aws

移植指南

FreeRTOS

FreeRTOS: 移植指南

Table of Contents

FreeRTOS 植

什麼是 FreeRTOS

與世界領先的晶片公司合作開發超過 20 年，現在每 170 秒下載一次，FreeRTOS 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，以及在合格主機板上展示 FreeRTOS 功能的示範應用程式。

如需詳細資訊，請造訪 FreeRantos.org。

將 FreeRTOS 移植到您的 IoT 主機板

您需要根據 FreeRTOS 軟體程式庫的功能和應用程式，將其移植到以微控制器為基礎的主機板。

將 FreeRTOS 移植到您的裝置

1. 依照中列出的指示，下載用於移植 FreeRTOS 服務以下載最新版本的 FreeRTOS，以進行移植。
2. 依照中的指示設設定您的工作區和專案以進行移植定 FreeRTOS 下載中的檔案和資料夾，以進行移植和測試。
3. 依照中的指示移植 FreeRTOS 式庫將 FreeRTOS 程式庫移植到您的裝置。每個移植主題都包含測試連接埠的指示。

移植常見問答集

什麼是 FreeRTOS 接埠？

> FreeRTOS 連接埠是針對所需 FreeRTOS 程式庫和您平台支援的 FreeRTOS 核心的主機板特定 API 實作。連接埠可讓 API 在電路板上運作，並實作與平台廠商所提供的裝置驅動程式和 BSP 所需的整合。您的連接埠還應該包含電路板所需的任何組態調整 (例如時脈速率、堆疊大小、堆積大小)。

如果您有關於移植的問題，但在本頁或 FreeRTOS 移植指南的其他部分中找不到解答，請參閱可用的 FreeRTOS 支援選項。

下載用於移植 FreeRTOS 服務

從 Freertos.org 下載最新的免費服務或長期 Support (L TS) 版本，或從 GitHub (免費翻譯) 或複製來源。

> **Note**
>
> 我們建議您複製儲存庫。克隆使您可以更輕鬆地在將更新推送到存儲庫時對主分支進行更新。

或者，將來自 FreeRTOS 或 FreerToS-LTS 存放庫的個別程式庫子模組。但是，請確定程式庫版本符合 FreeRTOS 或 FreerTos 存放庫中`manifest.yml`檔案中列出的組合。

下載或複製 FreeRTOS 之後，您就可以開始將 FreeRTOS 程式庫移植到您的主機板上。如需說明，請參閱 設定您的工作區和專案以進行移植，然後請參閱 移植 FreeRTOS 式庫。

設定您的工作區和專案以進行移植

請遵循下列步驟設定：

- 使用您選擇的專案結構和建置系統來匯入 FreeRTOS 程式庫。
- 使用主機板支援的整合式開發環境 (IDE) 和工具鏈來建立專案。
- 在您的專案中加入主機板支援套件 (BSP) 和主機板特定驅動程式。

設定好工作區之後，您就可以開始移植個別 FreeRTOS 程式庫。

移植 FreeRTOS 式庫

開始移植之前，請遵循中的指示設定您的工作區和專案以進行移植。

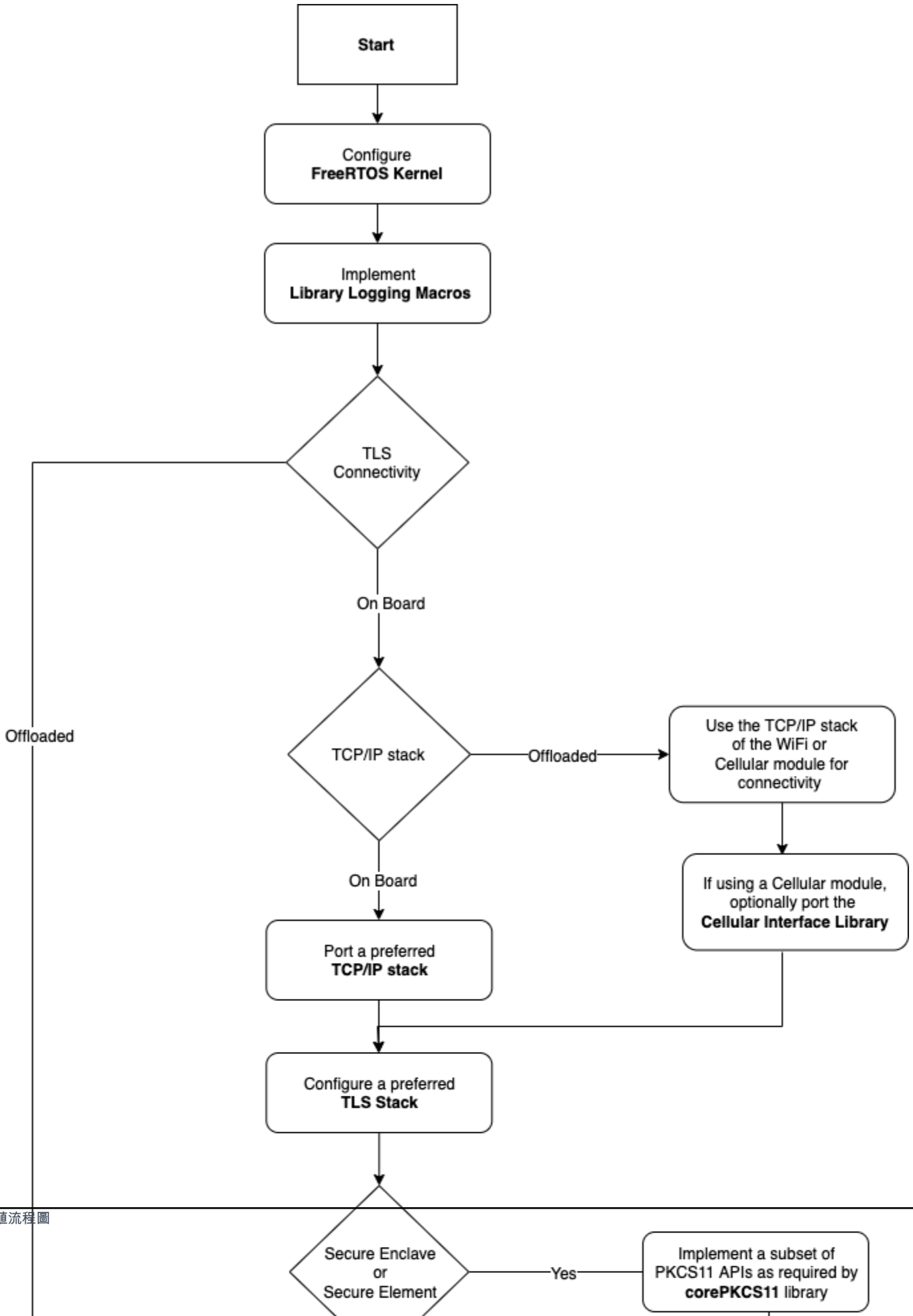
FreeRTOS 移植流程圖說明移植所需的程式庫。

若要將 FreeRTOS 移植到您的裝置，請依照以下主題中的指示操作。

1. 設定 FreeRTOS 核心連接埠
2. 實現庫日誌宏
3. 移植 TCP/IP 堆疊
4. 移植網路傳輸介面
5. 移植圖書館
6. 設定核心 QTT 程式庫
7. 設定核心 HTTP 程式庫
8. 移植AWS IoT over-the-air (OTA) 更新程式庫
9. 移植蜂窩接口庫

FreeRTOS 移植流程圖

將 FreeRTOS 移植到主機板時，請使用下方的移植流程圖做為視覺輔助工具。


Start
Configure
FreeRTOS Kernel
Implement
Library Logging Macros
TLS
Connectivity
Offloaded
On Board
TCP/IP stack
Offloaded
Use the TCP/IP stack
of the WiFi or
Cellular module for
connectivity
If using a Cellular module,
optionally port the
Cellular Interface Library
On Board
Port a preferred
TCP/IP stack
Configure a preferred
TLS Stack
Secure Enclave
or
Secure Element
Yes
Implement a subset of
PKCS11 APIs as required by
corePKCS11 library

設定 FreeRTOS 核心連接埠

本節提供將 FreeRTOS 核心的連接埠整合至 FreeRTOS 連接埠測試專案的指示。如需可用核心連接埠的清單，請參閱 FreeRTOS 核心連接埠。

FreeRTOS 使用 FreeRTOS 核心進行多工作業和任務間的通訊。如需詳細資訊，請參閱 FreeRTOS 使用者指南和 FreeRTOS 核心基礎知識。

> **Note**
>
> 本文件不包含將 FreeRTOS 核心移植到新架構。如果您有興趣，請聯絡 FreeRTOS 工程團隊。
> 對於 FreeRTOS 資格認證計劃，僅支援現有的 FreeRTOS 核心連接埠。程式內不接受對這些連接埠的修改。如需詳細資訊，請檢閱 FreeRTOS 核心連接埠原則。

先決條件

若要設定 FreeRTOS 核心以準備移植，您需要以下項目：

- 目標平台的官方 FreeRTOS 核心連接埠或 FreeRTOS 支援的連接埠。
- IDE 專案，包含適用於目標平台和編譯器的正確 FreeRTOS 核心連接埠檔案。如需有關設定測試專案的詳細資訊，請參閱 設定您的工作區和專案以進行移植。

設定 FreeRTOS 核心

FreeRTOS 核心是使用名為的組態檔來自訂的`FreeRTOSConfig.h`。此檔案指定核心的應用程式特定組態設定。如需每個組態選項的說明，請參閱 FreerTos.org 上的自訂。

若要設定 FreeRTOS 核心與您的裝置搭配使用，請包含`FreeRTOSConfig.h`和修改任何額外的 FreeRTOS 組態。

如需每個組態選項的說明，請參閱 FreerTos.org 上的自訂組態。

測試

- 執行簡單的 FreeRTOS 工作，將訊息記錄到序列輸出主控台。

- 確認訊息是否如預期般輸出至主控台。

實現庫日誌宏

FreeRTOS 程式庫會使用下列記錄巨集，依詳細程度的遞增順序列出。

- `LogError`
- `LogWarn`
- `LogInfo`
- `LogDebug`

必須提供所有巨集的定義。這些建議是：

- 宏應該支持C89樣式日誌記錄。
- 日誌記錄應該是線程安全的。來自多個任務的日誌行不得相互交錯。
- 記錄 API 不得封鎖，而且必須避免應用程式工作在 I/O 上封鎖。

如需實作詳細資訊，請參閱 FreerTos.org 上的記錄功能。你可以在這個例子中看到一個實現。

測試

- 運行具有多個任務的測試，以驗證日誌不交錯。
- 執行測試以確認記錄 API 不會在 I/O 上封鎖。
- 使用各種標準測試記錄巨集，例如C89,C99樣式記錄。
- 透過設定不同的記錄層級來測試記錄巨集Debug，例如InfoError、、和Warning。

移植 TCP/IP 堆疊

本節提供有關移植和測試板載 TCP/IP 堆疊的說明。如果您的平台將 TCP/IP 和 TLS 功能卸載到個別的網路處理器或模組，您可以略過此移植區段並造訪移植網路傳輸介面。

自由行 + TCP是 FreeRTOS 核心的原生 TCP/IP 堆疊。FreeRTOS 的工程團隊是由 FreeRTOS 工程團隊所開發與維護，是建議搭配 FreeRTOS 使用的 TCP/IP 堆疊。如需詳細資訊，請參閱移植 FreeRTOS+TCP。或者，您也可以使用第三方 TCP/IP 堆疊LWIP。本節提供的測試指示會針對 TCP 純文字使用傳輸介面測試，且不依賴於特定實作的 TCP/IP 堆疊。

移植 FreeRTOS+TCP

FreeRTOS+TCP 是 FreeRTOS 核心的原生 TCP/IP 堆疊。如需詳細資訊，請參閱 FreeRTOS.org。

先決條件

若要移植 FreeRTOS+TCP 程式庫，您需要以下項目：

- 包含供應商提供的乙太網路或 Wi-Fi 驅動程式的 IDE 專案。

  如需有關設定測試專案的詳細資訊，請參閱 設定您的工作區和專案以進行移植。
- FreeRTOS 核心的已驗證組態。

  如需有關為您的平台設定 FreeRTOS 核心的詳細資訊，請參閱 設定 FreeRTOS 核心連接埠。

移植

在您開始移植自由軟體 + TCP 程式庫之前，請先檢查GitHub目錄以查看主機板的連接埠是否已存在。

如果連接埠不存在，請執行下列動作：

1. 依照 FreeRTOS.org 上的將 FreeRTOS+TCP 移植到不同的微控制器指示，將 FreeRTOS+TCP 移植到您的裝置。
2. 如有必要，依照 FreeRTOS.org 上的將 FreeRTOS+TCP 移植到新的 Embedded C 編譯器指示，將 FreeRTOS+TCP 移植到新的編譯器。
3. 在名為的文件中實現使用供應商提供的以太網或 Wi-Fi 驅動程序的新端口`NetworkInterface.c`。造訪GitHub範本的儲存庫。

建立連接埠之後，或者如果連接埠已存在，請建立`FreeRTOSIPConfig.h`，然後編輯組態選項，使其適用於您的平台。如需組態選項的詳細資訊，請參閱 FreeRTOS.org 上的 FreeRTOS+TCP 組態。

測試

無論您使用 FreeRTO+TCP 庫還是第三方庫，請按照以下步驟進行測試：

- 提供的實作`connect/disconnect/send/receive`傳輸介面測試中的 API。
- 以純文字 TCP 連線模式設定 echo 伺服器，並執行傳輸介面測試。

> **Note**
>
> 若要正式符合 FreeRTOS 的裝置資格，如果您的架構需要連接 TCP/IP 軟體堆疊，您必須針對純文字 TCP 連線模式下的傳輸介面測試，驗證裝置的移植原始程式碼。AWS IoT Device Tester。按照中的說明操作使用AWS IoT Device Tester對於免費服務在自由使用者指南若要設定AWS IoT Device Tester用於端口驗證。若要測試特定程式庫的連接埠，必須在 Device Tester `configs` 資料夾的 `device.json` 檔案中啟用正確的測試群組。

移植圖書館

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。包裝套件 11指的是標準及其定義的 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加密對象。它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在應用程序的內存中。

FreeRTOS 程式庫和參考整合使用 PCKS #11 介面標準的子集，重點放在涉及非對稱金鑰、隨機數產生和雜湊的作業上。下表列出了要支援的使用案例和必要的 PKCS #11 API。

使用案例

使用案例	必要的 PKCS #11 應用程式介面系列
全部	初始化、完成、開啟/關閉工作階段、 GetSlotList, 登入
佈建	GenerateKeyPair, CreateObject, DestroyObject, InitToken, GetTokenInfo
TLS	隨機、簽署、 FindObject, GetAttributeValue
FreeRTOS+TCP	隨機、摘要
OTA	驗證、摘要、 FindObject, GetAttributeValue

何時實作完整 PKCS #11 模組

在評估和快速原型設計案例中，將私有金鑰存放在一般用途快閃記憶體中可能很方便。我們建議您使用專用的加密硬體，以減少在生產環境中遭到資料竊取和裝置複製的威脅。密碼編譯硬體包含的元件有功

能可防止匯出密碼編譯私密金鑰。若要支援此功能，您必須實作 PKCS #11 的子集，才能使用上表所定義的 FreeRTOS 程式庫。

何時使用 FreeRTOS 網路服務 11

CorepkCS11 程式庫包含以軟體為基礎的 PKCS #11 介面 (API) 實作，該介面使用由提供的密碼編譯功能姆貝德 TLS。這是針對硬體沒有專用加密硬體的快速原型製作和評估案例所提供的。在這種情況下，您只需要實作 CorepkCS11 PAL，即可讓 CorepkCS11 軟體式實作與您的硬體平台搭配使用。

移植公司 11

您必須實作才能將加密物件讀取和寫入非揮發性記憶體 (NVM)，例如板載快閃記憶體。加密對象必須存儲在 NVM 的一個部分，該部分未初始化，並且在設備重新編程時不會擦除。CorepkCS11 程式庫的使用者將使用認證佈建裝置，然後使用可透過 CorepkCS11 介面存取這些認證的新應用程式來重新編程裝置。CorepkCS11 PAL 連接埠必須提供儲存位置：

- 裝置用戶端憑證
- 裝置用戶端私密金鑰
- 裝置用戶端公開金鑰
- 受信任的根 CA
- 安全開機載入程式的程式碼驗證公開金鑰 (或包含程式碼驗證公開金鑰的憑證) over-the-air （太田）更新
- 即時佈建憑證

包括頭文件並實現定義的 PAL API。

菲航 API

函數	描述
初始化	初始化 PAL 層。由 CorepkCS11 程式庫在其初始化序列開始時呼叫。
帕爾 _SaveObject	將資料寫入非揮發性儲存。
帕爾 _FindObject	使用 PKCS #11 CKA_LABEL 在非揮發性儲存中搜尋對應的 PKCS #11 物件，並傳回該物件的控制代碼 (如果存在)。

函數	描述
帕爾 _GetObjectValue	根據控制代碼擷取物件的值。
帕爾 _GetObjectValueCleanup	清除 PKCS11_PAL_GetObjectValue 呼叫。可用於釋放 PKCS11_PAL_GetObjectValue 呼叫中配置的記憶體。

測試

如果您使用 FreeRTOS 體庫或實作 PKCS11 API 的必要子集，您必須通過 PKCS11 測試。這些測試 FreeRTOS 程式庫所需的功能是否如預期般執行。

本節也說明如何在本節中使用資格測試這個節也說明如何在本節中執行 FreeRTOS PKCS11 測試。

先決條件

若要設定 FreeRTOS PKCS11 測試，必須執行下列動作。

- PKCS11 API 的支援連接埠。
- FreeRTOS 資格測試平台功能的實作，其中包括：
  - `FRTest_ThreadCreate`
  - `FRTest_ThreadTimedJoin`
  - `FRTest_MemoryAlloc`
  - `FRTest_MemoryFree`

(請參閱雷德美. MDPKCS #11 上的 FreeRTOS 程式庫整合測試檔案 GitHub。)

移植測試

- 新增自由圖書館集成測試作為一個子模塊到你的項目。子模塊可以放置在項目的任何目錄中，只要它可以被構建。
- 複製`config_template/test_execution_config_template.h`和`config_template/test_param_config_template.h`到構建路徑中的項目位置，並將其重命名為`test_execution_config.h`和`test_param_config.h`。
- 在建置系統中包含相關檔案。如果使用CMake,`qualification_test.cmake`和`src/pkcs11_tests.cmake`可用於包括相關文件。

- 實施UNITY_OUTPUT_CHAR這樣測試輸出日誌和設備日誌就不會交錯。
- 整合 MBedTLS，以驗證加密操作結果。
- 呼叫RunQualificationTest()從應用程式中。

配置測試

PKCS11 測試套件必須根據 PKCS11 實作進行配置。下表列出在中所提供的 PKCS11 測試所需的配置test_param_config.h頭文件。

PKSC11 測試配置

組態	描述
支援金鑰支援	移植支援 RSA 關鍵功能。
支援鍵盤測試	該移植支持 EC 鍵功能。
支持私人密鑰支持	移植支援匯入私密金鑰。如果啟用了支援的關鍵功能，則會在測試中驗證 RSA 和 EC 金鑰匯入。
生成鍵盤支持	移植支持密鑰對生成。如果啟用了支持的關鍵功能，則會在測試中驗證 EC 密鑰對生成。
預先佈建支援	移植具有預先佈建的認證。PKCS11_TEST_LABEL_DEVICE_PRIVATE_KEY_FOR_TLS ,PKCS11_TEST_LABEL_DEVICE_PUBLIC_KEY_FOR_TLS 和PKCS11_TEST_LABEL_DEVICE_CERTIFICATE_FOR_TLS ，是認證的範例。
PKCS11 測試 _ 標籤設備 _ 私有密鑰形式	測試中使用的私密金鑰標籤。
PKCS11 測試標籤裝置公開密鑰	測試中使用的公開金鑰標籤。
PKCS11 測試標籤設備證書	測試中使用的憑證標籤。

組態	描述
支援的程式碼驗證碼	移植支援 JITP 的儲存裝置。將這個項目設定為 1 以啟用 JITP`codeverify` 測試。
驗證碼密鑰	JITP 中使用的代碼驗證密鑰的標籤`codeverify` 測試。
測試標籤設定證書	JITP 中使用的 JITP 憑證標籤`codeverify` 測試。
測試標籤根證書	JITP 中使用的根憑證標籤`codeverify` 測試。

FreeRTOS 程式庫和參考整合必須支援至少一個關鍵功能組態，例如 RSA 或橢圓形曲線索引鍵，以及 PKCS11 API 支援的一個金鑰佈建機制。測試必須啟用以下配置：

- 下列其中一個按鍵功能組態：
  - 支援金鑰支援
  - 支援鍵盤測試
- 下列其中一個金鑰佈建組態：
  - 支持私人密鑰支持
  - 生成鍵盤支持
  - 預先佈建支援

預先佈建的裝置認證測試必須在下列情況下執行：

- `PKCS11_TEST_PREPROVISIONED_SUPPORT`必須啟用且停用其他佈建機制。
- 只有一個按鍵功能`PKCS11_TEST_RSA_KEY_SUPPORT`或者`PKCS11_TEST_EC_KEY_SUPPORT`，已啟用。
- 根據您的按鍵功能設定預先佈建的金鑰標籤，包括`PKCS11_TEST_LABEL_DEVICE_PRIVATE_KEY_FOR_TLS`,`PKCS11_TEST_LABEL_DEVICE_PUBLIC_`執行測試之前，必須先存在這些認證。

如果實作支援預先佈建的認證和其他佈建機制，則測試可能需要使用不同的組態執行多次。

> **Note**
>
> 帶有標籤的對象PKCS11_TEST_LABEL_DEVICE_PRIVATE_KEY_FOR_TLS,PKCS11_TEST_LABEL_DEVICE_PUBLI果在測試期間被銷毀PKCS11_TEST_GENERATE_KEYPAIR_SUPPORT或者PKCS11_TEST_GENERATE_KEYPAIR_SUPPORT已啟用。

執行測試

本節說明如何透過資格測試在本機測試 PKCS11 介面。或者，也可以使用 IDT 來自動執行。請參閱AWS IoT Device Tester對於 FreeRTOS在FreeRTOS 使用指南如需詳細資訊。

下列指示說明如何運行測試：

- 打開test_execution_config.h並定義核心 _ 測試已啟用到 1 個。
- 構建應用程序並將其刷新到設備以運行。測試結果將輸出到串行端口。

下面是輸出測試結果的範例。

```
TEST(Full_PKCS11_StartFinish, PKCS11_StartFinish_FirstTest) PASS
TEST(Full_PKCS11_StartFinish, PKCS11_GetFunctionList) PASS
TEST(Full_PKCS11_StartFinish, PKCS11_InitializeFinalize) PASS
TEST(Full_PKCS11_StartFinish, PKCS11_GetSlotList) PASS
TEST(Full_PKCS11_StartFinish, PKCS11_OpenSessionCloseSession) PASS
TEST(Full_PKCS11_Capabilities, PKCS11_Capabilities) PASS
TEST(Full_PKCS11_NoObject, PKCS11_Digest) PASS
TEST(Full_PKCS11_NoObject, PKCS11_Digest_ErrorConditions) PASS
TEST(Full_PKCS11_NoObject, PKCS11_GenerateRandom) PASS
TEST(Full_PKCS11_NoObject, PKCS11_GenerateRandomMultiThread) PASS
TEST(Full_PKCS11_RSA, PKCS11_RSA_CreateObject) PASS
TEST(Full_PKCS11_RSA, PKCS11_RSA_FindObject) PASS
TEST(Full_PKCS11_RSA, PKCS11_RSA_GetAttributeValue) PASS
TEST(Full_PKCS11_RSA, PKCS11_RSA_Sign) PASS
TEST(Full_PKCS11_RSA, PKCS11_RSA_FindObjectMultiThread) PASS
TEST(Full_PKCS11_RSA, PKCS11_RSA_GetAttributeValueMultiThread) PASS
TEST(Full_PKCS11_RSA, PKCS11_RSA_DestroyObject) PASS
TEST(Full_PKCS11_EC, PKCS11_EC_GenerateKeyPair) PASS
TEST(Full_PKCS11_EC, PKCS11_EC_CreateObject) PASS
TEST(Full_PKCS11_EC, PKCS11_EC_FindObject) PASS
```

```
TEST(Full_PKCS11_EC, PKCS11_EC_GetAttributeValue) PASS
TEST(Full_PKCS11_EC, PKCS11_EC_Sign) PASS
TEST(Full_PKCS11_EC, PKCS11_EC_Verify) PASS
TEST(Full_PKCS11_EC, PKCS11_EC_FindObjectMultiThread) PASS
TEST(Full_PKCS11_EC, PKCS11_EC_GetAttributeValueMultiThread) PASS
TEST(Full_PKCS11_EC, PKCS11_EC_SignVerifyMultiThread) PASS
TEST(Full_PKCS11_EC, PKCS11_EC_DestroyObject) PASS

-----------------------
27 Tests 0 Failures 0 Ignored
OK
```

當所有測試都通過時，測試就完成。

> **Note**
>
> 若要正式符合 FreeRTOS 的裝置資格，您必須使用以下方式驗證裝置移植的原始程式碼AWS IoT Device Tester。按照中所提供的說明使用AWS IoT Device Tester對於 FreeRTOS在《FreeRTOS 使用者指南》中進行設定AWS IoT Device Tester用於連接埠驗證。若要測試特定程式庫的連接埠，必須在`device.json`檔案中的AWS IoT Device Tester `configs`資料夾中。

移植網路傳輸介面

整合 TLS 程式庫

如需 Transport Layer Security (TLS) 驗證，請使用您偏好的 TLS 堆疊。我們建議使用姆貝德 TLS因為它是使用 FreeRTOS 庫進行測試的。你可以在這裡找到一個例子GitHub儲存庫。

無論您的裝置所使用的 TLS 實作為何，您都必須針對具有 TCP/IP 堆疊的 TLS 堆疊實作基礎傳輸勾點。他們必須支持支援的 TLS 加密套件AWS IoT。

移植網路傳輸介面程式庫

您必須實作網路傳輸介面才能使用中央电台和核心 HTTP。網絡傳輸接口包含在單個網絡連接上發送和接收數據所需的功能指針和上下文數據。請參閱傳輸介面詳細資訊。FreeRTOS 提供一組內建的網路傳輸介面測試來驗證這些實作。以下部分將指導您如何設置項目以運行這些測試。

先決條件

若要移植此測試，您需要下列項目：

- 具有建置系統的專案，可以使用經過驗證的 FreeRTOS 核心連接埠來建置 FreeRTOS。
- 網絡驅動程序的工作實現。

移植

- 新增自由圖書館集成測試作為一個子模塊到你的項目。子模塊放置在項目中的位置並不重要，只要它可以構建即可。
- 複製config_template/test_execution_config_template.h和config_template/test_param_config_template.h到構建路徑中的項目位置，並將其重命名為test_execution_config.h和test_param_config.h。
- 在建置系統中包含相關檔案。如果使用CMake,qualification_test.cmake和src/transport_interface_tests.cmake用於包括相關文件。
- 在適當的專案位置實作下列功能：
- • 一個network connect function：簽名定義NetworkConnectFunc在src/common/network_connection.h。此函數會取得指向網路內容的指標、指向主機資訊的指標，以及指向網路認證的指標。它使用提供的網絡憑據與主機信息中指定的服務器建立連接。
  - 一個network disconnect function：簽名定義NetworkDisconnectFunc在src/common/network_connection.h。此函數會接收指向網路前後關聯的指標。它會中斷儲存在網路前後關聯中之前建立的連線。
  - setupTransportInterfaceTestParam()：定義於src/transport_interface/transport_interface_tests.h。實現必須具有與中定義的完全相同的名稱和簽名transport_interface_tests.h。這個函數接受一個指向TransportInterfaceTestParam結構。它將填充中的字段TransportInterfaceTestParam傳輸介面測試所使用的結構。
- 實施單位 _ 輸出字符以便測試輸出日誌不會與設備日誌交錯。
- 呼叫runQualificationTest()從應用程式。設備硬件必須正確初始化，並且必須在呼叫之前連接網絡。

憑證管理 (裝置上產生的金鑰)

何時強制 _ 生成 _ 新的密鑰 _ 對在test_param_config.h設置為 1，設備應用程序生成一個新的設備上的 key pair 並輸出公鑰。裝置應用程式使用回聲伺服器和運輸 _ 客戶 _ 證書作為回顯服務器根 CA 和客戶端證書，當建立與 echo 服務器的 TLS 連接時。IDT 會在限定執行期間設定這些參數。

認證管理 (匯入金鑰)

裝置應用程式使用回聲伺服器,運輸 _ 客戶 _ 證書和運輸 _ 用戶端 _ 私人密鑰在test_param_config.h作為 echo 服務器根 CA，客戶端證書和客戶端私鑰時建立與 echo 服務器的 TLS 連接。IDT 會在限定執行期間設定這些參數。

測試

本節說明如何透過資格測試在本機測試傳輸介面。如需詳細資訊，請參閱傳輸層詳細資訊，請參閱傳輸資訊。傳輸界面在自由圖書館集成測試的部分 GitHub。

或者，您也可以使用 IDT 來自動執行。請參閱AWS IoT Device Tester對於 FreeRTOS在FreeRTOS 用者指南詳細資訊。

啟用測試

打開test_execution_config.h並定義傳輸 _ 介面 _ 測試啟用到 1.

設置迴聲服務器進行測試

本地測試需要可從運行測試的設備訪問的 echo 服務器。如果傳輸介面實作支援 TLS，回應伺服器必須支援 TLS。如果你還沒有，自由圖書館集成測試 GitHub 存儲庫有一個 echo 服務器實現。

設定要測試的專案

在test_param_config.h，更新迴聲伺服器端點和迴聲伺服器連接埠前一步的端點和伺服器設定。

設置憑據（設備上生成的密鑰）

- 設置回聲伺服器到 echo 服務器的服務器證書。
- 設置強制 _ 生成 _ 新的密鑰 _ 對到 1 生成一個 key pair 並獲取公鑰。
- 設置強制 _ 生成 _ 新的密鑰 _ 對密鑰生成後返回 0。
- 使用公開金鑰、伺服器金鑰與憑證來產生用戶端憑證。
- 設置運輸 _ 客戶 _ 證書到生成的客戶端證書。

設定認證 (匯入金鑰)

- 設置回聲伺服器到 echo 服務器的服務器證書。
- 設置運輸 _ 客戶 _ 證書至預先產生的用戶端憑證。
- 設置運輸 _ 用戶端 _ 私人密鑰到預先生成的客戶端私鑰。

構建並刷新應用程序

使用您選擇的工具鏈來構建和刷新應用程序。何時`runQualificationTest()`被調用，傳輸接口測試將運行。測試結果會輸出至序列埠。

> **Note**
>
> 要正式使用 FreeRTOS 的設備資格，您必須根據 OTA PAL 和 OTA E2E 測試組驗證設備的移植源代碼AWS IoT Device Tester。按照說明進行操作使用AWS IoT Device Tester對於FreeRTOS在FreeRTOS 用者指南若要設定AWS IoT Device Tester用於連接埠驗證。若要測試特定程式庫的連接埠，必須在`device.json`檔案中的AWS IoT Device Tester `configs`資料夾。

設定核心 QTT 程式庫

邊緣上的裝置可以使用 MQTT 通訊協定，與 AWS 雲端通訊。AWS IoT 託管 MQTT 中介裝置，可往返於邊緣上連線的裝置來傳送和接收訊息。

CoremQTT 程式庫會為執行 FreeRTOS 的裝置實作 MQTT 通訊協定。CoremQtt 程式庫不需要移植，但是您的裝置的測試專案必須通過所有 MQTT 測試以取得資格。如需詳細資訊，請參閱《FreeRTOS 使用者指南》中的 CoremQTT 程式庫。

先決條件

若要設定 CoremQtt 程式庫測試，您需要網路傳輸介面連接埠。請參閱移植網路傳輸介面以瞭解更多資訊。

測試

執行核心 QTT 整合測試：

- 向 MQTT 代理程式註冊您的用戶端憑證。
- 在中設置代理端點`config`並運行集成測試。

建立參考 MQTT 示範

我們建議您使用 CoremQtt 代理程式來處理所有 MQTT 作業的執行緒安全性。使用者還需要發佈和訂閱工作，以及裝置建議程式測試，以驗證應用程式是否有效整合 TLS、MQTT 和其他 FreeRTOS 程式庫。

若要正式符合 FreeRTOS 的裝置資格，請使用AWS IoT Device Tester MQTT 測試案例驗證您的整合專案。如需設定和測試的指示，請參閱AWS IoT裝置建議程式工作流程。TLS 和 MQTT 的強制性測試用例如下所示：

TLS 測試案例

測試用例	測試用例	必要的測試
TLS	TLS Connect	是
TLS	TLS SupportAWS IoT 密碼套件	建議使用的密碼套件
TLS	TLS 不安全伺服器憑證	是
TLS	TLS 不正確的主旨名稱伺服器證書	是

MQTT 測試用例

測試用例	測試用例	必要的測試
MQTT	MQTT TT TT TT TT TT Connect	是
MQTT	MQTT Connect 抖動重試	是，沒有警告
MQTT	MQTT 訂閱	是
MQTT	MQTT 出版	是

測試用例	測試用例	必要的測試
MQTT	MQTT ClientPuback QoS1	是
MQTT	MQTT 沒有確認 PingResp	是

設定核心 HTTP 程式庫

邊緣裝置可以使用 HTTP 通訊協定與AWS雲端通訊。 AWS IoT服務主控 HTTP 伺服器，該伺服器會在邊緣向連線裝置傳送和接收訊息。

測試

請按照以下步驟進行測試：

- 將 PKI 設定為與AWS或 HTTP 伺服器進行 TLS 相互驗證。
- 執行核心 HTTP 整合測試。

移植AWS IoT over-the-air (OTA) 更新程式庫

您可以使用 FreeRTOS over-the-air (OTA) 更新執行下列操作：

- 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。
- 將韌體新增至群組、重設或重新佈建時，將韌體部署至裝置。
- 在新韌體部署到裝置後驗證其真偽和完整性。
- 監控部署進度。
- 對失敗的部署進行除錯。
- 使用適用於 AWS IoT 的程式碼簽署來數位簽署韌體。

如需詳細資訊，請參閱 FreeRT OS 使用者指南中的無線更新以及 AWS IoTOver-the-air 更新文件。

您可以使用 OTA 更新程式庫將 OTA 功能整合到 FreeRTOS 應用程式中。如需詳細資訊，請參閱 FreeRTOS 使用者指南中的《FreeRTOS OTA 更新程式庫》。

FreeRTOS 裝置必須在收到的 OTA 韌體映像上強制執行加密程式碼簽章驗證。我們建議以下演算法：

- 橢圓曲線數位簽章演算法 (ECDSA)
- NIST P256 曲線
- SHA-256 雜湊

先決條件

- 完成中的指示設定您的工作區和專案以進行移植。
- 建立網路傳輸介面連接埠。

  如需相關資訊，請參閱 移植網路傳輸介面。
- 整合核心 QTT 程式庫。請參閱 FreeRTOS 使用者指南中的 CoremQTT 程式庫。
- 創建一個可以支持 OTA 更新的引導加載程序。

平台移植

您必須提供 OTA 便攜式抽象層（PAL）的實現，以將 OTA 庫移植到新設備。PAL API 定義於必須提供實作特定詳細資訊的檔案中。

函數名稱	描述
`otaPal_Abort`	停止 OTA 更新。
`otaPal_CreateFileForRx`	創建一個文件來存儲接收的數據塊。
`otaPal_CloseFile`	關閉指定的檔案。如果您使用實作加密保護的儲存體，這可能會驗證檔案。
`otaPal_WriteBlock`	將資料區塊寫入指定檔案中的指定位移。如果成功，函數返回寫入的字節總數。否則，該函數返回一個負的錯誤代碼。塊大小將始終是 2 的冪，並將被對齊。如需詳細資訊，請參閱 OTA 程式庫設定。
`otaPal_ActivateNewImage`	啟用或啟動新的韌體映像。對於某些端口，如果設備以編程方式同步重置，則此功能將不會返回。

函數名稱	描述
otaPal_SetPlatformImageState	執行讓平台接受或拒絕最新 OTA 韌體映像 (或套件) 所需的動作。若要實作此功能，請參閱您主機板 (平台) 詳細資料和架構的說明文件。
otaPal_GetPlatformImageState	取得 OTA 更新映像的狀態。

如果您的裝置內建支援此表格中的函數，請實作這些函數。

函數名稱	描述
otaPal_CheckFileSignature	驗證指定檔案的簽章。
otaPal_ReadAndAssumeCertificate	從檔案系統讀取指定的簽署者憑證，並傳回給發起人。
otaPal_ResetDevice	重設裝置。

> **Note**
>
> 確定您具有可支援 OTA 更新的開機載入器。有關創建AWS IoT設備引導加載程序的說明，請參閱IoT 裝置開機載入器。

E2E 和 PAL 測試

運行 OTA PAL 和 E2E 測試。

端對端測試

OTA 端對端（E2E）測試用於驗證設備的 OTA 功能並從現實中模擬場景。此測試將包括錯誤處理。

先決條件

若要移至這個測試，您需要以下資訊：

- 一個集成了 OAWS TA 圖書館的項目。有關更多信息，請訪問 OTA 圖書館移植指南。

- 使用 OTA 庫移植演示應用程序以與之交互AWS IoT Core以進行 OTA 更新。請參閱 移植 OTA 演示應用程序。
- 設定 IDT 工具。這將運行 OTA E2E 主機應用程序以構建，刷新和監視具有不同配置的設備，並驗證 OTA 庫集成。

移植 OTA 演示應用程序

OTA E2E 測試必須具有 OTA 演示應用程序才能驗證 OTA 庫集成。演示應用程序必須具有執行 OTA 固件更新的能力。您可以在 FreeRT OS GitHub 儲存庫中找到免費伺服器 OTA 示範應用程式。我們建議您使用演示應用程序作為參考，並根據您的規格進行修改。

移植步驟

1. 初始化 OTA 代理程式。
2. 實現 OTA 應用程序回調函數。
3. 建立 OTA 代理程式事件處理工作。
4. 啟動 OTA 代理程式。
5. 監視 OTA 代理程式統計資料。
6. 關閉 OTA 代理程式。

通過 MQTT 訪問 FreeRTOS OTA-演示的入口點以獲取詳細說明。

組態

以下是與之互動所必需的配置AWS IoT Core：

- AWS IoT Core用戶端認證
  - 在亞馬遜信任服務端點中`Ota_Over_Mqtt_Demo/demo_config.h`設定示範設定 Root_CA_PEM。如需詳細資訊，請參閱AWS伺服器驗證
  - `Ota_Over_Mqtt_Demo/demo_config.h`使用您的客戶端憑據設置演示配置AWS IoT客戶端證書 請參閱用戶AWS端驗證詳細資料，以瞭解用戶端憑證和私密金鑰。
- 應用程式版本
- OTA 控制協定
- OTA 資訊協定
- 程式碼簽章認證
- 其他 OTA 庫配置

您可以在 FreeRTOS OTA 示範應用程式`ota_config.h`中找到上述資訊。`demo_config.h`如需詳細資訊，請造訪透過 MQTT 的 FreeRTOS OTA-設定裝置。

建立驗證

運行演示應用程序以運行 OTA 作業。成功完成後，您可以繼續運行 OTA E2E 測試。

FreeRTOS OTA 演示提供了有關在 FreeRTOS 窗口模擬器上設置 OAWS IoT Core TA 客戶端和 OTA 作業的詳細信息。 AWSOTA 同時支持 MQTT 和 HTTP 協議。如需詳細資訊，請參閱下列範例：

- 視窗模擬器上的 MQTT 演示 OTA
- 通過視窗模擬器上的 HTTP 演示 OTA

使用 IDT 工具執行測試

要運行 OTA E2E 測試，您必須使用AWS IoT Device Tester（IDT）自動執行。AWS IoT Device Tester如需詳細資訊，請參閱 FreeRTOS 使用者指南中的 FreeRTOS。

端子測試用例

測試案例	描述
`OTAE2EGreaterVersion`	定期 OTA 更新的快樂路徑測試。它會建立具有較新版本的更新，裝置會成功更新。
`OTAE2EBackToBackDownloads`	此測試會連續建立 3 次 OTA 更新。裝置預期會連續更新 3 次。
`OTAE2ERollbackIfUnableToConnectAfterUpdate`	如果設備無法連接到具有新固件的網絡，則此測試將驗證設備是否還原到先前的固件。
`OTAE2ESameVersion`	此測試確認如果版本保持不變，則設備拒絕傳入的固件。
`OTAE2EUnsignedImage`	如果映像未簽署，此測試會驗證裝置是否拒絕更新。
`OTAE2EUntrustedCertificate`	如果韌體使用不受信任的憑證簽署，此測試會驗證裝置是否拒絕更新。

測試案例	描述
OTAE2EPreviousVersion	此測試會驗證裝置是否拒絕較舊的更新版本。
OTAE2EIncorrectSigningAlgorithm	不同的設備支持不同的簽名和哈希算法。如果使用不支持的算法創建，則此測試驗證設備是否會失敗 OTA 更新。
OTAE2EDisconnectResume	這是暫停和恢復功能的快樂路徑測試。此測試會建立 OTA 更新並開始更新。然後，它會AWS IoT Core使用相同的用戶端 ID (物件名稱) 和認證連線到。 AWS IoT Core然後斷開設備的連接。預計該設備會檢測到它與斷開連接AWS IoT Core，並在一段時間後將自己移動到暫停狀態，並嘗試重新連接到AWS IoT Core並繼續下載。
OTAE2EDisconnectCancelUpdate	如果 OTA 作業處於暫停狀態時取消，此測試將檢查設備是否可以自行恢復。它與OTAE2EDisconnectResume 測試執行相同的操作，除了在連接到AWS IoT Core斷開設備的連接後，它會取消 OTA 更新。隨即建立新的更新。裝置預期會重新連線到AWS IoT Core、中止目前的更新、回到等待中狀態，然後接受並完成下一次更新。
OTAE2EPresignedUrlExpired	創建 OTA 更新時，您可以配置 S3 預簽名 URL 的生命週期。此測試驗證設備是否能夠執行 OTA，即使在 url 過期時無法完成下載也是如此。裝置預期會要求新的工作文件，其中包含可繼續下載的新 URL。
OTAE2E2UpdatesCancel1st	此測試連續創建兩個 OTA 更新。當設備報告正在下載第一個更新時，測試會強制取消第一次更新。裝置預期會中止目前的更新並取得第二個更新，然後完成更新。

測試案例	描述
OTAE2ECancelThenUpdate	此測試連續創建兩個 OTA 更新。當設備報告正在下載第一個更新時，測試會強制取消第一次更新。裝置預期會中止目前的更新並取得第二個更新，然後完成更新。
OTAE2EImageCrashed	此測試會檢查設備是否能夠在映像崩潰時拒絕更新。

PAL 測試

先決條件

若要移至網路傳輸界面測試，您需要以下資訊：

- 可以使用有效的 FreeRTOS 核心連接埠建置 FreeRTOS 的專案。
- OTA PAL 的工作實施。

移植

- 將 FreeRTOS 庫集成測試作為子模塊添加到您的項目中。項目中子模塊的位置必須是可以構建的位置。
- 將config_template/test_execution_config_template.h和複製config_template/test_param_config_template.h到構建路徑中的位置，並將其重命名為test_execution_config.h和test_param_config.h。
- 在建置系統中包含相關檔案。如果使用CMakequalification_test.cmake，則src/ota_pal_tests.cmake可用於包含相關文件。
- 透過實作下列功能來設定測試：
  - SetupOtaPalTestParam()：定義於src/ota/ota_pal_test.h。該實現必須具有與中定義相同的名稱和簽名ota_pal_test.h。目前，您不需要設定這項功能。
- 實施 UNITY_OUTPUT_CHAR，以便測試輸出日誌不會與設備日誌交錯。
- RunQualificationTest()從應用程序調用。設備硬件必須正確初始化，並且必須在呼叫之前連接網絡。

測試

本節介紹 OTA PAL 資格測試的本地測試。

啟用測試

打開`test_execution_config.h`並定義奧塔 _ 帕 _ 測試 _ 啟用為 1。

在中`test_param_config.h`，更新下列選項：

- 驗證類型：選取使用的憑證類型。
- 憑證檔案：裝置憑證的路徑 (如果適用)。
- 韌體檔案：韌體檔案的名稱 (如果適用)。
- 系統：如果 OTA PAL 使用文件系統抽象，則設置為 1。

使用您選擇的工具鏈來建置和更新應用程式。當`RunQualificationTest()`被調用時，OTA PAL 測試將運行。測試結果會輸出到序列埠。

整合 OTA 工作

- 將 OTA 代理程式新增至您目前的 MQTT 示範。
- 使用. 運行 OTA 端對端（E2E）測試AWS IoT。這會驗證整合是否如預期般運作。

> **Note**
>
> 要正式使用 FreeRTOS 的設備資格，您必須使用AWS IoT Device Tester. 請依照 FreeRTOS 使AWS IoT Device Tester用者指南中的使用 FreeRTOS 中的指示來設AWS IoT Device Tester 定連接埠驗證。若要測試特定程式庫的連接埠，必須在AWS IoT Device Tester`configs`資料夾中的`device.json`檔案中啟用正確的測試群組。

IoT 裝置開機載入器

您必須提供自己的安全引導加載程序應用程序。確保設計和實施可以適當緩解安全威脅。以下是威脅建模，供您參考。

IoT 裝置開機載入器的威脅模型

背景介紹

作為工作定義，此威脅模型引用的嵌入式AWS IoT設備是基於微控制器的產品，可與雲服務進行交互。它們可以部署在消費者、商業或工業環境。IoT 裝置可能會收集有關使用者、病患、機器或環境的資料，並可以控制燈泡和門鎖到工廠機械等任何項目。

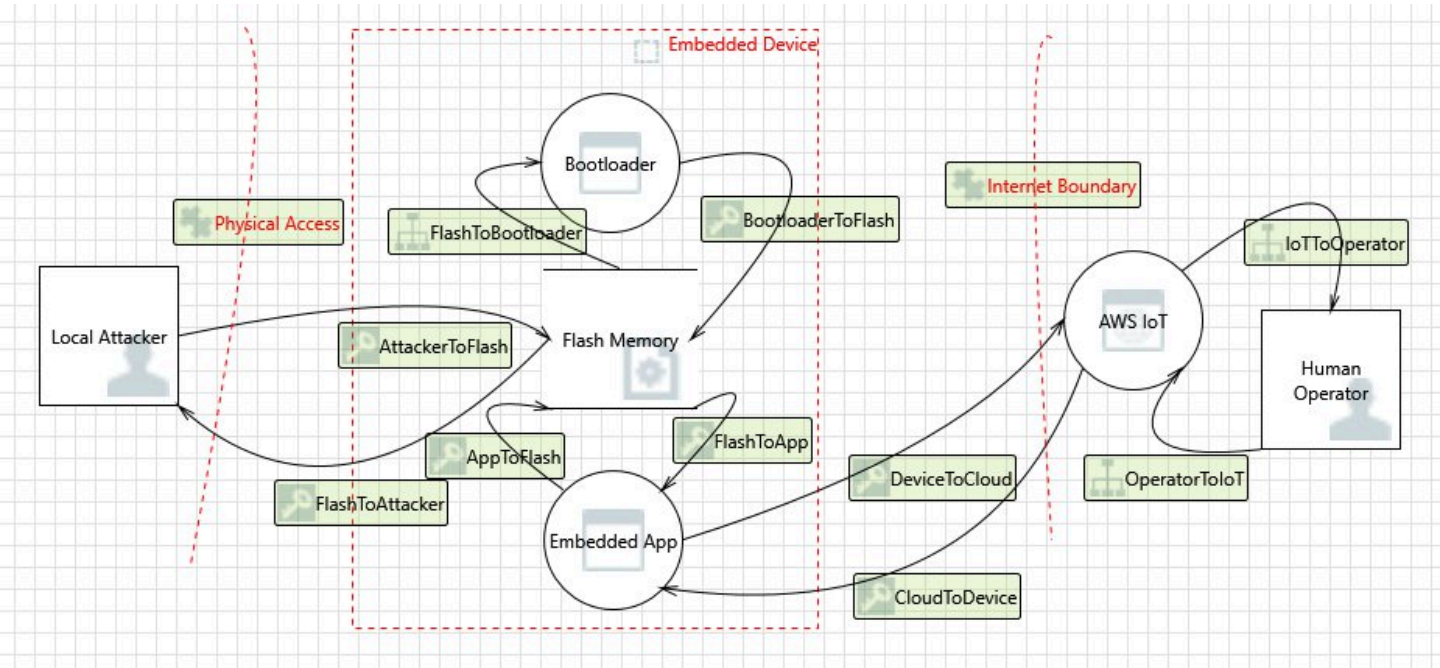
威脅模型是假設從對手的角度來檢視安全的方法。通過考慮對手的目標和方法，將創建一個威脅列表。威脅是對手對資源或資產的攻擊。此清單會排定優先順序，並用來識別和建立緩和解決方案。在選擇緩解解決方案時，實施和維護它的成本應與其提供的真正安全價值保持平衡。有多種 威脅模型方法。每個都能夠支持安全和成功的AWS IoT產品的開發。

FreeRTOS 為裝置提供 OTA (over-the-air) 軟AWS IoT體更新。更新設施結合雲端服務與裝置內軟體程式庫和合作夥伴提供的開機載入器。此威脅模型特別著重於對開機載入器的威脅。

開機載入器使用案例

- 在部署前數位簽署及加密韌體。
- 將新的韌體映像部署到單一裝置、裝置群組，或是整個機群。
- 在韌體部署到裝置後驗證其真確性及完整性。
- 裝置只能從信任來源執行未經修改的軟體。
- 裝置可透過 OTA 接收的故障軟體具備彈性。

資料流程圖



威脅

某些攻擊具有多種緩解模式；例如，透過驗證 TLS 伺服器所提供之憑證和新韌體映像檔的代碼簽署者憑證的信任，藉此減輕 man-in-the-middle 打算提供惡意韌體映像檔的網路。為了最大化引導加載程序的安全性，任何非引導加載程序緩解解決方案都被認為是不可靠的。引導加載程序應該具有針對每個攻擊的內在緩解解決方案。具有分層緩解解決方案被稱為 defense-in-depth.

威脅：

- 攻擊者劫持裝置與伺服器的連線，以傳遞惡意韌體映像。

  防護範例

  - 開機時，開機載入器會使用已知的憑證來驗證映像的加密簽章。如果驗證失敗，開機載入器會回復至前一個映像。
- 攻擊者利用緩衝區溢位，將惡意行為引入儲存在快閃中的現有韌體映像。

  防護範例

  - 開機時，開機載入器會進行驗證，如前所述。當驗證失敗且沒有可用的先前映像時，開機載入器會停止。
  - 開機時，開機載入器會進行驗證，如前所述。當驗證失敗且沒有先前的映像可用時，引導加載程序將進入僅限故障安全的 OTA 模式。
- 攻擊者可利用裝置開機至先前儲存的映像。

防護範例

- 成功安裝和測試新映像時，會清除儲存最後一個映像的快閃磁區。
- 每次成功升級時就會燒毀一個保險絲，而每個影像會拒絕執行，除非已燒毀正確的保險絲數量。

- OTA 會更新提供損壞裝置的故障或惡意映像。

防護範例

- 開機載入器會啟動硬體監視計時器，觸發回復到之前的映像。

- 攻擊者會修補開機載入器以繞過映像驗證，讓裝置接受未簽署的映像。

防護範例

- 開機載入器位於 ROM (唯讀記憶體) 中，且無法修改。
- 引導加載程序位於 OTP（one-time-programmable 內存）中，無法修改。
- 引導加載程序位於 ARM 的安全區域中 TrustZone，無法修改。

- 攻擊者會取代驗證憑證，讓裝置能夠接受惡意映像。

防護範例

- 憑證位於密碼編譯共同處理器中，且無法修改。
- 憑證位於 ROM (或 OTP 或安全區域) 中，且無法修改。

進一步建立威脅模型

此威脅模型只會考量開機載入器。進一步建立威脅模型可提升整體安全性。建議的方法是列出對手的目標、鎖定的資產目標以及資產的進入點。威脅清單可以透過考慮攻擊的進入點，取得資產的控制權。以下列出 IoT 裝置的目標、資產和進入點的範例。這些清單並不詳盡，旨在刺激進一步思考。

對手的目標

- 侵占金錢
- 損害名譽
- 偽造資料
- 轉移資源
- 遠端監視目標
- 取得對網站的實體存取
- 嚴重破壞

- 安裝惡意軟體

關鍵資產

- 私密金鑰
- 用戶端憑證
- CA 根憑證
- 安全登入資料和字符
- 客戶的個人識別資訊
- 商業秘密的實作
- 感應器資料
- 雲端分析資料存放區
- 雲端基礎設施

進入點

- DHCP 回應
- DNS 回應
- 透過 TLS 的 MQTT
- HTTPS 回應
- OTA 軟體映像
- 其他，如應用程式所指定，例如 USB
- 匯流排的實體通道
- 已宣告的 IC

移植蜂窩接口庫

FreeRTOS 支援 TCP 卸載的蜂巢式抽象層的 AT 命令。如需詳細資訊，請參閱行動網路介面程式庫和在 freertos.org 上移植行動網路介面程式庫。

先決條件

行動網路介面程式庫沒有直接相依性。但是，在 FreeRTOS 網路堆疊中，乙太網路、Wi-Fi 和行動網路無法共存，因此開發人員必須選擇其中一個與移植網路傳輸介面.

> **Note**
>
> 如果蜂窩模塊能夠支持 TLS 卸載，或不支持 AT 命令，則開發人員可以實現自己的蜂窩抽象以與移植網路傳輸介面.

從 MQTT 第三版移轉至核心

本移轉指南說明如何將應用程式從 MQTT 移轉至 CoremQTT。

從 OTA 應用程式遷移至第 3 版

本指南將幫助您將應用程序從 OTA 庫版本 1 遷移到版本 3。

> **Note**
>
> OTA 版本 2 API 與 OTA v3 API 相同，因此，如果您的應用程序使用的是第 2 版的 API，則 API 調用不需要更改，但我們建議您集成庫的版本 3。

OTA 版本 3 的演示可以在這裡找到：

- 發現 _ 核心 _ mqtt.
- 演示 _ 核心 _ HTTP.
- 發泡。

API 變更摘要

OTA 庫版本 1 和版本 3 之間的 API 更改摘要

OTA AUT版本 1 版	OTA AUT版本 3 版	變更說明
太田 _AgentInit	初始化	由於 OTA v3 中的實現發生變化，輸入參數以及從函數返回的值也會發生變化。有關詳細信息，請參閱以下 Ota_init 部分。
太田 _AgentShutdown	大關機	輸入參數的變更，包括選擇性取消訂閱 MQTT 主題的附加參數。有關詳細信息，請參閱下面的 OTA 關閉部分。
太田 _GetAgentState	太田 _GetState	API 名稱會變更，但不會變更輸入參數。返回值是相同的，但枚舉和成員被重命名。有關

OTA AUT版本 1 版	OTA AUT版本 3 版	變更說明
		詳細信息，請參閱GetState 下面的 OTA_ 部分。
N/A	太田 _GetStatistics	添加了新的 API，取代了 API OT_GetPacketsRecei ved，OT_GetPacketsQ ueued，OT_GetPacket sProcessed，OTA _GetPacke tsDropped。有關詳細信息，請參閱GetStatistics 下面的 OTA_ 部分。
太田 _GetPacketsReceived	N/A	此 API 已從版本 3 中移除，並由 OT_ 取代GetStatistics。
太田 _GetPacketsQueued	N/A	此 API 已從版本 3 中移除，並由 OT_ 取代GetStatistics。
太田 _GetPacketsProcessed	N/A	此 API 已從版本 3 中移除，並由 OT_ 取代GetStatistics。
太田 _GetPacketsDropped	N/A	此 API 已從版本 3 中移除，並由 OT_ 取代GetStatistics。
太田 _ActivateNewImage	太田 _ActivateNewImage	輸入參數相同，但返回 OTA 錯誤代碼被重命名，並在 OTA 庫的版本 3 中添加了新的錯誤代碼。有關詳細信息，請參見 OTA_ActivateNewImage 部分。
太田 _SetImageState	太田 _SetImageState	輸入參數相同並重命名，返回 OTA 錯誤代碼被重命名，並在 OTA 庫的版本 3 中添加了新的錯誤代碼。有關詳細信息，請參見 OTA_SetImageState 部分。

OTA AUT版本 1 版	OTA AUT版本 3 版	變更說明
太田 _GetImageState	太田 _GetImageState	輸入參數是相同的，返回枚舉在 OTA 庫的版本 3 中重命名。有關詳細信息，請參見 OTA_GetImageState 部分。
宅暫停	宅暫停	輸入參數相同，返回 OTA 錯誤代碼被重命名，並在 OTA 庫的版本 3 中添加了新的錯誤代碼。有關詳細信息，請參閱「發燒暫停」部分。
御宅履歷	御宅履歷	在 OTA 演示/應用程序中處理連接時，連接的輸入參數被刪除，返回的 OTA 錯誤代碼被重命名，並在 OTA 庫的版本 3 中添加了新的錯誤代碼。有關詳細信息，請參閱發展簡歷部分。
太田 _CheckForUpdate	太田 _CheckForUpdate	輸入參數相同，返回 OTA 錯誤代碼被重命名，並在 OTA 庫的版本 3 中添加了新的錯誤代碼。有關詳細信息，請參見 OTA_CheckForUpdate 部分。
N/A	太田 _EventProcessingTask	添加了新的 API，它是處理 OTA 更新事件的主要事件循環，必須由應用程序任務調用。有關詳細信息，請參見 OTA_EventProcessingTask 部分。

OTA AUT版本 1 版	OTA AUT版本 3 版	變更說明
N/A	太田 _SignalEvent	添加了新的 API，它將事件添加到 OTA 事件隊列的後面，並由內部 OTA 模塊用於向代理任務發出信號。有關詳細信息，請參見 OTA_SignalEvent 部分。
N/A	奧塔埃尔斯特朗	用於錯誤代碼到 OTA 錯誤的字符串轉換的新 API。
N/A	OTJobParse _ 斯特雷羅	用於將錯誤代碼轉換為字符串的新 API，用於 Job 分析錯誤。
N/A	OTOsStatus _ 斯特雷羅	用於 OTA OS 端口狀態的狀態代碼到字符串轉換的新 API。
N/A	OTPalStatus _ 斯特雷羅	用於 OTA PAL 端口狀態的狀態代碼到字符串轉換的新 API。

所需變更說明

初始化

在 v1 中初始化 OTA 代理時，將使用`OTA_AgentInit` API，該 API 將連接上下文，事物名稱，完整回調和超時的參數作為輸入。

```
OTA_State_t OTA_AgentInit( void * pvConnectionContext,
                           const uint8_t * pucThingName,
                           pxOTACompleteCallback_t xFunc,
                           TickType_t xTicksToWait );
```

此 API 現在已更改為`OTA_Init` ota，ota 接口，事物名稱和應用程序回調所需緩衝區的參數。

```
OtaErr_t OTA_Init( OtaAppBuffer_t * pOtaBuffer,
```

```
                    OtaInterfaces_t * pOtaInterfaces,
                    const uint8_t * pThingName,
                    OtaAppCallback OtaAppCallback );
```

刪除輸入參數-

pvConnectionContext -

連接上下文被刪除，因為 OTA 庫版本 3 不需要將連接上下文傳遞給它，並且 MQTT/HTTP 操作由 OTA 演示/應用程序中的各自接口處理。

xTicksTo等待-

在調用 Ota_init 之前，在 OTA 演示/應用程序中創建任務時，等待參數的刻度也被刪除。

重新命名輸入參數-

依 FUNC-

參數會重新命名為， OtaAppCallback 且其類型會變更為 OtaAppCallback _t。

新增輸入參數-

pOtaBuffer

應用程序必須分配緩衝區，並在初始化期間使用 OtaAppBuffer _t 結構將它們傳遞給 OTA 庫。根據用於下載檔案的通訊協定，所需的緩衝區會略有不同。對於 MQTT 協議，流名稱的緩衝區是必需的，對於 HTTP 協議，預先簽名的 url 和授權方案的緩衝區是必需的。

使用 MQTT 進行文件下載時需要的緩衝區-

```
static OtaAppBuffer_t otaBuffer =
{
    .pUpdateFilePath    = updateFilePath,
    .updateFilePathsize = otaexampleMAX_FILE_PATH_SIZE,
    .pCertFilePath      = certFilePath,
    .certFilePathSize   = otaexampleMAX_FILE_PATH_SIZE,
    .pStreamName        = streamName,
    .streamNameSize     = otaexampleMAX_STREAM_NAME_SIZE,
    .pDecodeMemory      = decodeMem,
    .decodeMemorySize   = ( 1U << otaconfigLOG2_FILE_BLOCK_SIZE ),
    .pFileBitmap        = bitmap,
    .fileBitmapSize     = OTA_MAX_BLOCK_BITMAP_SIZE
};
```

使用 HTTP 進行文件下載時所需的緩衝區-

```
static OtaAppBuffer_t otaBuffer =
{
    .pUpdateFilePath    = updateFilePath,
    .updateFilePathsize = otaexampleMAX_FILE_PATH_SIZE,
    .pCertFilePath      = certFilePath,
    .certFilePathSize   = otaexampleMAX_FILE_PATH_SIZE,
    .pDecodeMemory      = decodeMem,
    .decodeMemorySize   = ( 1U << otaconfigLOG2_FILE_BLOCK_SIZE ),
    .pFileBitmap        = bitmap,
    .fileBitmapSize     = OTA_MAX_BLOCK_BITMAP_SIZE,
    .pUrl               = updateUrl,
    .urlSize            = OTA_MAX_URL_SIZE,
    .pAuthScheme        = authScheme,
    .authSchemeSize     = OTA_MAX_AUTH_SCHEME_SIZE
};
```

哪裡-

```
    pUpdateFilePath    Path to store the files.
    updateFilePathsize Maximum size of the file path.
    pCertFilePath      Path to certificate file.
    certFilePathSize   Maximum size of the certificate file path.
    pStreamName        Name of stream to download the files.
    streamNameSize     Maximum size of the stream name.
    pDecodeMemory      Place to store the decoded files.
    decodeMemorySize   Maximum size of the decoded files buffer.
    pFileBitmap        Bitmap of the parameters received.
    fileBitmapSize     Maximum size of the bitmap.
    pUrl               Presigned url to download files from S3.
    urlSize            Maximum size of the URL.
    pAuthScheme        Authentication scheme used to validate download.
    authSchemeSize     Maximum size of the auth scheme.
```

pOtaInterfaces

Ota_init 的第二個輸入參數是對類型 OtaInterfaces _t 的 OTA 接口的引用。這組接口必須傳遞給 OTA 庫，並在操作系統接口中包含 MQTT 接口，HTTP 接口和平台抽象層接口。

OTA OS 界面

OTA OS 功能接口是一組必須實現的 API，該 API 必須為設備使用 OTA 庫。此接口的函數實現提供給用戶應用程序中的 OTA 庫。OTA 庫調用函數實現來執行通常由操作系統提供的

功能。這包括管理事件，計時器和內存分配。FreeRTOS 和 POSIX 的實現與 OTA 庫一起提供。

使用提供的 FreeRTOS 連接埠的 FreeRTOS 範例-

```
OtaInterfaces_t otaInterfaces;
otaInterfaces.os.event.init   = OtaInitEvent_FreeRTOS;
otaInterfaces.os.event.send   = OtaSendEvent_FreeRTOS;
otaInterfaces.os.event.recv   = OtaReceiveEvent_FreeRTOS;
otaInterfaces.os.event.deinit = OtaDeinitEvent_FreeRTOS;
otaInterfaces.os.timer.start  = OtaStartTimer_FreeRTOS;
otaInterfaces.os.timer.stop   = OtaStopTimer_FreeRTOS;
otaInterfaces.os.timer.delete = OtaDeleteTimer_FreeRTOS;
otaInterfaces.os.mem.malloc   = Malloc_FreeRTOS;
otaInterfaces.os.mem.free     = Free_FreeRTOS;
```

使用提供的 POSIX 端口的 Linux 示例-

```
OtaInterfaces_t otaInterfaces;
otaInterfaces.os.event.init    = Posix_OtaInitEvent;
otaInterfaces.os.event.send    = Posix_OtaSendEvent;
otaInterfaces.os.event.recv    = Posix_OtaReceiveEvent;
otaInterfaces.os.event.deinit  = Posix_OtaDeinitEvent;
otaInterfaces.os.timer.start   = Posix_OtaStartTimer;
otaInterfaces.os.timer.stop    = Posix_OtaStopTimer;
otaInterfaces.os.timer.delete  = Posix_OtaDeleteTimer;
otaInterfaces.os.mem.malloc    = STDC_Malloc;
otaInterfaces.os.mem.free      = STDC_Free;
```

MQTT MQTT MQTT

OTA MQTT 接口是一組必須在庫中實現的 API，以使 OTA 庫能夠從流媒體服務下載文件塊。

透過 MQTT 示範，使用 OTA 中的 CoremQtt 代理程式 API 的範例-

```
OtaInterfaces_t otaInterfaces;
otaInterfaces.mqtt.subscribe = prvMqttSubscribe;
otaInterfaces.mqtt.publish = prvMqttPublish;
otaInterfaces.mqtt.unsubscribe = prvMqttUnSubscribe;
```

HTTP Futface

OTA HTTP 接口是一組必須在庫中實現的 API，以使 OTA 庫通過連接到預簽名的 url 並獲取數據塊來下載文件塊。除非您將 OTA 庫配置為從預簽名的 URL 而不是流服務下載，否則它是可選的。

通過 HTTP 演示使用來自 OTA 的核心 HTTP API 的示例-

```
OtaInterfaces_t otaInterfaces;
otaInterfaces.http.init = httpInit;
otaInterfaces.http.request = httpRequest;
otaInterfaces.http.deinit = httpDeinit;
```

A塔 PAL 界面

OTA PAL 接口是一組必須實現的 API，該 API 必須為設備使用 OTA 庫。OTA PAL 的設備特定實現提供給用戶應用程序中的庫。庫使用這些功能來存儲，管理和驗證下載。

```
OtaInterfaces_t otaInterfaces;
otaInterfaces.pal.getPlatformImageState = otaPal_GetPlatformImageState;
otaInterfaces.pal.setPlatformImageState = otaPal_SetPlatformImageState;
otaInterfaces.pal.writeBlock = otaPal_WriteBlock;
otaInterfaces.pal.activate = otaPal_ActivateNewImage;
otaInterfaces.pal.closeFile = otaPal_CloseFile;
otaInterfaces.pal.reset = otaPal_ResetDevice;
otaInterfaces.pal.abort = otaPal_Abort;
otaInterfaces.pal.createFile = otaPal_CreateFileForRx;
```

回報的變化-

返回從 OTA 代理狀態更改為 OTA 錯誤代碼。請參閱 AWS IoTOver-the-air 更新版本 3.0.0： OtaErr_t。

大關機

在 OTA 庫版本 1 中，用於關閉 OTA 代理的 API 是 OT_AgentShutdown ，現在已更改為 Ota_Shutdown 以及輸入參數的更改。

OTA 代理程式關閉 (版本 1)

```
OTA_State_t OTA_AgentShutdown( TickType_t xTicksToWait );
```

OTA 代理程式關閉 (版本 3)

```
OtaState_t OTA_Shutdown( uint32_t ticksToWait,
                         uint8_t unsubscribeFlag );
```

ticksToWait -

等待 OTA 代理程式完成關機程序的刻度數。如果設置為零，該函數將立即返回，而無需等待。實際狀態會傳回呼叫者。代理程式在此期間不會進入睡眠狀態，而是用於忙碌迴圈。

新輸入參數-

取消訂閱旗標-

此旗標可指出呼叫 shutdown 時，是否應從工作主題執行取消訂閱作業。如果旗標為 0，則不會針對工作主題呼叫取消訂閱作業。如果應用程式必須從工作主題中取消訂閱，則呼叫 Ota_Shutdown 時必須將此旗標設定為 1。

回報的變化-

OtaState_t-

OTA 代理狀態及其成員的枚舉被重命名。請參閱 AWS IoTOver-the-air 更新版本 3.0.0。

太田 _GetState

API 名稱已從 OT_ 變更AgentGetState 為 OT_GetState。

OTA 代理程式關閉 (版本 1)

```
OTA_State_t OTA_GetAgentState( void );
```

OTA 代理程式關閉 (版本 3)

```
OtaState_t OTA_GetState( void );
```

回報的變化-

OtaState_t-

OTA 代理狀態及其成員的枚舉被重命名。請參閱 AWS IoTOver-the-air 更新版本 3.0.0。

太田 _GetStatistics

為統計添加了新的單個 API。它取代了 API OT_GetPacketsReceived，OTA _GetPacketsQueued，OTA _GetPacketsProcessed，OTA _GetPacketsDropped。此外，在 OTA 圖書館版本 3 中，統計數字僅與當前作業有關。

O塔圖書館第 1 版

```
uint32_t OTA_GetPacketsReceived( void );
uint32_t OTA_GetPacketsQueued( void );
uint32_t OTA_GetPacketsProcessed( void );
uint32_t OTA_GetPacketsDropped( void );
```

O塔圖書館第 3 版

```
OtaErr_t OTA_GetStatistics( OtaAgentStatistics_t * pStatistics );
```

個人統計數字-

輸入/輸出參數用於統計數據，例如當前作業接收，丟棄，排隊和處理的數據包。

輸出參數-

OTA 錯誤代碼。

範例用量-

```
OtaAgentStatistics_t otaStatistics = { 0 };
OTA_GetStatistics( &otaStatistics );
LogInfo( ( " Received: %u   Queued: %u   Processed: %u   Dropped: %u",
                           otaStatistics.otaPacketsReceived,
                           otaStatistics.otaPacketsQueued,
                           otaStatistics.otaPacketsProcessed,
                           otaStatistics.otaPacketsDropped ) );
```

太田 _ActivateNewImage

輸入參數相同，但返回的 OTA 錯誤代碼被重命名，並在 OTA 庫的版本 3 中添加了新的錯誤代碼。

O塔圖書館第 1 版

```
OTA_Err_t OTA_ActivateNewImage( void );
```

O塔圖書館第 3 版

```
OtaErr_t OTA_ActivateNewImage( void );
```

返回 OTA 錯誤代碼枚舉已更改，並添加了新的錯誤代碼。請參閱 AWS IoTOver-the-air 更新版本 3.0.0： OtaErr_t。

範例用量-

```
OtaErr_t otaErr = OtaErrNone;
otaErr = OTA_ActivateNewImage();
/* Handle error */
```

太田 _SetImageState

輸入參數相同並重命名，返回 OTA 錯誤代碼被重命名，並在 OTA 庫的版本 3 中添加了新的錯誤代碼。

O塔圖書館第 1 版

```
OTA_Err_t OTA_SetImageState( OTA_ImageState_t eState );
```

O塔圖書館第 3 版

```
OtaErr_t OTA_SetImageState( OtaImageState_t state );
```

輸入參數會重新命名為 OtaImageState _t。請參閱 AWS IoTOver-the-air 更新版本 3.0.0。

返回 OTA 錯誤代碼枚舉已更改，並添加了新的錯誤代碼。請參閱 AWS IoTOver-the-air 更新版本 3.0.0/ OtaErr_。

範例用量-

```
OtaErr_t otaErr = OtaErrNone;
otaErr = OTA_SetImageState( OtaImageStateAccepted );
/* Handle error */
```

太田 _GetImageState

輸入參數相同，返回枚舉在 OTA 庫的版本 3 中重命名。

O塔圖書館第 1 版

```
OTA_ImageState_t OTA_GetImageState( void );
```

O塔圖書館第 3 版

```
OtaImageState_t OTA_GetImageState( void );
```

返回枚舉重命名為 OtaImageState _t。請參閱 AWS IoTOver-the-air 更新版本 3.0.0：OtaImageState_t。

範例用量-

```
OtaImageState_t imageState;
imageState = OTA_GetImageState();
```

宅暫停

輸入參數相同，返回 OTA 錯誤代碼被重命名，並在 OTA 庫的版本 3 中添加了新的錯誤代碼。

O塔圖書館第 1 版

```
OTA_Err_t OTA_Suspend( void );
```

O塔圖書館第 3 版

```
OtaErr_t OTA_Suspend( void );
```

返回 OTA 錯誤代碼枚舉已更改，並添加了新的錯誤代碼。請參閱 AWS IoTOver-the-air 更新版本 3.0.0： OtaErr_t。

範例用量-

```
OtaErr_t xOtaError = OtaErrUninitialized;
xOtaError = OTA_Suspend();
```

```
/* Handle error */
```

御宅履歷

在 OTA 演示/應用程序中處理連接時，用於連接的輸入參數被刪除，返回的 OTA 錯誤代碼被重命名，並在 OTA 庫的版本 3 中添加了新的錯誤代碼。

O塔圖書館第 1 版

```
OTA_Err_t OTA_Resume( void * pxConnection );
```

O塔圖書館第 3 版

```
OtaErr_t OTA_Resume( void );
```

返回 OTA 錯誤代碼枚舉已更改，並添加了新的錯誤代碼。請參閱 AWS IoTOver-the-air 更新版本 3.0.0： OtaErr_t。

範例用量-

```
OtaErr_t xOtaError = OtaErrUninitialized;
xOtaError = OTA_Resume();
/* Handle error */
```

太田 _CheckForUpdate

輸入參數相同，返回 OTA 錯誤代碼被重命名，並在 OTA 庫的版本 3 中添加了新的錯誤代碼。

O塔圖書館第 1 版

```
OTA_Err_t OTA_CheckForUpdate( void );
```

O塔圖書館第 3 版

```
OtaErr_t OTA_CheckForUpdate( void )
```

返回 OTA 錯誤代碼枚舉已更改，並添加了新的錯誤代碼。請參閱 AWS IoTOver-the-air 更新版本 3.0.0： OtaErr_t。

太田 _EventProcessingTask

這是一個新的 API，是處理 OTA 更新事件的主要事件循環。它必須由應用程序任務調用。此循環將繼續處理並執行 OTA Update 收到的事件，直到應用程序終止此任務。

O塔圖書館第 3 版

```
void OTA_EventProcessingTask( void * pUnused );
```

免費使用者的範例-

```
/* Create FreeRTOS task*/
xTaskCreate( prvOTAAgentTask,
             "OTA Agent Task",
             otaexampleAGENT_TASK_STACK_SIZE,
             NULL,
             otaexampleAGENT_TASK_PRIORITY,
             NULL );

/* Call OTA_EventProcessingTask from the task */
static void prvOTAAgentTask( void * pParam )
{
    /* Calling OTA agent task. */
    OTA_EventProcessingTask( pParam );
    LogInfo( ( "OTA Agent stopped." ) );

    /* Delete the task as it is no longer required. */
    vTaskDelete( NULL );
}
```

示例-

```
/* Create posix thread.*/
if( pthread_create( &threadHandle, NULL, otaThread, NULL ) != 0 )
{
    LogError( ( "Failed to create OTA thread: "
                ",errno=%s",
                strerror( errno ) ) );

   /* Handle error. */
}
```

```
/* Call OTA_EventProcessingTask from the thread.*/
static void * otaThread( void * pParam )
{
    /* Calling OTA agent task. */
    OTA_EventProcessingTask( pParam );
    LogInfo( ( "OTA Agent stopped." ) );

    return NULL;
}
```

太田 _SignalEvent

這是一個新的 API，它將事件添加到事件隊列的後面，並且內部 OTA 模塊也用於信號代理任務。

O塔圖書館第 3 版

```
bool OTA_SignalEvent( const OtaEventMsg_t * const pEventMsg );
```

範例用量-

```
OtaEventMsg_t xEventMsg = { 0 };
xEventMsg.eventId = OtaAgentEventStart;
( void ) OTA_SignalEvent( &xEventMsg );
```

將 OTA 庫集成為應用程序中的子模塊

如果你想在你自己的應用程序中集成 OTA 庫，你可以使用 git 子模塊命令。Git 子模塊允許你保留一個 Git 倉庫作為另一個 Git 倉庫的子目錄。OTA 庫版本 3 保存在 ota-for-aws-iot嵌入式 SDK 存儲庫中。

```
git submodule add https://github.com/aws/ota-for-aws-iot-embedded-
sdk.git destination_folder
```

```
git commit -m "Added the OTA Library as submodule to the project."
```

```
git push
```

如需詳細資訊，請參閱《FreeRTOS 使用者指南》中的將 OTA 代理程式整合到您的應用程式中。

參考

- 奧塔 1。
- 宅第 3 版。

從第 1 版遷移到第 3 版的 OTA PAL 連接埠

Over-the-air Updates 程式庫引入了資料夾結構中的一些變更，以及元件庫和示範應用程式所需組態的位置。對於旨在與 v1.2.0 配合使用以遷移到庫 v3.0.0 的 OTA 應用程序，您必須更新 PAL 端口功能簽名，並按照本遷移指南中所述包含其他配置文件。

OTA PAL 的變更

- OTA PAL 端口目錄名稱已從更新ota為ota_pal_for_aws。此資料夾必須包含 2 個檔案：ota_pal.c和ota_pal.h。PAL 頭文件libraries/freertos_plus/aws/ota/src/aws_iot_ota_pal.h已從 OTA 庫中刪除，必須在端口內部定義。
- 返回代碼（OTA_Err_t）被翻譯成一個枚舉OTAMainStatus_t。如需已翻譯的傳回代碼，請參閱網站表單介面。 還提供了輔助宏來組合OtaPalMainStatus和OtaPalSubStatus編碼，並OtaMainStatus從中提取OtaPalStatus和類似。
- 登錄 PAL
  - 移除了DEFINE_OTA_METHOD_NAME巨集。
  - 早些時候：OTA_LOG_L1("[%s] Receive file created.\r\n", OTA_METHOD_NAME);。
  - 更新：LogInfo(("Receive file created."));使用LogDebugLogWarn和LogError適當的日誌。
- 變數cOTA_JSON_FileSignatureKey變更為OTA_JsonFileSignatureKey。

函數

函數簽名是在中定義的，ota_pal.h並以前綴開頭，otaPal而不是prvPAL。

> **Note**
>
> PAL 的確切名稱在技術上是開放式的，但是為了與資格測試兼容，名稱應符合以下指定的名稱。

- 第一版：OTA_Err_t prvPAL_CreateFileForRx(OTA_FileContext_t * const *C*);

第三版：OtaPalStatus_t otaPal_CreateFileForRx(OtaFileContext_t * const *pFileContext*);

附註：當資料區塊進入時，請為它們建立新的接收檔案。

- 第一版：int16_t prvPAL_WriteBlock(OTA_FileContext_t * const C, uint32_t ulOffset, uint8_t * const pcData, uint32_t ulBlockSize);

  第三版：int16_t otaPal_WriteBlock(OtaFileContext_t * const pFileContext, uint32_t ulOffset, uint8_t * const pData, uint32_t ulBlockSize);

  筆記：在給定的偏移量寫入數據塊到指定的文件。

- 第一版：OTA_Err_t prvPAL_ActivateNewImage(void);

  第三版：OtaPalStatus_t otaPal_ActivateNewImage(OtaFileContext_t * const *pFileContext*);

  注意：激活通過 OTA 接收的最新 MCU 映像。

- 第一版：OTA_Err_t prvPAL_ResetDevice(void);

  第三版：OtaPalStatus_t otaPal_ResetDevice(OtaFileContext_t * const *pFileContext*);

  注意：重置設備。

- 第一版：OTA_Err_t prvPAL_CloseFile(OTA_FileContext_t * const *C*);

  第三版：OtaPalStatus_t otaPal_CloseFile(OtaFileContext_t * const *pFileContext*);

  注意：在指定的 OTA 上下文中對基礎接收文件進行身份驗證並關閉。

- 第一版：OTA_Err_t prvPAL_Abort(OTA_FileContext_t * const *C*);

  第三版：OtaPalStatus_t otaPal_Abort(OtaFileContext_t * const *pFileContext*);

  注意：停止 OTA 轉移。

- 第一版：OTA_Err_t prvPAL_SetPlatformImageState(OTA_ImageState_t *eState*);

第三版：OtaPalStatus_t otaPal_SetPlatformImageState(OtaFileContext_t * const pFileContext, OtaImageState_t eState);

注意：嘗試設置 OTA 更新映像的狀態。

- 第一版：OTA_PAL_ImageState_t prvPAL_GetPlatformImageState(void);

  第三版：OtaPalImageState_t otaPal_GetPlatformImageState(OtaFileContext_t * const *pFileContext*);

  注意：獲取 OTA 更新映像的狀態。

資料類型

- 第一版：OTA_PAL_ImageState_t

  檔案：aws_iot_ota_agent.h

  第三版：OtaPalImageState_t

  檔案：ota_private.h

  附註：平台實作所設定的映像狀態。
- 第一版：OTA_Err_t

  檔案：aws_iot_ota_agent.h

  第三版：OtaErr_t OtaPalStatus_t (combination of OtaPalMainStatus_t and OtaPalSubStatus_t)

  檔案:ota.h,ota_platform_interface.h

  筆記：v1：這些是定義 32 個無符號整數的宏。v3：代表錯誤類型的專門枚舉並與錯誤代碼相關聯。
- 第一版：OTA_FileContext_t

  檔案：aws_iot_ota_agent.h

  第三版：OtaFileContext_t

  檔案：ota_private.h

筆記：v1：包含數據的枚舉和緩衝區。v3：包含其他數據長度變量。

- 第一版：OTA_ImageState_t

  檔案：aws_iot_ota_agent.h

  第三版：OtaImageState_t

  檔案：ota_private.h

  筆記：OTA 圖像狀態

組態變更

該文件aws_ota_agent_config.h被重命名為 ota_config.h，將包含警衛從更改_AWS_OTA_AGENT_CONFIG_H_為OTA_CONFIG_H_。

- 檔案aws_ota_codesigner_certificate.h已遭刪除。
- 包括用於打印調試消息的新日誌堆棧：

```
/************************************************/
/******* DO NOT CHANGE the following order ********/
/************************************************/

/* Logging related header files are required to be included in the following order:
 * 1. Include the header file "logging_levels.h".
 * 2. Define LIBRARY_LOG_NAME and  LIBRARY_LOG_LEVEL.
 * 3. Include the header file "logging_stack.h".
 */

/* Include header that defines log levels. */
#include "logging_levels.h"

/* Configure name and log level for the OTA library. */
#ifndef LIBRARY_LOG_NAME
    #define LIBRARY_LOG_NAME     "OTA"
#endif
#ifndef LIBRARY_LOG_LEVEL
    #define LIBRARY_LOG_LEVEL    LOG_INFO
#endif
```

```
#include "logging_stack.h"

/************ End of logging configuration ****************/
```

- 添加了常量配置：

```
/** * @brief Size of the file data block message (excluding the header). */
#define otaconfigFILE_BLOCK_SIZE ( 1UL << otaconfigLOG2_FILE_BLOCK_SIZE )
```

新文件：**ota_demo_config.h**包含 OTA 演示所需的配置，例如代碼簽名證書和應用程序版本。

- signingcredentialSIGNING_CERTIFICATE_PEM在中定義的內容demos/include/aws_ota_codesigner_certificate.h已被移動到ota_demo_config.h，otapalconfigCODE_SIGNING_CERTIFICATE並且可以從 PAL 文件中訪問，如下所示：

```
static const char codeSigningCertificatePEM[] = otapalconfigCODE_SIGNING_CERTIFICATE;
```

  檔案aws_ota_codesigner_certificate.h已遭刪除。
- 巨集APP_VERSION_BUILDAPP_VERSION_MAJOR已加入至ota_demo_config.h。APP_VERSION_MINOR已移除包含版本資訊的舊檔案，例如tests/include/aws_application_version.hlibraries/c_sdk/standard/common/include/iot_appversion32.h，demos/demo_runner/aws_demo_version.c。

OTA PAL 測試的更改

- 移除了「Full_Ota_Agent」測試群組以及所有相關檔案。此測試組以前是要求進行資格認證的。這些測試是針對 OTA 圖書館的，而不是針對 OTA PAL 端口的。OTA 庫現在具有在 OTA 存儲庫中託管的完整測試覆蓋範圍，因此不再需要此測試組。
- 移除了「全部」和「隔離區」測試群組以及所有相關檔案。這些測試不是資格測試的一部分。這些測試涵蓋的功能現在正在 OTA 存儲庫中進行測試。
- 將測試文件從庫目錄移動到目tests/integration_tests/ota_pal錄中。
- 更新了 OTA PAL 資格測試，以使用 OTA 圖書館 API 的 3.0.0 版。
- 更新了 OTA PAL 測試如何訪問代碼簽名證書進行測試。以前有一個專用的標頭文件用於代碼簽名憑證。對於新版本的庫而言，情況已不再如此。測試程式碼需要在中定義此變數ota_pal.c。該值被分配給在平台特定的 OTA 配置文件中定義的宏。

清單

使用此檢查清單確定您遵循移轉所需的步驟：

- 將 ota pal 端口文件夾的名稱從更新ota為ota_pal_for_aws。
- 添加ota_pal.h與上面提到的功能的文件。如需範例ota_pal.h檔案，請參閱 GitHub。
- 添加配置文件：
  - 將檔案名稱從變更aws_ota_agent_config.h為 (或建立)ota_config.h。
    - 新增：

      ```
      otaconfigFILE_BLOCK_SIZE ( 1UL << otaconfigLOG2_FILE_BLOCK_SIZE )
      ```

    - 包括：

      ```
      #include "ota_demo_config.h"
      ```

  - 將上述檔案複製到aws_test config資料夾中，並ota_demo_config.h使用取代的任何包含項目aws_test_ota_config.h。
  - 新增ota_demo_config.h檔案。
  - 新增aws_test_ota_config.h檔案。
- 對 ota_pal.c 進行下列變更：
  - 使用最新的 OTA 庫文件名更新包含。
  - 移除 DEFINE_OTA_METHOD_NAME 巨集。
  - 更新 OTA PAL 功能的簽名。
  - 將檔案上下文變數的名稱從更新C為pFileContext。
  - 更新結OTA_FileContext_t構和所有相關變量。
  - 更新cOTA_JSON_FileSignatureKey至OTA_JsonFileSignatureKey。
  - 更新OTA_PAL_ImageState_t和Ota_ImageState_t類型。
  - 更新錯誤類型和值。
  - 更新列印巨集以使用記錄堆疊。
  - 更新signingcredentialSIGNING_CERTIFICATE_PEM為otapalconfigCODE_SIGNING_CERTIFICA
  - 更新otaPal_CheckFileSignature和otaPal_ReadAndAssumeCertificate功能評論。
- 更新CMakeLists.txt檔案。

- 更新 IDE 專案。

文件歷史紀錄

下表說明 FreeRTOS Guide 的文件歷史。

日期	文件版本	變更歷史	FreeRTOS 版
2022 年 5 月	自由移植指南 免費駕駛資格指南	• 更新了現有測試，添加了新的測試，並刪除了基於 FreeRTOS 長期 Support（LTS）庫的冗餘測試。如需詳細資訊，請參閱上的 FreeRTOS 程式庫整合測試 202205.00 GitHub。 • 已更新FreeRTOS 移植流程圖。 • 添加了一個新的移植網路傳輸介面. • 移植AWS IoT over-the-air (OTA) 更新程式庫現在需要資格。 • 刪除了 Wi-Fi 和 TLS 抽象移植指南，因為它不再需要了。 • 如需 FreeRTOS 資格的進一步更新，請參閱最新變更。	四小型 202112.00
2021 年 7 月	移植指南 (英文版) 資格指南	• 發行版本	202107.00

日期	文件版本	變更歷史	FreeRTOS 版
		• 已變更移植AWS IoT over-the-air (OTA) 更新程式庫 • 已新增 從 OTA 應用程式遷移至第 3 版 • 已新增 從第 1 版遷移到第 3 版的 OTA PAL 連接埠	
2020 年 12 月	移植指南 資格指南	• 版本 • 已新增 設定核心 HTTP 程式庫 • 已新增 移植蜂窩接口庫	202012.00
2020 年 11 月	移植指南 資格指南	• 版本 • 已新增 設定核心 QTT 程式庫	202011.00
2020 年 7 月	移植指南 (資格指南)	• 二零零七	202007.00
2020 年 2 月 18 日	202002.00 (移植指南) 202002.00 (資格指南)	• 發行版本 202002.00 • Amazon FreeRTOS 現已成為 FreeRTOS	202002.00
2019 年 12 月 17 日	201912.00 (移植指南) 201912.00 (資格指南)	• 發行版本 201912.00 • 新增移植通用 I/O 程式庫。	201912.00

日期	文件版本	變更歷史	FreeRTOS 版
2019 年 10 月 29 日	201910.00 (移植指南) 201910.00 (資格指南)	• 發行版本 201910.00 • 已更新隨機數字產生器移轉信息。	201910.00
2019 年 8 月 26 日	201908.00 (移植指南) 201908.00 (資格指南)	• 201908.00 版 • 新增設定 HTTPS 用戶端程式庫以進行測試 已更新 移植圖書館	201908.00
2019 年 6 月 17 日	移植指南 資格指南	• 201906.00 版 • 更新目錄結構	201906.00 主要版本
2019 年 5 月 21 日	1.4.8 (移植指南) 1.4.8 (資格指南)	• 移植文件已移至 FreeRTOS 移植指南 • 資格文件已移至 FreeRTOS 資格指南	1.4.8
2019 年 2 月 25 日	1.1.6	• 從《入門指南範本附錄》移除下載和組態說明 (第 84 頁)	1.4.5 1.4.6 1.4.7
2018 年 12 月 27 日	1.1.5	• 在〈資格的檢查清單〉附錄中補充 CMake 要求 (第 70 頁)	1.4.5 1.4.6
2018 年 12 月 12 日	1.1.4	• 在 TCP/IP 移植附錄中增加 lwIP 移植指示 (第 31 頁)	1.4.5

日期	文件版本	變更歷史	FreeRTOS 版
2018 年 11 月 26 日	1.1.3	• 新增低功耗藍牙移植附錄 (第 52 頁) • 在整個文件中新增了 FreeRTOS 測試資訊的AWS IoT裝置測試儀 • 在 FreeRTOS 控制台附錄中添加了 CMake 鏈接到列出的信息 (第 85 頁)	1.4.4
2018 年 11 月 7 日	1.1.2	• 在 PKCS #11 移植附錄中更新 PKCS #11 PAL 界面移植指示 (第 38 頁) • 更新 `CertificateConfigurator.html` 的路徑 (第 76 頁) • 更新〈入門指南範本〉附錄 (第 80 頁)	1.4.3

日期	文件版本	變更歷史	FreeRTOS 版
2018 年 10 月 8 日	1.1.1	• 在 `aws_test_runner_config.h` 測試組態表格中新增新的「AFQP 需要」欄 (第 16 頁) • 在〈建立測試專案〉小節中更新 Unity 模組目錄路徑 (第 14 頁) • 更新「建議的移植順序」圖表 (第 22 頁) • 在 TLS 附錄〈測試設定〉中更新用戶端憑證和金鑰變數名稱 (第 40 頁) • 在 Secure Sockets 移植附錄〈測試設定〉(第 34 頁)；TLS 移植附錄〈測試設定〉(第 40 頁)；以及〈TLS 伺服器設定〉附錄 (第 57 頁) 中，變更檔案路徑	1.4.2
2018 年 8 月 27 日	1.1.0	• 新增〈OTA 更新〉移植附錄 (第 47 頁) • 新增〈開機載入器〉移植附錄 (第 51 頁)	1.4.0 1.4.1

日期	文件版本	變更歷史	FreeRTOS 版
2018 年 8 月 9 日	1.0.1	• 更新「建議的移植順序」圖表 (第 22 頁) • 更新 PKCS #11 移植附錄 (第 36 頁) • 在 TLS 移植附錄〈測試設定〉(第 40 頁) 和〈TLS 伺服器設定〉附錄步驟 9 (第 51 頁) 中，變更檔案路徑 • 在 MQTT 移植附錄〈先決條件〉中修正超連結 (第 45 頁) • 在〈建立 BYOC 的指示〉附錄的範例中新增 AWS CLI config 指示 (第 57 頁)	1.3.1 1.3.2
2018 年 7 月 31 日	1.0.0	FreeRTOS 資格計劃指南的初始版本	1.3.0

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。