aws

移植指南

FreeRTOS

FreeRTOS: 移植指南

Table of Contents

FreeRTOS 移植

什麼是 FreeRTOS

FreeRTOS 與世界領先的晶片公司合作開發了 20 年，現在每 170 秒下載一次，是微型控制器和小型微處理器的市場領導即時作業系統 (RTOS)。在 MIT 開放原始碼授權下自由分佈，FreeRTOS 包含核心和一組不斷增長的程式庫，適用於所有產業領域。FreeRTOS 的設計最強調的是可靠性和容易使用。FreeRTOS 包含用於連線、安全性和over-the-air(OTA) 更新的程式庫，以及可在合格主機板上示範 FreeRTOS 功能的示範應用程式。

如需詳細資訊，請造訪FreeRTOS.org。

將 FreeRTOS 移植到您的 IoT 主機板

您需要根據其功能和應用程式，將 FreeRTOS 軟體程式庫移植到您的微型控制器型主機板。

將 FreeRTOS 移植到您的裝置

1. 遵循 中的指示下載 FreeRTOS 以進行移植，下載最新版本的 FreeRTOS 以進行移植。
2. 請遵循 中的指示設定您的工作區和專案以進行移植，在 FreeRTOS 下載中設定檔案和資料夾以進行移植和測試。
3. 遵循 中的指示，將 FreeRTOS 程式庫移植 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 連接埠？

> FreeRTOS 連接埠是針對平台支援之必要 FreeRTOS 程式庫和 FreeRTOS 核心的主機板特定 APIs 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)。

如果您對此頁面或 FreeRTOS 移植指南的其餘部分中未回答的移植有任何疑問，請參閱可用的 FreeRTOS 支援選項。

下載 FreeRTOS 以進行移植

從 freertos.org：// 下載最新的 FreeRTOS 或長期支援 (LTS) 版本，或從 GitHub (FreeRTOS-LTS) 或 (FreeRTOS) 複製。

> **Note**
>
> 我們建議您複製儲存庫。複製可讓您在主分支推送到儲存庫時更輕鬆地取得更新。

或者，從 FreeRTOS 或 FreeRTOS-LTS 儲存庫子模組個別程式庫。不過，請確定程式庫版本符合 FreeRTOS 或 FreeRTOS-LTS 儲存庫中`manifest.yml`檔案中列出的組合。

下載或複製 FreeRTOS 之後，您可以開始將 FreeRTOS 程式庫移植到您的主機板。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 程式庫。

設定您的工作區和專案以進行移植

請依照下列步驟來設定您的工作區和專案：

- 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。
- 在您的專案中包含主機板支援套件 (BSP) 和主機板特定的驅動程式。

設定工作區後，您就可以開始移植個別 FreeRTOS 程式庫。

移植 FreeRTOS 程式庫

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。

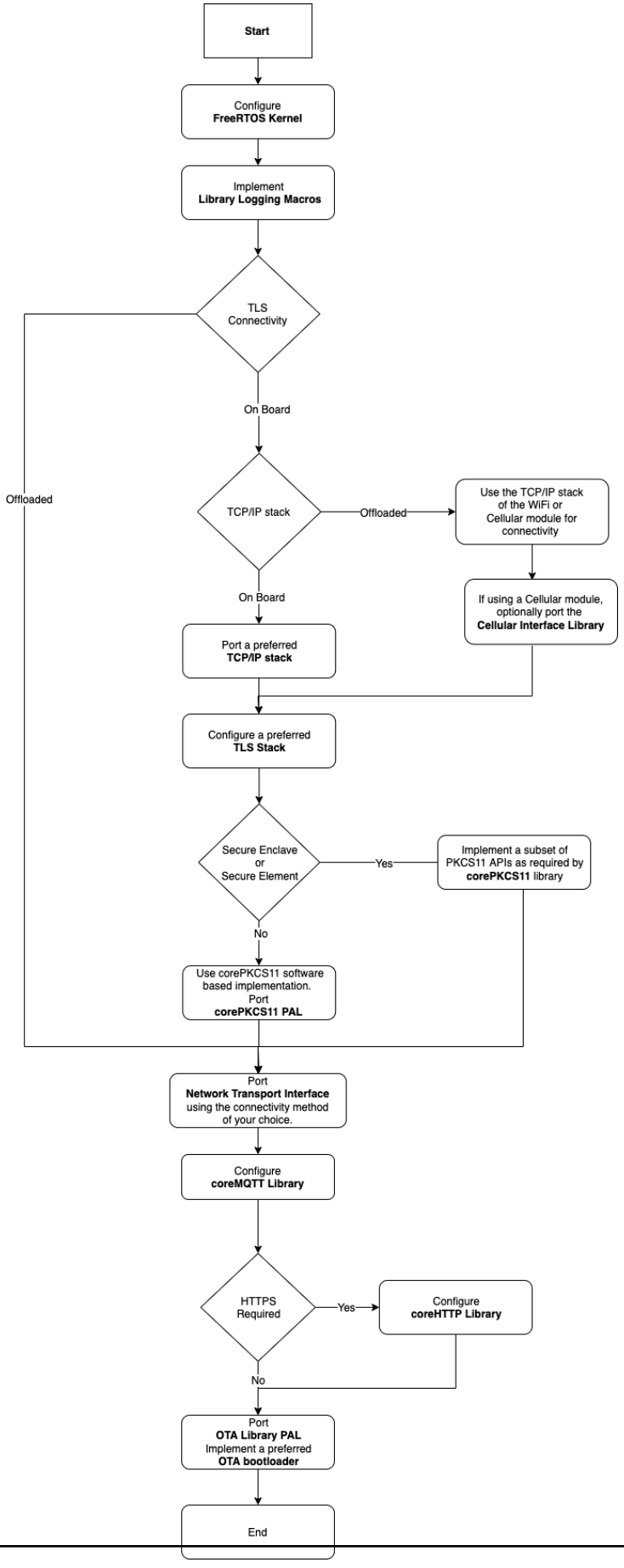
FreeRTOS 移植流程圖 說明移植所需的程式庫。

若要將 FreeRTOS 移植到您的裝置，請遵循下列主題中的指示。

1. 設定 FreeRTOS 核心連接埠
2. 實作程式庫日誌巨集
3. 移植 TCP/IP 堆疊
4. 移植網路傳輸界面
5. 移植 corePKCS11 程式庫
6. 設定 coreMQTT 程式庫
7. 設定 coreHTTP 程式庫
8. 移植 AWS IoT over-the-air(OTA) 更新程式庫
9. 移植行動介面程式庫

FreeRTOS 移植流程圖

當您將 FreeRTOS 移植到電路板時，請使用下面的移植流程圖做為視覺化輔助。


Start
Configure
FreeRTOS Kernel
Implement
Library Logging Macros
TLS
Connectivity
On Board
Offloaded
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
No
Use corePKCS11 software
based implementation.
Port
corePKCS11 PAL
Port
Network Transport Interface
using the connectivity method
of your choice.
Configure
coreMQTT Library
HTTPS
Required
Yes
Configure
coreHTTP Library
No
Port
OTA Library PAL
Implement a preferred
OTA bootloader
End

移植流程圖

設定 FreeRTOS 核心連接埠

本節提供將 FreeRTOS 核心的連接埠整合到 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 核心進行多工和任務間通訊。如需詳細資訊，請參閱 FreeRTOS 使用者指南中的 FreeRTOS 核心基本概念和 FreeRTOS.org FreeRTOS

> **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 資格計劃，僅支援現有的 FreeRTOS 核心連接埠。程式中不接受修改這些連接埠。如需詳細資訊，請參閱 FreeRTOS 核心連接埠政策。

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 的組態檔案進行自訂`FreeRTOSConfig.h`。此檔案會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。 FreeRTOS.org.

若要設定 FreeRTOS 核心以搭配您的裝置使用，請包含 `FreeRTOSConfig.h`，並修改任何其他 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。 FreeRTOS.org.

測試

- 執行簡單的 FreeRTOS 任務，將訊息記錄到序列輸出主控台。

- 確認訊息如預期輸出至主控台。

實作程式庫日誌巨集

FreeRTOS 程式庫使用下列日誌巨集，以增加的詳細程度順序列出。

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

必須提供所有巨集的定義。建議包括：

- Macros 應支援C89樣式記錄。
- 記錄應該是執行緒安全。來自多個任務的日誌行不得互相交錯。
- 記錄 APIs不得封鎖，而且必須釋放應用程式任務在 I/O 上封鎖。

如需實作詳細資訊，請參閱 FreeRTOS.org 上的記錄功能。您可以在此範例中看到 實作。

測試

- 執行具有多個任務的測試，以驗證日誌不會交錯。
- 執行測試以確認記錄 APIs I/O 上未封鎖。
- 使用各種標準測試記錄巨集，例如`C89`,`C99`樣式記錄。
- 透過設定不同的日誌層級來測試日誌巨集，例如 `Debug`、`Error`、 `Info`和 `Warning`。

移植 TCP/IP 堆疊

本節提供移植和測試內建 TCP/IP 堆疊的指示。如果您的平台將 TCP/IP 和 TLS 功能卸載至單獨的網路處理器或模組，您可以略過此移植區段並造訪 移植網路傳輸界面。

FreeRTOS+TCP 是 FreeRTOS 核心的原生 TCP/IP 堆疊。FreeRTOS+TCP 由 FreeRTOS 工程團隊開發和維護，是建議與 FreeRTOS 搭配使用的 TCP/IP 堆疊。如需詳細資訊，請參閱移植 FreeRTOS +TCP。或者，您可以使用第三方 TCP/IP 堆疊 lwIP 。本節中提供的測試指示使用 TCP 純文字的傳輸界面測試，並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：

- IDE 專案，其中包含廠商提供的乙太網路或 Wi-Fi 驅動程式。

  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

開始移植 FreeRTOS+TCP 程式庫之前，請檢查 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 的檔案中實作使用廠商提供乙太網路或 Wi-Fi 驅動程式的新連接埠`NetworkInterface.c`。請造訪 GitHub 儲存庫以取得範本。

建立連接埠後，或者如果連接埠已存在，請建立 `FreeRTOSIPConfig.h`，並編輯組態選項，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

無論您是使用 FreeRTOS+TCP 程式庫或第三方程式庫，請依照下列步驟進行測試：

- 在傳輸介面測試中提供 `connect/disconnect/send/receive` APIs的實作。
- 在純文字 TCP 連線模式中設定 echo 伺服器，並執行傳輸介面測試。

> **Note**
>
> 若要正式讓裝置符合 FreeRTOS 的資格，如果您的架構需要移植 TCP/IP 軟體堆疊，您需要使用純文字 TCP 連線模式，根據傳輸介面測試來驗證裝置的移植原始碼 AWS IoT Device Tester。請遵循 FreeRTOS 使用者指南中的使用 AWS IoT Device Tester for FreeRTOS 中的指示來設定 AWS IoT Device Tester 連接埠驗證。 FreeRTOS 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 corePKCS11 程式庫

公有金鑰密碼編譯標準 #11 定義了與平台無關的 API，以管理和使用密碼編譯字符。PKCS 11 是指標準及其定義的 APIs。PKCS #11 密碼編譯 API 會抽象金鑰儲存、密碼編譯物件的 get/set 屬性，以及工作階段語意。它廣泛用於操縱常見的密碼編譯物件。其 函數允許應用程式軟體使用、建立、修改和刪除密碼編譯物件，而不會將這些物件暴露到應用程式的記憶體。

FreeRTOS 程式庫和參考整合使用 PCKS #11 介面標準的子集，著重於涉及非對稱金鑰、隨機數量產生和雜湊的操作。下表列出要支援的使用案例和所需的 PKCS #11 APIs。

使用案例

使用案例	必要的 PKCS #11 API 系列
全部	初始化、完成、開啟/關閉工作階段、GetSlotList、登入
佈建中	GenerateKeyPair、CreateObject、DestroyObject、InitToken、GetTokenInfo
TLS	Random、Sign、FindObject、GetAttributeValue
FreeRTOS+TCP	隨機、摘要
OTA	Verify、Digest、FindObject、GetAttributeValue

何時實作完整的 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密碼編譯硬體，以減少生產案例中資料遭竊和裝置重複的威脅。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，您必須實作使用 FreeRTOS 程式庫所需的 PKCS #11 子集，如上表所定義。

何時使用 FreeRTOS corePKCS11

corePKCS11 程式庫包含以軟體為基礎的 PKCS #11 介面 (API) 實作，其使用 Mbed TLS 提供的密碼編譯功能。這適用於硬體沒有專用密碼編譯硬體的快速原型設計和評估案例。在此情況下，您只需要實作 corePKCS11 PAL 即可讓 corePKCS11 軟體型實作與硬體平台搭配使用。

移植 corePKCS11

您必須擁有將密碼編譯物件讀取和寫入非揮發性記憶體 (NVM) 的實作，例如內建快閃記憶體。密碼編譯物件必須存放在未初始化的 NVM 區段中，且不會在裝置重新程式設計時清除。corePKCS11 程式庫的使用者將使用登入資料佈建裝置，然後使用透過 corePKCS11 界面存取這些登入資料的新應用程式來重新程式設計裝置。corePKCS11 PAL 連接埠必須提供要存放的位置：

- 裝置用戶端憑證
- 裝置用戶端私有金鑰
- 裝置用戶端公有金鑰
- 信任的根 CA
- 用於安全開機載入器和over-the-air(OTA) 更新的程式碼驗證公有金鑰 （或包含程式碼驗證公有金鑰的憑證）
- Just-In-Time佈建憑證

包含 標頭檔案並實作定義的 PAL APIs。

PAL APIs

函式	說明
PKCS11_PAL_Initialize	初始化 PAL 層。在初始化序列開始時，由 corePKCS11 程式庫呼叫。
PKCS11_PAL_SaveObject	將資料寫入非揮發性儲存。

函式	說明
PKCS11_PAL_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)。
PKCS11_PAL_GetObjectValue	根據控制代碼擷取物件的值。
PKCS11_PAL_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 corePKCS11 程式庫或實作所需的 PKCS11 APIs 子集，您必須通過 FreeRTOS PKCS11 測試。這些測試 FreeRTOS 程式庫所需的函數是否如預期般執行。

本節也說明如何使用資格測試在本機執行 FreeRTOS PKCS11 測試。

先決條件

若要設定 FreeRTOS PKCS11 測試，必須實作下列項目。

- PKCS11 APIs支援的連接埠。
- FreeRTOS 資格測試平台函數的實作，包括下列項目：
  - FRTest_ThreadCreate
  - FRTest_ThreadTimedJoin
  - FRTest_MemoryAlloc
  - FRTest_MemoryFree

（請參閱 GitHub 上 PKCS #11 的 FreeRTOS 程式庫整合測試的 README.md 檔案。) FreeRTOS GitHub.)

移植測試

- 將 FreeRTOS-Libraries-Integration-Tests 作為子模組新增至您的專案。子模組可以放置在專案的任何目錄中，只要可以建置即可。

- 將 config_template/test_execution_config_template.h和 config_template/test_param_config_template.h 複製到建置路徑中的專案位置，並將其重新命名為 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可用來包含相關檔案。
- 實作 ，UNITY_OUTPUT_CHAR讓測試輸出日誌和裝置日誌不會交錯。
- 整合 MbedTLS，以驗證加密操作結果。
- RunQualificationTest() 從應用程式呼叫 。

設定測試

PKCS11 測試套件必須根據 PKCS11 實作進行設定。下表列出 test_param_config.h標頭檔案中 PKCS11 測試所需的組態。

PKSC11 測試組態

Configuration	說明
PKCS11_TEST_RSA_KEY_SUPPORT	移植支援 RSA 金鑰函數。
PKCS11_TEST_EC_KEY_SUPPORT	移植支援 EC 金鑰函數。
PKCS11_TEST_IMPORT_PRIVATE_KEY_SUPPORT	移植支援匯入私有金鑰。如果啟用支援金鑰函數，則會在測試中驗證 RSA 和 EC 金鑰匯入。
PKCS11_TEST_GENERATE_KEYPAIR_SUPPORT	移植支援金鑰對產生。如果啟用支援金鑰函數，則會在測試中驗證 EC 金鑰對產生。
PKCS11_TEST_PREPROVISIONED_SUPPORT	移植有預先佈建的登入資料。 PKCS11_TEST_LABEL_DEVICE_PRIVATE_KEY_FOR_TLS PKCS11_TEST_LABEL_DEVICE_PUBLIC_KEY_FOR_TLS 和 PKCS11_TEST_LABEL_DEVICE_CERTIFICATE_FOR_TLS 是登入資料的範例。
PKCS11_TEST_LABEL_DEVICE_PRIVATE_KEY_FOR_TLS	測試中使用的私有金鑰標籤。

Configuration	說明
PKCS11_TEST_LABEL_DEVICE_PUBLIC_KEY_FOR_TLS	測試中使用的公有金鑰標籤。
PKCS11_TEST_LABEL_DEVICE_CERTIFICATE_FOR_TLS	測試中使用的憑證標籤。
PKCS11_TEST_JITP_CODEVERIFY_ROOT_CERT_SUPPORTED	移植支援 JITP 儲存。將此設定為 1 以啟用 JITP `codeverify` 測試。
PKCS11_TEST_LABEL_CODE_VERIFICATION_KEY	JITP `codeverify` 測試中使用的程式碼驗證金鑰標籤。
PKCS11_TEST_LABEL_JITP_CERTIFICATE	JITP `codeverify` 測試中使用的 JITP 憑證標籤。
PKCS11_TEST_LABEL_ROOT_CERTIFICATE	JITP `codeverify` 測試中使用的根憑證標籤。

FreeRTOS 程式庫和參考整合必須至少支援一個金鑰函數組態，例如 RSA 或橢圓曲線金鑰，以及 PKCS11 APIs 支援的一個金鑰佈建機制。測試必須啟用下列組態：

- 至少下列其中一個金鑰函數組態：
  - PKCS11_TEST_RSA_KEY_SUPPORT
  - PKCS11_TEST_EC_KEY_SUPPORT
- 至少下列其中一個金鑰佈建組態：
  - PKCS11_TEST_IMPORT_PRIVATE_KEY_SUPPORT
  - PKCS11_TEST_GENERATE_KEYPAIR_SUPPORT
  - PKCS11_TEST_PREPROVISIONED_SUPPORT

預先佈建的裝置登入資料測試必須在下列條件下執行：

- `PKCS11_TEST_PREPROVISIONED_SUPPORT` 必須啟用 ，並停用其他佈建機制。
- 僅`PKCS11_TEST_EC_KEY_SUPPORT`啟用一個金鑰函數，`PKCS11_TEST_RSA_KEY_SUPPORT`或 。

- 根據您的金鑰函數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_KEY_FOR_TLS和 PKCS11_TEST_LABEL_DEVICE_CERTIFICATE_FOR_TLS。在執行測試之前，這些登入資料必須存在。

如果實作支援預先佈建的登入資料和其他佈建機制，則測試可能需要使用不同的組態執行數次。

> **Note**
>
> 如果 PKCS11_TEST_GENERATE_KEYPAIR_SUPPORT或 已啟用PKCS11_TEST_LABEL_DEVICE_PRIVATE_KEY_FOR_TLS，則在測試期間PKCS11_TEST_LABEL_DEVICE_CERTIFICATE_FOR_TLS會銷毀具有標籤 、 PKCS11_TEST_LABEL_DEVICE_PUBLIC_KEY_FOR_TLS和 PKCS11_TEST_GENERATE_KEYPAIR_SUPPORT的物件。

執行測試

本節說明如何使用資格測試在本機測試 PKCS11 介面。或者，您也可以使用 IDT 自動化執行。如需詳細資訊AWS IoT Device Tester ，請參閱FreeRTOS 使用者指南》中的 FreeRTOS。

下列指示說明如何執行測試：

- 開啟 CORE_PKCS11_TEST_ENABLED test_execution_config.h並定義為 1。
- 建置應用程式並刷新至您的裝置以執行。測試結果會輸出到序列連接埠。

以下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**
>
> 若要正式讓裝置符合 FreeRTOS 的資格，您必須使用 驗證裝置的移植原始碼 AWS IoT Device Tester。請遵循 FreeRTOS 使用者指南中的使用 AWS IoT Device Tester for FreeRTOS 中的指示來設定 AWS IoT Device Tester 連接埠驗證。若要測試特定程式庫的連接埠，必須在 `configs` 資料夾中的 `device.json` 檔案中 AWS IoT Device Tester 啟用正確的測試群組。

移植網路傳輸界面

整合 TLS 程式庫

針對 Transport Layer Security (TLS) 身分驗證，請使用您偏好的 TLS 堆疊。建議使用 Mbed TLS，因為它已使用 FreeRTOS 程式庫進行測試。您可以在此 GitHub 儲存庫中找到此範例。

無論您的裝置使用何種 TLS 實作，您都必須使用 TCP/IP 堆疊實作 TLS 堆疊的基礎傳輸掛鉤。它們必須支援 支援的 TLS 密碼套件 AWS IoT。

移植網路傳輸介面程式庫

您必須實作網路傳輸界面，才能使用 coreMQTT 和 coreHTTP。網路傳輸界面包含在單一網路連線上傳送和接收資料所需的函數指標和內容資料。如需詳細資訊，請參閱傳輸界面。FreeRTOS 提供一組內建的網路傳輸介面測試，以驗證這些實作。下一節會引導您設定專案以執行這些測試。

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，可使用經過驗證的 FreeRTOS 核心連接埠建置 FreeRTOS。
- 網路驅動程式的工作實作。

移植

- 將 FreeRTOS-Libraries-Integration-Tests 作為子模組新增至您的專案。子模組放置在專案中的位置並不重要，只要可以建置即可。
- 將 `config_template/test_execution_config_template.h`和 `config_template/test_param_config_template.h` 複製到建置路徑中的專案位置，並將其重新命名為 `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，`src/transport_interface_tests.cmake`則使用 `qualification_test.cmake`和 來包含相關檔案。
- 在適當的專案位置實作下列函數：
- • `network connect function`：簽章由 在 `NetworkConnectFunc`中定義`src/common/network_connection.h`。此函數接受指向網路內容的指標、託管資訊的指標，以及指向網路憑證的指標。它會使用提供的網路登入資料，與主機資訊中指定的伺服器建立連線。
  - `network disconnect function`：簽章由 在 `NetworkDisconnectFunc`中定義`src/common/network_connection.h`。此函數會將指標帶入網路內容。它會中斷之前在網路內容中存放的連線。
  - `setupTransportInterfaceTestParam()`：這是在 中定義的`src/transport_interface/transport_interface_tests.h`。實作必須具有與 中定義的完全相同的名稱和簽章`transport_interface_tests.h`。此函數會

將指標帶入 TransportInterfaceTestParam 結構。它會填入傳輸介面測試所使用的 TransportInterfaceTestParam 結構中的欄位。

- 實作 UNITY_OUTPUT_CHAR，讓測試輸出日誌不會與裝置日誌互斥。
- `runQualificationTest()`從應用程式呼叫 。裝置硬體必須正確初始化，且必須在呼叫之前連接網路。

登入資料管理 （裝置內產生的金鑰）

當 中的 FORCE_GENERATE_NEW_KEY_PAIR `test_param_config.h` 設定為 1 時，裝置應用程式會產生新的裝置內金鑰對，並輸出公有金鑰。裝置應用程式在建立與回應伺服器的 TLS 連線時，使用 echo_SERVER_ROOT_CA 和 TRANSPORT_CLIENT_CERTIFICATE 做為回應伺服器根 CA 和用戶端憑證。IDT 會在資格執行期間設定這些參數。

登入資料管理 （匯入金鑰）

裝置應用程式在與回應伺服器建立 TLS 連線時，使用 中的 echo_SERVER_ROOT_CA、TRANSPORT_CLIENT_CERTIFICATE 和 TRANSPORT_CLIENT_PRIVATE_KEY `test_param_config.h`做為回應伺服器根 CA、用戶端憑證和用戶端私有金鑰。IDT 會在資格執行期間設定這些參數。

測試

本節說明如何使用資格測試在本機測試傳輸界面。如需其他詳細資訊，請參閱 GitHub 上 FreeRTOS-Libraries-Integration-Tests 的 transport_interface 一節所提供的 README.md 檔案。

或者，您也可以使用 IDT 自動化執行。如需詳細資訊AWS IoT Device Tester ，請參閱FreeRTOS 使用者指南》中的 FreeRTOS。

啟用測試

開啟 TRANSPORT_INTERFACE_TEST_ENABLED `test_execution_config.h`並將其定義為 1。

設定用於測試的 echo 伺服器

本機測試需要從執行測試的裝置存取的 echo 伺服器。如果傳輸介面實作支援 TLS，回應伺服器必須支援 TLS。如果您還沒有，FreeRTOS-Libraries-Integration-Tests GitHub 儲存庫具有回應伺服器實作。

設定專案進行測試

在 中`test_param_config.h`，將 echo_SERVER_ENDPOINT 和 echo_SERVER_PORT 更新為上一個步驟中的端點和伺服器設定。

設定登入資料（裝置上產生的金鑰）

- 將 echo_SERVER_ROOT_CA 設定為 echo 伺服器的伺服器憑證。
- 將 FORCE_GENERATE_NEW_KEY_PAIR 設定為 1，以產生金鑰對並取得公有金鑰。
- 產生金鑰後，將 FORCE_GENERATE_NEW_KEY_PAIR 設回 0。
- 使用公有金鑰和伺服器金鑰和憑證來產生用戶端憑證。
- 將 TRANSPORT_CLIENT_CERTIFICATE 設定為產生的用戶端憑證。

設定登入資料（匯入金鑰）

- 將 echo_SERVER_ROOT_CA 設定為 echo 伺服器的伺服器憑證。
- 將 TRANSPORT_CLIENT_CERTIFICATE 設定為預先產生的用戶端憑證。
- 將 TRANSPORT_CLIENT_PRIVATE_KEY 設定為預先產生的用戶端私有金鑰。

建置和刷新應用程式

使用您選擇的工具鏈建置和刷新應用程式。調用 `runQualificationTest()` 時，將執行傳輸界面測試。測試結果會輸出到序列連接埠。

> **Note**
>
> 若要正式讓裝置符合 FreeRTOS 的資格，您必須使用 OTA PAL 和 OTA E2E 測試群組來驗證裝置的移植原始碼 AWS IoT Device Tester。請遵循 FreeRTOS 使用者指南中的使用 AWS IoT Device Tester for FreeRTOS 中的指示來設定 AWS IoT Device Tester 連接埠驗證。FreeRTOS 若要測試特定程式庫的連接埠，必須在 資料夾中的 `device.json` 檔案中 AWS IoT Device Tester `configs`啟用正確的測試群組。

設定 coreMQTT 程式庫

邊緣上的裝置可以使用 MQTT 通訊協定與 AWS Cloud 通訊。 AWS IoT 託管 MQTT 代理程式，在邊緣傳送和接收連線裝置的訊息。

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，但裝置的測試專案必須通過所有 MQTT 測試才能符合資格。如需詳細資訊，請參閱《FreeRTOS 使用者指南》中的 coreMQTT Library。 FreeRTOS

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。如需進一步了解，請參閱移植網路傳輸界面。

測試

執行 coreMQTT 整合測試：

- 向 MQTT 代理程式註冊您的用戶端憑證。
- 在 中設定中介裝置端點，`config`並執行整合測試。

建立參考 MQTT 示範

我們建議使用 coreMQTT 代理程式來處理所有 MQTT 操作的執行緒安全性。使用者還需要發佈和訂閱任務，而 Device Advisor 會測試 來驗證應用程式是否有效整合 TLS、MQTT 和其他 FreeRTOS 程式庫。

若要正式讓裝置符合 FreeRTOS 的資格，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 Device Advisor 工作流程。TLS 和 MQTT 的強制性測試案例如下所示：

TLS 測試案例

測試案例	測試案例	必要的測試
TLS	TLS Connect	是
TLS	TLS 支援 AWS IoT 密碼套件	建議的加密套件
TLS	TLS 不安全伺服器憑證	是
TLS	TLS 不正確的主體名稱伺服器憑證	是

MQTT 測試案例

測試案例	測試案例	必要的測試
MQTT	MQTT Connect	是
MQTT	MQTT Connect 抖動重試	是，沒有警告
MQTT	MQTT 訂閱	是
MQTT	MQTT 發佈	是
MQTT	MQTT ClientPuback QoS1	是
MQTT	MQTT No Ack PingResp	是

設定 coreHTTP 程式庫

邊緣上的裝置可以使用 HTTP 通訊協定與 AWS Cloud. AWS IoT services 託管 HTTP 伺服器，在邊緣傳送和接收連線裝置的訊息。

測試

請依照下列步驟進行測試：

- 使用 AWS 或 HTTP 伺服器設定 TLS 交互身分驗證的 PKI。
- 執行 CoreHTTP 整合測試。

移植 AWS IoT over-the-air(OTA) 更新程式庫

透過 FreeRTOS over-the-air(OTA) 更新，您可以執行下列動作：

- 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。
- 在將韌體新增至群組、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之後，驗證其真實性和完整性。
- 監控部署進度。
- 對失敗的部署進行除錯。
- 使用 Code Signing for 以數位方式簽署韌體 AWS IoT。

如需詳細資訊，請參閱 FreeRTOS Over-the-Air更新以及AWS IoT Over-the-air更新文件。 FreeRTOS

您可以使用 OTA 更新程式庫，將 OTA 功能整合到您的 FreeRTOS 應用程式。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

FreeRTOS 裝置必須在其收到的 OTA 韌體映像上強制執行密碼編譯程式碼簽署驗證。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界面。
- 整合 coreMQTT 程式庫。請參閱 FreeRTOS 使用者指南中的 coreMQTT 程式庫。
- 建立可支援 OTA 更新的開機載入器。

平台移植

您必須提供 OTA 可攜式抽象層 (PAL) 的實作，才能將 OTA 程式庫移植到新裝置。PAL APIs 是在 ota_platform_interface.h 檔案中定義，其中必須提供實作特定詳細資訊。

函數名稱	說明
`otaPal_Abort`	停止 OTA 更新。
`otaPal_CreateFileForRx`	建立檔案以存放接收的資料區塊。
`otaPal_CloseFile`	關閉指定的檔案。如果您使用實作密碼編譯保護的儲存體，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。成功時，函數會傳回寫入的位元組數。否則，函數會

函數名稱	說明
	傳回負錯誤代碼。區塊大小一律為 2 的功率，並會對齊。如需詳細資訊，請參閱 OTA 程式庫組態。
otaPal_ActivateNewImage	啟用或啟動新的韌體映像。對於某些連接埠，如果裝置以程式設計方式同步重設，則不會傳回此函數。
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。若要實作此函數，請參閱主機板 （平台） 詳細資訊和架構的文件。
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	說明
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。如需建立 AWS IoT 裝置開機載入器的說明，請參閱 IoT 裝置開機載入器。

E2E 和 PAL 測試

執行 OTA PAL 和 E2E 測試。

E2E 測試

OTA 端對端 (E2E) 測試用於驗證裝置的 OTA 功能，以及模擬來自現實的案例。此測試將包含錯誤處理。

先決條件

若要移植此測試，您需要下列項目：

- 整合 OTA AWS 程式庫的專案。如需其他資訊，請參閱 OTA 程式庫移植指南。
- 使用 OTA 程式庫移植示範應用程式，與 互動 AWS IoT Core 以進行 OTA 更新。請參閱 移植 OTA 示範應用程式。
- 設定 IDT 工具。這會執行 OTA E2E 主機應用程式，以使用不同的組態來建置、刷新和監控裝置，並驗證 OTA 程式庫整合。

移植 OTA 示範應用程式

OTA E2E 測試必須具有 OTA 示範應用程式，才能驗證 OTA 程式庫整合。示範應用程式必須具有執行 OTA 韌體更新的容量。您可以在 FreeRTOS GitHub 儲存庫找到 FreeRTOS OTA 示範應用程式。我們建議您使用示範應用程式做為參考，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. 實作 OTA 應用程式回呼函數。
3. 建立 OTA 代理程式事件處理任務。
4. 啟動 OTA 代理程式。
5. 監控 OTA 代理程式統計資料。
6. 關閉 OTA 代理程式。

如需詳細說明，請造訪示範的 FreeRTOS OTA over MQTT - 進入點。

Configuration

下列組態是與 互動的必要組態 AWS IoT Core：

- AWS IoT Core 用戶端登入資料

  - `Ota_Over_Mqtt_Demo/demo_config.h` 使用 Amazon Trust Services 端點在 中設定 democonfigROOT_CA_PEM。如需詳細資訊，請參閱AWS 伺服器身分驗證。
  - `Ota_Over_Mqtt_Demo/demo_config.h` 使用 AWS IoT 用戶端憑證在 中設定 democonfigCLIENT_CERTIFICATE_PEM 和 democonfigCLIENT_PRIVATE_KEY_PEM。請參閱AWS 用戶端身分驗證詳細資訊，以了解用戶端憑證和私有金鑰。
- 應用程式版本
- OTA 控制通訊協定
- OTA 資料通訊協定
- 程式碼簽署憑證
- 其他 OTA 程式庫組態

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

組建驗證

執行示範應用程式以執行 OTA 任務。成功完成後，您可以繼續執行 OTA E2E 測試。

FreeRTOS OTA 示範提供在 FreeRTOS Windows 模擬器上設定 OTA 用戶端和 AWS IoT Core OTA 任務的詳細資訊。 AWS OTA 同時支援 MQTT 和 HTTP 通訊協定。如需詳細資訊，請參閱下列範例：

- Windows 模擬器上的 OTA over MQTT 示範
- 在 Windows 模擬器上透過 HTTP 進行 OTA 示範

使用 IDT 工具執行測試

若要執行 OTA E2E 測試，您必須使用 AWS IoT Device Tester (IDT) 來自動化執行。如需詳細資訊AWS IoT Device Tester ，請參閱FreeRTOS 使用者指南》中的 FreeRTOS。

E2E 測試案例

測試案例	說明
`OTAE2EGreaterVersion`	定期 OTA 更新的快樂路徑測試。它使用更新版本建立更新，裝置會成功更新。

測試案例	說明
OTAE2EBackToBackDownloads	此測試會建立連續 3 次的 OTA 更新。裝置預期會連續更新 3 次。
OTAE2ERollbackIfUnableToConnectAfterUpdate	如果裝置無法使用新韌體連線至網路，此測試會驗證裝置是否復原至先前的韌體。
OTAE2ESameVersion	如果版本保持不變，此測試會確認裝置拒絕傳入的韌體。
OTAE2EUnsignedImage	如果影像未簽署，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。
OTAE2EPreviousVersion	此測試會驗證裝置是否拒絕較舊的更新版本。
OTAE2EIncorrectSigningAlgorithm	不同的裝置支援不同的簽署和雜湊演算法。如果使用不支援的演算法建立 OTA 更新，此測試會驗證裝置是否失敗。
OTAE2EDisconnectResume	這是暫停和繼續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）和登入資料連線至 。 AWS IoT Core 然後， 會中斷裝置連線。裝置預期會偵測到與 中斷連線 AWS IoT Core，並在一段時間後將自己移至暫停狀態，並嘗試重新連線至 AWS IoT Core 並繼續下載。

測試案例	說明
OTAE2EDisconnectCancelUpdate	如果 OTA 任務在處於暫停狀態時遭到取消，則此測試會檢查裝置是否可以自行復原。它會執行與OTAE2EDisconnectResume 測試相同的動作，除了 連線後 AWS IoT Core中斷裝置連線的裝置，它會取消 OTA 更新。建立新的更新。裝置預期會重新連線至 AWS IoT Core、中止目前的更新、返回等待狀態，並接受並完成下一次更新。
OTAE2EPresignedUrlExpired	建立 OTA 更新時，您可以設定 S3 預先簽章 URL 的生命週期。此測試會驗證裝置是否能夠執行 OTA，即使裝置無法在 URL 過期時完成下載。裝置預期會請求新的任務文件，其中包含新的 URL 以繼續下載。
OTAE2E2UpdatesCancel1st	此測試會連續建立兩個 OTA 更新。當裝置回報正在下載第一次更新時，測試力會取消第一次更新。裝置預期會中止目前的更新，並取得第二個更新，並完成它。
OTAE2ECancelThenUpdate	此測試會連續建立兩個 OTA 更新。當裝置回報正在下載第一次更新時，測試力會取消第一次更新。裝置預期會中止目前的更新並取得第二個更新，然後完成它。
OTAE2EImageCrashed	此測試會檢查裝置是否能夠在映像當機時拒絕更新。

PAL 測試

先決條件

若要移植網路傳輸介面測試，您需要下列項目：

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

移植

- 將 FreeRTOS-Libraries-Integration-Tests 作為子模組新增至您的專案。專案中子模組的位置必須是可建置的位置。
- 將 config_template/test_execution_config_template.h和 config_template/test_param_config_template.h 複製到建置路徑中的位置，並將其重新命名為 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/ota_pal_tests.cmake可用來包含相關檔案。
- 透過實作下列函數來設定測試：
  - SetupOtaPalTestParam()：在 中定義src/ota/ota_pal_test.h。實作必須具有與 中定義的相同名稱和簽章ota_pal_test.h。目前，您不需要設定此函數。
- 實作 UNITY_OUTPUT_CHAR，讓測試輸出日誌不會與裝置日誌交錯。
- RunQualificationTest() 從應用程式呼叫 。裝置硬體必須正確初始化，且必須在呼叫之前連接網路。

測試

本節說明 OTA PAL 資格測試的本機測試。

啟用測試

開啟 OTA_PAL_TEST_ENABLED test_execution_config.h並定義為 1。

在 中test_param_config.h，更新下列選項：

- OTA_PAL_TEST_CERT_TYPE：選取使用的憑證類型。
- OTA_PAL_CERTIFICATE_FILE：裝置憑證的路徑，如適用。
- OTA_PAL_FIRMWARE_FILE：韌體檔案的名稱，如適用。
- OTA_PAL_USE_FILE_SYSTEM：如果 OTA PAL 使用檔案系統抽象，請將 設定為 1。

使用您選擇的工具鏈建置和刷新應用程式。RunQualificationTest() 呼叫 時，OTA PAL 測試將會執行。測試結果會輸出到序列連接埠。

整合 OTA 任務

- 將 OTA 代理程式新增至您目前的 MQTT 示範。

- 使用 執行 OTA 端對端 (E2E) 測試 AWS IoT。這會驗證整合是否如預期般運作。

> **Note**
>
> 若要正式讓裝置符合 FreeRTOS 的資格，您必須使用 OTA PAL 和 OTA E2E 測試群組來驗證裝置的移植原始碼 AWS IoT Device Tester。請遵循 FreeRTOS 使用者指南中的使用 AWS IoT Device Tester for FreeRTOS 中的指示來設定 AWS IoT Device Tester 連接埠驗證。FreeRTOS 若要測試特定程式庫的連接埠，必須在 資料夾中的 `device.json` 檔案中 AWS IoT Device Tester `configs`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開機載入器應用程式。請確定設計和實作提供安全威脅的適當緩解措施。以下是供您參考的威脅建模。

IoT 裝置開機載入器的威脅模型

背景介紹

作為工作定義，此威脅模型參考的內嵌 AWS IoT 裝置是與雲端服務互動的微控制器型產品。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。透過考慮對手的目標和方法，會建立威脅清單。威脅是對手對資源或資產的攻擊。清單會排定優先順序，並用於識別和建立緩解解決方案。選擇緩解解決方案時，實作和維護它的成本應與它提供的實際安全值平衡。有多種 威脅模型方法。每個 都能夠支援開發安全且成功的 AWS IoT 產品。

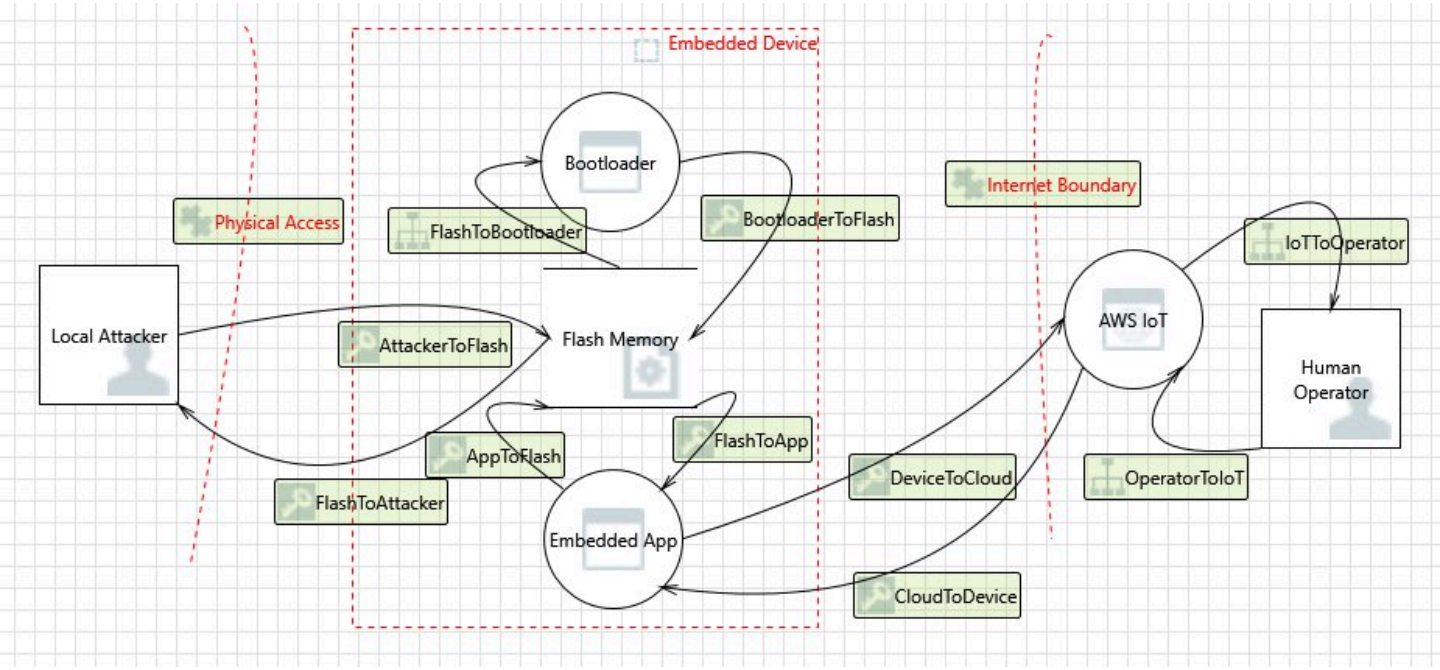
FreeRTOS 為 AWS IoT 裝置提供 OTA over-the-air) 軟體更新。更新設施結合雲端服務與裝置內軟體程式庫和合作夥伴提供的開機載入器。此威脅模型特別著重於對開機載入器的威脅。

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

有些攻擊有多個緩解模型；例如，透過驗證對 TLS 伺服器所提供憑證的信任，以及新韌體映像的程式碼簽署者憑證，來緩解旨在交付惡意韌體映像的網路man-in-the-middle。為了最大化開機載入器的安全性，任何非開機載入器緩解解決方案都會被視為不可靠。開機載入器應具有每次攻擊的內部緩解解決方案。擁有分層緩解解決方案稱為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。當驗證失敗且沒有可用的先前映像時，開機載入器會進入故障安全僅限 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。
  - 開機載入器位於 OTP (一次性可程式化記憶體) 中，且無法修改。
  - 開機載入器位於 ARM TrustZone 的安全區域中，且無法修改。
- 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

移植行動介面程式庫

FreeRTOS 支援 TCP 卸載行動抽象層的 AT 命令。如需詳細資訊，請參閱 https：//freertos.org 上的行動介面程式庫和移植行動介面程式庫。

先決條件

行動介面程式庫沒有直接相依性。不過，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來與 整合移植網路傳輸界面。

> **Note**
>
> 如果行動模組能夠支援 TLS 卸載，或不支援 AT 命令，開發人員可以實作自己的行動抽象，以與 整合移植網路傳輸界面。

從 MQTT 第 3 版遷移至 coreMQTT

本遷移指南說明如何將應用程式從 MQTT 遷移至 coreMQTT。

從 OTA 應用程式第 1 版遷移到第 3 版

本指南將協助您將應用程式從 OTA 程式庫第 1 版遷移到第 3 版。

> **Note**
>
> OTA 第 2 版 APIs 與 OTA v3 APIs 相同，因此如果您的應用程式使用第 2 版 APIs，則 API 呼叫不需要變更，但建議您整合第 3 版的程式庫。

以下提供 OTA 第 3 版的示範：

- ota_demo_core_mqtt。
- ota_demo_core_http。
- ota_ble。

API 變更摘要

OTA Library 第 1 版和第 3 版之間的 API 變更摘要

OTA 第 1 版 API	OTA 第 3 版 API	變更說明
OTA_AgentInit	OTA_Init	由於 OTA v3 中的實作變更，輸入參數以及從函數傳回的值也會變更。如需詳細資訊，請參閱以下 OTA_Init 一節。
OTA_AgentShutdown	OTA_Shutdown	輸入參數的變更，包括選擇性取消訂閱 MQTT 主題的額外參數。如需詳細資訊，請參閱以下 OTA_Shutdown 一節。
OTA_GetAgentState	OTA_GetState	API 名稱會變更，而輸入參數不會變更。傳回值相同，但列舉和成員會重新命名。如需詳細資訊，請參閱以下 OTA_GetState 一節。

OTA 第 1 版 API	OTA 第 3 版 API	變更說明
N/A	OTA_GetStatistics	新增了取代 APIs的新 API：OTA_GetPacketsReceived、OTA_GetPacketsQueued、OTA_GetPacketsProcessed、OTA_GetPacketsDropped。如需詳細資訊，請參閱以下 OTA_GetStatistics 一節。
OTA_GetPacketsReceived	N/A	此 API 已從第 3 版中移除，並由 OTA_GetStatistics 取代。
OTA_GetPacketsQueued	N/A	此 API 已從第 3 版中移除，並由 OTA_GetStatistics 取代。
OTA_GetPacketsProcessed	N/A	此 API 已從第 3 版中移除，並由 OTA_GetStatistics 取代。
OTA_GetPacketsDropped	N/A	此 API 已從第 3 版中移除，並由 OTA_GetStatistics 取代。
OTA_ActivateNewImage	OTA_ActivateNewImage	輸入參數相同，但傳回的 OTA 錯誤碼會重新命名，並在 OTA 程式庫的第 3 版中新增新的錯誤碼。如需詳細資訊，請參閱 OTA_ActivateNewImage 一節。
OTA_SetImageState	OTA_SetImageState	輸入參數相同且已重新命名，傳回的 OTA 錯誤碼會重新命名，並在 OTA 程式庫第 3 版中新增新的錯誤碼。如需詳細資訊，請參閱 OTA_SetImageState 一節。

OTA 第 1 版 API	OTA 第 3 版 API	變更說明
OTA_GetImageState	OTA_GetImageState	輸入參數相同，傳回列舉在 OTA 程式庫第 3 版中重新命名。如需詳細資訊，請參閱 OTA_GetImageState 一節。
OTA_暫停	OTA_暫停	輸入參數相同，傳回的 OTA 錯誤代碼會重新命名，並在 OTA 程式庫的第 3 版中新增新的錯誤代碼。如需詳細資訊，請參閱 OTA_Suspend 一節。
OTA_繼續	OTA_繼續	連線的輸入參數會移除，因為連線是在 OTA 示範/應用程式中處理、傳回的 OTA 錯誤碼會重新命名，而且新的錯誤碼會新增至 OTA 程式庫的第 3 版。如需詳細資訊，請參閱 OTA_Resume 一節。
OTA_CheckForUpdate	OTA_CheckForUpdate	輸入參數相同，傳回的 OTA 錯誤代碼會重新命名，並在 OTA 程式庫的第 3 版中新增新的錯誤代碼。如需詳細資訊，請參閱 OTA_CheckForUpdate 一節。
N/A	OTA_EventProcessingTask	新增 API，這是處理 OTA 更新事件的主要事件迴圈，必須由應用程式任務呼叫。如需詳細資訊，請參閱 OTA_Event ProcessingTask 一節。

OTA 第 1 版 API	OTA 第 3 版 API	變更說明
N/A	OTA_SignalEvent	新增 API，並將事件新增至 OTA 事件佇列的背面，內部 OTA 模組會使用它來發出代理程式任務的訊號。如需詳細資訊，請參閱 OTA_SignalEvent 一節。
N/A	OTA_Err_strerror	用於 OTA 錯誤之錯誤碼轉換為字串的新 API。
N/A	OTA_JobParse_strerror	用於錯誤碼的新 API 轉換為任務剖析錯誤的字串。
N/A	OTA_OsStatus_strerror	狀態碼的新 API 轉換為 OTA 作業系統連接埠狀態的字串。
N/A	OTA_PalStatus_strerror	狀態碼的新 API 轉換為 OTA PAL 連接埠狀態的字串。

所需變更的說明

OTA_Init

在 v1 中初始化 OTA 代理程式時，會使用 `OTA_AgentInit` API，其會取得連線內容、物件名稱、完成回呼和逾時做為輸入的參數。

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

此 API 現在已變更為 `OTA_Init`，其中包含 ota、ota 介面、實物名稱和應用程式回呼所需的緩衝區參數。

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

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

已移除輸入參數 -

pvConnectionContext -

連線內容會移除，因為 OTA 程式庫第 3 版不需要將連線內容傳遞給它，而且 MQTT/HTTP 操作是由其各自的界面在 OTA 示範/應用程式中處理。

xTicksToWait -

呼叫 OTA_Init 之前，在 OTA 示範/應用程式中建立任務時，也會移除等待參數的刻度。

重新命名輸入參數 -

xFunc -

參數重新命名為 OtaAppCallback，其類型變更為 OtaAppCallback_t。

新的輸入參數 -

pOtaBuffer

應用程式必須配置緩衝區，並在初始化期間使用 OtaAppBuffer_t 結構將其傳遞至 OTA 程式庫。所需的緩衝區略有不同，具體取決於用於下載檔案的通訊協定。對於 MQTT 通訊協定，串流名稱的緩衝區是必要的，對於 HTTP 通訊協定，需要預先簽章的 URL 和授權機制的緩衝區。

使用 MQTT 進行檔案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,
    .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 進行檔案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
};
```

其中 -

```
    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 OtaInterfaces_t 類型的 OTA 介面參考。這組界面必須傳遞至 OTA 程式庫，並在作業系統界面中包含 MQTT 界面、HTTP 界面和平台抽象層界面。

OTA 作業系統界面

OTA 作業系統功能介面是一組 APIs，必須實作才能讓裝置使用 OTA 程式庫。此界面的函數實作會提供給使用者應用程式中的 OTA 程式庫。OTA 程式庫會呼叫函數實作，以執行通常

由作業系統提供的功能。這包括管理事件、計時器和記憶體配置。FreeRTOS 和 POSIX 的實作隨附於 OTA 程式庫。

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 連接埠的 Linux 範例 -

```
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 介面

OTA MQTT 介面是在程式庫中實作APIs，讓 OTA 程式庫能夠從串流服務下載檔案區塊。

透過 MQTT 示範從 OTA 使用 coreMQTT https://github.com/aws/amazon-freertos/blob/main/demos/ota/ota_demo_core_mqtt/ota_demo_core_mqtt.c代理程式 APIs 的範例

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

HTTP 介面

OTA HTTP 介面是一組 APIs，必須在程式庫中實作，讓 OTA 程式庫透過連線至預先簽章的 URL 並擷取資料區塊來下載檔案區塊。除非您將 OTA 程式庫設定為從預先簽章的 URL 而非串流服務下載，否則這是選用的。

透過 HTTP 的 OTA 使用 coreHTTP APIs的範例示範- https://github.com/aws/amazon-freertos/blob/main/demos/ota/ota_demo_core_http/ota_demo_core_http.c

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

OTA PAL 界面

OTA PAL 界面是一組 APIs，必須實作才能讓裝置使用 OTA 程式庫。OTA PAL 的裝置特定實作會提供給使用者應用程式中的程式庫。程式庫使用這些函數來存放、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;
```

傳回的變更 -

傳回會從 OTA 代理程式狀態變更為 OTA 錯誤碼。請參閱AWS IoT Over-the-air更新 3.0.0 版：OtaErr_t。

OTA_Shutdown

在 OTA 程式庫第 1 版中，用於關閉 OTA 代理程式的 API 是 OTA_AgentShutdown，現在會變更為 OTA_Shutdown，以及輸入參數的變更。

OTA 代理程式關機 (第 1 版)

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

OTA 代理程式關機 (第 3 版)

```
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量。如果將此設為零，則函數會立即傳回而不等待。實際狀態會傳回給發起人。代理程式不會就此休眠，但用於忙碌迴圈。

新的輸入參數 -

unsubscribeFlag -

指出在呼叫關機時是否應從任務主題執行取消訂閱操作的旗標。如果旗標為 0，則不會針對任務主題呼叫取消訂閱操作。如果應用程式必須取消訂閱任務主題，則在呼叫 OTA_Shutdown 時，此旗標必須設定為 1。

傳回的變更 -

OtaState_t -

OTA 代理程式狀態及其成員的列舉會重新命名。請參閱AWS IoT Over-the-air更新 v3.0.0。

OTA_GetState

API 名稱會從 OTA_AgentGetState 變更為 OTA_GetState。

OTA 代理程式關機 (第 1 版)

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

OTA 代理程式關機 (第 3 版)

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

傳回的變更 -

OtaState_t -

OTA 代理程式狀態及其成員的列舉會重新命名。請參閱AWS IoT Over-the-air更新 v3.0.0。

OTA_GetStatistics

為統計資料新增新的單一 API。它會取代 APIs
OTA_GetPacketsReceived、OTA_GetPacketsQueued、OTA_GetPacketsProcessed、OTA_GetPacketsDro
此外，在 OTA Library 第 3 版中，統計資料編號僅與目前的任務相關。

OTA 程式庫第 1 版

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

OTA 程式庫第 3 版

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

pStatistics -

目前任務所接收、捨棄、排入佇列和處理之封包等統計資料的輸入/輸出參數。

輸出參數 -

OTA 錯誤碼。

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 ) );
```

OTA_ActivateNewImage

輸入參數相同，但傳回的 OTA 錯誤碼會重新命名，並在 OTA 程式庫的第 3 版中新增新的錯誤碼。

OTA 程式庫第 1 版

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

OTA 程式庫第 3 版

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

傳回的 OTA 錯誤碼列舉已變更，並新增新的錯誤碼。請參閱AWS IoT Over-the-air更新 3.0.0 版：OtaErr_t。

範例用量 -

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

OTA_SetImageState

輸入參數相同且重新命名，傳回的 OTA 錯誤碼會重新命名，並在 OTA 程式庫第 3 版中新增新的錯誤碼。

OTA 程式庫第 1 版

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

OTA 程式庫第 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 IoT Over-the-air更新 v3.0.0。

傳回的 OTA 錯誤碼列舉已變更，並新增新的錯誤碼。請參閱AWS IoT Over-the-air更新 v3.0.0 / OtaErr_t。

範例用量 -

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

OTA_GetImageState

輸入參數相同，傳回列舉會在 OTA 程式庫的第 3 版中重新命名。

OTA 程式庫第 1 版

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

OTA 程式庫第 3 版

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

傳回列舉會重新命名為 OtaImageState_t。請參閱AWS IoT Over-the-air更新 3.0.0 版：OtaImageState_t 。

範例用量 -

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

OTA_暫停

輸入參數相同，傳回的 OTA 錯誤代碼會重新命名，並在 OTA 程式庫的第 3 版中新增新的錯誤代碼。

OTA 程式庫第 1 版

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

OTA 程式庫第 3 版

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

傳回的 OTA 錯誤碼列舉已變更，並新增新的錯誤碼。請參閱AWS IoT Over-the-air更新 3.0.0 版：OtaErr_t。

範例用量 -

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

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

OTA_繼續

連線的輸入參數會移除，因為連線是在 OTA 示範/應用程式中處理、傳回的 OTA 錯誤碼會重新命名，而且新的錯誤碼會新增至 OTA 程式庫的第 3 版。

OTA 程式庫第 1 版

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

OTA 程式庫第 3 版

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

傳回的 OTA 錯誤碼列舉已變更，並新增新的錯誤碼。請參閱AWS IoT Over-the-air更新 3.0.0 版：OtaErr_t。

範例用量 -

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

OTA_CheckForUpdate

輸入參數相同，傳回的 OTA 錯誤代碼會重新命名，並在 OTA 程式庫的第 3 版中新增新的錯誤代碼。

OTA 程式庫第 1 版

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

OTA 程式庫第 3 版

```
OtaErr_t OTA_CheckForUpdate( void )
```

傳回的 OTA 錯誤碼列舉已變更，並新增新的錯誤碼。請參閱AWS IoT Over-the-air更新 3.0.0 版：OtaErr_t。

OTA_EventProcessingTask

這是新的 API，也是處理 OTA 更新事件的主要事件迴圈。應用程式任務必須呼叫它。此迴圈將繼續處理和執行針對 OTA 更新接收的事件，直到應用程式終止此任務為止。

OTA 程式庫第 3 版

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

FreeRTOS - 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 );
}
```

POSIX 的範例 -

```
/* 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;
}
```

OTA_SignalEvent

這是新的 API，可將事件新增至事件佇列的背面，內部 OTA 模組也會使用它來發出代理程式任務的訊號。

OTA 程式庫第 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 程式庫整合為應用程式中的子模組

如果您想要將 OTA 程式庫整合到您自己的應用程式中，您可以使用 git 子模組命令。Git 子模組可讓您將 Git 儲存庫保留為另一個 Git 儲存庫的子目錄。OTA 程式庫版本 3 維護在 ota-for-aws-iot-embedded-sdk 儲存庫中。

```
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。

參考

- OTAv1。
- OTAv3。

從 OTA PAL 連接埠的第 1 版遷移到第 3 版

Over-the-air更新程式庫在資料夾結構中引入了一些變更，以及程式庫和示範應用程式所需的組態配置。對於旨在使用 v1.2.0 遷移至程式庫 v3.0.0 的 OTA 應用程式，您必須更新 PAL 連接埠函數簽章，並包含本遷移指南中所述的其他組態檔案。

OTA PAL 的變更

- OTA PAL 連接埠目錄名稱已從 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 標頭檔案libraries/freertos_plus/aws/ota/src/aws_iot_ota_pal.h已從 OTA 程式庫中刪除，且必須在連接埠內定義。
- 傳回碼 (OTA_Err_t) 會轉譯為列舉 OTAMainStatus_t。如需翻譯的傳回代碼，請參閱 ota_platform_interface.h。 也提供協助程式巨集來合併 OtaPalMainStatus 和 OtaPalSubStatus程式碼，以及OtaMainStatus從 OtaPalStatus 和類似 擷取。
- 在 PAL 中記錄
  - 已移除DEFINE_OTA_METHOD_NAME巨集。
  - 先前版本：OTA_LOG_L1("[%s] Receive file created.\r\n", OTA_METHOD_NAME);。
  - 已更新：針對適當的日誌LogInfo(("Receive file created."));使用 LogDebug、LogError LogWarn和 。
- 變數cOTA_JSON_FileSignatureKey已變更為 OTA_JsonFileSignatureKey。

函數

函數簽章是在 中定義ota_pal.h，並以 字首開頭，otaPal而不是 prvPAL。

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

- 第 1 版： OTA_Err_t prvPAL_CreateFileForRx(OTA_FileContext_t * const *C*);

  第 3 版： OtaPalStatus_t otaPal_CreateFileForRx(OtaFileContext_t * const *pFileContext*);

注意：在資料區塊進來時，為資料區塊建立新的接收檔案。

- 第 1 版： `int16_t prvPAL_WriteBlock( OTA_FileContext_t * const C, uint32_t ulOffset, uint8_t * const pcData, uint32_t ulBlockSize );`

  第 3 版： `int16_t otaPal_WriteBlock( OtaFileContext_t * const pFileContext, uint32_t ulOffset, uint8_t * const pData, uint32_t ulBlockSize );`

  注意：以指定的位移將資料區塊寫入指定的檔案。

- 第 1 版： `OTA_Err_t prvPAL_ActivateNewImage( void );`

  第 3 版： `OtaPalStatus_t otaPal_ActivateNewImage( OtaFileContext_t * const *pFileContext* );`

  注意：啟用透過 OTA 接收的最新 MCU 映像。

- 第 1 版： `OTA_Err_t prvPAL_ResetDevice( void );`

  第 3 版： `OtaPalStatus_t otaPal_ResetDevice( OtaFileContext_t * const *pFileContext* );`

  注意：重設裝置。

- 第 1 版： `OTA_Err_t prvPAL_CloseFile( OTA_FileContext_t * const *C* );`

  第 3 版： `OtaPalStatus_t otaPal_CloseFile( OtaFileContext_t * const *pFileContext* );`

  注意：在指定的 OTA 內容中驗證和關閉基礎接收檔案。

- 第 1 版： `OTA_Err_t prvPAL_Abort( OTA_FileContext_t * const *C* );`

  第 3 版： `OtaPalStatus_t otaPal_Abort( OtaFileContext_t * const *pFileContext* );`

  注意：停止 OTA 傳輸。

- 第 1 版： `OTA_Err_t prvPAL_SetPlatformImageState( OTA_ImageState_t *eState* );`

  第 3 版： `OtaPalStatus_t otaPal_SetPlatformImageState( OtaFileContext_t * const pFileContext, OtaImageState_t eState );`

注意：嘗試設定 OTA 更新映像的狀態。

- 第 1 版： `OTA_PAL_ImageState_t prvPAL_GetPlatformImageState( void );`

  第 3 版： `OtaPalImageState_t otaPal_GetPlatformImageState( OtaFileContext_t * const *pFileContext* );`

  注意：取得 OTA 更新映像的狀態。

資料類型

- 第 1 版： `OTA_PAL_ImageState_t`

  檔案： `aws_iot_ota_agent.h`

  第 3 版： `OtaPalImageState_t`

  檔案： `ota_private.h`

  注意：平台實作設定的影像狀態。

- 第 1 版： `OTA_Err_t`

  檔案： `aws_iot_ota_agent.h`

  第 3 版： `OtaErr_t OtaPalStatus_t (combination of OtaPalMainStatus_t and OtaPalSubStatus_t)`

  檔案：`ota.h`、`ota_platform_interface.h`

  注意：v1：這些是定義 32 個未簽署整數的巨集。v3：專用列舉，代表錯誤類型並與錯誤代碼相關聯。

- 第 1 版： `OTA_FileContext_t`

  檔案： `aws_iot_ota_agent.h`

  第 3 版： `OtaFileContext_t`

  檔案： `ota_private.h`

  注意：v1：包含資料的列舉和緩衝區。v3：包含額外的資料長度變數。

- 第 1 版： OTA_ImageState_t

  檔案： aws_iot_ota_agent.h

  第 3 版： OtaImageState_t

  檔案： ota_private.h

  注意：OTA 映像狀態

組態變更

檔案aws_ota_agent_config.h已重新命名為 ota_config.h，而 會將包含防護從 _AWS_OTA_AGENT_CONFIG_H_變更為 OTA_CONFIG_H_。

- 檔案aws_ota_codesigner_certificate.h已刪除。
- 包含新的記錄堆疊以列印偵錯訊息：

```
/************************************************/
/******* 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 ****************/
```

- 新增常數組態：

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

新檔案：ota_demo_config.h包含 OTA 示範所需的組態，例如程式碼簽署憑證和應用程式版本。

- signingcredentialSIGNING_CERTIFICATE_PEM 在 中定義的 demos/include/aws_ota_codesigner_certificate.h 已移至 ota_demo_config.h ，otapalconfigCODE_SIGNING_CERTIFICATE並且可以從 PAL 檔案存取為：

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

  檔案aws_ota_codesigner_certificate.h已刪除。
- 巨集 APP_VERSION_BUILD、 APP_VERSION_MINORAPP_VERSION_MAJOR已新增至 ota_demo_config.h。包含版本資訊的舊檔案已移除，例如 tests/include/aws_application_version.h、libraries/c_sdk/standard/common/include/iot_appversion32.h、demos/demo_runner/aws_demo_version.c。

OTA PAL 測試的變更

- 已移除「Full_OTA_AGENT」測試群組以及所有相關檔案。先前需要此測試群組才能符合資格。這些測試適用於 OTA 程式庫，而非特定於 OTA PAL 連接埠。OTA 程式庫現在具有 OTA 儲存庫中託管的完整測試涵蓋範圍，因此不再需要此測試群組。
- 已移除「Full_OTA_CBOR」和「Quarantine_OTA_CBOR」測試群組，以及所有相關檔案。這些測試不屬於資格測試的一部分。這些測試涵蓋的功能現在正在 OTA 儲存庫中進行測試。
- 將測試檔案從程式庫目錄移至 tests/integration_tests/ota_pal 目錄。
- 更新 OTA PAL 資格測試，以使用 OTA 程式庫 API 的 v3.0.0。
- 更新 OTA PAL 測試如何存取程式碼簽署憑證以進行測試。先前有程式碼簽署登入資料的專用標頭檔案。新版本的程式庫不再如此。測試程式碼預期此變數會在 中定義ota_pal.c。此值會指派給平台特定 OTA 組態檔案中定義的巨集。

檢查清單

使用此檢查清單以確保您遵循遷移所需的步驟：

- 將 ota pal 連接埠資料夾的名稱從 更新ota為 ota_pal_for_aws。
- ota_pal.h 使用上述函數新增 檔案。如需範例ota_pal.h檔案，請參閱 GitHub。
- 新增組態檔案：
  - 將檔案名稱從 aws_ota_agent_config.h變更為 （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，並以 ota_demo_config.h 取代 的任何 包含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 程式庫檔案名稱更新 包含 。
  - 移除 DEFINE_OTA_METHOD_NAME 巨集。
  - 更新 OTA PAL 函數的簽章。
  - 將檔案內容變數的名稱從 更新C為 pFileContext。
  - 更新OTA_FileContext_t結構和所有相關變數。
  - 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。
  - 將 更新signingcredentialSIGNING_CERTIFICATE_PEM為 otapalconfigCODE_SIGNING_CERTIFICATE。
  - 更新 otaPal_CheckFileSignature 和 otaPal_ReadAndAssumeCertificate 函數註解。
- 更新 CMakeLists.txt 檔案。

- 更新 IDE 專案。

文件歷史記錄

下表說明 FreeRTOS 移植指南和 FreeRTOS 資格指南的文件歷史記錄。

日期	文件版本	變更歷史記錄	FreeRTOS 版本
2022 年 5 月	FreeRTOS 移植指南 FreeRTOS 資格指南	• 更新現有測試、新增測試，並根據 FreeRTOS 長期支援 (LTS) 程式庫移除備援測試。如需詳細資訊，請參閱 GitHub 上的 FreeRTOS 程式庫整合測試 202205.00。 • 已更新FreeRTOS 移植流程圖。 • 新增 移植網路傳輸界面。 • 移植 AWS IoT over-the-air(OTA) 更新程式庫 現在需要 才能符合資格。 • 已移除 Wi-Fi 和 TLS 抽象移植指南，因為不再需要。 • 如需 FreeRTOS 資格的進一步更新，請參閱最新變更。	202012.04-LTS 202112.00
2021 年 7 月	202107.00（移植指南）	• 版本 202107.00 • 已變更 移植 AWS IoT over-the-	202107.00

日期	文件版本	變更歷史記錄	FreeRTOS 版本
	202107.00（資格指南）	air(OTA) 更新程式庫 • 已新增 從 OTA 應用程式第 1 版遷移到第 3 版 • 已新增 從 OTA PAL 連接埠的第 1 版遷移到第 3 版	
2020 年 12 月	202012.00（移植指南） 202012.00（資格指南）	• 版本 202012.00 • 已新增 設定 coreHTTP 程式庫 • 已新增 移植行動介面程式庫	202012.00
2020 年 11 月	202011.00（移植指南） 202011.00（資格指南）	• 版本 202011.00 • 已新增 設定 coreMQTT 程式庫	202011.00
2020 年 7 月	202007.00（移植指南） 202007.00（資格指南）	• 版本 202007.00	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 現在是 FreeRTOS	202002.00

日期	文件版本	變更歷史記錄	FreeRTOS 版本
2019 年 12 月 17 日	201912.00 (移植指南) 201912.00 (資格指南)	• 發行版本 201912.00 • 新增共用 I/O 程式庫的移植。	201912.00
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 用戶端程式庫進行測試 已更新 移植 corePKCS11 程式庫	201908.00
2019 年 6 月 17 日	201906.00 （移植指南） 201906.00 （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)	• 移植文件移至 FreeRTOS 移植指南 • 資格文件已移至 FreeRTOS 資格指南	1.4.8

日期	文件版本	變更歷史記錄	FreeRTOS 版本
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
2018 年 11 月 26 日	1.1.3	• 新增低功耗藍牙移植附錄 (第 52 頁) • 在整份文件中新增 AWS IoT 了 Device Tester for FreeRTOS 測試資訊 • 已將 CMake 連結新增至 FreeRTOS 主控台附錄上列出的資訊 （ 第 85 頁 ）	1.4.4

日期	文件版本	變更歷史記錄	FreeRTOS 版本
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	變更歷史記錄	FreeRTOS 版本
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	變更歷史記錄	FreeRTOS 版本
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 附錄的指示 （第 AWS CLI 57 頁） 中新增了組態指示至範例	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。