 發出要求的詳細資訊 AWS IoT，請參閱裝置通訊協定-HTTPS。系統會使用相同的 CoreHTTP API 呼叫來收到回應。`HTTPClient_Send`

Amazon S3 基本上傳演示

> **Important**
>
> 此演示託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。我們建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱. 亞馬遜自由 Github 存儲庫遷移指南

簡介

此範例示範如何將 PUT 請求傳送至亞馬遜簡單儲存服務 (Amazon S3) HTTP 伺服器並上傳小檔案。它還會執行 GET 請求，以在上傳後驗證文件的大小。本範例使用使用 mbedTLS 的網路傳輸界面，在執行 CoreHTTP 的 IoT 裝置用戶端與 Amazon S3 HTTP 伺服器之間建立互相驗證的連線。

> **Note**
>
> 若要設定並執行 FreeRTOS 示範，請遵循中的步驟。FreeRTOS 入

單螺紋與多線程

有兩種 CoreHTTP 使用模式，單執行緒和多執行緒 (多工處理)。雖然本節中的示範會在執行緒中執行 HTTP 程式庫，但實際上會示範如何在單一執行緒環境中使用 CoreHTTP。此示範中只有一項工作會使用 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 (或精靈) 工作中，在背景中傳送 HTTP 要求。

源代碼組織

演示源文件被命名`http_demo_s3_upload.c`，可以在目*`freertos`*`/demos/coreHTTP/`錄和 GitHub網站上找到。

設定 Amazon S3 HTTP 伺服器連線

本示範使用預先簽署的 URL 連接到 Amazon S3 HTTP 伺服器，並授權要下載的物件的存取權。Amazon S3 HTTP 伺服器的 TLS 連線僅使用伺服器身份驗證。在應用程式層級，會使用預先簽署的 URL 查詢中的參數驗證物件的存取權。請按照以下步驟配置您的連接 AWS。

1. 設置一個 AWS 帳戶：
   a. 如果您還沒有，請創建一個 AWS 帳戶。
   b. 帳戶和許可是使用 AWS Identity and Access Management (IAM) 設定的。您可以使用 IAM 管理帳戶中每個使用者的許可。根據預設，在根擁有者授與之前，使用者才具有權限。
      i. 若要將使用者新增至您的 AWS 帳戶，請參閱 IAM 使用者指南。
      ii. 授予您 AWS 帳戶存取 FreeRTOS 的權限，並新增此 AWS IoT 政策：
         - 亞馬遜 FullAccess
2. 按照如何建立 S3 儲存貯體中的步驟在 Amazon S3 中建立儲存貯體？ 在 Amazon 簡單存儲服務用戶指南。

3. 按照如何將檔案和資料夾上傳到 S3 儲存貯體中的步驟，將檔案上傳到 Amazon S3？ 。
4. 使用位於檔案中的指令碼產生預先簽署的 FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/presigned_urls_gen.py URL。

   如需使用指示，請參閱FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/README.md檔案。

功能

示範會先連線到具有 TLS 伺服器身份驗證的 Amazon S3 HTTP 伺服器。然後，它會創建一個 HTTP 請求來上傳在中指定的數據democonfigDEMO_HTTP_UPLOAD_DATA。上傳文件後，它通過請求文件的大小來檢查文件是否成功上傳。演示的源代碼可以在GitHub網站上找到。

連接到 Amazon S3 HTTP 伺服器

connectToServerWithBackoff重試功能會嘗試與 HTTP 伺服器建立 TCP 連線。如果連線失敗，則會在逾時後重試。逾時值會以指數方式增加，直到達到嘗試次數上限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，則connectToServerWithBackoffRetries函數會傳回失敗狀態。

此prvConnectToServer函數示範如何透過僅使用伺服器身份驗證來建立與 Amazon S3 HTTP 伺服器的連線。它使用在文件中實現的基於 MBEDTLS 的傳輸接口。FreeRTOS-Plus/Source/Application-Protocols/network_transport/freertos_plus_tcp/using_mbedtls/using_mbedtls.c的定義prvConnectToServer可以在GitHub網站上找到。

上傳資料

該prvUploadS3ObjectFile函數演示了如何創建 PUT 請求並指定要上傳的文件。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署的 URL 中指定。為了節省內存，相同的緩衝區用於兩個請求頭和接收響應。回應會使用 HTTPClient_Send API 函數同步接收。Amazon S3 HTTP 伺服器需要200 OK回應狀態碼。任何其他狀態碼都是錯誤。

的源代碼prvUploadS3ObjectFile()可以在 GitHub網站上找到。

驗證上傳

prvVerifyS3ObjectFileSize函數會呼叫prvGetS3ObjectFileSize以擷取 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署網址的 HEAD 請求，因此會要求第 0 個位

元組。該文件的大小包含在響應的`Content-Range`頭字段。服務器預期`206 Partial Content`響應。任何其他回應狀態碼都是錯誤。

的源代碼`prvGetS3ObjectFileSize()`可以在 GitHub網站上找到。

核心 HTTP 基本 S3 下載演示

> **Important**
>
> 此演示託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。我們建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱. 亞馬遜自由 Github 存儲庫遷移指南

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。當您使用建立 HTTP 要求時，CoreHTTP API 原生支援範圍`HTTPClient_AddRangeHeader`要求。對於微控制器環境，高度鼓勵範圍請求。通過下載不同範圍的大文件，而不是在單個請求中，可以在不阻止網絡套接字的情況下處理文件的每個部分。範圍請求降低了丟棄數據包的風險，這需要在 TCP 連接上重新傳輸，因此它們可以改善設備的功耗。

本範例使用使用 mbedTLS 的網路傳輸界面，在執行 CoreHTTP 的 IoT 裝置用戶端與 Amazon S3 HTTP 伺服器之間建立互相驗證的連線。

> **Note**
>
> 若要設定並執行 FreeRTOS 示範，請遵循中的步驟。FreeRTOS 入

單螺紋與多線程

有兩種 CoreHTTP 使用模式，單執行緒和多執行緒 (多工處理)。雖然本節中的示範會在執行緒中執行 HTTP 程式庫，但實際上會示範如何在單一執行緒環境中使用 CoreHTTP (在示範中只有一項工作使用 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 (或精靈) 工作中，在背景中傳送 HTTP 要求。

源代碼組織

演示項目被命名`http_demo_s3_download.c`，可以在目*`freertos`*`/demos/coreHTTP/`錄和GitHub網站上找到。

設定 Amazon S3 HTTP 伺服器連線

本示範使用預先簽署的 URL 連接到 Amazon S3 HTTP 伺服器，並授權要下載的物件的存取權。Amazon S3 HTTP 伺服器的 TLS 連線僅使用伺服器身份驗證。在應用程式層級，會使用預先簽署的 URL 查詢中的參數驗證物件的存取權。請按照以下步驟配置您的連接 AWS。

1. 設置一個 AWS 帳戶：
   a. 如果您還沒有，請創建並激活一個 AWS 帳戶。
   b. 帳戶和許可是使用 AWS Identity and Access Management (IAM) 設定的。IAM 可讓您管理帳戶中每個使用者的許可。根據預設，在根擁有者授與之前，使用者才具有權限。
      i. 若要將使用者新增至您的 AWS 帳戶，請參閱 IAM 使用者指南。
      ii. 授予您 AWS 帳戶存取 FreeRTOS 的權限，並新增 AWS IoT 下列原則：
         - 亞馬遜 FullAccess
2. 按照如何建立 S3 儲存貯體？中的步驟在 S3 建立儲存貯體？ 在 Amazon 簡單儲存服務主控台使用者指南中。
3. 按照如何將檔案和資料夾上傳到 S3 儲存貯體中的步驟將檔案上傳到 S3 儲存貯體？ 。
4. 使用位於的指令碼產生預先簽署的 URL。`FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/presigned_urls_gen.py`如需使用指示，請參閱`FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/README.md`。

功能

演示首先檢索文件的大小。然後它按順序請求每個字節範圍，在一個循環中，範圍大`democonfigRANGE_REQUEST_LENGTH`小為.

演示的源代碼可以在GitHub網站上找到。

連接到 Amazon S3 HTTP 伺服器

函數 connectToServerWithBackoff重試（ ）嘗試使 TCP 連接到 HTTP 服務器。如果連線失敗，則會在逾時後重試。逾時值會以指數方式增加，直到達到嘗試次數上限或達到最大逾時值為止。`connectToServerWithBackoffRetries()`如果在設定的嘗試次數之後無法建立與伺服器的 TCP 連線，則會傳回失敗狀態。

此函數prvConnectToServer()示範如何僅使用伺服器身份驗證來建立與 Amazon S3 HTTP 伺服器的連線。它使用在文件中實現的基於 MBEDTLS 的傳輸接口自由加/源/應用程序協議/網絡傳輸/自由托斯加/使用 _mbedtls.c。

的原始程式碼prvConnectToServer()可在上找到 GitHub。

建立範圍請求

API 函數HTTPClient_AddRangeHeader()支援將位元組範圍序列化為 HTTP 要求標頭，以形成範圍要求。範圍請求在本演示中用於檢索文件大小並請求文件的每個部分。

函數prvGetS3ObjectFileSize()會擷取 S3 儲存貯體中檔案的大小。標Connection: keep-alive頭會在此第一個請求中新增至 Amazon S3，以便在傳送回應後保持連線開啟狀態。S3 HTTP 伺服器目前不支援使用預先簽署 URL 的 HEAD 請求，因此會要求第 0 個位元組。該文件的大小包含在響應的Content-Range頭字段。從服務器預期206 Partial Content響應; 收到的任何其他響應狀態代碼是一個錯誤。

的原始程式碼prvGetS3ObjectFileSize()可在上找到 GitHub。

擷取檔案大小之後，此示範會為要下載的檔案的每個位元組範圍建立新的範圍請求。它用HTTPClient_AddRangeHeader()於文件的每個部分。

發送範圍請求和接收響應

該函數在一個循環中prvDownloadS3ObjectFile()發送範圍請求，直到整個文件被下載。API 函數HTTPClient_Send()發送請求並同步接收響應。當函數傳回時，回應會以xResponse. 然後驗證狀態碼是，到目前為止下載的字節數由頭值遞增。206 Partial Content Content-Length

的原始程式碼prvDownloadS3ObjectFile()可在上找到 GitHub。

核心 HTTP 基本多執行緒示範

> **Important**
>
> 此演示託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。我們建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱. 亞馬遜自由 Github 存儲庫遷移指南

簡介

此示範使用 FreeRTOS 的執行緒安全佇列來保留要求和回應等待處理。在此示範中，有三個工作需要注意。

- 主要工作會等待要求出現在要求佇列中。它會通過網絡發送這些請求，然後將響應放入響應隊列。
- 要求工作會建立要傳送至伺服器的 HTTP 程式庫要求物件，並將它們置於要求佇列中。每個請求物件都會指定應用程式設定要下載的 S3 檔案的位元組範圍。
- 響應任務等待響應出現在響應隊列中。它記錄它收到的每個響應。

這個基本的多執行緒示範設定為僅使用 TLS 連線與伺服器身份驗證，這是 Amazon S3 HTTP 伺服器所要求的。應用程式層驗證是使用預先簽署的 URL 查詢中的簽章版本 4 參數來完成。

源代碼組織

演示項目被命名`http_demo_s3_download_multithreaded.c`，可以在*`freertos`*`/demos/coreHTTP/`目錄和 GitHub網站中找到。

建立示範專案

示範專案使用免費社群版的視覺工作室。要構建演示：

1. 從內開啟`mqtt_multitask_demo.sln`視覺工作室解決方案檔案。
2. 從 IDE 的 [建置] 功能表中選取 [建置方案]。

> **Note**
>
> 如果您使用的是 Microsoft Visual Studio 2017 年或更早版本，那麼你必須選擇一個平台工具集兼容您的版本：項目-> RTOsdemo 屬性-> 平台工具集。

設定示範專案

此示範使用自由式 + TCP TCP/IP 堆疊，因此請依照 TCP/IP 入門專案提供的指示執行下列指示：

1. 安裝先決條件元件 (例如 WinPCap)。
2. 選擇性地設定靜態或動態 IP 位址、閘道位址和網路遮罩。
3. 選擇性地設定 MAC 位址。

4. 選取主機上的乙太網路介面。
5. 重要的是，在嘗試運行 HTTP 演示之前測試您的網絡連接。

設定 Amazon S3 HTTP 伺服器連線

請依照 CoreHTTP 基本下載示範設定 Amazon S3 HTTP 伺服器連線中的指示進行。

功能

該演示總共創建三個任務：

- 一種通過網絡發送請求和接收響應。
- 一個創建請求發送。
- 一個處理收到的響應。

在本演示中，主要任務：

1. 創建請求和響應隊列。
2. 建立與伺服器的連線。
3. 創建請求和響應任務。
4. 等待要求佇列透過網路傳送要求。
5. 將透過網路接收到的回應放置到回應佇列中。

請求任務：

1. 創建每個範圍請求。

響應任務：

1. 處理收到的每個回應。

类型定義

此示範定義下列結構以支援多執行緒。

請求項目

下列結構定義要放入要求佇列的要求項目。要求工作建立 HTTP 要求之後，要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;
```

響應物件

下列結構定義要放入回應佇列的回應項目。在主要 HTTP 工作透過網路接收回應之後，回應項目會複製到佇列中。

```
/**
 * @brief Data type for the response queue.
 *
 * Contains the response data type and its corresponding buffer, to be enqueued
 * by the main task, and interpreted by the response task. The buffer is
 * included to avoid pointer inaccuracy during queue copy operations.
 */
typedef struct ResponseItem
{
    HTTPResponse_t xResponse;
    uint8_t ucResponseBuffer[ democonfigUSER_BUFFER_LENGTH ];
} ResponseItem_t;
```

主要發送任務

主要應用任務：

1. 剖析主機位址的預先簽署 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署 URL。

3. 使用 TLS 與伺服器身份驗證連接到 Amazon S3 HTTP 伺服器。
4. 創建請求和響應隊列。
5. 創建請求和響應任務。

該功能prvHTTPDemoTask()執行此設置，並給出了演示狀態。這個函數的原始程式碼可以在 Github 上找到。

在函數中prvDownloadLoop()，主要任務塊並等待來自請求隊列的請求。當它收到一個請求時，它使用 API 函數發送它HTTPClient_Send()。如果 API 函數成功，則將響應放入響應隊列中。

的原始程式碼prvDownloadLoop()可以在 Github 上找到。

請求任務

請求任務在函數中指定prvRequestTask。這個函數的原始程式碼可以在 Github 上找到。

請求任務會擷取 Amazon S3 儲存貯體中檔案的大小。這是在函數中完成的prvGetS3ObjectFileSize。「連線：保持活動」標頭會新增至 Amazon S3 的此請求，以便在傳送回應後保持連線開啟狀態。Amazon S3 HTTP 伺服器目前不支援使用預先簽署 URL 的 HEAD 請求，因此會要求第 0 個位元組。該文件的大小包含在響應的Content-Range頭字段。從服務器預期206 Partial Content響應; 收到的任何其他響應狀態代碼是一個錯誤。

的原始程式碼prvGetS3ObjectFileSize可以在 Github 上找到。

擷取檔案大小後，要求工作會繼續要求檔案的每個範圍。每個範圍請求都放置在要求佇列中，以便傳送主要工作。檔案範圍由巨集中的示範使用者設定democonfigRANGE_REQUEST_LENGTH。使用函數HTTPClient_AddRangeHeader的 HTTP 用戶端程式庫 API 原生支援範圍要求。該函數prvRequestS3ObjectRange演示了如何使用HTTPClient_AddRangeHeader()。

該函數的源代碼prvRequestS3ObjectRange可以在 Github 上找到。

響應任務

響應任務在響應隊列上等待通過網絡接收的響應。主要工作成功接收 HTTP 回應時，會填入回應佇列。此工作會記錄狀態碼、標頭和內文來處理回應。例如，實際應用程式可以透過將回應主體寫入快閃記憶體來處理回應。如果回應狀態碼不是206 partial content，則工作會通知主要工作示範應該失敗。響應任務在函數中指定prvResponseTask。這個函數的原始程式碼可以在 Github 上找到。

AWS IoT工作庫演示

> **Important**
>
> 此演示託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。建議您在建立任務時從此處開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱亞馬遜自由 Github 存儲庫遷移指南.

簡介

JAWS IoT obs 程式庫示範將示範如何透過 MQTT 連線連線至 J AWS IoTobs 服務、從中擷取工作 AWS IoT，以及在裝置上進行處理。AWS IoT工作示範專案使用 FreeRTOS 視窗連接埠，因此可以使用視窗上的視覺工作室社群版本來建置和評估。不需要微控制器硬件。此示範會以相同的方式建立與 AWS IoT MQTT 代理程式的安全連線交互身分驗證示範。

> **Note**
>
> 若要設定並執行 FreeRTOS 示範，請遵循中的步驟FreeRTOS 入。

原始程式碼組織

演示代碼是在`jobs_demo.c`文件中，可以在GitHub網站上或*`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交互身分驗證示範.

功能

此示範顯示用來接收工作AWS IoT並在裝置上處理工作的工作流程。示範是互動式的，需要您使用 AWS IoT主控台或AWS Command Line Interface (AWS CLI) 來建立工作。如需建立任務的詳細資訊，請參閱：建立任務。AWS CLI此示範要求工作文件必須設定`action`金鑰，才能`print`將訊息列印到主控台。

採用下列格式：

```
{
```

```
    "action": "print",
    "message": "ADD_MESSAGE_HERE"
}
```

您可以使用AWS CLI來建立工作，如下列範例指令所示。

```
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金鑰設定為的工作文件，publish將訊息重新發行至主題。採用下列格式：

```
{
    "action": "publish",
    "message": "ADD_MESSAGE_HERE",
    "topic": "topic/name/here"
}
```

該演示循環，直到它收到一個工作文檔，action鍵設置exit為退出演示。工作文件的格式如下。

```
{
    "action: "exit"
}
```

工作演示的入口點

您可以在上找到 Jobs 示範入口點功能的原始程式碼 GitHub。此函數執行下列操作：

1. 使用中的輔助函數建立 MQTT 連線mqtt_demo_helpers.c。
2. 使用中的輔助函數，訂閱NextJobExecutionChanged API 的 MQTT 主題mqtt_demo_helpers.c。先前會使用AWS IoT Jobs 程式庫定義的巨集來組合主題字串。
3. 使用中的說明函StartNextPendingJobExecution式，發佈至 API 的 MQTT 主題mqtt_demo_helpers.c。先前會使用AWS IoT Jobs 程式庫定義的巨集來組合主題字串。
4. 反複呼叫MQTT_ProcessLoop以接收傳入的消息，這些消息被交給prvEventCallback進行處理。
5. 示範收到結束動作後，使用mqtt_demo_helpers.c檔案中的輔助函式，取消訂閱 MQTT 主題並中斷連線。

已接收 MQTT 訊息的回呼

prvEventCallback函數會Jobs_MatchTopic從「AWS IoT工作」程式庫呼叫，以將傳入的 MQTT 訊息分類。如果訊息類型對應於新工作，prvNextJobHandler()則會呼叫。

prvNextJobHandler 函數及其呼叫的函數、從 JSON 格式的訊息剖析工作文件，然後執行工作指定的動作。特別感興趣的是prvSendUpdateForJob功能。

傳送執行中工作的更新

函數 prvSendUpdateForJob()Jobs_Update() 從 Jobs 程式庫呼叫，以填入緊接在 MQTT 發佈作業中使用的主題字串。

演示

> **Important**
>
> 此演示託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱亞馬遜自由 Github 存儲庫遷移指南.

這些示範將會協助您了解如何使用 Coremqtt 程式庫。

主題

- 交互身分驗證示範
- CoremQtt 代理程式連線共用示範

交互身分驗證示範

> **Important**
>
> 此演示託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱亞馬遜自由 Github 存儲庫遷移指南.

簡介

CoremQtt 相互驗證示範專案說明如何使用 TLS 與用戶端與伺服器之間的相互驗證，建立與 MQTT 代理程式的連線。此示範使用 mBedTLS 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。它會訂閱主題篩選器，然後發佈至符合篩選條件的主題，並在 QoS 1 層級等待從伺服器接收這些郵件。此發佈至代理程式並從代理程式接收相同訊息的循環會無限期重複。此示範中的訊息會以 QoS 1 傳送，根據 MQTT 規格保證至少一次傳送。

> **Note**
>
> 若要設定並執行 FreeRTOS 示範，請遵循中的步驟FreeRTOS 入。

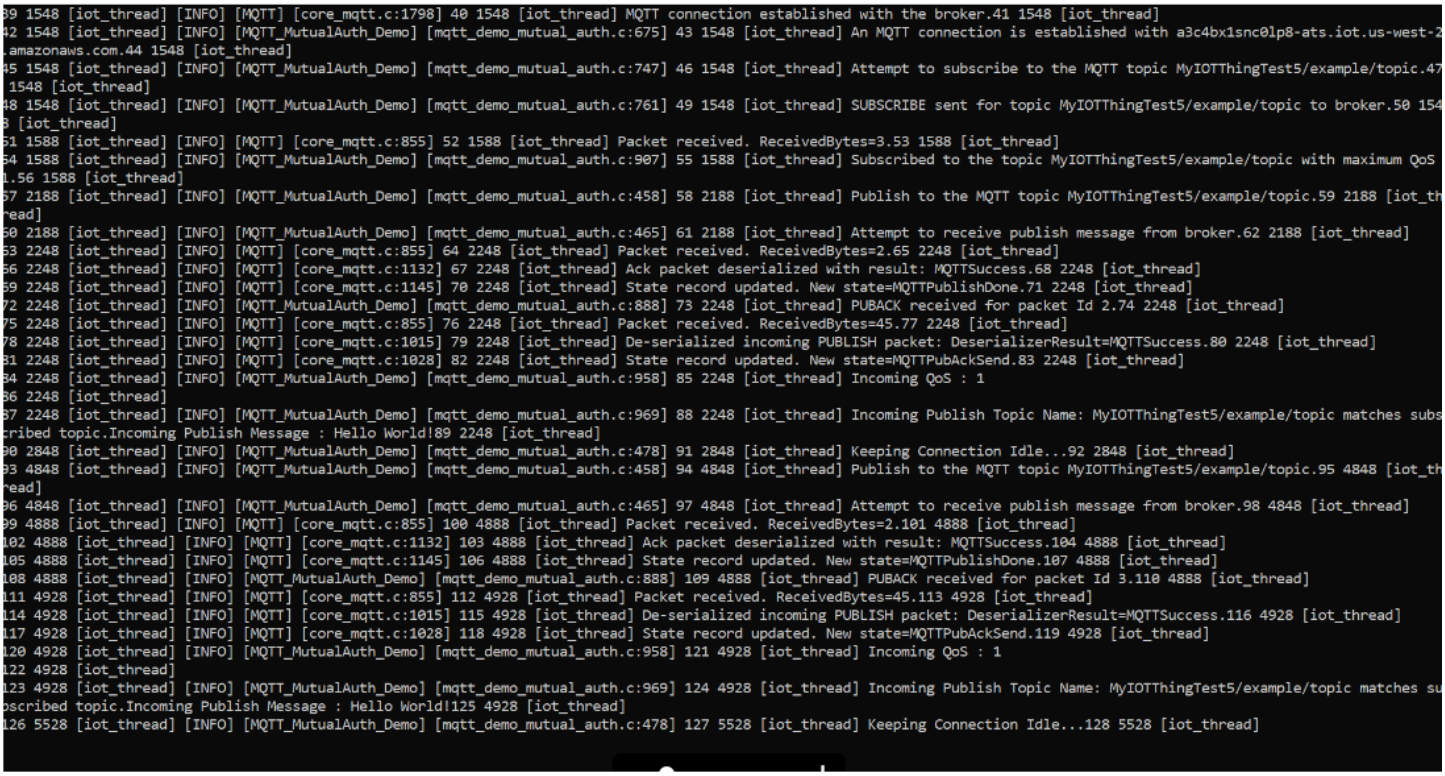
來源碼

演示源文件被命名`mqtt_demo_mutual_auth.c`，可以在*`freertos`*`/demos/coreMQTT/`目錄和 GitHub網站中找到。

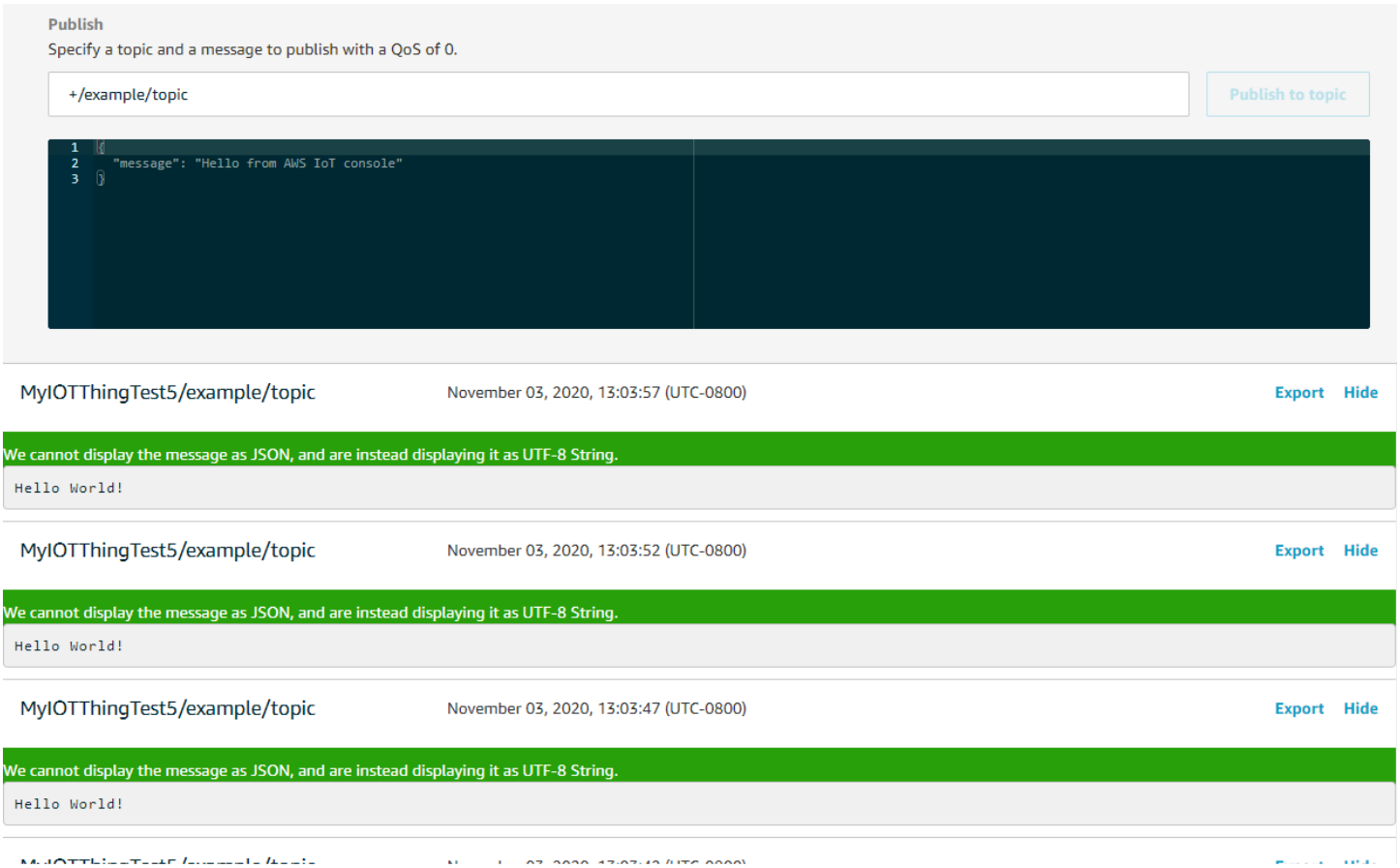
功能

演示創建一個單一的應用程序任務，通過一組示例演示如何連接到代理，訂閱代理的主題，發布到代理的主題，然後最後，斷開與代理的主題的例子。示範應用程式都會訂閱並發佈至相同的主題。每次示範將訊息發佈至 MQTT 代理程式時，代理程式就會將相同的訊息傳回示範應用程式。

成功完成演示將會產生與下圖類似的輸出。



AWS IoT控制台將會產生與下圖類似的輸出。



具有指數輪詢和抖動的重試邏輯

「prvBackoffFor重試」功能會顯示伺服器失敗的網路作業 (例如 TLS 連線或 MQTT 訂閱要求) 如何以指數輪詢和抖動來重試。函數會計算下一次重試嘗試的輪詢期間，如果重試嘗試尚未用盡，則會執行輪詢延遲。由於輪詢週期的計算需要產生隨機數，因此函數會使用 PKCS11 模組來產生隨機數。如果供應商平台支持，則使用 PKCS11 模塊允許訪問真隨機數生成器（TRNG）。我們建議您使用特定於設備的熵源植入隨機數生成器，以減輕連接重試期間從設備發生衝突的可能性。

連線至 MQTT 代理程式

該 prvConnectToServerWithBackoffRetries函數嘗試與 MQTT 代理人建立相互驗證的 TLS 連接。如果連線失敗，則會在輪詢期間後重試。輪詢期間將呈指數級增加，直到達到嘗試次數上限或達到最大輪詢期間為止。該BackoffAlgorithm_GetNextBackoff函數提供了一個指數增加的輪詢值，並RetryUtilsRetriesExhausted在達到最大嘗試次數時返回。如果在設定的嘗試次數之後無法建立與 Broker 的 TLS 連線，prvConnectToServerWithBackoffRetries函數會傳回失敗狀態。

P rvCreaTemqttConnectionWithBroker 函數示範如何透過乾淨的工作階段建立 MQTT 代理程式的 MQTT 連線。它使用 TLS 傳輸接口，這是在FreeRTOS-Plus/Source/Application-

Protocols/platform/freertos/transport/src/tls_freertos.c文件中實現。請記住，我們正在為經紀人設置保持活動秒數xConnectInfo。

下一個函數顯示如何使用該函數在 MQTT 內容中設定 TLS 傳輸介面和時間MQTT_Init函數。它還顯示了一個事件回調函數指針（prvEventCallback）是如何設置的。此回調用於報告傳入消息。

訂閱 MQTT 主題

P rvmqttSubscribeWithBackoffRetries 函數示範如何訂閱 MQTT 代理程式上的主題篩選器。此範例示範如何訂閱一個主題篩選器，但可以在相同的訂閱 API 呼叫中傳遞主題篩選器清單，以訂閱多個主題篩選器。此外，如果 MQTT 代理拒絕訂閱請求，則訂閱將重試，並具有指數輪詢RETRY_MAX_ATTEMPTS。

發佈到主題

P rvmqttPublishToTopic 函數示範如何發佈至 MQTT 代理程式的主題。

接收內送訊息

如先前所述，應用程式會在連線至 Broker 之前註冊事件回呼函數。該prvMQTTDemoTask函數調用該MQTT_ProcessLoop函數以接收傳入消息。當收到傳入的 MQTT 訊息時，它會呼叫應用程式所註冊的事件回呼函數。該 prvEventCallback函數是這樣的事件回調函數的一個例子。prvEventCallback檢查傳入的封包類型，並呼叫適當的處理常式。在下面的範例中，函數會呼叫prvMQTTProcessIncomingPublish()處理傳入的發佈訊息或prvMQTTProcessResponse()處理確認 (ACK)。

處理傳入 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。

取消訂閱主題

工作流程的最後一個步驟是取消訂閱主題，以便代理人不會從中傳送任何已發佈的訊息mqttexampleTOPIC。以下是函數的定義UnsubscribeFromTopic。

變更示範中使用的根 CA

根據預設，FreeRTOS 示範會使用亞馬遜根 CA 1 憑證 (RSA 2048 位元金鑰) 向AWS IoT Core伺服器進行驗證。您可以使用其他 CA 憑證進行伺服器身分驗證，包括 Amazon 根 CA 3 憑證 (ECC 256 位元金鑰)。若要變更 CoremQtt 相互驗證示範的根 CA，請執行下列動作：

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，找出下行。

```
* #define democonfigROOT_CA_PEM    "...insert here..."
```

取消註釋這一行，並在必要時將其移到註釋塊結尾處 */。

3. 複製您要使用的 CA 憑證，然後將其貼到"...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. (選擇性) 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 代理程式連線共用示範

> **Important**
>
> 此演示託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱亞馬遜自由 Github 存儲庫遷移指南.

簡介

CoremQtt 連線共用示範專案將示範您如何使用多執行緒應用程式，透過用戶端與伺服器之間的相互驗證，使用 TLS 建立與AWS MQTT 代理程式的連線。此示範使用 mBedTLS 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器、發佈至符合篩選器的主題，然後等待從 QoS 1 層級的伺服器接收這些訊息。對於每個創建的任

務，重複多次發布到代理人並從代理接收相同的消息的這個週期。此示範中的訊息會以 QoS 1 傳送，根據 MQTT 規格保證至少一次傳送。

> **Note**
>
> 若要設定並執行 FreeRTOS 示範，請遵循中的步驟FreeRTOS 入。

此示範使用執行緒安全佇列來保存與 MQTT API 互動的命令。此示範中有兩項工作需要注意。

- MQTT 代理程式 (主要) 工作會處理來自命令佇列的命令，而其他工作將它們排入佇列中。此工作會進入迴圈，在此迴圈期間會處理來自命令佇列的指令。如果收到終止命令，則此任務將中斷循環。
- 示範子酒吧工作會建立 MQTT 主題的訂閱，然後建立發佈作業並將其推送至命令佇列。然後，MQTT 代理程式工作會執行這些發佈作業。demo 子酒吧任務等待發布完成，由命令完成回調的執行指示，然後在開始下一次發布之前輸入一個短暫的延遲。此工作會顯示應用程式工作如何使用 CoremQtt 代理程式 API 的範例。

對於傳入的發行訊息，CoremQtt 代理程式會叫用單一回呼函數。此示範也包含一個訂閱管理員，可讓工作針對已訂閱的主題，針對內送發佈訊息指定要叫用的回呼。此示範中的代理程式傳入發佈回呼會叫用訂閱管理員，將發佈散佈至已註冊訂閱的任何工作。

此示範使用具有相互驗證的 TLS 連線來連線到AWS。如果在示範期間網路意外中斷連線，則用戶端會嘗試使用指數輪詢邏輯重新連線。如果用戶端成功重新連線，但 Broker 無法繼續先前的工作階段，則用戶端將重新訂閱與上一個工作階段相同的主題。

單線程與多線程

有兩種 CoremQtt 使用模式，分別是單執行緒和多執行緒 (多工處理)。單線程模型僅從一個線程使用 CoremQtt 庫，並要求您在 MQTT 庫中進行重複的顯式調用。多執行緒使用案例可改為在代理程式 (或協助程式) 工作的背景中執行 MQTT 通訊協定，如此處所述的示範所示。當您在代理程式工作中執行 MQTT 通訊協定時，您不需要明確管理任何 MQTT 狀態或呼叫`MQTT_ProcessLoop` API 函數。此外，當您使用代理程式工作時，多個應用程式工作可以共用單一 MQTT 連線，而不需要同步化基元 (例如互斥體)。

來源碼

演示源文件被命名，`mqtt_agent_task.csimple_sub_pub_demo.c`並且可以在*`freertos`*`/demos/coreMQTT_Agent/`目錄和GitHub網站中找到。

功能

此示範至少會建立兩個工作：處理 MQTT API 呼叫要求的主要工作，以及可設定數量的建立這些要求的子工作。在此示範中，主要工作會建立子工作、呼叫處理迴圈，然後再進行清除。主要任務會建立與子任務之間共用的代理人的單一 MQTT 連線。子任務與代理商創建 MQTT 訂閱，然後向其發布消息。每個子任務都使用一個獨特的主題進行發佈。

主要工作

主要應用程式工作 RunCoreMQTTAgentDemo 會建立 MQTT 工作階段、建立子工作，並執行處理迴圈 MQTTagent_，CommandLoop直到收到終止命令為止。如果網路意外中斷連線，示範會在背景中重新連線至代理程式，並與代理人重新建立訂閱。處理循環終止後，它與代理斷開連接。

命令

當您呼叫 CoremQtt 代理程式 API 時，它會建立傳送至代理程式工作佇列的命令，該命令會在中處理MQTTAgent_CommandLoop()。在創建命令時，可以傳遞可選的完成回調和上下文參數。一旦相應的命令完成後，完成回調將與傳遞的上下文以及作為命令結果創建的任何返回值調用。完成回調的簽名如下：

```
typedef void (* MQTTAgentCommandCallback_t )( void * pCmdCallbackContext,
                                              MQTTAgentReturnInfo_t * pReturnInfo );
```

命令完成上下文是用戶定義的; 對於本演示，它是：結構 MQTTAgentCommandContext。

在下列情況下，指令會被視為

- 以 QoS > 0 的方式訂閱、取消訂閱和發佈：收到對應的確認封包後。
- 所有其他操作：一旦調用了相應的 CoremQtt API。

指令使用的任何結構 (包括發行資訊、訂閱資訊和完成前後關聯) 都必須保持在範圍內，直到指令完成為止。調用任務不得在調用完成回調之前重複使用任何命令的結構。請注意，由於完成回調是由 MQTT Agent 調用，因此它將與代理程序任務的線程內容一起運行，而不是創建命令的任務。處理序間通訊機制，例如工作通知或佇列，可用來指示命令完成的呼叫工作。

執行命令迴圈

指令會在中連續處理MQTTAgent_CommandLoop()。如果沒有要處理的指令，迴圈會等待最多一個指令新增至佇列，如果沒有新增指令，則會執行的單一反覆運

算MQTT_ProcessLoop()。MQTT_AGENT_MAX_EVENT_QUEUE_WAIT_TIME這樣可確保管理 MQTT Keep-Alive，並且即使隊列中沒有命令，也可以接收任何傳入的發布。

命令迴圈函數會因下列原因而傳回：

- 命令會傳回除此之外的任何狀態碼MQTTSuccess。錯誤狀態由命令循環返回，因此您可以決定如何處理它。在此示範中，TCP 連線會重新建立，並進行重新連線嘗試。如果發生任何錯誤，則可能會在背景中重新連線，而不需要使用 MQTT 的其他任務進行任何干預。
- 會處理中斷連線指令 (從MQTTAgent_Disconnect)。指令迴圈會結束，以便 TCP 可以中斷連線。
- 處理終止命令（從MQTTAgent_Terminate）。此命令也會將仍在佇列中或等待確認封包的任何指令標示為錯誤，傳回碼為MQTTRecvFailed。

訂閱管理員

由於示範使用多個主題，因此訂閱管理員是將訂閱主題與唯一回呼或工作產生關聯的便捷方式。此示範中的訂閱管理員是單執行緒，因此不應同時由多個工作使用。在此示範中，訂閱管理員函數只會從傳遞至 MQTT 代理程式的回呼函式呼叫，而且只能在代理程式工作的執行緒內容中執行。

簡單的訂閱發布任務

的每個執行個體都 prvSimpleSubscribePublishTask會建立 MQTT 主題的訂閱，並為該主題建立發佈作業。若要示範多種發佈類型，即使是編號的工作也會使用 QoS 0 (在傳送發佈封包後即完成)，而奇數工作則使用 QoS 1 (在收到 PUBACK 封包後即完成)。

Over-the-air 更新演示應用

FreeRTOS 包含一個演示應用程序，演示了 over-the-air（OTA）庫的功能。OTA 演示應用程序位於*freertos*/demos/ota/ota_demo_core_mqtt/ota_demo_core_mqtt.c或*freertos*/demos/ota/ota_demo_core_http/ota_demo_core_http.c文件中。

OTA 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. 使用創建一個任務來練習 OTA 庫vRunOTAUpdateDemo()。
3. 使用 _establishMqttConnection() 建立 MQTT 用戶端。
4. 使用連接到AWS IoT MQTT 代理商，IotMqtt_Connect()並註冊 MQTT 斷開回調：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. 重複使用 MQTT 連接`xOTAConnectionCtx.pvControlClient = _mqttConnection;`
7. 如果 MQTT 斷開連接，應用程序將暫停 OTA 代理程序，嘗試使用帶抖動的指數延遲重新連接，然後恢復 OTA 代理程序。

在使用 OTA 更新之前，請先完成中的所有先決條件空中免費更新

完成 OTA 更新的設置後，在支持 OTA 功能的平台上下載，構建，刷新和運行 FreeRTOS OTA 演示。特定於裝置的示範指示可用於下列免費使用者資格的裝置：

- Texas Instruments CC3220SF-LAUNCHXL
- Microchip Curiosity PIC32MZEF
- Espressif ESP32
- 在瑞薩 RX65N 上下載、建置、快閃及執行 FreeRTOS OTA 示範

在您的裝置上建置、更新並執行 OTA 示範應用程式之後，您可以使用 AWS IoT 主控台或 AWS CLI 建立 OTA 更新任務。在您建立 OTA 更新任務後，請連線終端機模擬器，以查看 OTA 的進度更新。請記下程序中產生的任何錯誤。

成功的 OTA 更新任務，會顯示如下所示的輸出。為簡潔起見，此範例中的某些行已從清單中移除。

```
    249 21207 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT]  Received: 0
 Queued: 0   Processed: 0   Dropped: 0
    250 21247 [MQTT Agent Task] [core_mqtt.c:886] [INFO] [MQTT] Packet received.
 ReceivedBytes=601.
    251 21247 [MQTT Agent Task] [core_mqtt.c:1045] [INFO] [MQTT] De-serialized incoming
 PUBLISH packet: DeserializerResult=MQTTSuccess.
    252 21248 [MQTT Agent Task] [core_mqtt.c:1058] [INFO] [MQTT] State record updated.
 New state=MQTTPubAckSend.
    253 21249 [MQTT Agent Task] [ota_demo_core_mqtt.c:976] [INFO] [MQTT] Received job
 message callback, size 548.
    254 21252 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[execution.jobId: AFR_OTA-9702f1a3-b747-4c3e-a0eb-a3b0cf83ddbb]
    255 21253 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[execution.jobDocument.afr_ota.streamname: AFR_OTA-945d320b-a18b-441b-
b435-4a18d4e7671f]
    256 21255 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[execution.jobDocument.afr_ota.protocols: ["MQTT"]]
    257 21256 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[filepath: aws_demos.bin]
```

```
    258 21257 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
 value]=[filesize: 1164016]
    259 21258 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
 value]=[fileid: 0]
    260 21259 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[certfile: ecdsa-sha256-signer.crt.pem]
    261 21260 [OTA Agent Task] [ota.c:1575] [INFO] [OTA] Extracted parameter [ sig-
sha256-ecdsa: MEQCIE1SFkIHHiZAvkPpu6McJtx7SYoD... ]
    262 21261 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
 value]=[fileType: 0]
    263 21262 [OTA Agent Task] [ota.c:2199] [INFO] [OTA] Job document was accepted.
 Attempting to begin the update.
    264 21263 [OTA Agent Task] [ota.c:2323] [INFO] [OTA] Job parsing success:
 OtaJobParseErr_t=OtaJobParseErrNone, Job name=AFR_OTA-9702f1a3-b747-4c3e-a0eb-
a3b0cf83ddbb
    265 21318 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT]  Received: 0
 Queued: 0   Processed: 0   Dropped: 0
    266 21418 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT]  Received: 0
 Queued: 0   Processed: 0   Dropped: 0
    267 21469 [OTA Agent Task] [ota.c:938] [INFO] [OTA] Setting OTA data interface.
    268 21470 [OTA Agent Task] [ota.c:2839] [INFO] [OTA] Current State=[CreatingFile],
 Event=[ReceivedJobDocument], New state=[CreatingFile]
    269 21482 [MQTT Agent Task] [core_mqtt.c:886] [INFO] [MQTT] Packet received.
 ReceivedBytes=3.
    270 21483 [OTA Agent Task] [ota_demo_core_mqtt.c:1503] [INFO] [MQTT] SUBSCRIBED
 to topic $aws/things/__test_infra_thing71/streams/AFR_OTA-945d320b-a18b-441b-
b435-4a18d4e7671f/data/cbor to bro
    271 21484 [OTA Agent Task] [ota.c:2839] [INFO] [OTA] Current
 State=[RequestingFileBlock], Event=[CreateFile], New state=[RequestingFileBlock]
    272 21518 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT]  Received: 0
 Queued: 0   Processed: 0   Dropped: 0
    273 21532 [MQTT Agent Task] [core_mqtt_agent_command_functions.c:76] [INFO] [MQTT]
 Publishing message to $aws/things/__test_infra_thing71/streams/AFR_OTA-945d320b-
a18b-441b-b435-4a18d4e7671f/
    274 21534 [OTA Agent Task] [ota_demo_core_mqtt.c:1553] [INFO] [MQTT] Sent PUBLISH
 packet to broker $aws/things/__test_infra_thing71/streams/AFR_OTA-945d320b-a18b-441b-
b435-4a18d4e7671f/get/cbor
    275 21534 [OTA Agent Task] [ota_mqtt.c:1112] [INFO] [OTA] Published to MQTT
 topic to request the next block: topic=$aws/things/__test_infra_thing71/streams/
AFR_OTA-945d320b-a18b-441b-b435-4a1
    276 21537 [OTA Agent Task] [ota.c:2839] [INFO] [OTA] Current
 State=[WaitingForFileBlock], Event=[RequestFileBlock], New state=[WaitingForFileBlock]
    277 21558 [MQTT Agent Task] [core_mqtt.c:886] [INFO] [MQTT] Packet received.
 ReceivedBytes=4217.
```

```
   278 21559 [MQTT Agent Task] [core_mqtt.c:1045] [INFO] [MQTT] De-serialized incoming
PUBLISH packet: DeserializerResult=MQTTSuccess.
   279 21560 [MQTT Agent Task] [core_mqtt.c:1058] [INFO] [MQTT] State record updated.
New state=MQTTPublishDone.
   280 21561 [MQTT Agent Task] [ota_demo_core_mqtt.c:1026] [INFO] [MQTT] Received data
message callback, size 4120.
   281 21563 [OTA Agent Task] [ota.c:2464] [INFO] [OTA] Received valid file block:
Block index=0, Size=4096
   282 21566 [OTA Agent Task] [ota.c:2683] [INFO] [OTA] Number of blocks remaining:
284

   ... // Output removed for brevity

   3672 42745 [OTA Agent Task] [ota.c:2464] [INFO] [OTA] Received valid file block:
Block index=284, Size=752
   3673 42747 [OTA Agent Task] [ota.c:2633] [INFO] [OTA] Received final block of the
update.
   (428298) ota_pal: No such certificate file: ecdsa-sha256-signer.crt.pem. Using
certificate in ota_demo_config.h.
   3674 42818 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT]  Received: 285
Queued: 285   Processed: 284   Dropped: 0
   3675 42918 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT]  Received: 285
Queued: 285   Processed: 284   Dropped: 0

   ... // Output removed for brevity

   3678 43197 [OTA Agent Task] [ota.c:2654] [INFO] [OTA] Received entire update and
validated the signature.
   3685 43215 [OTA Agent Task] [ota_demo_core_mqtt.c:862] [INFO] [MQTT] Received
OtaJobEventActivate callback from OTA Agent.

   ... // Output removed for brevity

   2 39 [iot_thread] [INFO ][DEMO][390] ---------STARTING DEMO---------

   [0;32mI (3633) WIFI: WIFI_EVENT_STA_CONNECTED
   [0;32mI (4373) WIFI: SYSTEM_EVENT_STA_GOT_IP

   ... // Output removed for brevity

   4 351 [sys_evt] [INFO ][DEMO][3510] Connected to WiFi access point, ip address:
255.255.255.0.
   5 351 [iot_thread] [INFO ][DEMO][3510] Successfully initialized the demo. Network
type for the demo: 1
```

```
    6 351 [iot_thread] [ota_demo_core_mqtt.c:1902] [INFO] [MQTT] OTA over MQTT demo,
 Application version 0.9.1
    7 351 [iot_thread] [ota_demo_core_mqtt.c:1323] [INFO] [MQTT] Creating a TLS
 connection to <endpoint>-ats.iot.us-west-2.amazonaws.com:8883.
    9 718 [iot_thread] [core_mqtt.c:886] [INFO] [MQTT] Packet received.
 ReceivedBytes=2.
    10 718 [iot_thread] [core_mqtt_serializer.c:970] [INFO] [MQTT] CONNACK session
 present bit not set.
    11 718 [iot_thread] [core_mqtt_serializer.c:912] [INFO] [MQTT] Connection accepted.

    ... // Output removed for brevity

    17 736 [OTA Agent Task] [ota_demo_core_mqtt.c:1503] [INFO] [MQTT] SUBSCRIBED to
 topic $aws/things/__test_infra_thing71/jobs/notify-next to broker.
    18 737 [OTA Agent Task] [ota_mqtt.c:381] [INFO] [OTA] Subscribed to MQTT topic:
 $aws/things/__test_infra_thing71/jobs/notify-next
    30 818 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT]  Received: 0
 Queued: 0   Processed: 0   Dropped: 0
    31 819 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[execution.jobId: AFR_OTA-9702f1a3-b747-4c3e-a0eb-a3b0cf83ddbb]
    32 820 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
 value]=[execution.statusDetails.updatedBy: 589824]
    33 822 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[execution.jobDocument.afr_ota.streamname: AFR_OTA-945d320b-a18b-441b-
b435-4a18d4e7671f]
    34 823 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[execution.jobDocument.afr_ota.protocols: ["MQTT"]]
    35 824 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[filepath: aws_demos.bin]
    36 825 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
 value]=[filesize: 1164016]
    37 826 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
 value]=[fileid: 0]
    38 827 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[certfile: ecdsa-sha256-signer.crt.pem]
    39 828 [OTA Agent Task] [ota.c:1575] [INFO] [OTA] Extracted parameter [ sig-sha256-
ecdsa: MEQCIE1SFkIHHiZAvkPpu6McJtx7SYoD... ]
    40 829 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
 value]=[fileType: 0]
    41 830 [OTA Agent Task] [ota.c:2102] [INFO] [OTA] In self test mode.
    42 830 [OTA Agent Task] [ota.c:1936] [INFO] [OTA] New image has a higher version
 number than the current image: New image version=0.9.1, Previous image version=0.9.0
    43 832 [OTA Agent Task] [ota.c:2120] [INFO] [OTA] Image version is valid: Begin
 testing file: File ID=0
```

```
    53 896 [OTA Agent Task] [ota.c:794] [INFO] [OTA] Beginning self-test.
    62 971 [OTA Agent Task] [ota_demo_core_mqtt.c:1553] [INFO] [MQTT] Sent PUBLISH
 packet to broker $aws/things/__test_infra_thing71/jobs/AFR_OTA-9702f1a3-b747-4c3e-
a0eb-a3b0cf83ddbb/update to br63 971 [MQTT Agent Task] [core_mqtt.c:1045] [INFO] [MQTT]
 De-serialized incoming PUBLISH packet: DeserializerResult=MQTTSuccess.
    65 973 [MQTT Agent Task] [core_mqtt.c:1058] [INFO] [MQTT] State record updated. New
 state=MQTTPublishDone.
    64 973 [OTA Agent Task] [ota_demo_core_mqtt.c:902] [INFO] [MQTT] Successfully
 updated with the new image.
```

Over-the-air 演示配置

OTA 演示配置是中提供的特定於演示的配置選項`aws_iot_ota_update_demo.c`。這些配置與 OTA 庫配置文件中提供的 OTA 庫配置不同。

反映保持活躍 _ 秒

對於 MQTT 用戶端，此設定是在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以經過的最大時間間隔。在沒有控制封包的情況下，會傳送 PINGREQ。代理人必須在此保持活動時間間隔的一半倍內中斷不發送消息或 PINGREQ 數據包的客戶端斷開連接。此配置應根據應用程序的要求進行調整。

輸入演示 _ 重試 _ 基本間隔 _ 秒

重試網路連線之前的基本間隔 (以秒為單位)。OTA 演示將在此基本時間間隔後嘗試重新連接。每次嘗試失敗後，間隔都會加倍。此時間間隔中也會新增一個隨機延遲 (最多為此基本延遲的最大值)。

輸入演示 _ 重新嘗試 _ 最大間隔 _ 秒

重試網路連線之前的最大間隔 (以秒為單位)。每次嘗試失敗時，重新連線延遲都會加倍，但最多只能達到此最大值，再加上相同間隔的抖動。

在德州儀器 CC3220SF-LAUNCHXL 上下載、建置、快閃記憶體和執行 FreeRTOS OTA 示範

> **Important**
>
> 此參考集成託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱亞馬遜自由 Github 存儲庫遷移指南.

若要下載 FreeRTOS 者和 OTA 示範程式碼

- 您可以在 GitHub 網站上下載開放原始碼：https://github.com/FreeRTOS/FreeRTOS。

建置示範應用程式

1. 遵循 FreeRTOS 入中的說明，將 aws_demos 專案匯入 Code Composer Studio，並設定 AWS IoT 端點、Wi-Fi SSID 和密碼，以及主機板的私有金鑰和憑證。
2. 開啟*freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h、註解掉#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED，以及定義CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED或CONFIG_OTA_HTTP_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 1000 [Tmr Svc] Simple Link task created
Device came up in Station mode
1 2534 [Tmr Svc] Write certificate...
2 5486 [Tmr Svc] [ERROR] Failed to destroy object. PKCS11_PAL_DestroyObject failed.
3 5486 [Tmr Svc] Write certificate...
4 5776 [Tmr Svc] Security alert threshold = 15
5 5776 [Tmr Svc] Current number of alerts = 1
6 5778 [Tmr Svc] Running Demos.
7 5779 [iot_thread] [INFO ][DEMO][5779] ---------STARTING DEMO---------
8 5779 [iot_thread] [INFO ][INIT][5779] SDK successfully initialized.
Device came up in Station mode
[WLAN EVENT] STA Connected to the AP: afrlab-pepper , BSSID: 74:83:c2:b4:46:27
[NETAPP EVENT] IP acquired by the device
Device has connected to afrlab-pepper
Device IP Address is 192.168.36.176
```

```
    9 8283 [iot_thread] [INFO ][DEMO][8282] Successfully initialized the demo. Network
 type for the demo: 1
    10 8283 [iot_thread] [INFO] OTA over MQTT demo, Application version 0.9.0
    11 8283 [iot_thread] [INFO] Creating a TLS connection to <endpoint>-ats.iot.us-
west-2.amazonaws.com:8883.
    12 8852 [iot_thread] [INFO] Creating an MQTT connection to <endpoint>-ats.iot.us-
west-2.amazonaws.com.
    13 8914 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
    14 8914 [iot_thread] [INFO] CONNACK session present bit not set.
    15 8914 [iot_thread] [INFO] Connection accepted.
    16 8914 [iot_thread] [INFO] Received MQTT CONNACK successfully from broker.
    17 8914 [iot_thread] [INFO] MQTT connection established with the broker.
    18 8915 [iot_thread] [INFO]  Received: 0   Queued: 0   Processed: 0   Dropped: 0
    19 8953 [OTA Agent T] [INFO] Current State=[RequestingJob], Event=[Start], New
 state=[RequestingJob]
    20 9008 [MQTT Agent ] [INFO] Packet received. ReceivedBytes=3.
    21 9015 [OTA Agent T] [INFO] SUBSCRIBED to topic $aws/things/__test_infra_thing73/
jobs/notify-next to broker.
    22 9015 [OTA Agent T] [INFO] Subscribed to MQTT topic: $aws/things/
__test_infra_thing73/jobs/notify-next
    23 9504 [MQTT Agent ] [INFO] Publishing message to $aws/things/
__test_infra_thing73/jobs/$next/get.
    24 9535 [MQTT Agent ] [INFO] Packet received. ReceivedBytes=2.
    25 9535 [MQTT Agent ] [INFO] Ack packet deserialized with result: MQTTSuccess.
    26 9536 [MQTT Agent ] [INFO] State record updated. New state=MQTTPublishDone.
    27 9537 [OTA Agent T] [INFO] Sent PUBLISH packet to broker $aws/things/
__test_infra_thing73/jobs/$next/get to broker.
    28 9537 [OTA Agent T] [WARN] OTA Timer handle NULL for Timerid=0, can't stop.
    29 9537 [OTA Agent T] [INFO] Current State=[WaitingForJob],
 Event=[RequestJobDocument], New state=[WaitingForJob]
    30 9539 [MQTT Agent ] [INFO] Packet received. ReceivedBytes=120.
    31 9539 [MQTT Agent ] [INFO] De-serialized incoming PUBLISH packet:
 DeserializerResult=MQTTSuccess.
    32 9540 [MQTT Agent ] [INFO] State record updated. New state=MQTTPublishDone.
    33 9540 [MQTT Agent ] [INFO] Received job message callback, size 62.
    34 9616 [OTA Agent T] [INFO] Failed job document content check: Required job
 document parameter was not extracted: parameter=execution
    35 9616 [OTA Agent T] [INFO] Failed job document content check: Required job
 document parameter was not extracted: parameter=execution.jobId
    36 9617 [OTA Agent T] [INFO] Failed job document content check: Required job
 document parameter was not extracted: parameter=execution.jobDocument
    37 9617 [OTA Agent T] [INFO] Failed job document content check: Required job
 document parameter was not extracted: parameter=execution.jobDocument.afr_ota
```

```
   38 9617 [OTA Agent T] [INFO] Failed job document content
check: Required job document parameter was not extracted:
parameter=execution.jobDocument.afr_ota.protocols
   39 9618 [OTA Agent T] [INFO] Failed job document content check: Required job
document parameter was not extracted: parameter=execution.jobDocument.afr_ota.files
   40 9618 [OTA Agent T] [INFO] Failed job document content check: Required job
document parameter was not extracted: parameter=filesize
   41 9618 [OTA Agent T] [INFO] Failed job document content check: Required job
document parameter was not extracted: parameter=fileid
   42 9619 [OTA Agent T] [INFO] Failed to parse JSON document as AFR_OTA job:
DocParseErr_t=7
   43 9619 [OTA Agent T] [INFO] No active job available in received job document:
OtaJobParseErr_t=OtaJobParseErrNoActiveJobs
   44 9619 [OTA Agent T] [ERROR] Failed to execute state transition handler: Handler
returned error: OtaErr_t=OtaErrJobParserError
   45 9620 [OTA Agent T] [INFO] Current State=[WaitingForJob],
Event=[ReceivedJobDocument], New state=[CreatingFile]
   46 9915 [iot_thread] [INFO]  Received: 0   Queued: 0   Processed: 0   Dropped: 0
   47 10915 [iot_thread] [INFO]  Received: 0   Queued: 0   Processed: 0   Dropped: 0
   48 11915 [iot_thread] [INFO]  Received: 0   Queued: 0   Processed: 0   Dropped: 0
   49 12915 [iot_thread] [INFO]  Received: 0   Queued: 0   Processed: 0   Dropped: 0
   50 13915 [iot_thread] [INFO]  Received: 0   Queued: 0   Processed: 0   Dropped: 0
   51 14915 [iot_thread] [INFO]  Received: 0   Queued: 0   Processed: 0   Dropped: 0
```

在微晶片好奇心 PIC32MZEF 上下載、建置、快閃記憶體和執行 FreeRTOS OTA 示範

> **Important**
>
> 此參考集成託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱亞馬遜自由 Github 存儲庫遷移指南.

> **Note**
>
> 在與微晶協議下，我們將從 FreeRTOS 參考整合儲存庫主要分支移除好奇心 PIC32MZEF (DM320104)，並且將不再於新版本中攜帶。微晶已發出正式通知，表示不再建議將 PIC32MZEF (DM320104) 用於新設計。PIC32MZEF 專案和原始程式碼仍可透過舊版標籤存取。微晶片建議客戶在新設計上使用好奇心 PIC32MZ-EF-2.0 開發板

(DM320209)。Pic32MZv1 平台仍然可以在 FreeRTOS 參考整合儲存庫的 v202012.00 中找到。不過，《自由服務參考》的 v202107.00 已不再支援該平台。

若要下載免費使用者 OTA 示範程式碼

- 您可以在 GitHub 網站上下載開放原始碼：https://github.com/FreeRTOS/FreeRTOS。

建置 OTA 更新示範應用程式

1. 遵循FreeRTOS 入 中的說明，將 aws_demos 專案匯入 MPLAB X IDE，設定您的 AWS IoT 端點、您的 Wi-Fi SSID 及密碼，以及您電路板的私有金鑰和憑證。
2. 開啟檔vendors/*vendor*/boards/*board*/aws_demos/config_files/ota_demo_config.h案，然後輸入您的憑證。

   ```
   [] = "your-certificate-key";
   ```

3. 將代碼簽名證書的內容粘貼到此處：

   ```
   #define otapalconfigCODE_SIGNING_CERTIFICATE [] = "your-certificate-key";
   ```

   遵循與aws_clientcredential_keys.h-相同的格式，每一行必須以新行字元 ('\ n') 結尾，並用引號括住。

   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、註解掉#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED，以及定義CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED或CONFIG_OTA_HTTP_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. 從您的主機板拔除除錯器，並在您的主機板上執行專案，確認它可以連線到 Wi-Fi 及 AWS IoT MQTT 訊息中介裝置。

當您執行專案時，MPLAB X IDE 應會開啟輸出視窗。確認已選取 ICD4 索引標籤。您應該會看到下列輸出。

```
Bootloader version 00.09.00
[prvBOOT_Init] Watchdog timer initialized.
[prvBOOT_Init] Crypto initialized.

[prvValidateImage] Validating image at Bank : 0
[prvValidateImage] No application image or magic code present at: 0xbd000000
[prvBOOT_ValidateImages] Validation failed for image at 0xbd000000

[prvValidateImage] Validating image at Bank : 1
[prvValidateImage] No application image or magic code present at: 0xbd100000
[prvBOOT_ValidateImages] Validation failed for image at 0xbd100000

[prvBOOT_ValidateImages] Booting default image.


>0 36246 [IP-task] 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 可以連線到 AWS IoT，並且訂閱 OTA 更新所需要的 MQTT 主題。`Missing job parameter` 訊息是在預期中的內容，因為沒有任何擱置中的 OTA 更新任務。

下載，構建，閃存和運行 FreeRTOS OTA 演示在濃縮咖啡 ESP32

> **Important**
>
> 此參考集成託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱亞馬遜自由 Github 存儲庫遷移指南.

1. 從下載 FreeRTOS 的原始碼GitHub。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、註解掉#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED，以及定義CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED或CONFIG_OTA_HTTP_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、註解掉#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED，以及定義CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED或CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED。
- 在以下位置開啟vendors/*vendor*/boards/*board*/aws_demos/config_files/ota_demo_config.h並複製您的 SHA-256/ECDSA 程式碼簽署憑證：

```
#define otapalconfigCODE_SIGNING_CERTIFICATE [] = "your-certificate-key";
```

在瑞薩 RX65N 上下載、建置、快閃及執行 FreeRTOS OTA 示範

> **Important**
>
> 此參考集成託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。我們建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱. 亞馬遜自由 Github 存儲庫遷移指南

本章說明如何在瑞薩 RX65N 上下載、建置、快速執行 FreeRTOS OTA 示範應用程式。

主題

- 設定您的作業環境
- 設定您的 AWS 資源
- 導入，配置頭文件並構建 aws_demo 和引導加載器

設定您的作業環境

本節中的程序使用下列環境：

- IDE：電子工作室 7.8.0，和 2 工作室
- 工具鏈：CCRX 編譯器
- 目標裝置：RSKRX65N-2MB
- 調試器：E 2，E 2 精簡版模擬器
- 軟體：瑞薩快閃記憶體程式設計師，瑞薩安全快閃記憶體 Programmer.exe，Tera 術語

若要設定您的硬體

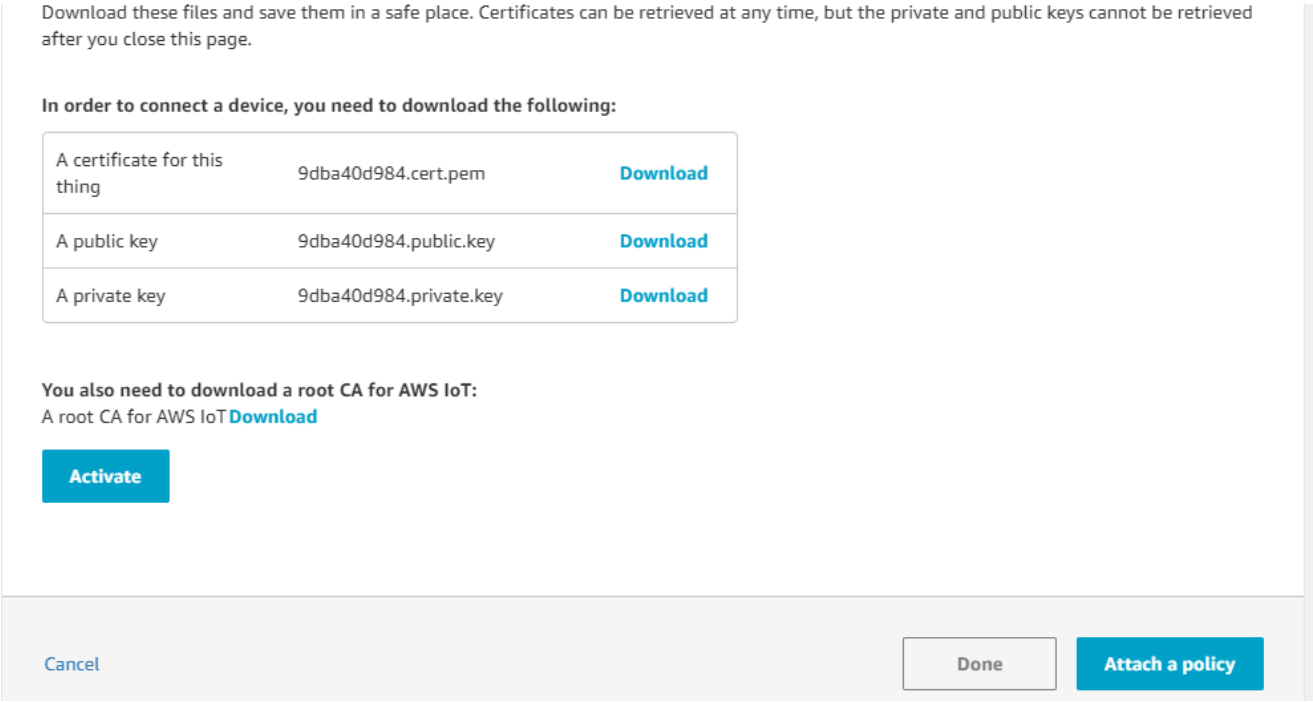
1. 將 E ^{2} 精簡版模擬器和 USB 串行端口 Connect 到您的 RX65N 主板和 PC。
2. 將電源 Connect 至 RX65N。

設定您的 AWS 資源

1. 若要執行 FreeRTOS 示範，您必須擁有具有存 AWS IoT 取服務權限的 IAM 使用者 AWS 帳戶。如果您還沒有，請按照中的步驟操作設定您的 AWS 帳戶和權限。
2. 若要設定 OTA 更新，請遵循中的步驟OTA 更新先決條件。尤其是，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「物件」來建立物件。

   物件是中裝置或邏輯實體的表示 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，例如應用程式執行個體或實體實體不連線 AWS IoT，但與裝置有關 (例如，具有引擎感應器或控制面板的汽車)。 AWS IoT 提供了一個東西註冊表，幫助你管理你的事情。

   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. 選取您在中建立的策略裝置政策。

使用 MQTT 接收 OTA 更新的每個設備都必須註冊為物件， AWS IoT 並且該物件必須具有與列出的策略相同的附加策略。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權限允許您的設備通 AWS IoT 過 MQTT 連接。
- AWS IoT 工作 (.../jobs/*) 主題的 iot:Subscribe 和 iot:Publish 許可允許連接的裝置接收工作通知和工作文件，並發佈工作執行的完成狀態。
- AWS IoT OTA 流（ .../streams/* ）主題的iot:Subscribe和iot:Publish權限允許連接的設備從中獲取 OTA 更新數據 AWS IoT。需要這些許可才能透過 MQTT 執行韌體更新。
- iot:Receive權限允許 AWS IoT Core 將這些主題的消息發佈到連接的設備。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. 若要建立程式碼簽署設定檔，並在上註冊程式碼簽署憑證。 AWS

a. 若要建立金鑰與認證，請參閱瑞薩 MCU 韌體更新設計原則中的第 7.3 節「使用 OpenSSL 產生 ECDSA-SHA256 金鑰配對」。

b. 開啟 AWS IoT 主控台。在左側導覽窗格中，選取管理，然後選取工作。選取建立 Job，然後選取建立 OTA 更新工作。

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項目 選取下一步。

d. 在 [建立 FreeRTOS OTA 更新工作] 頁面上，執行下列動作：

   i. 對於選擇韌體影像傳輸的通訊協定，請選擇 MQ TT。

   ii. 對於選擇並簽署您的固件映像，請選擇為我簽署新的固件映像。

   iii. 針對程式碼簽章設定檔，選擇 [建立]

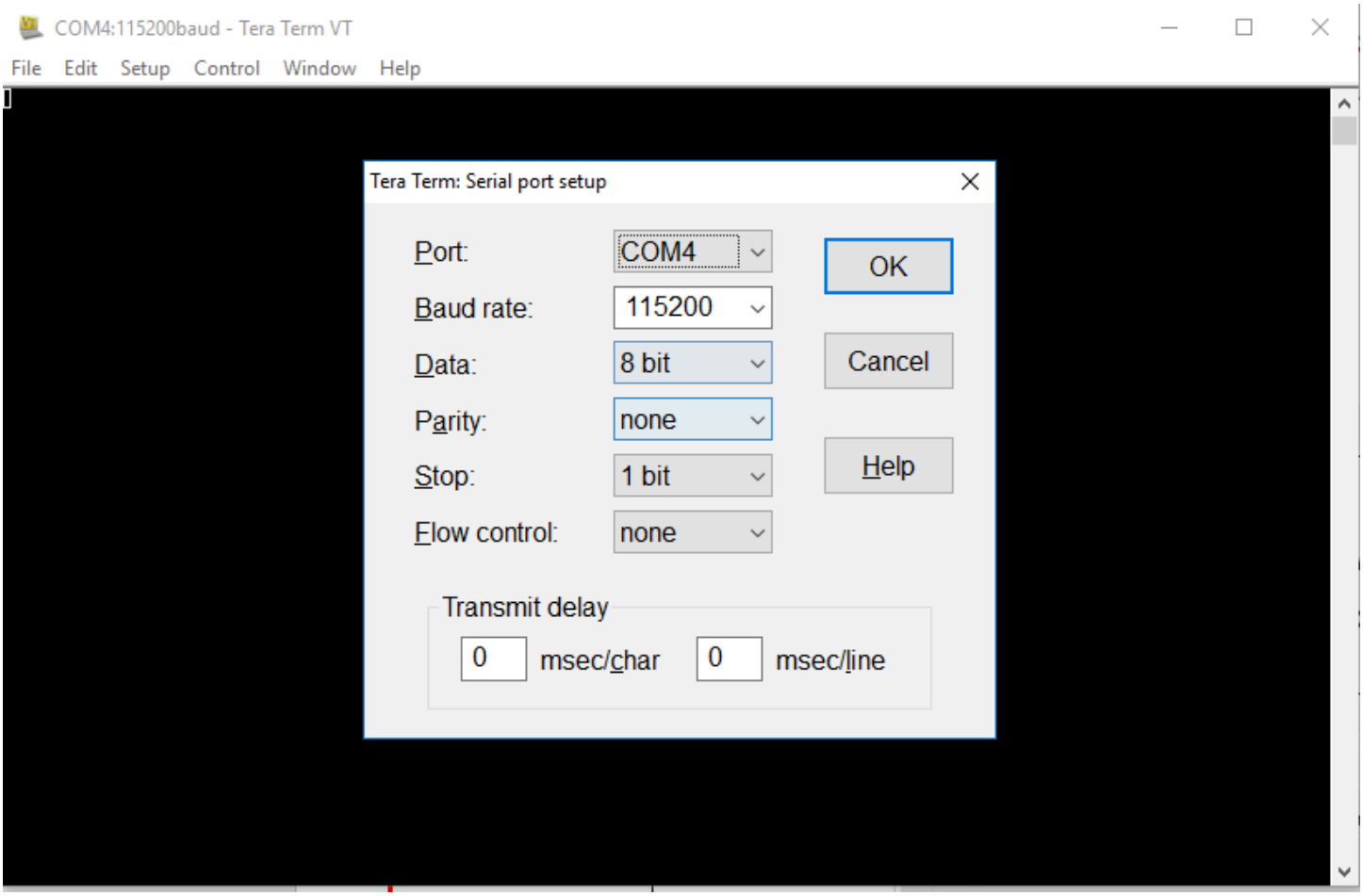
   iv. 在「建立程式碼簽章設定檔」視窗中，輸入設定檔名稱。針對裝置硬體平台，選取視窗模擬器。對於代碼簽名證書，選擇導入。

   v. 瀏覽以選取憑證 (`secp256r1.crt`)、憑證私密金鑰 (`secp256r1.key`) 和憑證鏈結 (`ca.crt`)。

   vi. 輸入裝置上程式碼簽章憑證的路徑名稱。然後選擇 Create (建立)。

6. 若要授與的程式碼簽章存取權 AWS IoT，請遵循中的步驟將存取權限授予適用於 AWS IoT 的程式碼簽署。

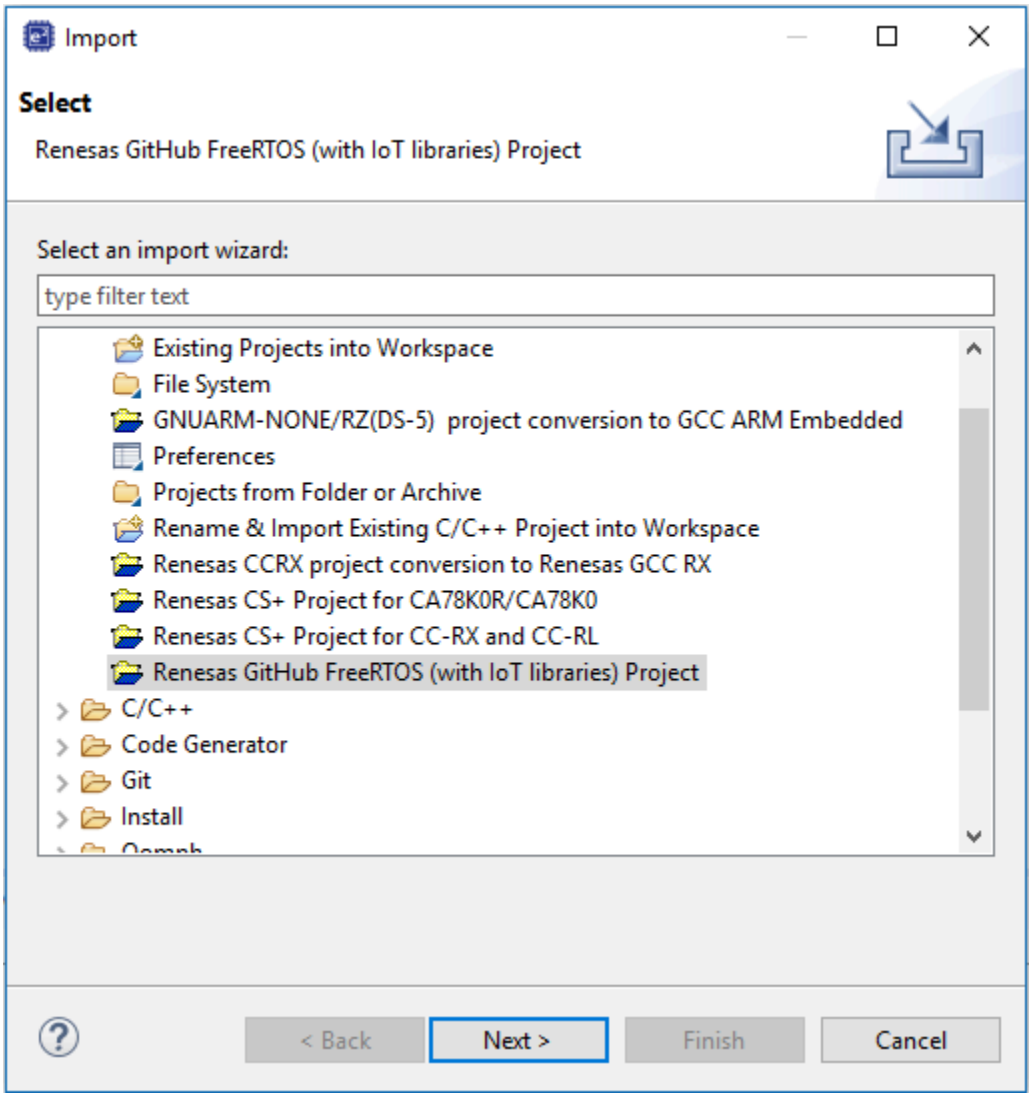
如果您的電腦上沒有安裝 Tera Term，您可以從 https://ttssh2.osdn.jp/index.html.en 下載並進行設定，如下所示。確保將 USB 串行端口從設備插入 PC。



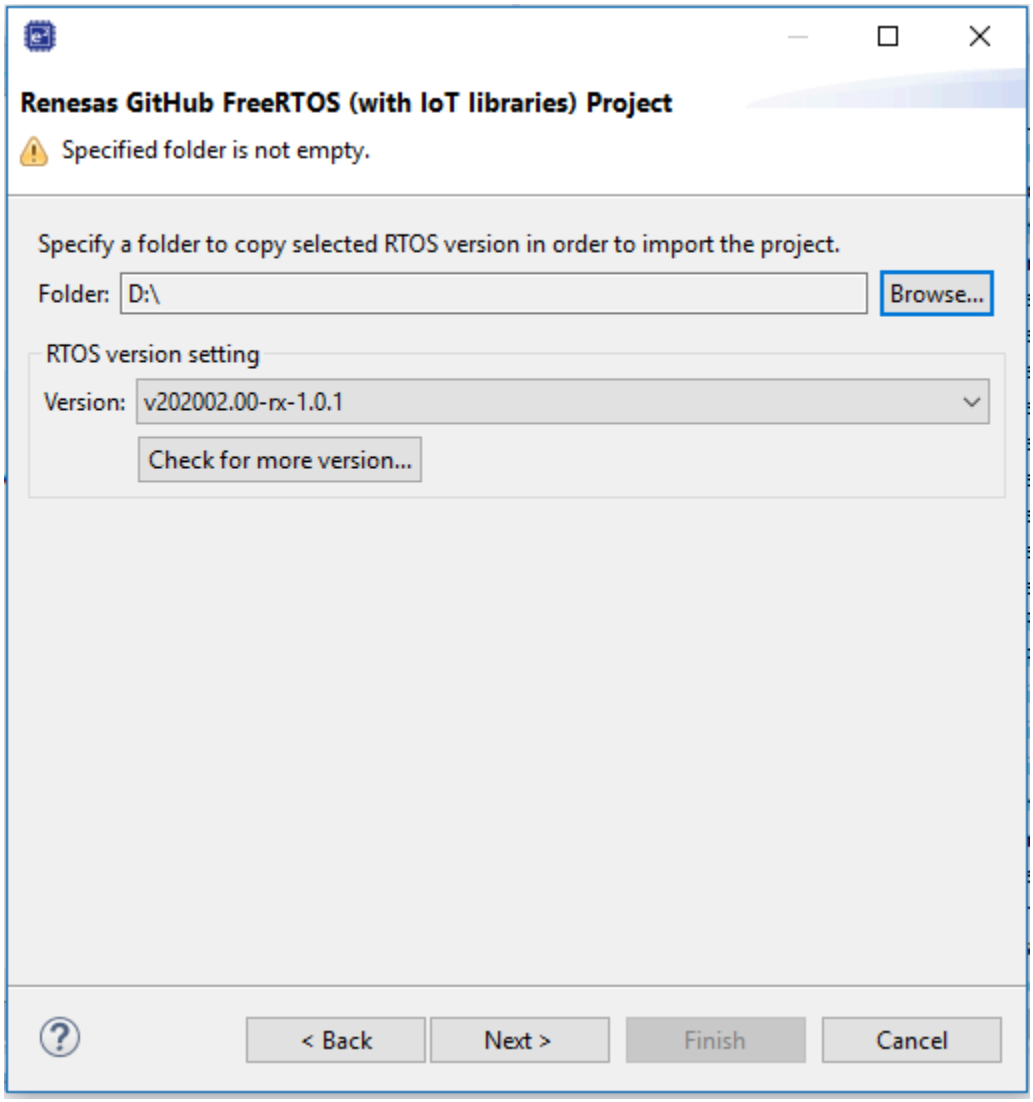
導入，配置頭文件並構建 aws_demo 和引導加載器

若要開始，請選取 FreeRTOS 套件的最新版本，這會從下載 GitHub 並自動匯入到專案中。這樣，您就可以專注於配置 FreeRTOS 和編寫應用程序代碼。

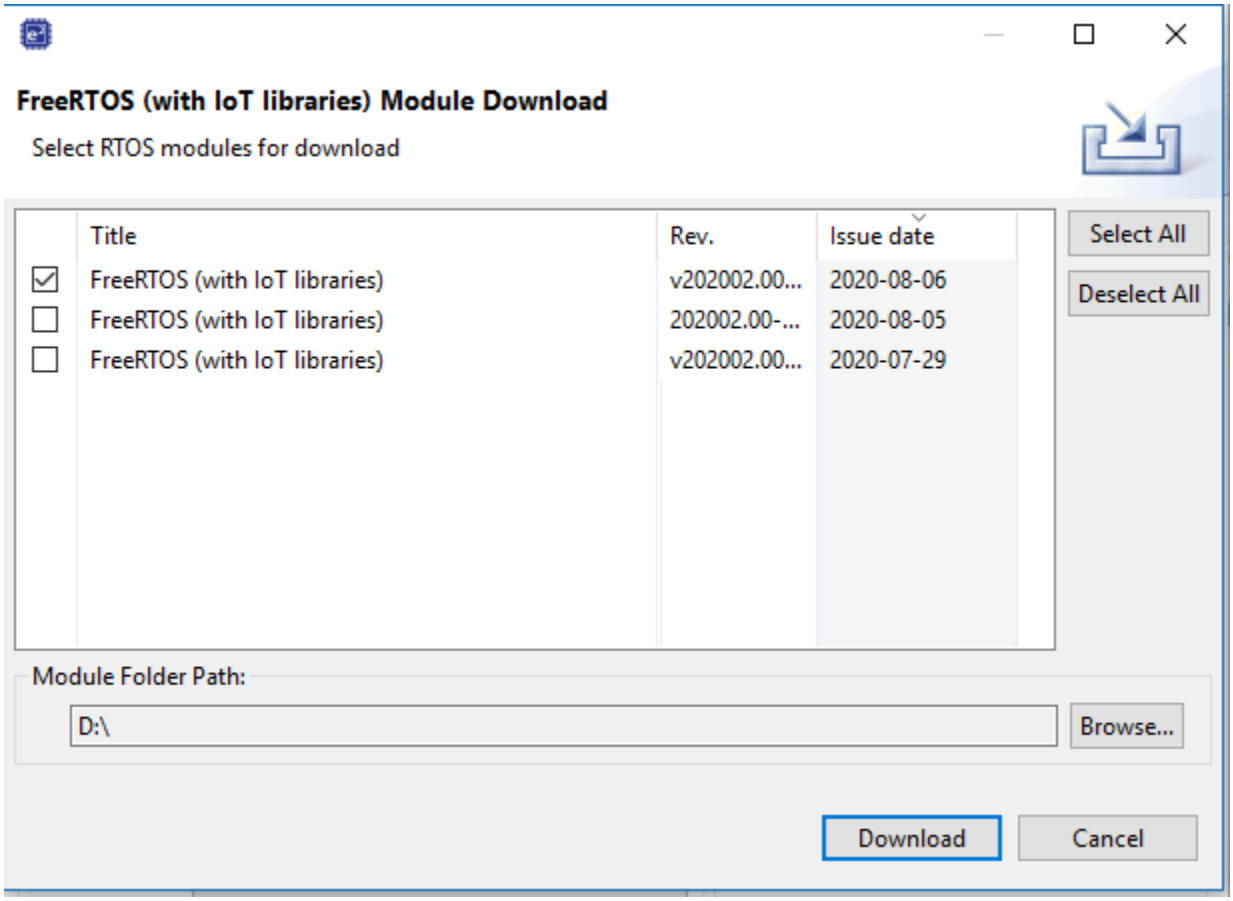
1. 啟動 e ^{2} 工作室。
2. 選擇 [檔案]，然後選擇 [匯入...]。
3. 選取「瑞薩 GitHub FreeRTOS (含 IoT 程式庫) 專案」。



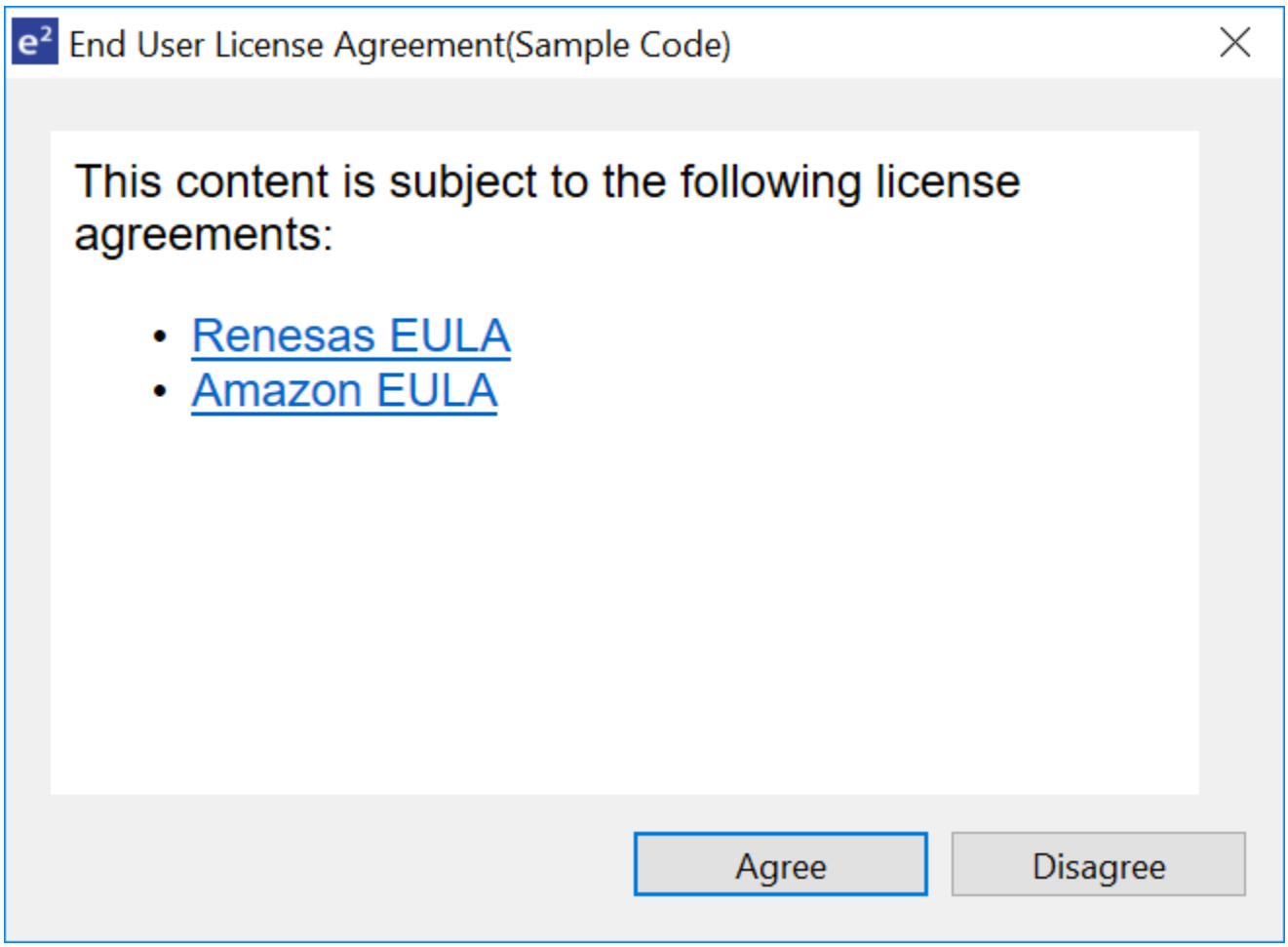
4. 選擇檢查更多版本... 以顯示下載對話方塊。


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

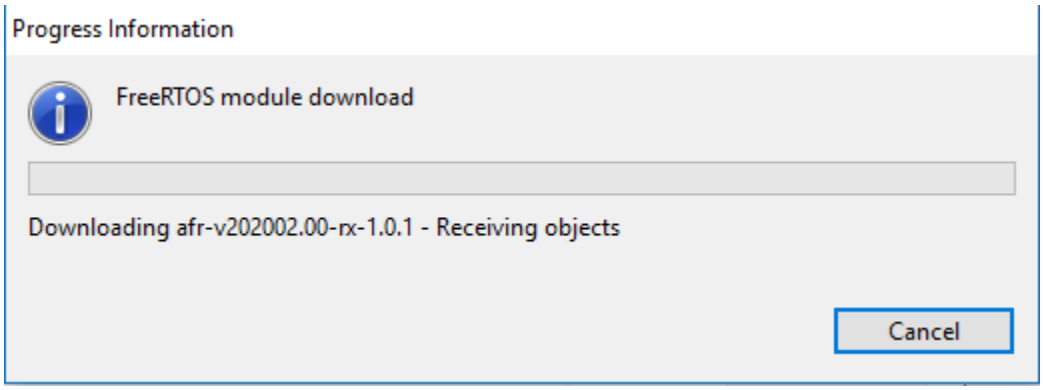
5. 選取最新的套件。



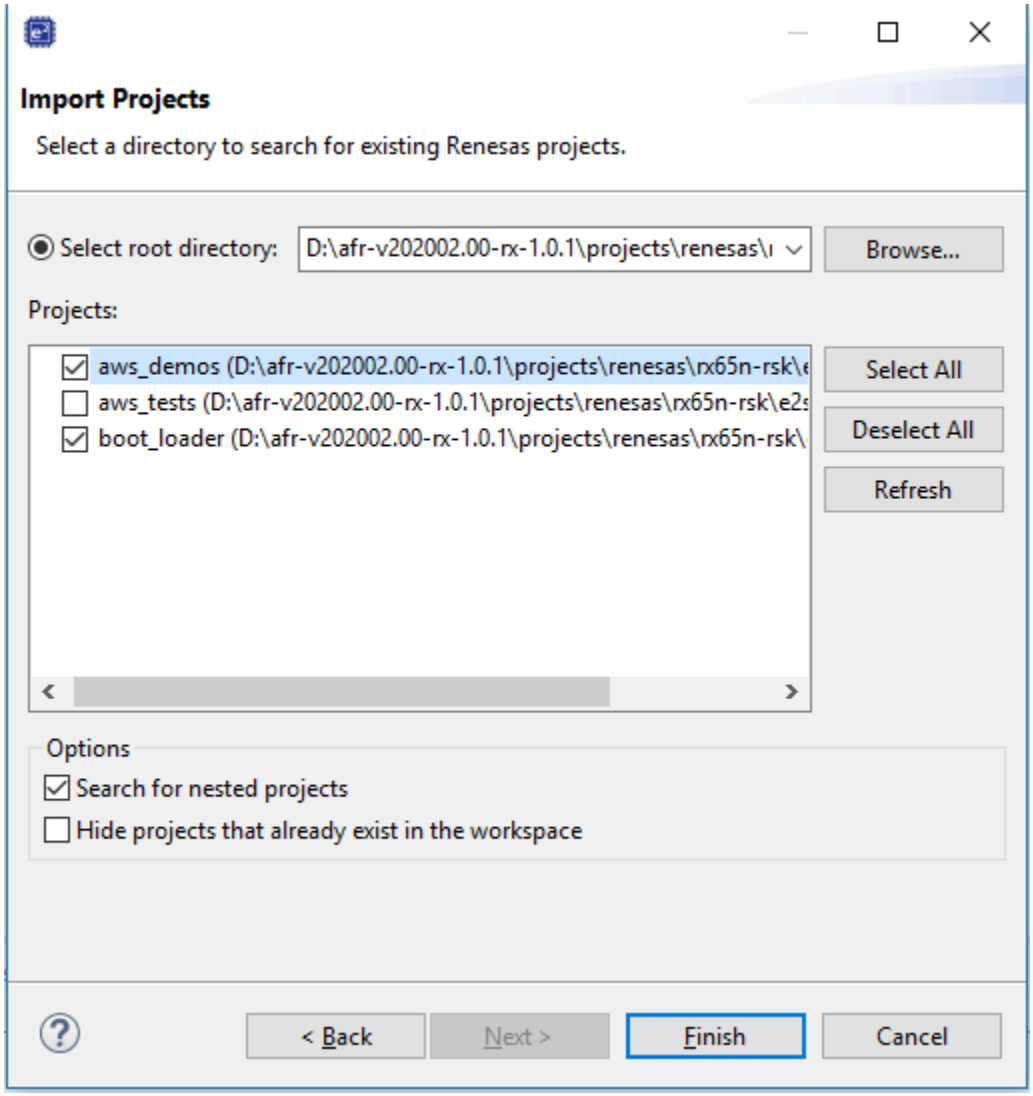
6. 選擇 [同意] 以接受使用者授權合約。



7. 等待下載完成。



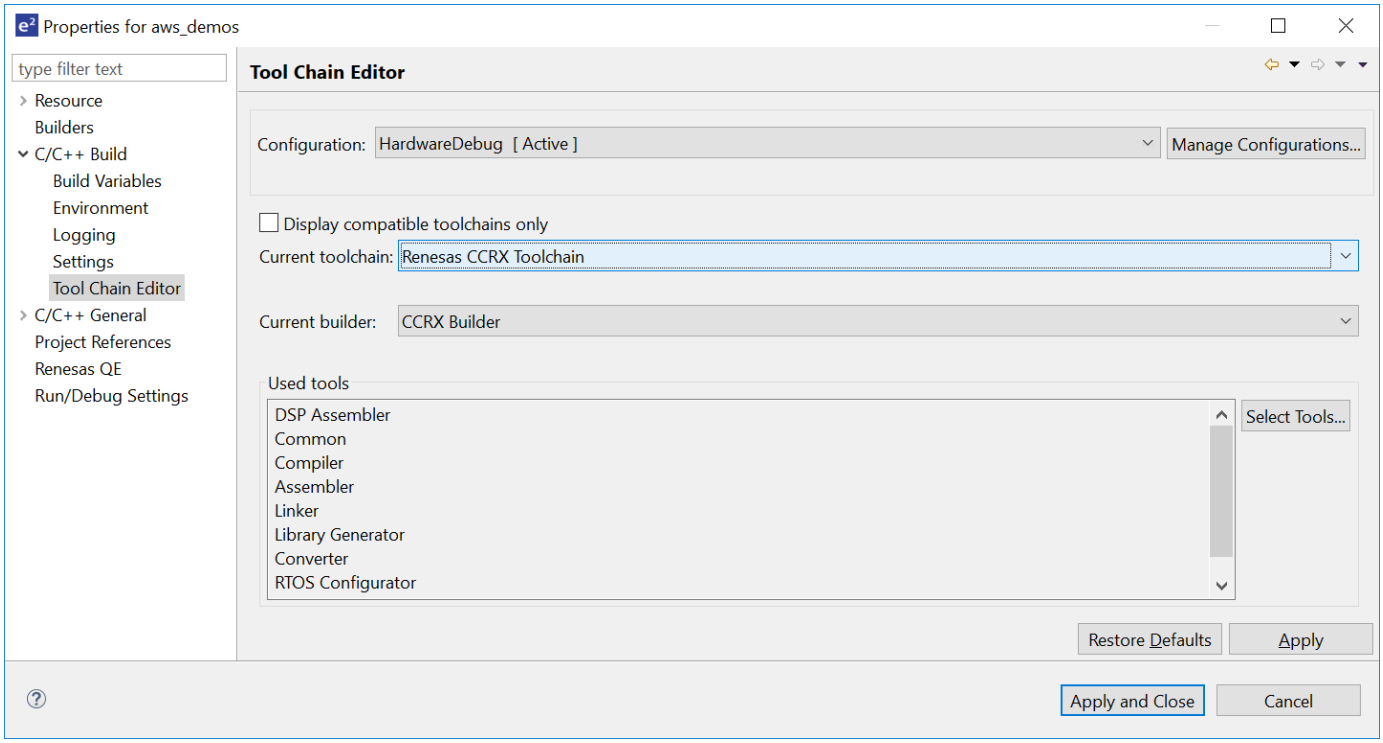
8. 選擇 aws_demo 和引導加載程序項目，然後選擇完成導入它們。



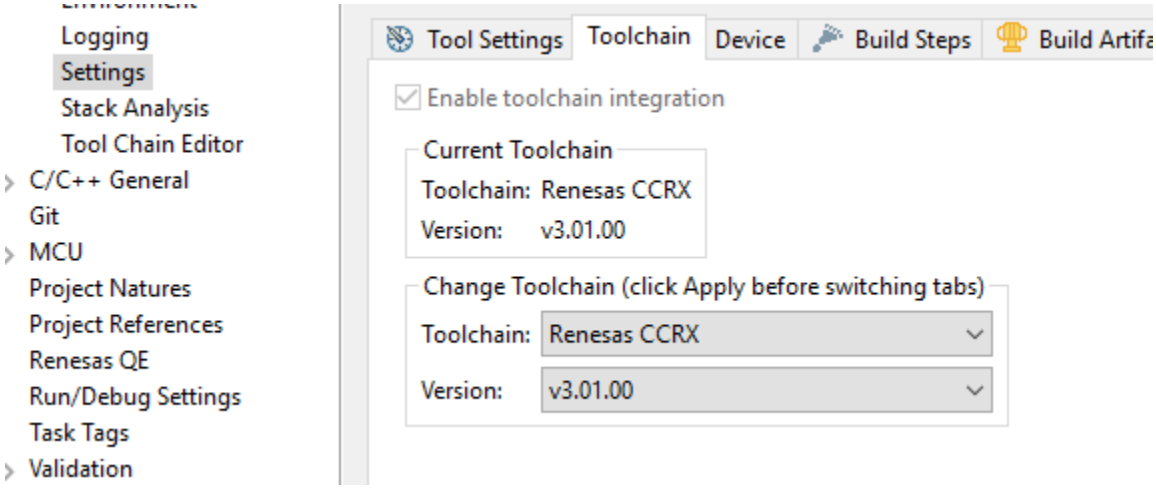
9. 對於這兩個項目，請打開項目屬性。在導覽窗格中，選擇「工具鏈編輯器」。

    a. 選擇「目前」工具鏈。

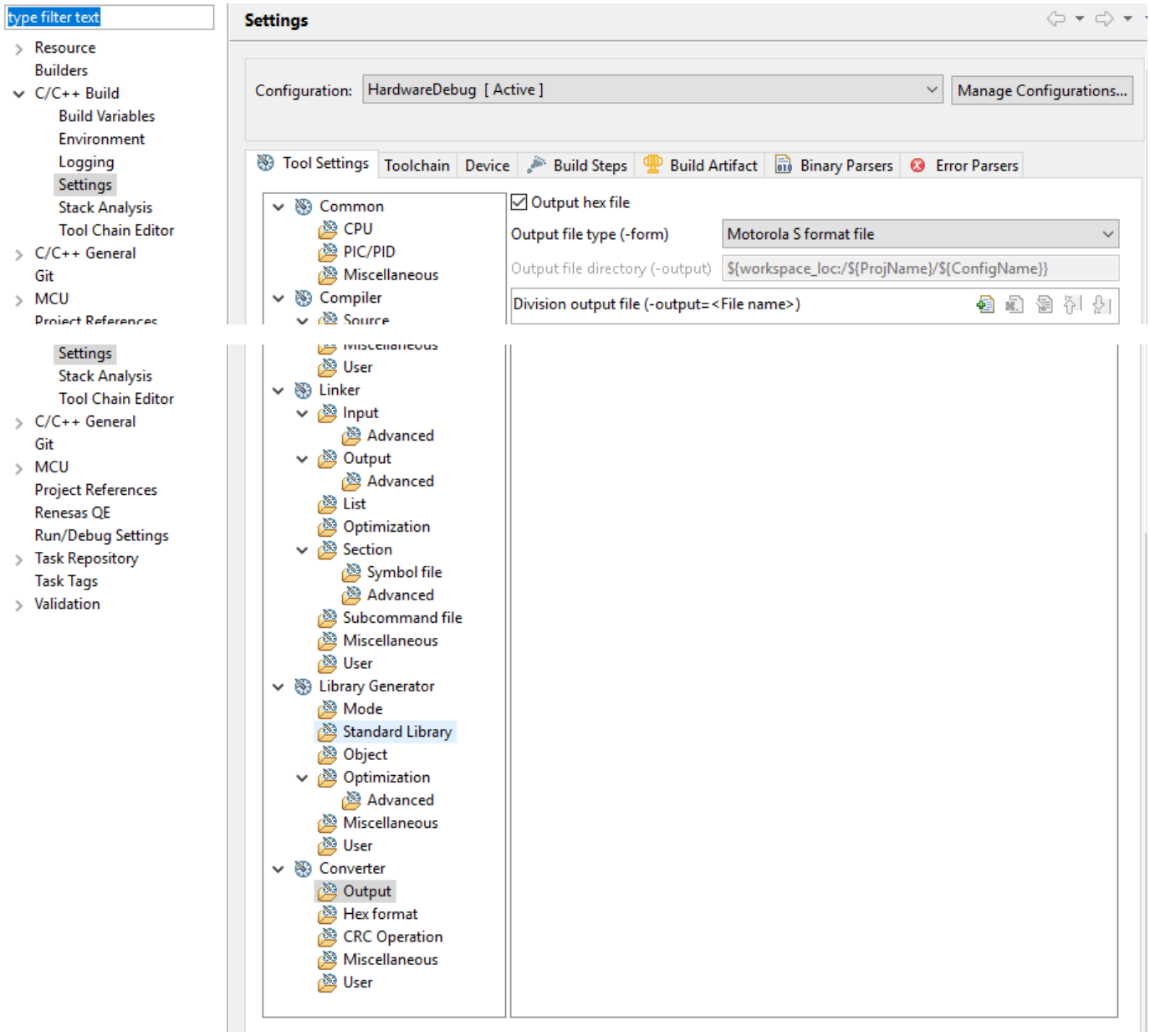
    b. 選擇目前的產生器。



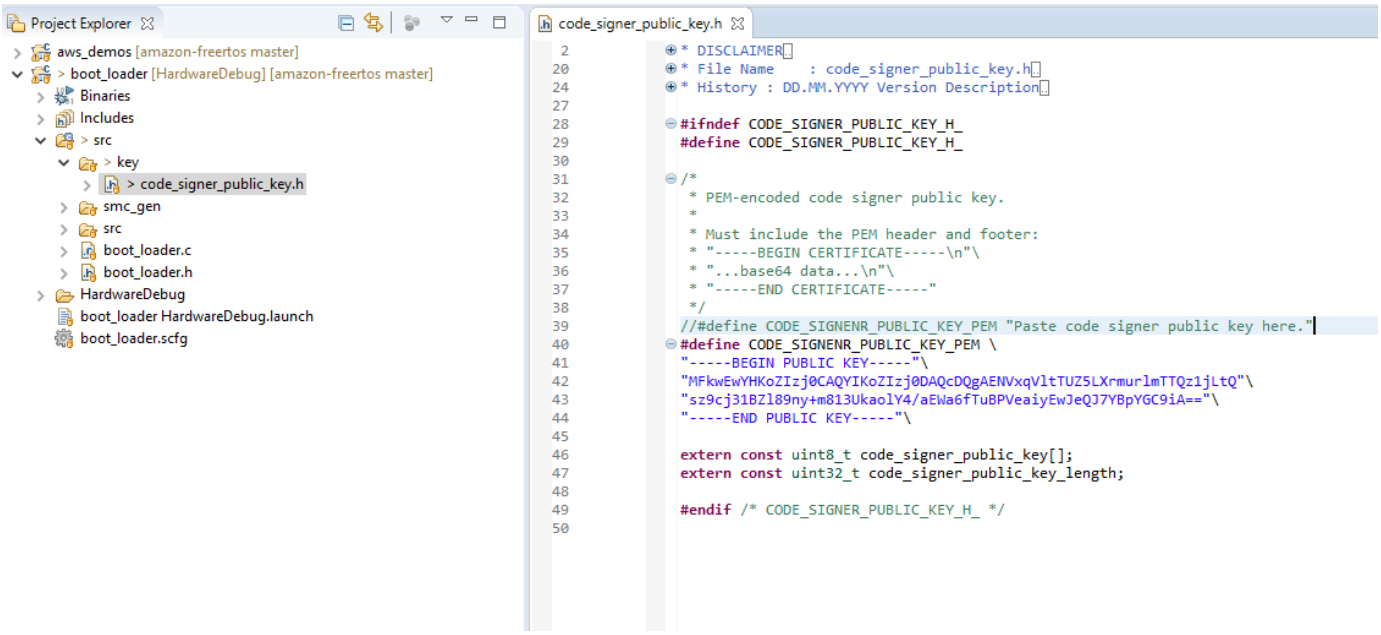
10. 在導覽窗格中，選擇設定。選擇工具鏈標籤，然後選擇工具鏈版本。



選擇工具設定標籤，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. 在引導加載程序項目中，打開`projects\renesas\rx65n-rsk\e2studio\boot_loader\src\key\code_signer_public_key.h`並輸入公鑰。如需如何建立公開金鑰的詳細資訊，請參閱[如何在 RX65N 上使用 Amazon Web Services 實作 FreeRTOS OTA，以及瑞薩 MC U 韌體更新設計政策中的第 7.3 節「使用 OpenSSL 產生 ECDSA-SHA256 金鑰配對」。



然後建立要建立的專案boot_loader.mot。

12. 開啟aws_demos專案。

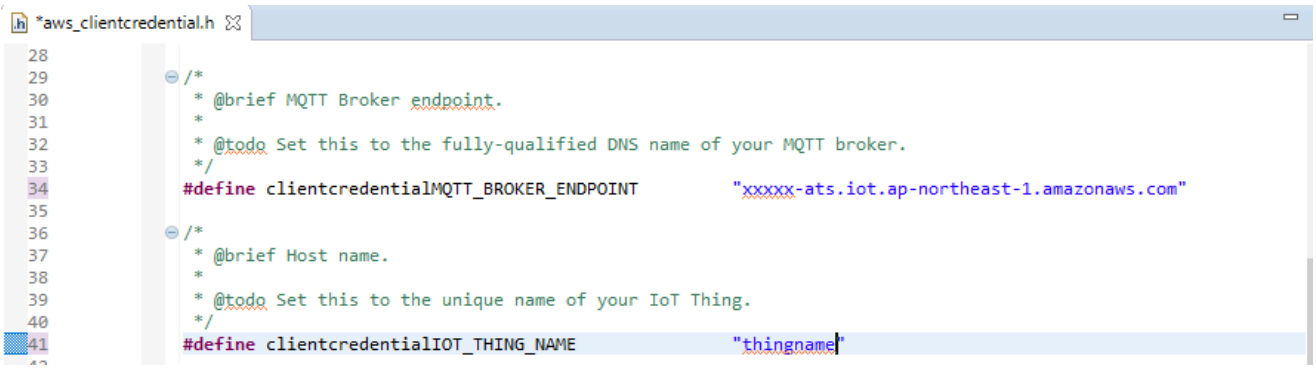
    a. 開啟 AWS IoT 主控台。

    b. 在左側的導覽窗格中，選擇設定。在裝置資料端點文字方塊中記下您的自訂端點。

    c. 選擇 [管理]，然後選擇 [物件]。記下您主機板的 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。

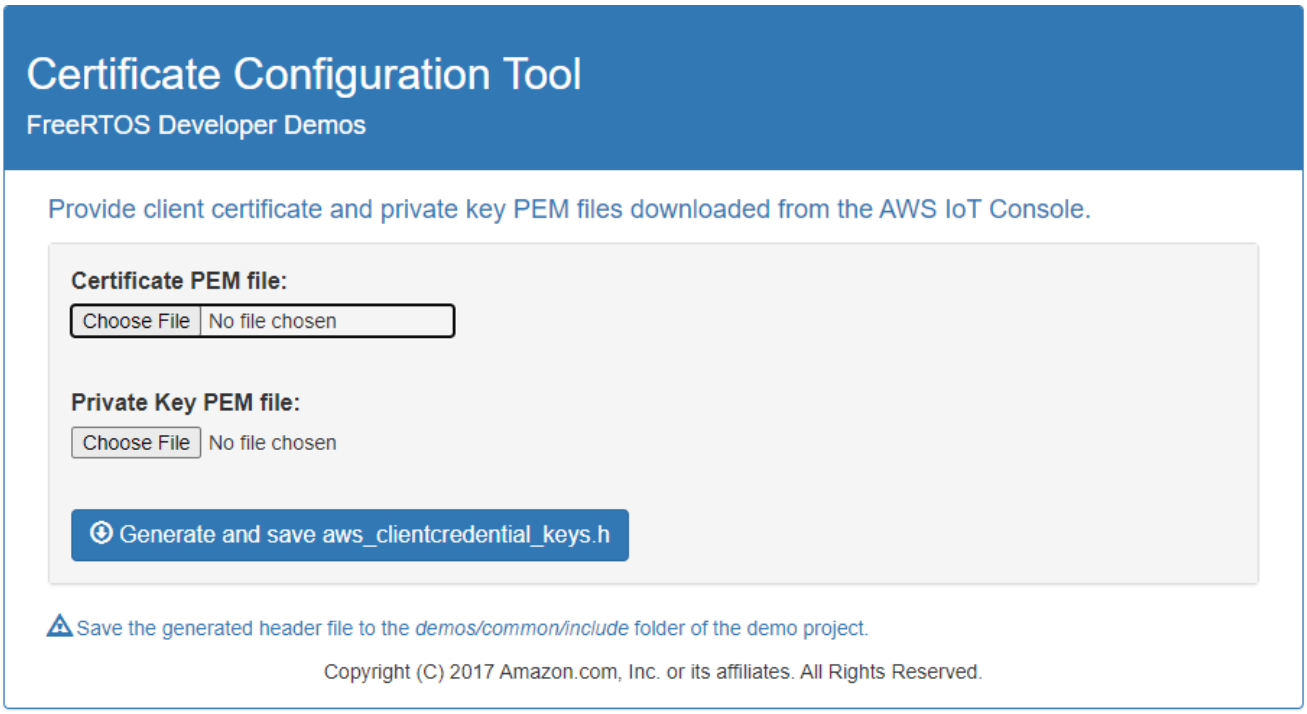
```
#define clientcredentialMQTT_BROKER_ENDPOINT[] = "Your AWS IoT endpoint";
#define clientcredentialIOT_THING_NAME "The AWS IoT thing name of your board"
```



    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。

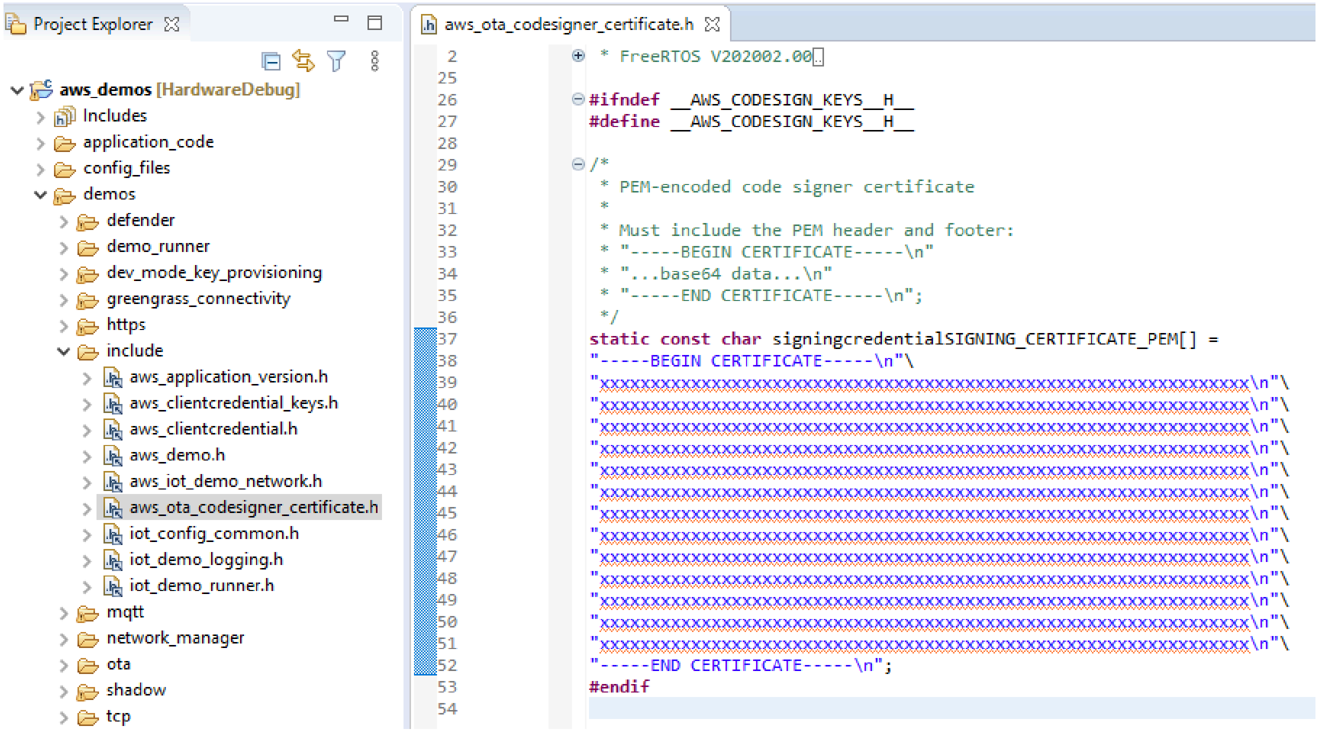
g. 選擇產生並儲存 aws_Client 認證 _keys .h，並取代目錄中的這個檔案。demos/include/



h. 開啟vendors/renesas/boards/rx65n-rsk/aws_demos/config_files/ota_demo_config.h檔案，並指定這些值。

```
#define otapalconfigCODE_SIGNING_CERTIFICATE [] = "your-certificate-key";
```

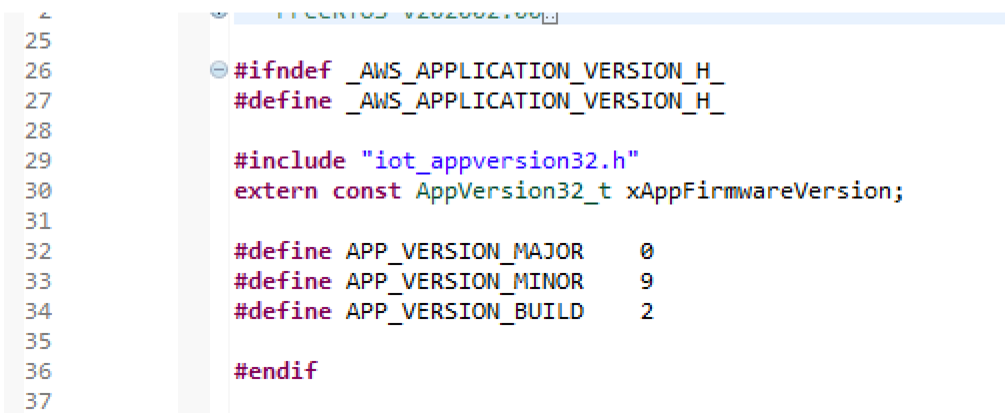
######是文件中的值。secp256r1.crt請記住在認證中的每一行之後添加「\」。如需建立secp256r1.crt檔案的詳細資訊，請參閱如何在 RX65N 上使用 Amazon Web Services 實作 FreeRTOS OTA，以及瑞薩 MC U 韌體更新設計政策中的第 7.3 節「使用 OpenSSL 產生 ECDSA-SHA256 金鑰配對」。



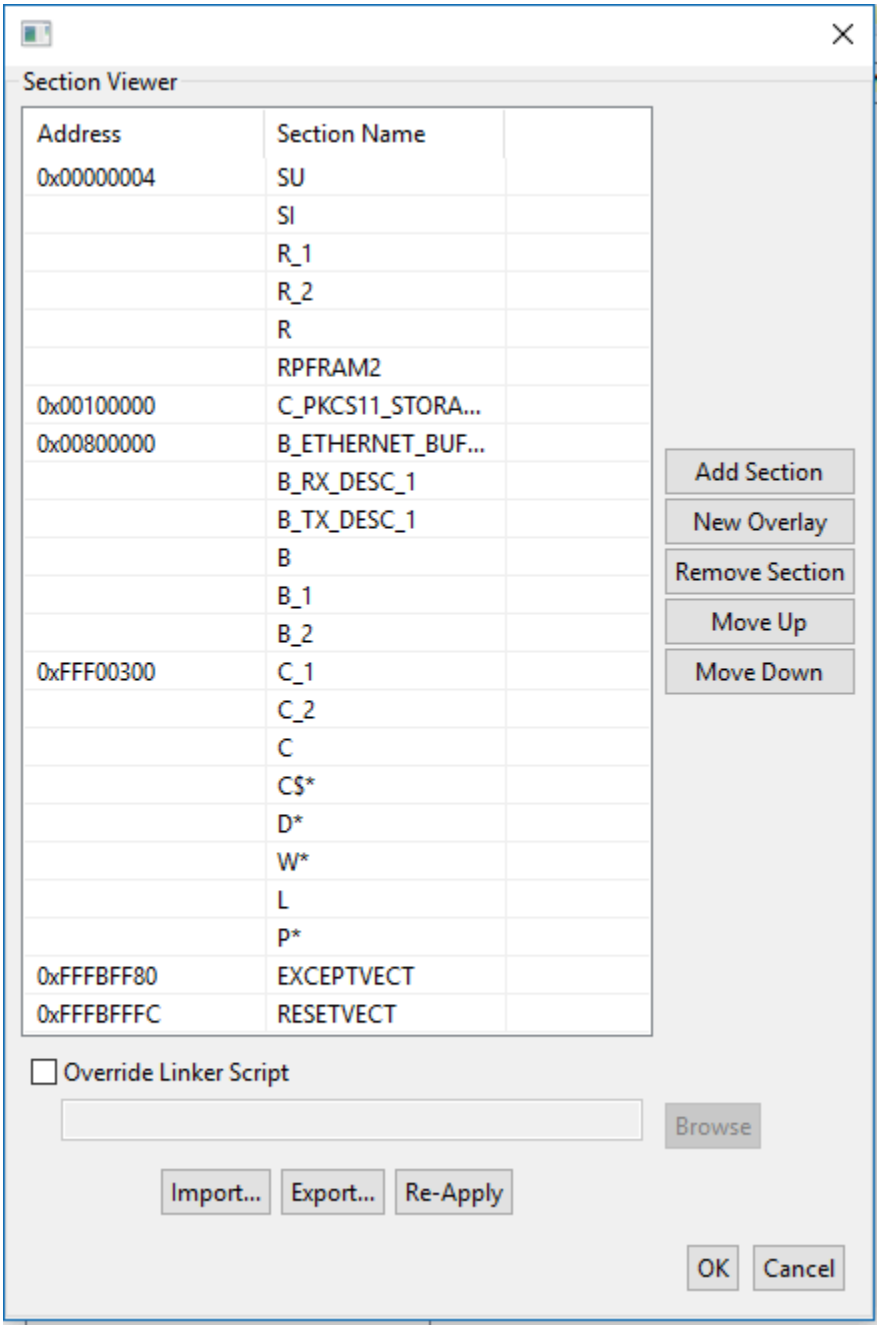
13. 工作 A：安裝韌體的初始版本

a. 開啟vendors/renesas/boards/board/aws_demos/config_files/aws_demo_config.h檔案、註解#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED，然後定義CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED或CONFIG_OTA_HTTP_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。



c. 在 「橫斷面檢視器」 中變更以下設定。



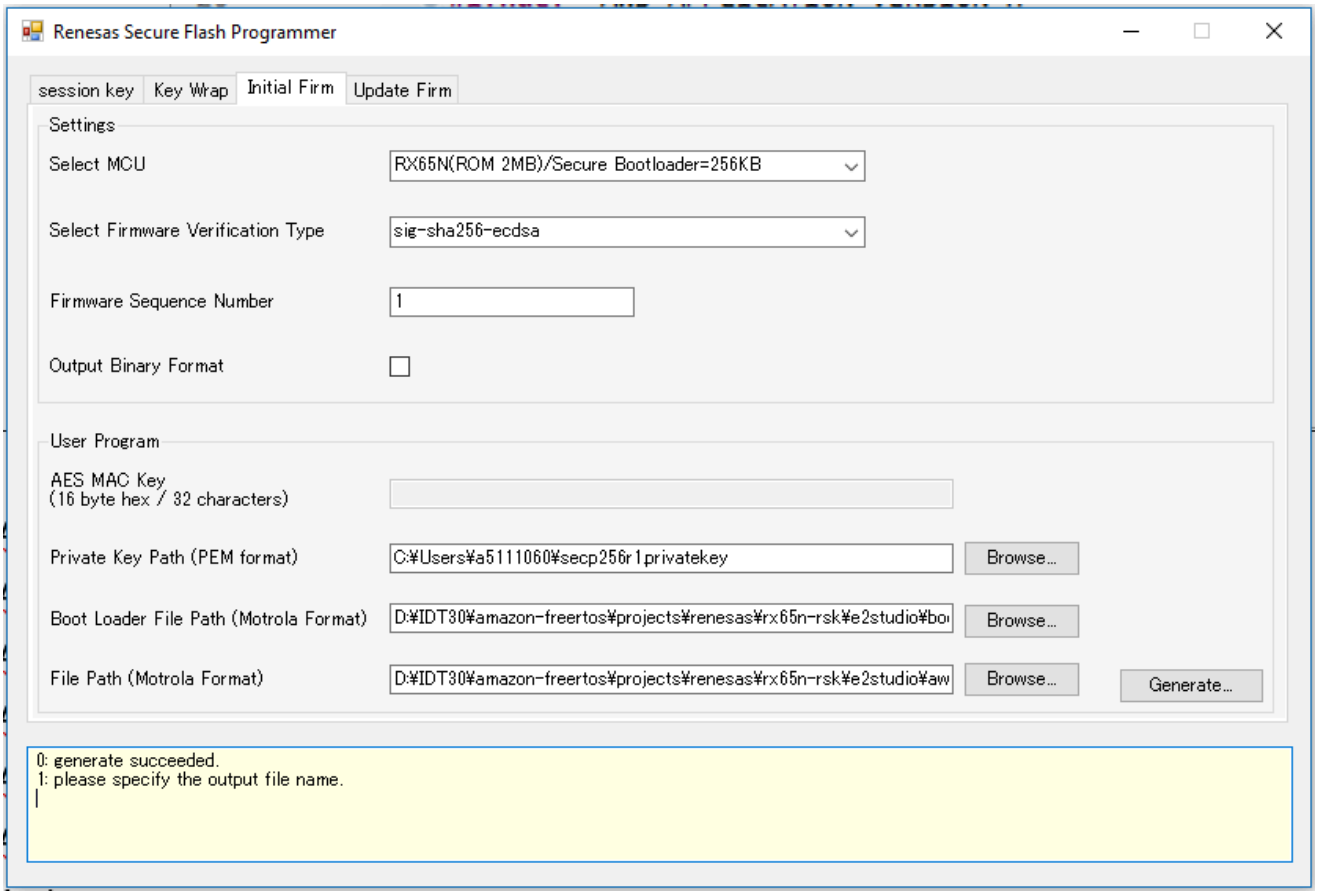
d. 選擇「建置」以建立aws_demos.mot檔案。

14. userprog.mot使用瑞薩安全快閃記憶體程式設計師建立檔案。
userprog.mot是aws_demos.mot與的組合boot_loader.mot。您可以將此檔案更新至 RX65N-RSK 以安裝初始韌體。

a. 下載 https://github.com/renesas/Amazon-FreeRTOS-Tools 並打開Renesas Secure Flash Programmer.exe.

b. 選擇「初始固定」頁標，然後設定下列參數：

- 私密金鑰路徑 — 的位置secp256r1.privatekey。
- 開機載入程式檔案路徑 — boot_loader.mot (projects\renesas\rx65n-rsk\e2studio\boot_loader\HardwareDebug) 的位置。
- 檔案路徑 — aws_demos.mot (projects\renesas\rx65n-rsk\e2studio\aws_demos\HardwareDebug) 的位置。



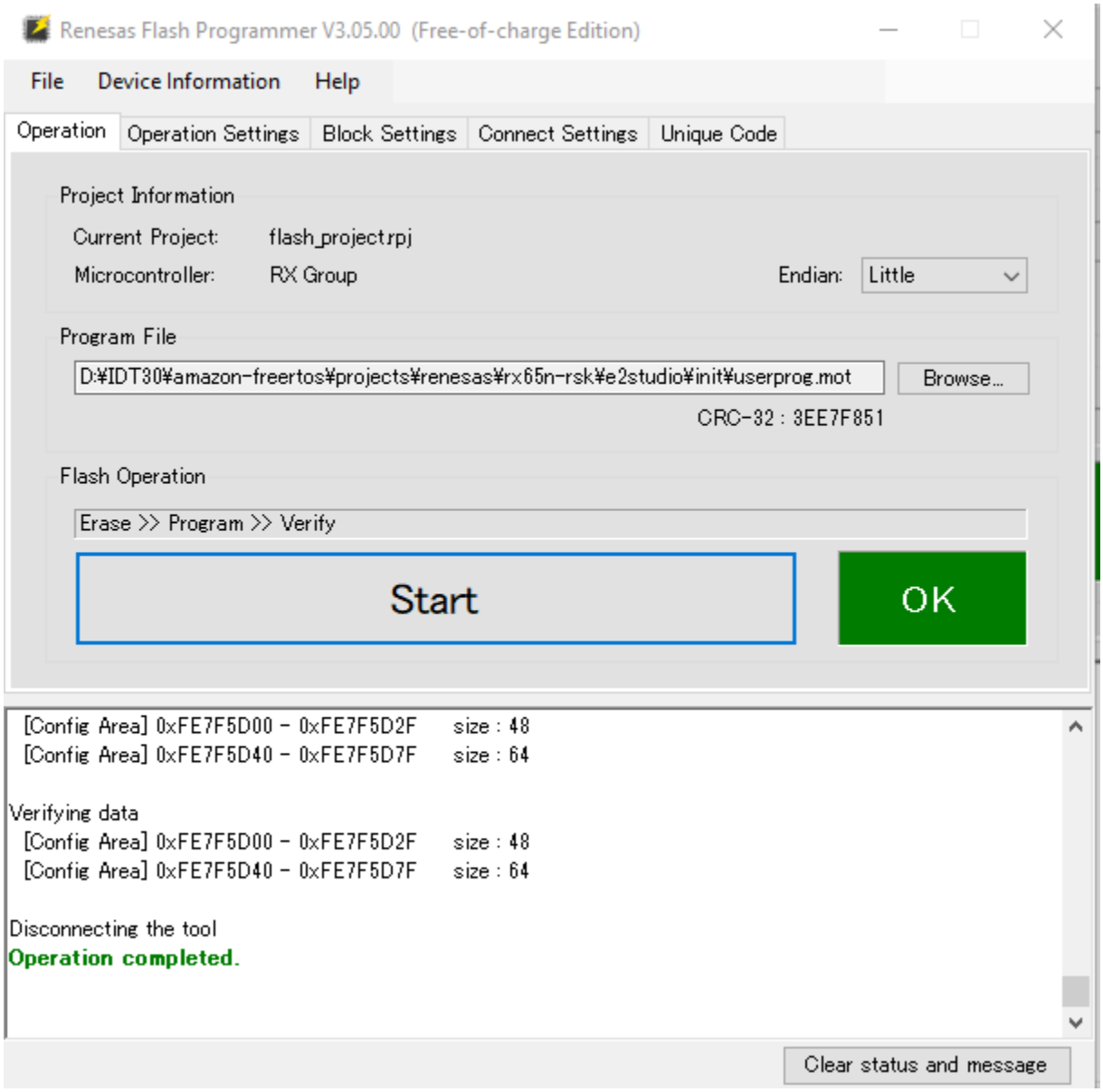
c. 建立名為 init_firmware「產生」的目錄userprog.mot，並將其儲存到目init_firmware錄中。確認產生成功。

15. 刷新 RX65N-RSK 上的初始韌體。

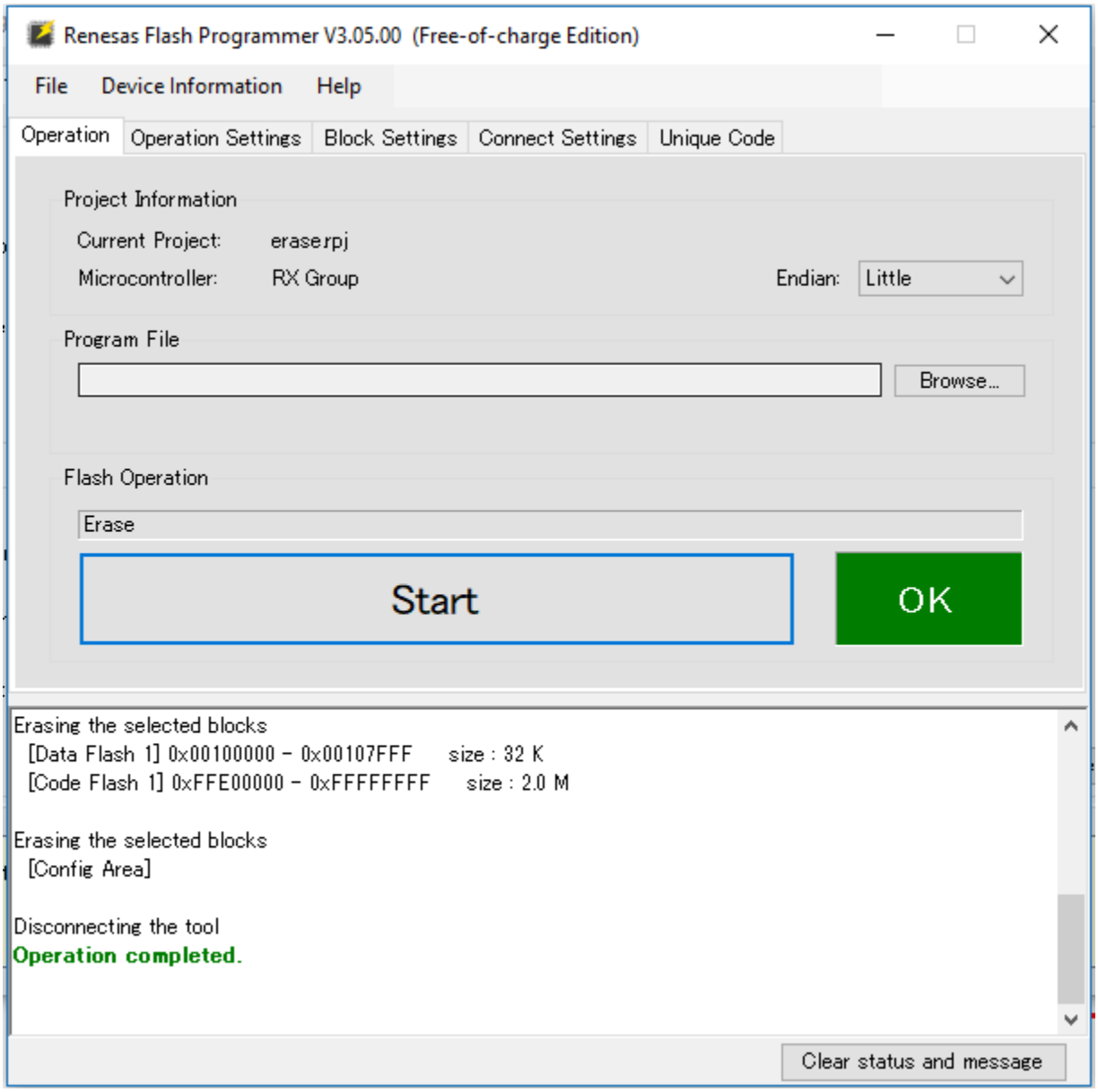
a. 請從 https://www.renesas.com/tw/en/products/software-tools/tools/programmer/renesas-flash-programmer-programming-gui.html 下載最新版本的瑞薩快閃記憶體程式設計師 (程式設計 GUI)。

b. 打開文vendors\renesas\rx_mcu_boards\boards\rx65n-rsk\aws_demos\flash_project\erase_from_bank\ erase.rpj件以擦除銀行上的數據。

c. 選擇 [開始] 以清除銀行。



d. 若要閃爍userprog.mot，請選擇 [瀏覽...] 並導航到目init_firmware錄，選擇userprog.mot文件並選擇開始。



16. 韌體版本 0.9.2 (初始版本) 已安裝至您的 RX65N-RSK。RX65N-RSK 板現在正在監聽 OTA 更新。如果您已在電腦上開啟 Tera Term，當初始韌體執行時，您會看到類似以下內容。

```
-------------------------------------------------
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. 工作 B：更新韌體版本

    a. 開啟demos/include/aws_application_version.h檔案並將APP_VERSION_BUILD權杖值遞增至0.9.3。

    b. 重新建置專案。

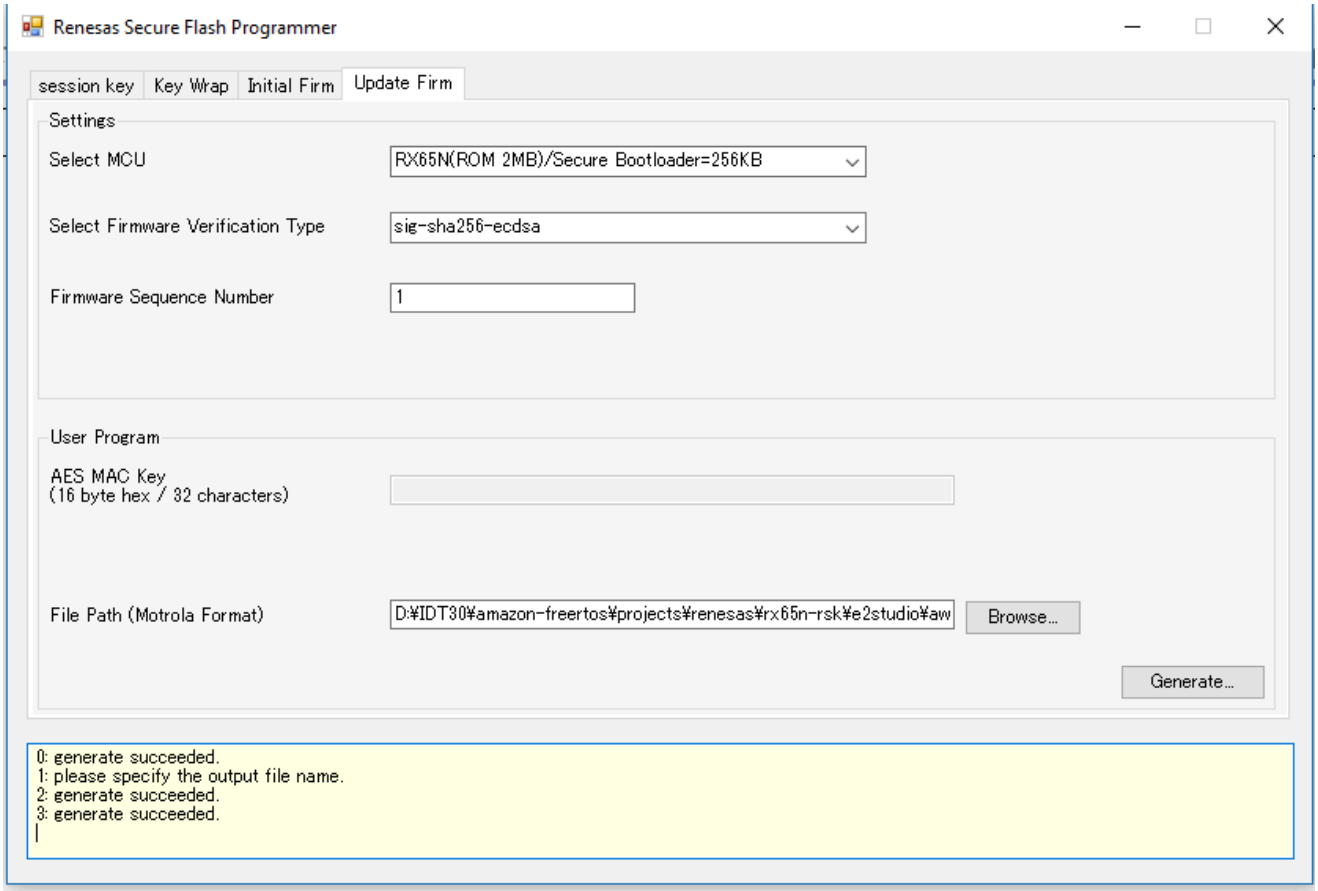
18. 使用瑞薩安全快閃記憶體程式設計師建立userprog.rsu檔案，以更新韌體版本。

    a. 開啟 Amazon-FreeRTOS-Tools\Renesas Secure Flash Programmer.exe 檔案。

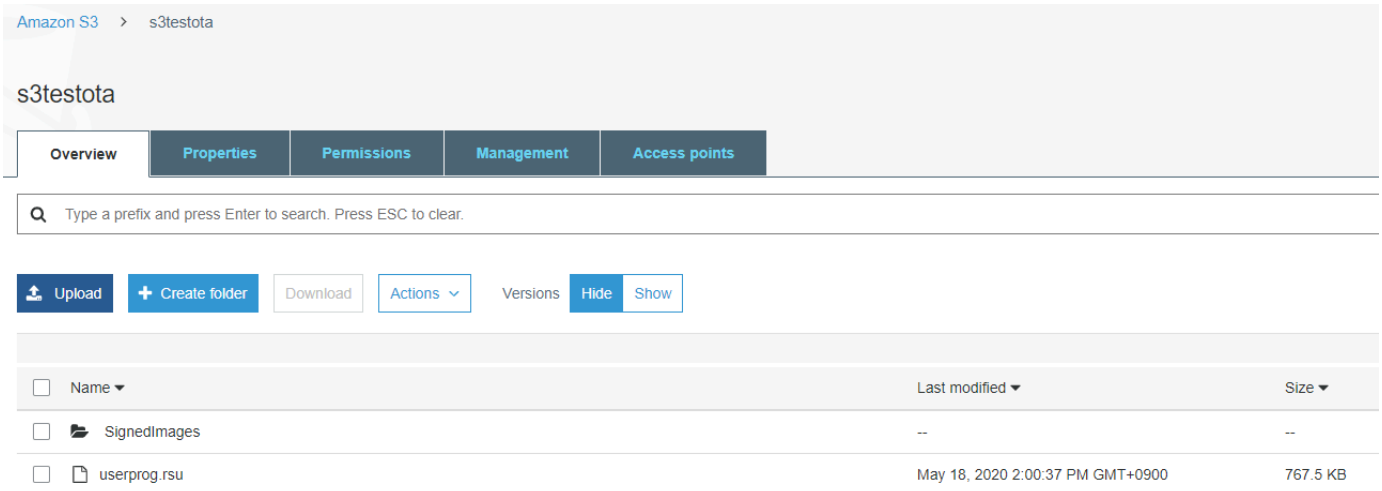
    b. 選擇「更新固定」頁標，並設定下列參數：

    - 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。生成userprog.rsu並將其保存到目update_firmware錄中。確認產生成功。



19. 如中所述userproj.rsu，將韌體更新上傳到 Amazon S3 儲存貯體建立 Amazon S3 儲存貯體來存放您的更新。



20. 建立工作以更新 RX65N-RSK 上的韌體。

    AWS IoT Jobs 是一種服務，用於通知一個或多個連接的設備有待處理的 Job。工作可用於管理裝置群組、更新裝置上的韌體和安全憑證，或執行系統管理工作，例如重新啟動裝置和執行診斷。

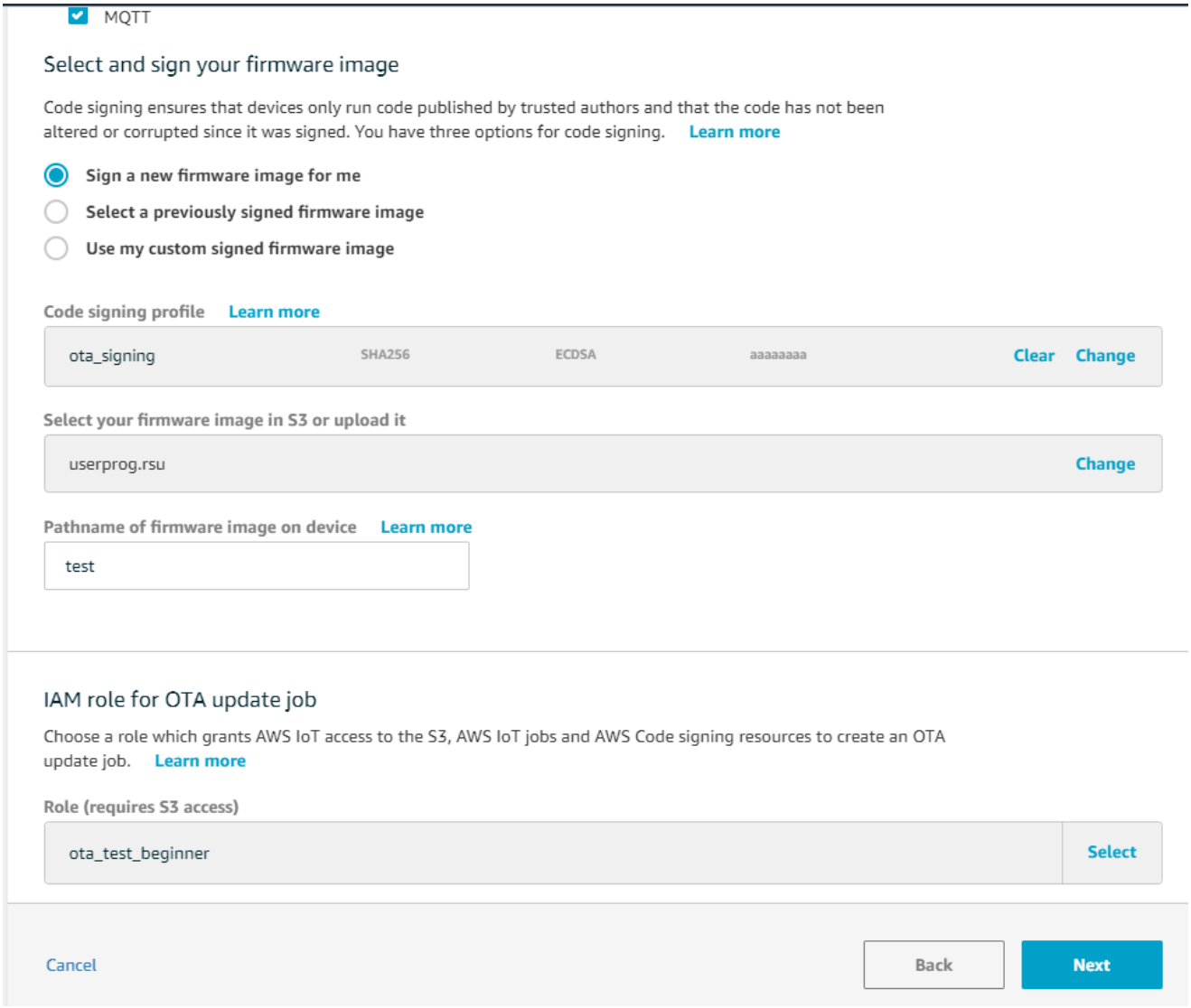
    a. 登入 AWS IoT 主控台。在導覽窗格中，選擇 [管理]，然後選擇 [工作]。

    b. 選擇「建立工作」，然後選擇「建立 OTA 更新工作」。選取項目，然後選擇「下一步」。

    c. 建立 FreeRTOS OTA 更新工作，如下所示：

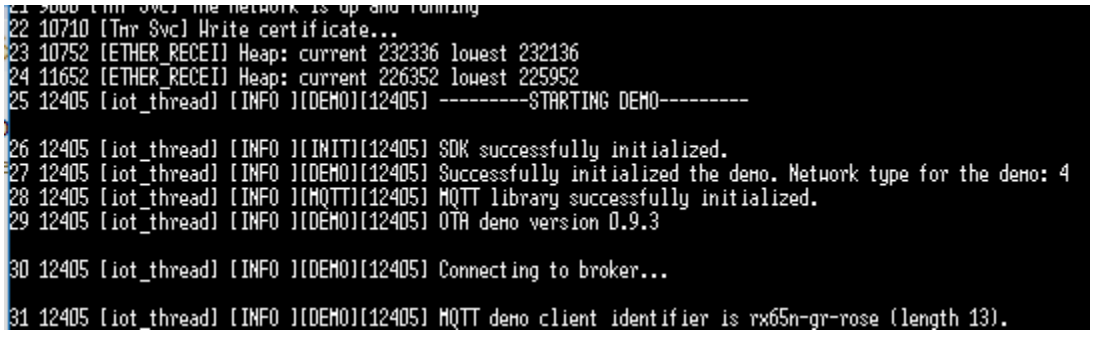
    - 選擇 MQTT。
    - 選取您在上一節中建立的程式碼簽章設定檔。
    - 選取您上傳到 Amazon S3 儲存貯體的韌體映像。
    - 針對裝置上韌體影像的路徑名稱，輸入**`test`**。
    - 選擇您在上一節中建立的 IAM 角色。

    d. 選擇下一步。

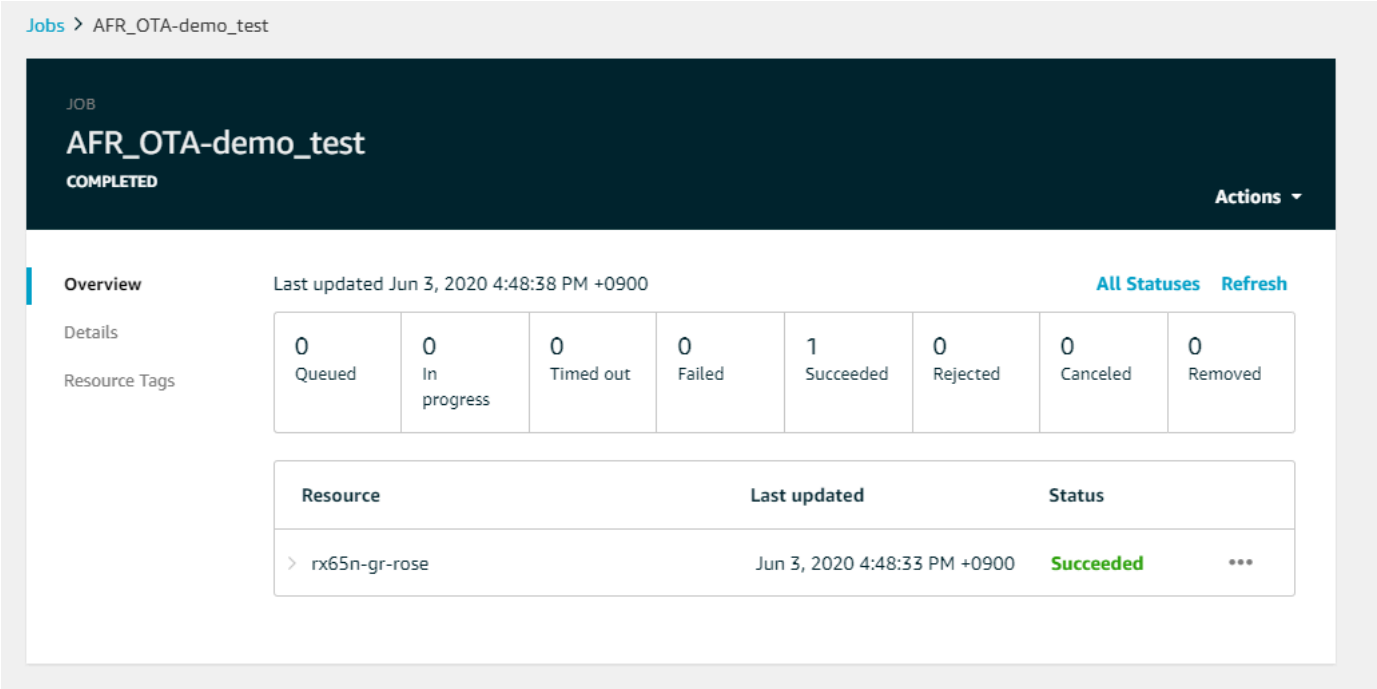


e. 輸入 ID，然後選擇 [建立]。

21. 重新打開 Tera 術語以驗證固件是否已成功更新為 OTA 演示版本 0.9.3。



22. 在 AWS IoT 主控台上，確認工作狀態為 [成功]。



教程：使用 FreeRTOS 藍牙低功耗在濃縮咖啡 ESP32 上執行 OTA 更新

> **Important**
>
> 此參考集成託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱亞馬遜自由 Github 存儲庫遷移指南.

本教程將向您展示如何更新連接到 Android 設備上的 MQTT 藍牙低功耗代理的濃縮咖啡 ESP32 微控制器。它使用AWS IoT Over-the-air （OTA）更新作業更新設備。該設備連接到AWS IoT使用在 Android 演示應用程序中輸入的 Amazon Cognito 憑據。授權運營商從雲啟動 OTA 更新。當設備通過 Android 演示應用程序連接時，將啟動 OTA 更新，並在設備上更新固件。

FreeRTOS 版本 2019.06.00 主要及更新版本包含藍牙低功耗 MQTT 代理支援，可用於 Wi-Fi 佈建和與 AWS IoT服務的安全連線。通過使用藍牙低功耗功能，您可以構建低功耗設備，無需 Wi-Fi 即可與移動設備配對以進行連接。設備可以通過使用通用訪問配置文件（GAP）和通用屬性（GATT）配置文件的 Android 或 iOS 低功耗藍牙 SDK 進行連接來使用 MQTT 進行通信。

以下是我們將遵循的步驟，以允許通過藍牙低功耗進行 OTA 更新：

1. 設定儲存：建立 Amazon S3 儲存貯體和政策，並設定可執行更新的使用者。

2. 建立程式碼簽署憑證：建立簽署憑證，並允許使用者簽署韌體更新。
3. 設定 Amazon Cognito 身份驗證：建立登入資料提供者、使用者集區和應用程式對使用者集區的存取權。
4. 設定 FreeRTOS：設定低功耗藍牙、用戶端認證和程式碼簽署公用憑證。
5. 配置 Android 應用程序：設置憑據提供程序，用戶池並將應用程序部署到 Android 設備。
6. 運行 OTA 更新腳本：要啟動 OTA 更新，請使用 OTA 更新腳本。

如需有關更新如何運作的詳細資訊，請參閱空中免費更新。如需有關如何設定藍牙低功耗 MQTT 代理功能的其他資訊，請參閱 Richard Kang 在 Espressif ESP32 上使用藍牙低功耗搭配 FreeRTOS 文章。

先決條件

若要執行此教學課程中的步驟，您需要以下資源：

- 一個 ESP32 開發板。
- 一條微型 USB 連接 USB 連接線。
- AWS帳戶（免費方案就足夠了）。
- 具備安卓 v 6.0 或更高版本的安卓手機，以及藍牙 4.2 或更高版本。

在您的開發計算機上，您需要：

- 適用於 Xtensa 工具鏈和 FreeRTOS 原始程式碼和範例的足夠磁碟空間 (約 500 Mb)。
- 安卓工作室安裝。
- 已AWS CLI安裝。
- 已安裝蟒蛇 3。
- 適用於 Python 的博托 3AWS 軟件開發工具包（SDK）。

本教學課程中的步驟假設 Xtensa 工具鏈、ESP-IDF 和 FreeRTOS 程式碼已安裝在主/esp目錄中的目錄中。您必須新增~/esp/xtensa-esp32-elf/bin至$PATH變數。

步驟 1：設定儲存

1. 建立 Amazon S3 儲存貯體來存放您的更新啟用版本控制以保存固件映像。

2. 建立 OTA 更新服務角色並將下列受管政策中新增至該角色：

   - AWSIotLogging
   - AWSIotRuleActions
   - AWSIotThingsRegistration
   - AWSFreeRTOSOTAUpdate

3. 創建一個可以執行 OTA 更新的用戶。此使用者可在帳戶中簽署並部署韌體更新至 IoT 裝置，並可存取在所有裝置上執行 OTA 更新。存取權限應限於受信任的實體。
4. 按照步驟操作建立 OTA 使用者政策並將其附加到您的用戶。

步驟 2：建立程式碼簽名憑證

1. 建立已啟用版本控制的 Amazon S3 儲存貯體來存放韌體映像。
2. 建立可用於簽署韌體的程式碼簽署憑證。請記下憑證匯入憑證時的憑證 Amazon Resource Name (ARN)。

```
aws acm import-certificate --profile=ota-update-user --certificate file://
ecdsasigner.crt --private-key file://ecdsasigner.key
```

輸出範例：

```
{
"CertificateArn": "arn:aws:acm:us-east-1:<account>:certificate/<certid>"
}
```

您稍後將使用 ARN 來創建簽名配置文件。如果需要，您現在可以使用以下命令創建配置文件：

```
aws signer put-signing-profile --profile=ota-update-user --profile-
name esp32Profile --signing-material certificateArn=arn:aws:acm:us-
east-1:account:certificate/certid --platform AmazonFreeRTOS-Default --signing-
parameters certname=/cert.pem
```

輸出範例：

```
{
"arn": "arn:aws:signer::<account>:/signing-profiles/esp32Profile"
}
```

步驟 3：Amazon Cognito 份驗證配置

建立 AWS IoT 政策

1. 登入 AWS IoT 主控台。
2. 在主機的右上角，選擇 My registry (我的帳戶)。在「帳戶設定」 下，記下您的 12 位數帳戶 ID。
3. 在左側的導覽窗格中，選擇 Settings (設定)。在裝置資料端點中，記下端點值。端點外觀大致如下`xxxxxxxxxxxxxx.iot.us-west-2.amazonaws.com`。在此範例中，「AWS區域」為「US-西 -2」。
4. 在左側導覽窗格中，選擇 [安全]，選擇 [原則]，然後選擇 [建立]。如果您的帳戶中沒有任何政策，您會看到「您還沒有任何政策」的訊息，您可以選擇 [建立政策]。
5. 輸入原則的名稱，例如，「政策」。
6. 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。`aws-account-id`使用您的帳戶 ID 和您`aws-region`的地區取代 (例如，「us-west-2")。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:Connect",
      "Resource": "arn:aws:iot:aws-region:aws-account-id:*"
    },
    {
      "Effect": "Allow",
      "Action": "iot:Publish",
      "Resource": "arn:aws:iot:aws-region:aws-account-id:*"
    },
    {
      "Effect": "Allow",
      "Action": "iot:Subscribe",
      "Resource": "arn:aws:iot:aws-region:aws-account-id:*"
    },
    {
      "Effect": "Allow",
      "Action": "iot:Receive",
      "Resource": "arn:aws:iot:aws-region:aws-account-id:*"
    }
  ]
```

```
}
```

7. 選擇 建立 。

建立物AWS IoT件

1. 登入 AWS IoT 主控台。
2. 在左側導覽窗格中，選擇 Manage (管理)，然後選擇 Things (實物)。
3. 選擇右上角的「建立」。如果您的帳戶中沒有註冊任何項目，則會顯示「您還沒有任何物件」的訊息，您可以選擇註冊物件。
4. 在 Creating AWS IoT things (建立 IoT 物件) 頁面上，選擇 Create a single thing (建立單一物件)。
5. 在 Add registry (將裝置新增至物件登錄檔) 頁面上，輸入您物件的名稱 (例如 esp32-ble)。只允許英數字元、連字號 (-) 和底線 (_) 字元。選擇 下一步。
6. 在 [為您的物件新增憑證] 頁面的 [略過憑證並建立物件] 下，選擇 [建立不含憑證的物件]。由於我們使用使用 Amazon Cognito 登入資料進行身份驗證和授權的 BLE 代理行動應用程式，因此不需要裝置憑證。

創建一個亞馬遜應用程序客戶端

1. 登入 Amazon Cognito 主控台。
2. 在右上方導覽橫幅中，選擇 [建立使用者集區]。
3. 輸入儲存池名稱 (例如，「儲存池」)。
4. 選擇 Review defaults (檢閱預設值)。
5. 在 [應用程式用戶端] 中，選擇 [新增應用程式用戶端]，然後選擇 [新增應用
6. 輸入應用程式用戶端名稱 (例如「mqtt_app_Client」)。
7. 確定已選取 [產生用戶端密碼]。
8. 選擇 Create app client (建立應用程式用戶端)。
9. 選擇 Return to pool details (回到集區的詳細資訊)。
10. 在使用者集區的 [檢閱] 頁面上，選擇 [建立集區]。您應該會看到一則訊息，指出您的使用者集區已成功建立。請記下集區 ID。
11. 在導覽窗格中，選擇 App 用戶端。
12. 選擇顯示詳細資料。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. 登入 Amazon Cognito 主控台。
2. 選擇 Create new identity pool (建立新的身分集區)。
3. 輸入識別集區的名稱 (例如，「mqtt_PROXY」)。
4. 展開驗證提供者。
5. 選擇「Co gnito」索引標籤。
6. 輸入您在先前步驟中記下的使用者集區 ID 和應用程式用戶端 ID。
7. 選擇 Create Pool (建立集區)。
8. 在下一頁上，若要為已驗證和未驗證身分建立新角色，請選擇 [允許]。
9. 記下格式的識別集區 IDus-east-1:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx。

將 IAM 政策連接到已驗證的身分

1. 開啟 Amazon Cognito 主控台。
2. 選取您剛才建立的身分集區 (例如，「mqtt_proxy_id」)。
3. 選擇 Edit identity pool (編輯身分集區)。
4. 記下指派給已驗證角色的 IAM 角色 (例如，「驗證角色」)。
5. 開啟 IAM 主控台。
6. 在導覽窗格中，選擇 Roles (角色)。
7. 搜尋指定的角色 (例如，「核心角色」)，然後加以選取。
8. 選擇 [新增內嵌原則]，然後選擇 [JSON]。
9. 輸入下列政策：

```
{
       "Version": "2012-10-17",
       "Statement": [
       {
          "Effect": "Allow",
          "Action": [
             "iot:AttachPolicy",
             "iot:AttachPrincipalPolicy",
             "iot:Connect",
             "iot:Publish",
             "iot:Subscribe"
```

```
            ],
            "Resource": "*"
        }]
    }
```

10. 選擇 Review Policy (檢閱政策)。
11. 輸入政策名稱 (例如mqttProxyCognito政策)。
12. 選擇 建立政策。

步驟 4：設定 Amazon FreeRTOS

1. 從免費伺服器 GitHub 軟體庫下載最新版本的亞馬遜 FreeRTOS 程式碼。
2. 若要啟用 OTA 更新示範，請依照中的步驟操作開始使用意式濃縮咖啡 ESP32-DevKit C 和歐洲環保工具組。
3. 在下列檔案中進行這些額外的修改：

   a. 開啟vendors/espressif/boards/esp32/aws_demos/config_files/aws_demo_config.h並定義CONFIG_OTA_UPDATE_DEMO_ENABLED。

   b. 開啟vendors/espressif/boards/esp32/aws_demos/common/config_files/aws_demo_config.h並變更democonfigNETWORK_TYPES為AWSIOT_NETWORK_TYPE_BLE。

   c. 開啟demos/include/aws_clientcredential.h並輸入的端點URLclientcredentialMQTT_BROKER_ENDPOINT。

      輸入您的物件名稱clientcredentialIOT_THING_NAME (例如，「esp32」)。使用 Amazon Cognito 登入資料時，不必新增憑證。

   d. 打開vendors/espressif/boards/esp32/aws_demos/config_files/aws_iot_network_config.h並更configENABLED_NETWORKS改configSUPPORTED_NETWORKS和僅包括AWSIOT_NETWORK_TYPE_BLE。

   e. 開啟檔vendors/*vendor*/boards/*board*/aws_demos/config_files/ota_demo_config.h案，然後輸入您的憑證。

      ```
      #define otapalconfigCODE_SIGNING_CERTIFICATE [] = "your-certificate-key";
      ```

   應用程序應該啟動並打印演示版本：

```
11 13498 [iot_thread] [INFO ][DEMO][134980] Successfully initialized the demo.
 Network type for the demo: 2
12 13498 [iot_thread] [INFO ][MQTT][134980] MQTT library successfully initialized.
13 13498 [iot_thread] OTA demo version 0.9.20
14 13498 [iot_thread] Creating MQTT Client...
```

步驟 5：設定安卓應用

1. 從 amazon-freertos-ble-android-sdk 軟件 GitHub 庫下載安卓低功耗藍牙 SDK 和示例應用程序。
2. 開啟檔案app/src/main/res/raw/awsconfiguration.json並填寫集區 ID、區域 AppClientId，並 AppClientSecret 使用下列 JSON 範例中的指示。

```
{
  "UserAgent": "MobileHub/1.0",
  "Version": "1.0",
  "CredentialsProvider": {
    "CognitoIdentity": {
      "Default": {
        "PoolId": "Cognito->Manage Identity Pools->Federated Identities-
>mqtt_proxy_identity_pool->Edit Identity Pool->Identity Pool ID",
        "Region": "Your region (for example us-east-1)"
      }
    }
  },

  "IdentityManager": {
    "Default": {}
  },

  "CognitoUserPool": {
    "Default": {
      "PoolId": "Cognito-> Manage User Pools -> esp32_mqtt_proxy_user_pool ->
 General Settings -> PoolId",
      "AppClientId": "Cognito-> Manage User Pools -> esp32_mqtt_proxy_user_pool ->
 General Settings -> App clients ->Show Details",
      "AppClientSecret": "Cognito-> Manage User Pools -> esp32_mqtt_proxy_user_pool
 -> General Settings -> App clients ->Show Details",
      "Region": "Your region (for example us-east-1)"
    }
  }
```

```
}
```

3. 開啟app/src/main/java/software/amazon/freertos/DemoConstants.java並輸入您先前建立的原則名稱 (例如，*esp32_mqtt_proxy_iot_##*) 以及 [區域] (例如，*##-## -1*)。
4. 構建並安裝演示應用程序。
   a. 在 Android 工作室中，選擇生成，然後選擇製作模塊應用程序。
   b. 選擇運行，然後運行應用程序。你可以去日誌貓窗格中的 Android 工作室監視日誌消息。
   c. 在 Android 設備上，從登錄屏幕創建一個帳戶。
   d. 建立使用者。如果使用者已存在，請輸入認證。
   e. 允許 Amazon FreeRTOS 演示訪問設備的位置。
   f. 掃描藍牙低功耗設備。
   g. 將找到的裝置的滑桿移至「開啟」。
   h. 按下 ESP32 的序列埠除錯主控台上的 y。
   i. 選擇 [配對並 Connect]。
5. 更多... 建立連線後，連結會變成作用中的狀態。連接完成後，連接狀態應更改為「BLE_CONNECTED」在 Android 設備日誌中：

```
2019-06-06 20:11:32.160 23484-23497/software.amazon.freertos.demo I/FRD: BLE
 connection state changed: 0; new state: BLE_CONNECTED
```

6. 在可以傳輸消息之前，Amazon FreeRTOS 備和安卓設備就 MTU 進行了協商。您應該會在 logcat 中看到下列輸出：

```
2019-06-06 20:11:46.720 23484-23497/software.amazon.freertos.demo I/FRD:
 onMTUChanged : 512 status: Success
```

7. 設備連接到應用程序，並開始使用 MQTT 代理發送 MQTT 消息。若要確認裝置可以通訊，請確定 MQTT_CONT_CONTROL 特性資料值變更為 01：

```
2019-06-06 20:12:28.752 23484-23496/software.amazon.freertos.demo D/FRD: <-<-<-
 Writing to characteristic: MQTT_CONTROL with data: 01
2019-06-06 20:12:28.839 23484-23496/software.amazon.freertos.demo D/FRD:
 onCharacteristicWrite for: MQTT_CONTROL; status: Success; value: 01
```

8. 裝置配對完成後，您會在 ESP32 主控台上看到提示。若要啟用 BLE，請按 y。在您執行此步驟之前，示範將無法運作。

```
E (135538) BT_GATT: GATT_INSUF_AUTHENTICATION: MITM Required
W (135638) BT_L2CAP: l2cble_start_conn_update, the last connection update command
 still pending.
E (135908) BT_SMP: Value for numeric comparison = 391840
15 13588 [InputTask] Numeric comparison:391840
16 13589 [InputTask] Press 'y' to confirm
17 14078 [InputTask] Key accepted
W (146348) BT_SMP: FOR LE SC LTK IS USED INSTEAD OF STK
18 16298 [iot_thread] Connecting to broker...
19 16298 [iot_thread] [INFO ][MQTT][162980] Establishing new MQTT connection.
20 16298 [iot_thread] [INFO ][MQTT][162980] (MQTT connection 0x3ffd5754, CONNECT
 operation 0x3ffd586c) Waiting for operation completion.
21 16446 [iot_thread] [INFO ][MQTT][164450] (MQTT connection 0x3ffd5754, CONNECT
 operation 0x3ffd586c) Wait complete with result SUCCESS.
22 16446 [iot_thread] [INFO ][MQTT][164460] New MQTT connection 0x3ffc0ccc
 established.
23 16446 [iot_thread] Connected to broker.
```

步驟 6：執行 OTA 更新指令碼

1. 若要安裝先決條件，請執行下列命令：

   ```
   pip3 install boto3
   ```

   ```
   pip3 install pathlib
   ```

2. 在中增加 FreeRTOS 應用程式版本`demos/include/aws_application_version.h`。
3. 建立新的 .bin 檔案。
4. 下載蟒蛇腳本 start_ota.py. 若要查看指令碼的說明內容，請在終端機視窗中執行下列命令：

   ```
   python3 start_ota.py -h
   ```

   您應該會看到類似下列的內容：

   ```
   usage: start_ota.py [-h] --profile PROFILE [--region REGION]
                       [--account ACCOUNT] [--devicetype DEVICETYPE] --name NAME
                       --role ROLE --s3bucket S3BUCKET --otasigningprofile
                       OTASIGNINGPROFILE --signingcertificateid
   ```

```
                    SIGNINGCERTIFICATEID [--codelocation CODELOCATION]
Script to start OTA update
optional arguments:
-h, --help            show this help message and exit
--profile PROFILE     Profile name created using aws configure
--region REGION       Region
--account ACCOUNT     Account ID
--devicetype DEVICETYPE thing|group
--name NAME           Name of thing/group
--role ROLE           Role for OTA updates
--s3bucket S3BUCKET   S3 bucket to store firmware updates
--otasigningprofile OTASIGNINGPROFILE
                      Signing profile to be created or used
--signingcertificateid SIGNINGCERTIFICATEID
                      certificate id (not arn) to be used
--codelocation CODELOCATION
                      base folder location (can be relative)
```

5. 如果您使用提供的AWS CloudFormation範本來建立資源，請執行下列命令：

```
python3 start_ota_stream.py --profile otausercf --name esp32-ble --role
 ota_ble_iot_role-sample --s3bucket afr-ble-ota-update-bucket-sample --
otasigningprofile abcd --signingcertificateid certificateid
```

您應該會在 ESP32 偵錯主控台中看到更新開始：

```
38 2462 [OTA Task] [prvParseJobDoc] Job was accepted. Attempting to start transfer.
---
49 2867 [OTA Task] [prvIngestDataBlock] Received file block 1, size 1024
50 2867 [OTA Task] [prvIngestDataBlock] Remaining: 1290
51 2894 [OTA Task] [prvIngestDataBlock] Received file block 2, size 1024
52 2894 [OTA Task] [prvIngestDataBlock] Remaining: 1289
53 2921 [OTA Task] [prvIngestDataBlock] Received file block 3, size 1024
54 2921 [OTA Task] [prvIngestDataBlock] Remaining: 1288
55 2952 [OTA Task] [prvIngestDataBlock] Received file block 4, size 1024
56 2953 [OTA Task] [prvIngestDataBlock] Remaining: 1287
57 2959 [iot_thread] State: Active  Received: 5   Queued: 5   Processed: 5
 Dropped: 0
```

6. OTA 更新完成後，設備將根據 OTA 更新過程的要求重新啟動。然後，它會嘗試使用更新的韌體進行連線。如果升級成功，則更新的韌體會標示為作用中，您應該會在主控台中看到更新的版本：

```
13 13498 [iot_thread] OTA demo version 0.9.21
```

AWS IoT 裝置影子示範應用程式

> **Important**
>
> 此演示託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。建議您在建立新專案時從這裡開始。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱亞馬遜自由 Github 存儲庫遷移指南.

簡介

此示範示範如何使用AWS IoT Device Shadow 程式庫連線至 AWSDevice Shadow 服務。它會使用建立與 TLS (相互驗證) 到 MQTT 代理程式和 CoreJSON 程式庫剖析器的AWS IoT MQTT 連線，以剖析從陰影服務接收到的AWS陰影文件。通訊協定圖書館此示範會顯示基本陰影作業，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註冊 CoremQtt 程式庫的回呼函式，以處理訊息，例如陰影/update和從AWS IoT Device Shadow 服務傳送的/update/delta訊息。

此示範僅用於學習練習，因為更新陰影文件 (狀態) 和更新回應的要求都是由相同的應用程式完成的。在實際的生產場景中，即使設備當前未連接，外部應用程序也會遠程請求更新設備的狀態。裝置會在連線時確認更新要求。

> **Note**
>
> 若要設定並執行 FreeRTOS 示範，請遵循中的步驟FreeRTOS 入。

功能

示範會建立單一應用程式工作，以迴圈執行一組範例，這些範例會示範陰影/update和/update/delta回呼，以模擬切換遠端裝置的狀態。它會傳送具有新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/update回調用於打印不斷變化的陰影狀態。此示範也會使用 MQTT 代理程式的安全AWS IoT MQTT 連線，並假設裝置陰影中有power0n狀態。

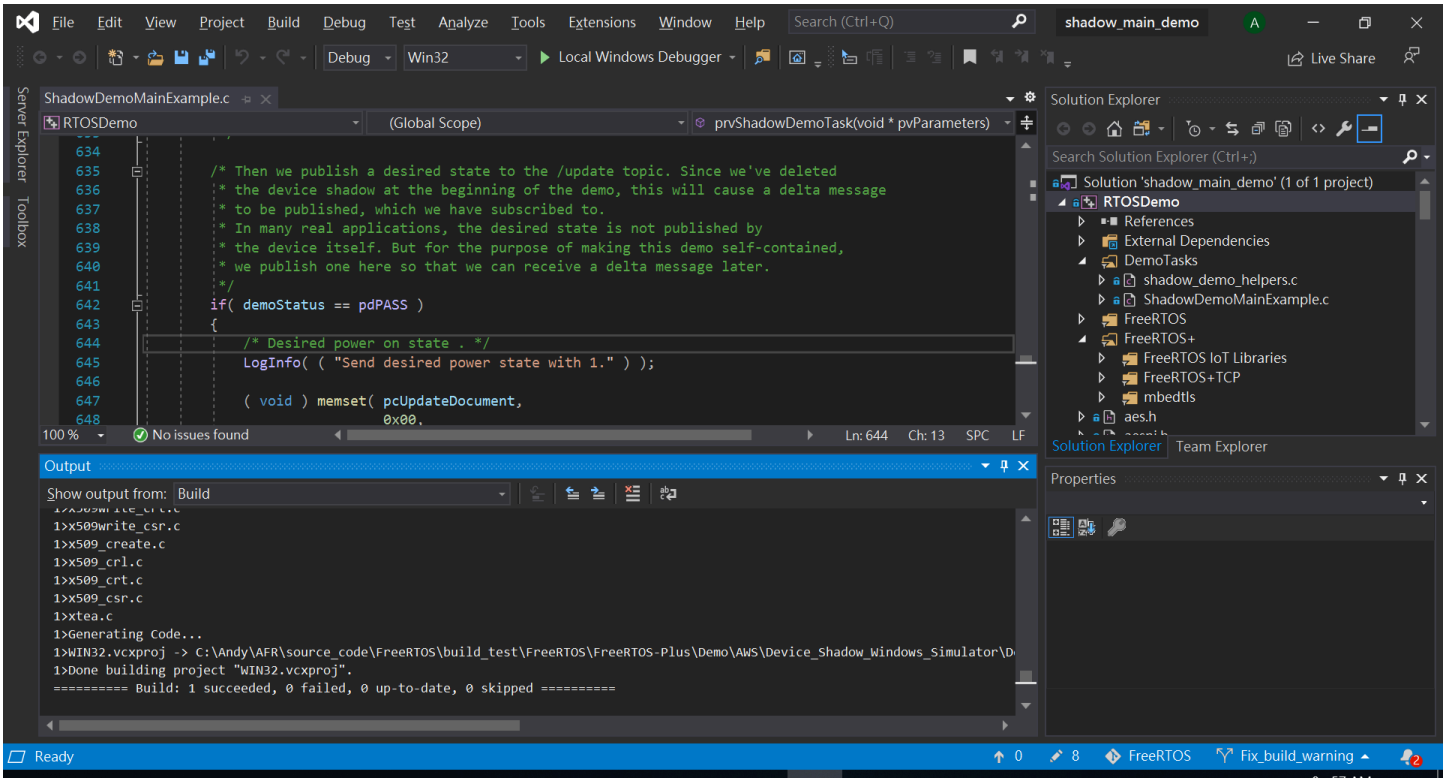
示範執行下列操作：

1. 使用中的輔助函數來建立 MQTT 連線shadow_demo_helpers.c。
2. 使用 Device Shadow 程式庫定義的巨集，為裝置陰影作業組合 MQTT 主題字串。AWS IoT
3. 發佈至 MQTT 主題，用於刪除裝置陰影以刪除任何現有裝置陰影。
4. 訂閱的 MQTT 主題/update/delta，/update/accepted並在中/update/rejected使用輔助函數shadow_demo_helpers.c。
5. 在中發佈powerOn使用輔助函數的所需狀態shadow_demo_helpers.c。這會將/update/delta訊息傳送到裝置。
6. 處理中的傳入 MQTT 訊息prvEventCallback，並使用 Device Shadow 程式庫 (Shadow_MatchTopic) 定義的函數判斷訊息是否與AWS IoT裝置陰影相關。如果消息是設備陰影/update/delta消息，則主演示功能將發布第二條消息以將報告的狀態更新為powerOn。如果收到/update/accepted訊息，請確認郵件與先前在更新訊息中發佈的相同clientToken。這將標誌著演示的結束。

```
82 9136 [ShadowDemo] [INFO] [SHADOW] [prvShadowDemoTask:641] 83 9136 [ShadowDemo] Send desired power state with 1.84 9136 [ShadowDemo]
85 9296 [ShadowDemo] [INFO] [SHADOW] [prvEventCallback:482] 86 9296 [ShadowDemo] pPublishInfo->pTopicName:$aws/things/testClient16:34:41/shadow/update/delta.87 9296 [Shad
owDemo]
88 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateDeltaHandler:256] 89 9296 [ShadowDemo] /update/delta json payload:{"version":1,"timestamp":1602751002,"state":{"powerOn":1}
,"metadata":{"powerOn":{"timestamp":1602751002}},"clientToken":"009136"}.90 9296 [ShadowDemo]
91 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateDeltaHandler:298] 92 9296 [ShadowDemo] version: 193 9296 [ShadowDemo]
94 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateDeltaHandler:308] 95 9296 [ShadowDemo] version:1, ulCurrentVersion:0
96 9296 [ShadowDemo]
97 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateDeltaHandler:342] 98 9296 [ShadowDemo] The new power on state newState:1, ulCurrentPowerOnState:0
99 9296 [ShadowDemo]
100 9296 [ShadowDemo] [INFO] [SHADOW] [prvEventCallback:482] 101 9296 [ShadowDemo] pPublishInfo->pTopicName:$aws/things/testClient16:34:41/shadow/update/accepted.102 9296
 [ShadowDemo]
103 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:376] 104 9296 [ShadowDemo] /update/accepted json payload:{"state":{"desired":{"powerOn":1}},"metadata":{"d
esired":{"powerOn":{"timestamp":1602751002}}},"version":1,"timestamp":1602751002,"clientToken":"009136"}.105 9296 [ShadowDemo]
106 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:424] 107 9296 [ShadowDemo] clientToken: 009136108 9296 [ShadowDemo]
109 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:429] 110 9296 [ShadowDemo] receivedToken:9136, clientToken:0
111 9296 [ShadowDemo]
112 9296 [ShadowDemo] [WARN] [SHADOW] [prvUpdateAcceptedHandler:442] 113 9296 [ShadowDemo] The received clientToken=9136 is not identical with the one=0 we sent 114 9296
[ShadowDemo]
115 9696 [ShadowDemo] [INFO] [SHADOW] [prvShadowDemoTask:670] 116 9696 [ShadowDemo] Report to the state change: 1117 9696 [ShadowDemo]
118 9856 [ShadowDemo] [INFO] [SHADOW] [prvEventCallback:482] 119 9856 [ShadowDemo] pPublishInfo->pTopicName:$aws/things/testClient16:34:41/shadow/update/accepted.120 9856
 [ShadowDemo]
121 9856 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:376] 122 9856 [ShadowDemo] /update/accepted json payload:{"state":{"reported":{"powerOn":1}},"metadata":{"
reported":{"powerOn":{"timestamp":1602751003}}},"version":2,"timestamp":1602751003,"clientToken":"009696"}.123 9856 [ShadowDemo]
124 9856 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:424] 125 9856 [ShadowDemo] clientToken: 009696126 9856 [ShadowDemo]
127 9856 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:429] 128 9856 [ShadowDemo] receivedToken:9696, clientToken:9696
129 9856 [ShadowDemo]
130 9856 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:437] 131 9856 [ShadowDemo] Received response from the device shadow. Previously published update with clie
ntToken=9696 has been accepted. 132 9856 [ShadowDemo]
133 10256 [ShadowDemo] [INFO] [SHADOW] [prvShadowDemoTask:698] 134 10256 [ShadowDemo] Start to unsubscribe shadow topics and disconnect from MQTT.
135 10256 [ShadowDemo]
136 12036 [ShadowDemo] [INFO] [SHADOW] [prvShadowDemoTask:747] 137 12036 [ShadowDemo] Demo completed successfully.138 12036 [ShadowDemo]
139 12036 [ShadowDemo] [INFO] [SHADOW] [prvShadowDemoTask:796] 140 12036 [ShadowDemo] Deleting Shadow Demo task.141 12036 [ShadowDemo]
```

演示可以在文件中找到*freertos*/demos/device_shadow_for_aws/shadow_demo_main.c或上 GitHub。

下列螢幕擷取畫面顯示示範成功時，預期的輸出。



Connect 到AWS IoT MQTT 代理商

若要連線至AWS IoT MQTT 代理程式，我們使用與MQTT_Connect()中相同的方法交互身分驗證示範。

刪除陰影文件

若要刪除陰影文件，請xPublishToTopic使用AWS IoT Device Shadow 元件庫定義的巨集以空白訊息呼叫。這會用MQTT_Publish來發佈到/delete主題。下列程式碼區段會示範如何在函數中完成此作業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，以接收來自AWS IoT代理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。下列程式碼區段會示範如何在prvShadowDemoTask函數中完成此作業。

```
/* Then try to subscribe shadow topics. */
if( returnStatus == EXIT_SUCCESS )
{
    returnStatus = SubscribeToTopic(
                     SHADOW_TOPIC_STRING_UPDATE_DELTA( THING_NAME ),
                     SHADOW_TOPIC_LENGTH_UPDATE_DELTA( THING_NAME_LENGTH ) );
}

if( returnStatus == EXIT_SUCCESS )
{
    returnStatus = SubscribeToTopic(
                     SHADOW_TOPIC_STRING_UPDATE_ACCEPTED( THING_NAME ),
                     SHADOW_TOPIC_LENGTH_UPDATE_ACCEPTED( THING_NAME_LENGTH ) );
}

if( returnStatus == EXIT_SUCCESS )
{
    returnStatus = SubscribeToTopic(
                     SHADOW_TOPIC_STRING_UPDATE_REJECTED( THING_NAME ),
                     SHADOW_TOPIC_LENGTH_UPDATE_REJECTED( THING_NAME_LENGTH ) );
}
```

傳送陰影更新

若要傳送陰影更新，示範會使用 Device Shadow 程式庫所定義的巨集，以 JSON 格式呼xPublishToTopic叫訊息。這會用MQTT_Publish來發佈到/delete主題。下列程式碼區段會示範如何在prvShadowDemoTask函數中完成此作業。

```
#define SHADOW_REPORTED_JSON    \
    "{"                         \
    "\"state\":{"               \
    "\"reported\":{"            \
    "\"powerOn\":%01d"          \
    "}"                         \
```

```
    "},"                          \
    "\"clientToken\":\"%06lu\"" \
    "}"
snprintf( pcUpdateDocument,
          SHADOW_REPORTED_JSON_LENGTH + 1,
          SHADOW_REPORTED_JSON,
           ( int ) ulCurrentPowerOnState,
           ( long unsigned ) ulClientToken );

xPublishToTopic( SHADOW_TOPIC_STRING_UPDATE( THING_NAME ),
                 SHADOW_TOPIC_LENGTH_UPDATE( THING_NAME_LENGTH ),
                 pcUpdateDocument,
                 ( SHADOW_DESIRED_JSON_LENGTH + 1 ) );
```

處理陰影增量訊息和陰影更新訊息

使用此函數註冊到 CoremQtt 用戶端程式庫的使用者回呼`MQTT_Init`函式，將會通知我們有關傳入封包事件的相關資訊。請參閱（詳見）的回呼函數 prvEventCallback GitHub。

回呼函式會確認傳入封包的類型`MQTT_PACKET_TYPE_PUBLISH`，並使用 Device Shadow 程式庫 API`Shadow_MatchTopic` 來確認傳入訊息是陰影訊息。

如果傳入消息是帶有類型的陰影消息`ShadowMessageTypeUpdateDelta`，那麼我們調用 prvUpdateDeltaHandler 來處理此消息。處理常式`prvUpdateDeltaHandler`使用 CoreJSON 程式庫來剖析訊息，以取得`powerOn`狀態的差異值，並將其與目前在本機維護的裝置狀態進行比較。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。

如果傳入消息是帶有類型的陰影消息`ShadowMessageTypeUpdateAccepted`，那麼我們調用 prvUpdateAcceptedHandler 來處理此消息。處理常式`prvUpdateAcceptedHandler`會使用 CoreJSON 程式庫剖析訊息，以`clientToken`從訊息取得。此處理常式函數會檢查 JSON 訊息中的用戶端權杖是否與應用程式使用的用戶端權杖相符。如果不符合，則會記錄警告訊息。

Secure Sockets echo 用戶端示範

> **Important**
>
> 此演示託管在亞馬遜的 FreeRTOS 存儲庫上，該存儲庫已被棄用。建議您為建立的專案開始操作時。如果您已經有一個現有的 FreeRTOS 專案以目前已取代的亞馬遜免費伺服器儲存庫為基礎，請參閱亞馬遜自由 Github 存儲庫遷移指南.

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 。

開始操作前，請確認您已將 FreeRTOS 下載到微控制器，並且建立即執行 FreeRTOS 示範專案。您可以從中克隆或下載 FreeRTOS GitHub。如需說明，請參閱 README.md 檔案。

執行示範

> **Note**
>
> 若要設定並執行 FreeRTOS 示範，請遵循中的步驟FreeRTOS 入。
> 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. 請依照《FreeRTOS 移植指南》中的〈設定 TLS 回應伺服器〉中的指示進行。

   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、註解掉#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED，以及定義CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED或CONFIG_OTA_HTTP_UPDATE_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。

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。