aws

Guía de portabilidad

FreeRTOS

FreeRTOS: Guía de portabilidad

Table of Contents

Portabilidad de FreeRTOS

Qué es FreeRTOS

Desarrollado en colaboración con las principales compañías de chips del mundo durante un período de 20 años y ahora descargado cada 170 segundos, FreeRTOS es un sistema operativo en tiempo real (RTOS) líder del mercado para microcontroladores y microprocesadores. Distribuido libremente bajo la licencia de código abierto del MIT, FreeRTOS incluye un kernel y un conjunto creciente de bibliotecas apropiadas para su uso en todos los sectores de la industria. FreeRTOS se basa en la fiabilidad y la facilidad de uso. FreeRTOS incluye bibliotecas de conectividad, seguridad y actualizaciones over-the-air (OTA), así como aplicaciones de demostración que muestran las funciones de FreeRTOS en placas compatibles.

Para obtener más información, consulte FreeRTOS.org.

Portabilidad de FreeRTOS a su placa de IoT

Deberá realizar la portabilidad de las bibliotecas de software FreeRTOS a su placa basada en microcontroladores en función de sus características y de su aplicación.

Portabilidad de FreeRTOS a su dispositivo

1. Siga las instrucciones indicadas en Descarga de FreeRTOS para la portabilidad para descargar la última versión de FreeRTOS para la portabilidad.
2. Siga las instrucciones indicadas en Configuración del espacio de trabajo y el proyecto para la portabilidad para configurar los archivos y carpetas de la descarga de FreeRTOS para la portabilidad y las pruebas.
3. Siga las instrucciones indicadas en Portabilidad de las bibliotecas de FreeRTOS para realizar la portabilidad de las bibliotecas de FreeRTOS a su dispositivo. Cada tema de portabilidad incluye instrucciones sobre cómo probar los puertos.

Portabilidad FAQs

¿Qué es una portabilidad de FreeRTOS?

Un puerto de FreeRTOS es una implementación específica de una placa para las bibliotecas de FreeRTOS requeridas y APIs el núcleo de FreeRTOS que admite su plataforma. El puerto permite

APIs que funcione en la placa e implementa la integración necesaria con los controladores del dispositivo proporcionados por el proveedor de BSPs la plataforma. El puerto también debe incluir los demás ajustes de configuración que requiera la tarjeta (por ejemplo, velocidad de reloj, tamaño de pila o tamaño de montón).

Si tiene alguna pregunta sobre la portabilidad que no esté respondida en esta página o en el resto de la Guía de portabilidad de FreeRTOS, consulte las opciones de soporte de FreeRTOS disponibles.

Descarga de FreeRTOS para la portabilidad

Descarga la última versión de FreeRTOS o Long Term Support (LTS) de freertos.org o clona desde (FreerTOS-LTS) o (GitHub FreeRTOS).

> **Note**
>
> Le recomendamos clonar el repositorio. La clonación facilita la propagación de las actualizaciones de la rama principal cuando se envían al repositorio.

Como alternativa, submodule las bibliotecas individuales del repositorio FreeRTOS o FreeRTOS-LTS. Sin embargo, asegúrese de que las versiones de la biblioteca coincidan con la combinación que aparece en el archivo `manifest.yml` del repositorio FreeRTOS o FreeRTOS-LTS.

Después de descargar o clonar FreeRTOS, puede empezar a realizar la portabilidad de las bibliotecas de FreeRTOS a su placa. Para obtener instrucciones, consulte Configuración del espacio de trabajo y el proyecto para la portabilidad y Portabilidad de las bibliotecas de FreeRTOS.

Configuración del espacio de trabajo y el proyecto para la portabilidad

Siga los pasos que se indican a continuación para configurar el espacio de trabajo y el proyecto:

- Utilice la estructura de proyecto y el sistema de creación de su elección para importar las bibliotecas de FreeRTOS.
- Cree un proyecto utilizando un entorno de desarrollo integrado (IDE) y una cadena de herramientas compatibles con su placa.
- Incluya en el proyecto los paquetes de soporte de la placa (BSP) y los controladores específicos de la placa.

Una vez que el espacio de trabajo esté configurado, puede empezar a realizar la portabilidad de bibliotecas individuales de FreeRTOS.

Portabilidad de las bibliotecas de FreeRTOS

Antes de comenzar la portabilidad, siga las instrucciones que se indican en Configuración del espacio de trabajo y el proyecto para la portabilidad.

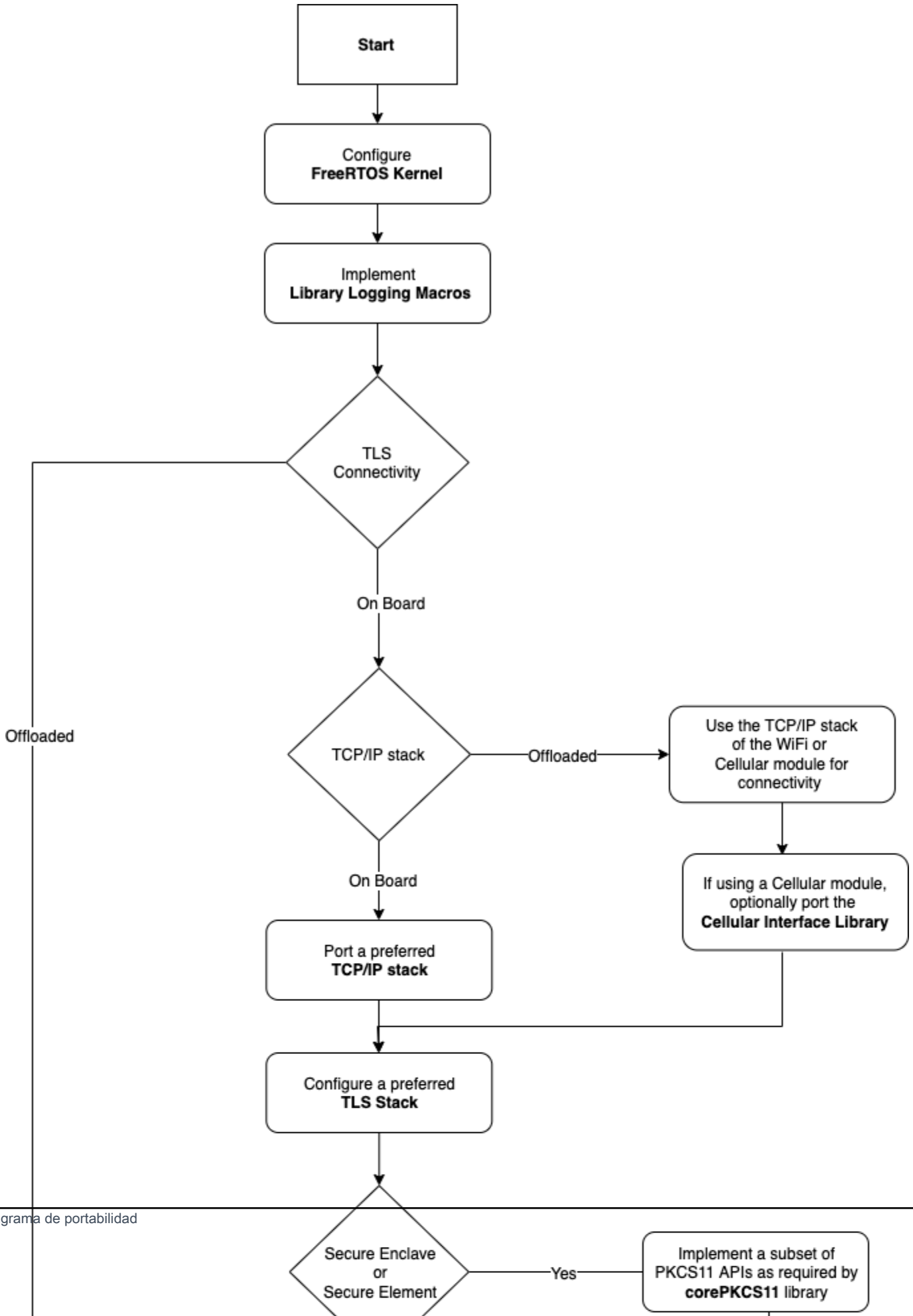
En el Diagrama de flujo de portabilidad de FreeRTOS se describen las bibliotecas necesarias para realizar la portabilidad.

Para realizar la portabilidad de FreeRTOS a su dispositivo, siga las instrucciones que se indican en los temas siguientes.

1. Configuración de un puerto de kernel FreeRTOS
2. Implementación de las macros de registro de la biblioteca
3. Portabilidad de una pila TCP/IP
4. Portabilidad de la interfaz de transporte de red
5. Migración de la PKCS11 biblioteca principal
6. Configuración de la biblioteca coreMQTT
7. Configuración de la biblioteca coreHTTP
8. Portabilidad de la biblioteca de actualizaciones AWS IoT over-the-air (OTA)
9. Portabilidad de la biblioteca de interfaces móviles

Diagrama de flujo de portabilidad de FreeRTOS

Use el siguiente diagrama de flujo de portabilidad como ayuda visual, mientras realiza la portabilidad de FreeRTOS a su placa.


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

Configuración de un puerto de kernel FreeRTOS

En esta sección se proporcionan instrucciones para integrar una portabilidad del kernel de FreeRTOS en un proyecto de prueba de portabilidad de FreeRTOS. Para obtener una lista de las transferencias de kernel disponibles, consulte FreeRTOS Kernel Ports.

FreeRTOS utiliza el kernel de FreeRTOS para las comunicaciones multitarea e intertarea. Para obtener más información, consulte Aspectos fundamentales del kernel de FreeRTOS en la Guía del usuario de FreeRTOS y en FreeRTOS.org.

> **Note**
>
> La portabilidad del kernel de FreeRTOS a una nueva arquitectura no se trata en esta documentación. Si está interesado, póngase en contacto con el equipo de ingeniería de FreeRTOS.
>
> En el programa de calificación de FreeRTOS, solo son compatibles las portabilidades de kernel de FreeRTOS existentes. No se admiten modificaciones de estas portabilidades dentro del programa. Consulte la Política de portabilidad de kernel de FreeRTOS para obtener más información.

Requisitos previos

Para configurar el kernel FreeRTOS para la portabilidad, necesita lo siguiente:

- Una portabilidad oficial del kernel de FreeRTOS o portabilidades compatibles de FreeRTOS para la plataforma de destino.
- Un proyecto en el IDE que incluya los archivos de puerto del kernel FreeRTOS correctos para la plataforma y compilador de destino. Para obtener información sobre el modo de configurar un proyecto de prueba, consulte Configuración del espacio de trabajo y el proyecto para la portabilidad.

Configuración del kernel FreeRTOS

El kernel de FreeRTOS se personaliza mediante un archivo de configuración llamado `FreeRTOSConfig.h`. El archivo especifica los ajustes de configuración del kernel específicos de la aplicación. Para obtener una descripción de cada opción de configuración, consulte la sección Personalización en FreeRTOS.org.

Para configurar el kernel de FreeRTOS para que funcione con su dispositivo, incluya `FreeRTOSConfig.h` y modifique cualquier configuración adicional de FreeRTOS.

Para obtener una descripción de cada opción de configuración, consulte las configuraciones de Personalización en FreeRTOS.org.

Testeo

- Ejecute una sencilla tarea de FreeRTOS para registrar un mensaje en la consola de salida en serie.
- Compruebe que el mensaje llega a la consola según lo esperado.

Implementación de las macros de registro de la biblioteca

Las bibliotecas de FreeRTOS utilizan las siguientes macros de registro, enumeradas en orden creciente de detalle.

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

Debe proporcionarse una definición para todas las macros. Las recomendaciones son las siguientes:

- Las macros deberían admitir el registro de estilos `C89`.
- El registro debe ser seguro para subprocesos. Las líneas de registro de varias tareas no deben intercalarse entre sí.
- El registro no APIs debe bloquear y debe liberar las tareas de la aplicación del bloqueo de E/S.

Consulte la funcionalidad de registro en FreeRTOS.org para obtener información específica sobre la implementación. Puede ver una implementación en este ejemplo.

Testeo

- Ejecute una prueba con varias tareas para comprobar que los registros no se intercalan.
- Realice una prueba para comprobar que el registro APIs no bloquea la E/S.

- Pruebe las macros de registro con varios estándares, como el registro de estilos `C89,C99`.
- Pruebe las macros de registro configurando diferentes niveles de registro, como `Debug`, `Info`, `Error` y `Warning`.

Portabilidad de una pila TCP/IP

En esta sección se proporcionan instrucciones para portar y probar la funcionalidad integrada TCP/IP stacks. If your platform offloads TCP/IP y de TLS a un procesador o módulo de red independiente. Puede omitir esta sección de portabilidad y visitar. Portabilidad de la interfaz de transporte de red

FreerTOS+TCP es una TCP/IP stack for the FreeRTOS kernel. FreeRTOS+TCP is developed and maintained by the FreeRTOS engineering team and is the recommended TCP/IP pila nativa para usar con FreeRTOS. Para obtener más información, consulte Portabilidad de FreeRTOS+TCP. Como alternativa, puede utilizar la pila TCP/IP de terceros lwIP. La instrucción de prueba que se proporciona en esta sección utiliza las pruebas de la interfaz de transporte para texto simple de TCP y no depende de la pila de TCP/IP específica implementada.

Portabilidad de FreeRTOS+TCP

FreeRTOS+TCP es una pila TCP/IP nativa para el kernel de FreeRTOS. Para obtener más información, consulte FreeRTOS.org.

Requisitos previos

Para realizar la portabilidad de la biblioteca FreeRTOS+TCP, necesita lo siguiente:

- Un proyecto de IDE que incluya los controladores de Ethernet o Wi-Fi suministrados por el proveedor.

  Para obtener información sobre el modo de configurar un proyecto de prueba, consulte Configuración del espacio de trabajo y el proyecto para la portabilidad.
- Una configuración validada del kernel de FreeRTOS.

  Para obtener información sobre cómo configurar el kernel de FreeRTOS para su plataforma, consulte Configuración de un puerto de kernel FreeRTOS.

Portabilidad

Antes de empezar a portar la biblioteca Freertos+TCP, comprueba en el GitHubdirectorio si ya existe un puerto para tu placa.

Si no existe ningún puerto, haga lo siguiente:

1. Siga las instrucciones ofrecidas en Porting FreeRTOS+TCP to a Different Microcontroller en FreeRTOS.org para realizar la portabilidad de FreeRTOS+TCP al dispositivo.
2. Si es necesario, siga las instrucciones de Porting FreeRTOS+TCP to a New Embedded C Compiler en FreeRTOS.org para realizar la portabilidad de FreeRTOS+TCP a un nuevo compilador.
3. Implemente una nueva portabilidad que use los controladores Ethernet o Wi-Fi proporcionados por el proveedor en un archivo llamado `NetworkInterface.c`. Visita el GitHubrepositorio para obtener una plantilla.

Después de crear una portabilidad, o si ya existe una, cree `FreeRTOSIPConfig.h` y edite las opciones de configuración para que sean correctas para su plataforma. Para obtener más información acerca de las opciones de configuración, consulte FreeRTOS+TCP Configuration en FreeRTOS.org.

Testeo

Tanto si utiliza la biblioteca FreeRTOS+TCP o una biblioteca de terceros, siga los pasos que se indican a continuación para realizar las pruebas:

- Proporcione una implementación para las pruebas `connect/disconnect/send/receive` APIs de la interfaz de transporte.
- Configure un servidor echo en el modo de conexión TCP de texto plano y ejecute las pruebas de la interfaz de transporte.

> **Note**
>
> Para calificar oficialmente un dispositivo para FreeRTOS, si su arquitectura requiere portar una pila de software TCP/IP, debe validar el código fuente portado del dispositivo con las pruebas de interfaz de transporte en modo de conexión TCP de texto plano con. AWS IoT Device Tester Siga las instrucciones de Uso AWS IoT Device Tester para FreeRTOS de

la Guía del usuario de FreeRTOS para configurar AWS IoT Device Tester la validación de puertos. Para probar el puerto de una biblioteca concreta, se debe habilitar el grupo de pruebas correcto en el archivo `device.json` de la carpeta `configs` de Device Tester.

Migración de la PKCS11 biblioteca principal

El estándar de criptografía de clave pública 11 define una API independiente de la plataforma para administrar y usar tokens criptográficos. PKCS 11 hace referencia al estándar y a lo que éste APIs define. La API criptográfica de PKCS 11 abstrae propiedades get/set, de almacenamiento de claves para objetos criptográficos y semántica de la sesión. Se usa ampliamente para manipular objetos criptográficos comunes. Sus funciones permiten al software de la aplicación utilizar, crear, modificar y eliminar objetos criptográficos sin exponerlos a la memoria de la aplicación.

Las bibliotecas FreeRTOS y las integraciones de referencia utilizan un subconjunto del estándar de interfaz PCKS 11 y se centran en las operaciones que implican claves asimétricas, generación de números aleatorios y hash. En la siguiente tabla se enumeran los casos de uso y el PKCS #11 necesario APIs para su compatibilidad.

Casos de uso

Caso de uso	Familia de API PKCS 11 necesaria
Todos	Inicializar, finalizar, abrir/cerrar sesión, iniciar sesión GetSlotList
Aprovisionando	GenerateKeyPair, CreateObject, DestroyObject, InitToken, GetTokenInfo
TLS	Aleatorio, Firmar, FindObject GetAttributeValue
FreeRTOS+TCP	Random, Digest
OTA	Verificar, digerir FindObject, GetAttributeValue

Cuándo implementar un módulo PKCS 11 completo

Almacenar claves privadas en una memoria flash de uso general puede resultar práctico en entornos de evaluación y creación rápida de prototipos. Le recomendamos que utilice hardware criptográfico

dedicado para reducir las amenazas de robo de datos y duplicación de dispositivos en escenarios de producción. El hardware criptográfico incluye componentes con características que impiden la exportación de las claves criptográficas secretas. Para ello, tendrá que implementar un subconjunto de PKCS 11 necesario para trabajar con las bibliotecas de FreeRTOS, tal y como se define en la tabla anterior.

Cuándo usar el núcleo de FreeRTOS PKCS11

La PKCS11 biblioteca principal contiene una implementación basada en software de la interfaz (API) PKCS #11 que utiliza la funcionalidad criptográfica proporcionada por Mbed TLS. Se proporciona para escenarios de creación rápida de prototipos y evaluación en los que el hardware no tiene un hardware criptográfico dedicado. En este caso, solo tiene que implementar la PKCS11 PAL básica para que la implementación básica PKCS11 basada en software funcione con su plataforma de hardware.

Portabilidad del núcleo PKCS11

Deberá disponer de implementaciones para leer y escribir objetos criptográficos en la memoria no volátil (NVM), como la memoria flash integrada. Los objetos criptográficos deben almacenarse en una sección de la NVM que no esté inicializada y que no se borre en caso de reprogramación del dispositivo. Los usuarios de la PKCS11 biblioteca principal suministrarán credenciales a los dispositivos y, a continuación, reprogramarán el dispositivo con una nueva aplicación que acceda a estas credenciales a través de la interfaz principalPKCS11 . Los puertos PKCS11 PAL principales deben proporcionar una ubicación para almacenar:

- El certificado de cliente del dispositivo
- La clave privada de cliente del dispositivo
- La clave pública de cliente del dispositivo
- Una CA raíz de confianza
- Una clave pública de verificación de código (o un certificado que contenga la clave pública de verificación de código) para garantizar la seguridad de las actualizaciones del gestor de arranque y (OTA) over-the-air
- Just-In-TimeUn certificado de aprovisionamiento

Incluya el archivo de cabecera e implemente el PAL APIs definido.

PAL APIs

Función	Descripción
PKCS11_PAL_Inicializar	Inicializa la capa de PAL. Lo llama la PKCS11 biblioteca principal al comienzo de su secuencia de inicialización.
PKCS11_PAL_ SaveObject	Escribe datos en almacenamiento no volátil.
PKCS11_PAL_ FindObject	Utiliza una etiqueta `CKA_LABEL` de PKCS #11 para buscar el objeto de PKCS # 11 correspon diente en el almacenamiento no volátil y devuelve un identificador del mismo, si existe.
PKCS11_PAL_ GetObjectValue	Obtiene el valor de un objeto a partir del identificador.
PKCS11_PAL_ GetObjectValueCleanup	Limpieza para la llamada `PKCS11_PA L_GetObjectValue` . Se puede utilizar para liberar la memoria asignada en una llamada `PKCS11_PAL_GetObjectValue` .

Testeo

Si utilizas la PKCS11 biblioteca principal de FreeRTOS o implementas el subconjunto necesario de PKCS11 APIs, debes superar las pruebas de FreeRTOS. PKCS11 Estas prueban si las funciones requeridas para las bibliotecas de FreeRTOS funcionan según lo esperado.

En esta sección también se describe cómo puede ejecutar localmente las pruebas de FreeRTOS con PKCS11 las pruebas de calificación.

Requisitos previos

Para configurar las PKCS11 pruebas de Freertos, se debe implementar lo siguiente.

- Un puerto compatible de. PKCS11 APIs
- Una implementación de las funciones de la plataforma de pruebas de calificación de FreeRTOS, que incluye las siguientes:

- `FRTest_ThreadCreate`
- `FRTest_ThreadTimedJoin`
- `FRTest_MemoryAlloc`
- `FRTest_MemoryFree`

(Consulte el archivo README.md para ver las pruebas de integración de las bibliotecas FreeRTOS para PKCS #11 en adelante). GitHub

Portabilidad de pruebas

- FreeRTOS-Libraries-Integration-TestsAñádelo como submódulo a tu proyecto. El submódulo se puede colocar en cualquier directorio del proyecto, siempre que se pueda crear.
- Copie `config_template/test_execution_config_template.h` y `config_template/test_param_config_template.h` en una ubicación del proyecto en la ruta de creación y cámbieles el nombre a `test_execution_config.h` y `test_param_config.h`.
- Incluya los archivos relevantes en el sistema creación. Si utiliza `CMake`, `qualification_test.cmake` y `src/pkcs11_tests.cmake` se pueden usar para incluir los archivos relevantes.
- Implemente `UNITY_OUTPUT_CHAR` de forma que los registros de salida de las pruebas y los registros del dispositivo no se intercalen.
- Integre el MbedTLS, que verifica el resultado de la operación cryptoki.
- Llame a `RunQualificationTest()` desde la aplicación.

Configuración de las pruebas

El conjunto PKCS11 de pruebas debe configurarse de acuerdo con la PKCS11 implementación. La siguiente tabla muestra la configuración requerida para las PKCS11 pruebas del archivo de `test_param_config.h` cabecera.

PKSC11 configuraciones de prueba

Configuración	Descripción
PKCS11_TEST_RSA_KEY_SUPPORT	La portabilidad admite las funciones clave de RSA.

Configuración	Descripción
PKCS11_TEST_EC_KEY_SUPPORT	La portabilidad admite las funciones clave de EC.
PKCS11_TEST_IMPORT_PRIVATE_KEY_SUPPORT	La portabilidad admite la importación de la clave privada. La importación de claves RSA y EC se valida en la prueba si las funciones clave auxiliares están habilitadas.
PKCS11_TEST_GENERATE_KEYPAIR_SUPPORT	La portabilidad admite la generación de pares de claves. La importación de pares de claves EC se valida en la prueba si las funciones clave auxiliares están habilitadas.
PKCS11_TEST_PREPROVISIONED_SUPPORT	La portabilidad tiene credenciales previamente aprovisionadas. `PKCS11_TEST_LABEL_DEVICE_PRIVATE_KEY_FOR_TLS`, `PKCS11_TEST_LABEL_DEVICE_PUBLIC_KEY_FOR_TLS` y `PKCS11_TEST_LABEL_DEVICE_CERTIFICATE_FOR_TLS` son ejemplos de las credenciales.
PKCS11_TEST_LABEL_DEVICE_PRIVATE_KEY_FOR_TLS	La etiqueta de la clave privada utilizada en la prueba.
PKCS11_TEST_LABEL_DEVICE_PUBLIC_KEY_FOR_TLS	La etiqueta de la clave pública utilizada en la prueba.
PKCS11_TEST_LABEL_DEVICE_CERTIFICATE_PARA_TLS	La etiqueta del certificado utilizado en la prueba.
PKCS11_TEST_JITP_CODEVERIFY_ROOT_CERT_SUPPORTED	La portabilidad admite el almacenamiento de JITP. Establézcalo en 1 para habilitar la prueba `codeverify` de JITP.

Configuración	Descripción
PKCS11_TEST_LABEL_CODE_VERIFICATION_KEY	La etiqueta de la clave de verificación de código utilizada en la prueba `codeverify` de JITP.
PKCS11_TEST_LABEL_JITP_CERTIFICATE	La etiqueta del certificado de JITP utilizado en la prueba `codeverify` de JITP.
PKCS11_TEST_LABEL_ROOT_CERTIFICATE	La etiqueta del certificado raíz utilizado en la prueba `codeverify` de JITP.

Las bibliotecas Freertos y las integraciones de referencia deben admitir como mínimo una configuración de función clave, como RSA o claves de curva elíptica, y un mecanismo de aprovisionamiento clave compatible con. PKCS11 APIs La prueba debe habilitar las siguientes configuraciones:

- Al menos una de las siguientes configuraciones de funciones de clave:
  - PKCS11_TEST_RSA_KEY_SUPPORT
  - PKCS11_TEST_EC_KEY_SUPPORT
- Al menos una de las siguientes configuraciones de aprovisionamiento de clave:
  - PKCS11_TEST_IMPORT_PRIVATE_KEY_SUPPORT
  - PKCS11_TEST_GENERATE_KEYPAIR_SUPPORT
  - PKCS11_TEST_PREPROVISIONED_SUPPORT

La prueba de credenciales del dispositivo aprovisionada previamente debe ejecutarse en las siguientes condiciones:

- `PKCS11_TEST_PREPROVISIONED_SUPPORT` debe estar activado y otros mecanismos de aprovisionamiento inhabilitados.
- Solo debe estar habilitada una función: `PKCS11_TEST_RSA_KEY_SUPPORT` o `PKCS11_TEST_EC_KEY_SUPPORT`.
- Configure las etiquetas de clave previamente aprovisionadas de acuerdo con su función de clave, incluidas `PKCS11_TEST_LABEL_DEVICE_PRIVATE_KEY_FOR_TLS`, `PKCS11_TEST_LABEL_DEVICE_PUBLIC_KEY_FOR_TLS` y

`PKCS11_TEST_LABEL_DEVICE_CERTIFICATE_FOR_TLS`. Estas credenciales deben existir antes de ejecutar la prueba.

Es posible que la prueba deba ejecutarse varias veces con diferentes configuraciones, si la implementación admite credenciales previamente aprovisionadas y otros mecanismos de aprovisionamiento.

> **Note**
>
> Los objetos con las etiquetas `PKCS11_TEST_LABEL_DEVICE_PRIVATE_KEY_FOR_TLS`, `PKCS11_TEST_LABEL_DEVICE_PUBLIC_KEY_FOR_TLS` y `PKCS11_TEST_LABEL_DEVICE_CERTIFICATE_FOR_TLS` se destruyen durante la prueba si `PKCS11_TEST_GENERATE_KEYPAIR_SUPPORT` o `PKCS11_TEST_GENERATE_KEYPAIR_SUPPORT` está habilitada.

Ejecución de pruebas

En esta sección se describe cómo se puede probar localmente la interfaz con las pruebas de calificación. PKCS11 También puede utilizar IDT para automatizar la ejecución. Consulte AWS IoT Device Tester para FreeRTOS en la Guía del usuario de FreeRTOS para obtener más información.

Las siguientes instrucciones describen cómo ejecutar las pruebas:

- Abra `test_execution_config.h` y defina CORE_ PKCS11 _TEST_ENABLED en 1.
- Cree e instale la aplicación en su dispositivo para ejecutarla. El resultado de la prueba se envía al puerto serie.

A continuación se muestra un ejemplo del resultado de la prueba de salida.

```
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
```

Las pruebas se completan cuando se superan todas.

> **Note**
>
> Para calificar oficialmente un dispositivo para FreeRTOS, debes validar con él el código fuente portado del dispositivo. AWS IoT Device Tester Siga las instrucciones de Uso AWS IoT Device Tester para FreeRTOS de la Guía del usuario de FreeRTOS para configurar AWS IoT Device Tester la validación de puertos. Para probar el puerto de una biblioteca específica, debe estar habilitado el grupo de prueba correcto en el `device.json` archivo de la AWS IoT Device Tester `configs` carpeta.

Portabilidad de la interfaz de transporte de red

Integración de la biblioteca TLS

Para la autenticación de la seguridad de la capa de transporte (TLS), utilice la pila de TLS que prefiera. Recomendamos usar Mbed TLS porque se ha probado con las bibliotecas FreeRTOS. Puedes encontrar un ejemplo de esto en este GitHubrepositorio.

Independientemente de la implementación de TLS que utilice su dispositivo, debe implementar los enlaces de transporte subyacentes para la pila de TLS con la pila de TCP/IP. Deben ser compatibles con los conjuntos de cifrado TLS compatibles con AWS IoT.

Portabilidad de la biblioteca de interfaces de transporte de red

Debe implementar una interfaz de transporte de red para usar coreMQTT y coreHTTP. La interfaz de transporte de red contiene los punteros de función y los datos de contexto necesarios para enviar y recibir datos en una sola conexión de red. Consulte Interfaz de transporte para obtener más información. FreeRTOS proporciona un conjunto de pruebas de interfaz de transporte de red integradas para validar estas implementaciones. La siguiente sección le guía sobre cómo configurar su proyecto para ejecutar estas pruebas.

Requisitos previos

Para realizar la portabilidad de esta prueba, necesita lo siguiente:

- Un proyecto con un sistema de creación que permita crear FreeRTOS con un puerto de kernel de FreeRTOS validado.
- Implementación funcional de los controladores de red.

Portabilidad

- FreeRTOS-Libraries-Integration-TestsAñádelo como submódulo a tu proyecto. No importa dónde se coloque el submódulo en el proyecto, siempre que se pueda crear.
- Copie `config_template/test_execution_config_template.h` y `config_template/test_param_config_template.h` en una ubicación del proyecto en la ruta de creación y cámbieles el nombre a `test_execution_config.h` y `test_param_config.h`.

- Incluya los archivos relevantes en el sistema creación. Si utiliza `CMake`, `qualification_test.cmake` y `src/transport_interface_tests.cmake` se usan para incluir los archivos relevantes.
- Implemente las siguientes funciones en la ubicación adecuada del proyecto:
- • Una `network connect function`: la firma se define mediante `NetworkConnectFunc` en `src/common/network_connection.h`. Esta función incorpora un puntero al contexto de red, un puntero a la información del host y un puntero a las credenciales de red. Establece una conexión con el servidor especificado en la información del host con las credenciales de red proporcionadas.
  - Una `network disconnect function`: la firma se define mediante `NetworkDisconnectFunc` en `src/common/network_connection.h`. Esta función incorpora un puntero a un contexto de red. Desconecta una conexión establecida previamente almacenada en el contexto de red.
  - `setupTransportInterfaceTestParam()`: se define en `src/transport_interface/transport_interface_tests.h`. La implementación debe tener exactamente el mismo nombre y firma que se definen en `transport_interface_tests.h`. Esta función lleva un puntero a una TransportInterfaceTestParamestructura. Rellenará los campos de la TransportInterfaceTestParamestructura que utiliza la prueba de la interfaz de transporte.
- Implemente UNITY_OUTPUT_CHAR para que los registros de salida de la prueba no se intercalen con los registros del dispositivo.
- Llame a `runQualificationTest()`desde la aplicación. El hardware del dispositivo debe estar correctamente inicializado y la red debe estar conectada antes de la llamada.

Gestión de credenciales (clave generada en el dispositivo)

Cuando FORCE_GENERATE_NEW_KEY_PAIR en `test_param_config.h` se establece en 1, la aplicación del dispositivo genera un nuevo par de claves en el dispositivo y genera la clave pública. La aplicación del dispositivo utiliza ECHO_SERVER_ROOT_CA y TRANSPORT_CLIENT_CERTIFICATE como CA raíz del servidor echo y certificado de cliente al establecer una conexión TLS con el servidor echo. IDT establece estos parámetros durante la ejecución de la calificación.

Gestión de credenciales (clave de importación)

La aplicación del dispositivo utiliza ECHO_SERVER_ROOT_CA, TRANSPORT_CLIENT_CERTIFICATE y TRANSPORT_CLIENT_PRIVATE_KEY en

`test_param_config.h` como CA raíz del servidor echo, certificado de cliente y clave privada de cliente al establecer una conexión TLS con el servidor echo. IDT establece estos parámetros durante la ejecución de la calificación.

Testeo

En esta sección se describe cómo puede probar localmente la interfaz de transporte con las pruebas de calificación. Puede encontrar más información en el archivo README.md que se proporciona en la sección transport_interface de la on. FreeRTOS-Libraries-Integration-Tests GitHub

También puede utilizar IDT para automatizar la ejecución. Consulte AWS IoT Device Tester para FreeRTOS en la Guía del usuario de FreeRTOS para obtener más información.

Habilitación de la prueba

Abra `test_execution_config.h` y defina TRANSPORT_INTERFACE_TEST_ENABLED en 1.

Configuración del servidor echo para realizar las pruebas

Para realizar las pruebas locales, es necesario disponer de un servidor de eco al que se pueda acceder desde el dispositivo que ejecuta las pruebas. El servidor echo debe admitir TLS si la implementación de la interfaz de transporte admite TLS. Si aún no tienes uno, el FreeRTOS-Libraries-Integration-Tests GitHub repositorio tiene una implementación de servidor echo.

Configuración del proyecto para probarlo

En `test_param_config.h`, actualice ECHO_SERVER_ENDPOINT y ECHO_SERVER_PORT con la configuración del punto de conexión y del servidor del paso anterior.

Configuración de las credenciales (clave generada en el dispositivo)

- Establezca ECHO_SERVER_ROOT_CA con el certificado de servidor del servidor echo.
- Establezca FORCE_GENERATE_NEW_KEY_PAIR en 1 para generar un par de claves y obtener la clave pública.
- Vuelva a establecer FORCE_GENERATE_NEW_KEY_PAIR en 0 tras la generación de la clave.
- Utilice la clave pública y la clave y el certificado de servidor para generar el certificado de cliente.
- Establezca TRANSPORT_CLIENT_CERTIFICATE con el certificado de cliente generado.

Configuración de las credenciales (clave de importación)

- Establezca ECHO_SERVER_ROOT_CA con el certificado de servidor del servidor echo.
- Establezca TRANSPORT_CLIENT_CERTIFICATE con el certificado de cliente generado previamente.
- Establezca TRANSPORT_CLIENT_PRIVATE_KEY con la clave privada de cliente generada previamente.

Creación e instalación de la aplicación

Cree e instale la aplicación con la cadena de herramientas que prefiera. Cuando se invoque a `runQualificationTest()`, se ejecutarán las pruebas de la interfaz de transporte. Los resultados de la prueba se envían al puerto de serie.

> **Note**
>
> Para calificar oficialmente un dispositivo para FreeRTOS, debes validar el código fuente portado del dispositivo con los grupos de prueba OTA PAL y OTA E2E con. AWS IoT Device Tester Siga las instrucciones de Uso AWS IoT Device Tester para FreeRTOS de la Guía del usuario de FreeRTOS para configurar AWS IoT Device Tester la validación de puertos. Para probar el puerto de una biblioteca específica, debe estar habilitado el grupo de prueba correcto en el `device.json` archivo de la AWS IoT Device Tester `configs` carpeta.

Configuración de la biblioteca coreMQTT

Los dispositivos periféricos pueden usar el protocolo MQTT para comunicarse con la AWS nube. AWS IoT aloja un intermediario MQTT que envía y recibe mensajes desde y hacia los dispositivos conectados en la periferia.

La biblioteca coreMQTT implementa el protocolo MQTT para los dispositivos que ejecutan FreeRTOS. No es necesario realizar la portabilidad de la biblioteca coreMQTT, pero el proyecto de prueba del dispositivo debe pasar todas las pruebas de MQTT para la calificación. Para obtener más información, consulte Biblioteca coreMQTT en la Guía del usuario de FreeRTOS.

Requisitos previos

Para configurar las pruebas de la biblioteca coreMQTT, necesita un puerto de interfaz de transporte de red. Consulte Portabilidad de la interfaz de transporte de red para obtener más información.

Testeo

Ejecute pruebas de integración de coreMQTT:

- Registre su certificado de cliente en el agente MQTT.
- Establezca el punto de conexión del agente en `config` y ejecute las pruebas de integración.

Creación de una demostración de MQTT de referencia

Recomendamos utilizar el agente coreMQTT para gestionar la seguridad de los subprocesos en todas las operaciones de MQTT. El usuario también necesitará tareas de publicación y suscripción, y pruebas de Device Advisor para validar si la aplicación integra TLS, MQTT y otras bibliotecas de FreeRTOS de forma eficaz.

Para calificar oficialmente un dispositivo para FreeRTOS, valide su proyecto de integración con casos de prueba de AWS IoT Device Tester MQTT. Consulte Flujo de trabajo de AWS IoT Device Advisor para obtener instrucciones de configuración y pruebas. Los casos de prueba obligatorios para TLS y MQTT se enumeran a continuación:

Casos de prueba de TLS

Caso de prueba	Casos de prueba	Pruebas obligatorias
TLS	Conexión TLS	Sí
TLS	Conjuntos de AWS IoT cifrado TLS Support	Un conjunto de cifrado recomendado
TLS	Certificado de servidor no seguro de TLS	Sí
TLS	Certificado de servidor con nombre de asunto incorrecto de TLS	Sí

Casos de prueba de MQTT

Caso de prueba	Casos de prueba	Pruebas obligatorias
MQTT	Conexión MQTT	Sí
MQTT	Reintentos de fluctuación de conexión MQTT	Sí, sin advertencias
MQTT	Suscripción a MQTT	Sí
MQTT	Publicación MQTT	Sí
MQTT	QoS1 de MQTT ClientPuback	Sí
MQTT	MQTT sin Amp PingResp	Sí

Configuración de la biblioteca coreHTTP

Los dispositivos periféricos pueden usar el protocolo HTTP para comunicarse con la AWS nube. AWS IoT los servicios alojan un servidor HTTP que envía y recibe mensajes hacia y desde los dispositivos conectados en la periferia.

Testeo

Siga los pasos que se indican a continuación para realizar las pruebas:

- Configure la PKI para la autenticación mutua de TLS con un servidor HTTP AWS o con él.
- Ejecute pruebas de integración de CoreHTTP.

Portabilidad de la biblioteca de actualizaciones AWS IoT over-the-air (OTA)

Con las actualizaciones de Freertos over-the-air (OTA), puede hacer lo siguiente:

- Implementar nuevas imágenes de firmware en un único dispositivo, grupo de dispositivos o toda la flota.

- Implementar firmware en dispositivos a medida que estos se añaden a grupos, restablecen o reaprovisionan.
- Verificar la autenticidad y la integridad del nuevo firmware una vez implementado en los dispositivos.
- Monitorizar el progreso de una implementación.
- Depurar una implementación infructuosa.
- Firme digitalmente el firmware mediante la firma de código para AWS IoT.

Para obtener más información, consulte Over-the-AirActualizaciones de FreeRTOS en la Guía del usuario de FreeRTOS junto con la documentación de actualización.AWS IoT Over-the-air

Puede utilizar la biblioteca de actualizaciones OTA para integrar la funcionalidad OTA en sus aplicaciones de FreeRTOS. Para obtener más información, consulte Biblioteca de actualizaciones OTA de FreeRTOS en la Guía del usuario de FreeRTOS.

Los dispositivos con FreeRTOS deben aplicar la verificación de la firma de código criptográfica de las imágenes de firmware OTA que reciban. Le recomendamos los siguientes algoritmos:

- Algoritmo de firma digital de curva elíptica (ECDSA)
- Curva NIST P256
- Hash SHA-256

Requisitos previos

- Complete las instrucciones que se indican en Configuración del espacio de trabajo y el proyecto para la portabilidad.
- Cree un puerto de interfaz de transporte de red.

  Para obtener más información, consulte Portabilidad de la interfaz de transporte de red.
- Integre la biblioteca coreMQTT. Consulte la Biblioteca coreMQTT en la Guía del usuario de FreeRTOS.
- Cree un cargador de arranque compatible con actualizaciones OTA.

Portabilidad de la plataforma

Debe proporcionar una implementación de la capa de abstracción portátil (PAL) OTA para realizar la portabilidad de la biblioteca OTA a un dispositivo nuevo. APIs Las PAL se definen en el archivo ota_platform_interface.h, para lo cual se deben proporcionar detalles específicos de la implementación.

Nombre de la función	Descripción
`otaPal_Abort`	Detiene una actualización OTA.
`otaPal_CreateFileForRx`	Crea un archivo para almacenar los fragmentos de datos recibidos.
`otaPal_CloseFile`	Cierra el archivo especificado. Esto puede autenticar el archivo si se utiliza almacenam iento con protección criptográfica.
`otaPal_WriteBlock`	Escribe un bloque de datos en el archivo especificado en el desplazamiento dado. Si se completa correctamente, la función devuelve el número de bytes escritos. De lo contrario , la función devuelve un código de error negativo. El tamaño del bloque siempre será una potencia de dos y estará alineado. Para obtener más información, consulte Configura ción de la biblioteca OTA.
`otaPal_ActivateNewImage`	Activa o lanza la nueva imagen de firmware. Para algunas portabilidades, si el dispositivo se restablece de forma síncrona mediante programación, esta función podría no retornar.
`otaPal_SetPlatformImageState`	Hace lo que requiera la plataforma para aceptar o rechazar la imagen (o paquete) de firmware OTA más reciente. Para implementar esta función, consulte los detalles y la arquitect

Nombre de la función	Descripción
	ura de la placa (plataforma) en la documenta ción.
`otaPal_GetPlatformImageState`	Obtiene el estado de la imagen de actualiza ción OTA.

Implemente las funciones de esta tabla si su dispositivo tiene compatibilidad integrada para ellas.

Nombre de la función	Descripción
`otaPal_CheckFileSignature`	Verifica la firma del archivo especificado.
`otaPal_ReadAndAssumeCertificate`	Lee el certificado de firma especificado del sistema de archivos y lo devuelve al intermedi ario.
`otaPal_ResetDevice`	Restablece el dispositivo.

> **Note**
>
> Asegúrese de disponer de un cargador de arranque compatible con actualizaciones OTA. Para obtener instrucciones sobre cómo crear el cargador de arranque de AWS IoT , consulte Cargador de arranque del dispositivo con IoT.

Pruebas E2E y PAL

Ejecución de las pruebas OTA PAL y E2E.

Pruebas E2E

La prueba integral (E2E) OTA se utiliza para verificar la capacidad OTA de un dispositivo y simular escenarios a partir de la realidad. Esta prueba incluirá la gestión de errores.

Requisitos previos

Para realizar la portabilidad de esta prueba, necesita lo siguiente:

- Un proyecto con una biblioteca OTA integrada. AWS Consulte la Guía de portabilidad de la biblioteca OTA para obtener información adicional.
- Realice la portabilidad de la aplicación de demostración con la biblioteca OTA para interactuar con AWS IoT Core para realizar las actualizaciones OTA. Consulte Portabilidad de la aplicación de demostración OTA.
- Configure la herramienta de IDT. Esto ejecuta la aplicación host E2E OTA para crear, instalar y supervisar el dispositivo con diferentes configuraciones, y valida la integración de la biblioteca OTA.

Portabilidad de la aplicación de demostración OTA

La prueba E2E OTA debe tener una aplicación de demostración OTA para validar la integración de la biblioteca OTA. La aplicación de demostración debe tener la capacidad de realizar actualizaciones de firmware OTA. Puede encontrar la aplicación de demostración de FreeRTOS OTA en el repositorio de FreeRTOS. GitHub Le recomendamos que utilice la aplicación de demostración como referencia y que la modifique según sus especificaciones.

Pasos de la portabilidad

1. Inicialice el agente de OTA.
2. Implemente la función de devolución de llamada de la aplicación OTA.
3. Cree la tarea de procesamiento de eventos del agente OTA.
4. Inicie el agente OTA.
5. Supervise las estadísticas del agente OTA.
6. Apague al agente OTA.

Consulte FreeRTOS OTA over MQTT - Entry point of the demo para obtener instrucciones detalladas.

Configuración

Se necesitan las siguientes configuraciones para interactuar con: AWS IoT Core

- AWS IoT Core credenciales de cliente
  - Configure democonfigROOT_CA_PEM en `Ota_Over_Mqtt_Demo/demo_config.h` con los puntos de conexión de Amazon Trust Services. Consulte server-authentication de AWS para obtener más información.
  - Configure DemoConfigClient_Certificate_PEM y DemoConfigClient_Private_KEY_PEM con las credenciales de su cliente. `Ota_Over_Mqtt_Demo/demo_config.h` AWS IoT Consulte los detalles de client-authentication de AWS para obtener información sobre los certificados de cliente y las claves privadas.
- Versión de la aplicación
- Protocolo de control OTA
- Protocolo de datos OTA
- Credenciales de firma de código
- Otras configuraciones de bibliotecas OTA

Puede encontrar la información anterior en las aplicaciones de demostración OTA de FreeRTOS `demo_config.h` y `ota_config.h`. Consulte FreeRTOS OTA over MQTT - Setting up the device para obtener más información.

Verificación de la creación

Ejecute la aplicación de demostración para ejecutar el trabajo OTA. Cuando se complete correctamente, podrá seguir ejecutando las pruebas E2E OTA.

La demostración OTA de FreeRTOS proporciona información detallada sobre la configuración de un cliente OTA y un trabajo de OTA en el AWS IoT Core simulador de Windows de FreeRTOS. AWS OTA es compatible con los protocolos MQTT y HTTP. Consulte los siguientes ejemplos para obtener más detalles:

- Demostración de OTA sobre MQTT en el simulador de Windows
- Demostración de OTA sobre HTTP en el simulador de Windows

Ejecución de pruebas con la herramienta IDT

Para ejecutar las pruebas OTA E2E, debe utilizar AWS IoT Device Tester (IDT) para automatizar la ejecución. Consulte AWS IoT Device Tester para FreeRTOS en la Guía del usuario de FreeRTOS para obtener más información.

Casos de prueba E2E

Caso de prueba	Descripción
`OTAE2EGreaterVersion`	Prueba de la ruta de Happy para actualizaciones periódicas OTA. Crea una actualización con una versión más reciente, que el dispositivo actualiza correctamente.
`OTAE2EBackToBackDownloads`	Esta prueba crea 3 actualizaciones OTA consecutivas. Se espera que el dispositivo se actualice 3 veces consecutivas.
`OTAE2ERollbackIfUnableToConnectAfterUpdate`	Esta prueba verifica que el dispositivo vuelva al firmware anterior si no puede conectarse a la red con el nuevo firmware.
`OTAE2ESameVersion`	Esta prueba confirma que el dispositivo rechaza el firmware entrante si la versión sigue siendo la misma.
`OTAE2EUnsignedImage`	Esta prueba verifica que el dispositivo rechaza una actualización si la imagen no está firmada.
`OTAE2EUntrustedCertificate`	Esta prueba verifica que el dispositivo rechaza una actualización si el firmware está firmado con un certificado que no es de confianza.
`OTAE2EPreviousVersion`	Esta prueba verifica que el dispositivo rechaza una versión de una actualización anterior.
`OTAE2EIncorrectSigningAlgorithm`	Los diferentes dispositivos admiten diferentes algoritmos de firma y hash. Esta prueba verifica que el dispositivo no supera la actualización OTA si se creó con un algoritmo no compatible.
`OTAE2EDisconnectResume`	Esta es la prueba de happy path para la característica de suspensión y reanudación. Esta prueba crea una actualización OTA

Caso de prueba	Descripción
	e inicia la actualización. A continuación, se conecta AWS IoT Core con el mismo ID de cliente (nombre de la cosa) y las mismas credenciales. AWS IoT Core a continuación, desconecta el dispositivo. Se espera que el dispositivo detecte que está desconectado y AWS IoT Core, transcurrido un tiempo, pase a un estado suspendido e intente volver a conectarse AWS IoT Core y reanudar la descarga.
`OTAE2EDisconnectCancelUpdate`	Esta prueba comprueba si el dispositivo puede recuperarse por sí solo si se cancela el trabajo OTA mientras está suspendido. Hace lo mismo que en la `OTAE2EDisconnectResume` prueba, excepto que tras conectarse a AWS IoT Core, lo que desconecta el dispositivo, cancela la actualización OTA. Se crea una nueva actualización. Se espera que el dispositivo se vuelva a conectar AWS IoT Core, aborte la actualización actual, vuelva al estado de espera y acepte y finalice la próxima actualización.
`OTAE2EPresignedUrlExpired`	Cuando se crea una actualización OTA, puede configurar la duración de la URL prefirmada de S3. Esta prueba verifica que el dispositivo puede realizar una OTA, incluso si no puede finalizar la descarga cuando caduca la URL. Se espera que el dispositivo solicite un nuevo documento de trabajo que contenga una nueva URL para reanudar la descarga.

Caso de prueba	Descripción
`OTAE2E2UpdatesCancel1st`	Esta prueba crea dos actualizaciones OTA seguidas. Cuando el dispositivo informa que está descargando la primera actualización, la prueba la cancela forzosamente. Se espera que el dispositivo aborte la actualización actual, recoja la segunda actualización y la complete.
`OTAE2ECancelThenUpdate`	Esta prueba crea dos actualizaciones OTA seguidas. Cuando el dispositivo informa que está descargando la primera actualización, la prueba la cancela forzosamente. Se espera que el dispositivo aborte la actualización actual, recoja la segunda actualización y la complete.
`OTAE2EImageCrashed`	Esta prueba comprueba que el dispositivo es capaz de rechazar una actualización cuando la imagen se bloquea.

Pruebas PAL

Requisitos previos

Para realizar la portabilidad de las pruebas de la interfaz de transporte de red, necesita lo siguiente:

- Un proyecto que pueda crear FreeRTOS con un puerto de kernel de FreeRTOS validado.
- Una implementación funcional de PAL OTA.

Portabilidad

- Añada FreeRTOS-Libraries-Integration-Tests como un submódulo al proyecto. La ubicación del submódulo en el proyecto debe ser donde se pueda crear.
- Copie `config_template/test_execution_config_template.h` y `config_template/test_param_config_template.h` en una ubicación en la ruta de creación y cámbieles el nombre a `test_execution_config.h` y `test_param_config.h`.

- Incluya los archivos relevantes en el sistema creación. Si utiliza `CMake`, `qualification_test.cmake` y `src/ota_pal_tests.cmake` se pueden usar para incluir los archivos relevantes.
- Configure la prueba implementando las siguientes funciones:
  - `SetupOtaPalTestParam()`: definido en `src/ota/ota_pal_test.h`. La implementación debe tener el mismo nombre y firma que se definen en `ota_pal_test.h`. Actualmente, no es necesario configurar esta función.
- Implemente UNITY_OUTPUT_CHAR para que los registros de salida de la prueba no se intercalen con los registros del dispositivo.
- Llame a `RunQualificationTest()` desde la aplicación. El hardware del dispositivo debe estar correctamente inicializado y la red debe estar conectada antes de la llamada.

Testeo

En esta sección se describen las pruebas locales de calificación de PAL OTA.

Habilitación de la prueba

Abra `test_execution_config.h` y defina OTA_PAL_TEST_ENABLED en 1.

En `test_param_config.h`, actualice las siguientes opciones:

- OTA_PAL_TEST_CERT_TYPE: seleccione el tipo de certificado utilizado.
- OTA_PAL_CERTIFICATE_FILE: ruta al certificado del dispositivo, si corresponde.
- OTA_PAL_FIRMWARE_FILE: nombre del archivo de firmware, si corresponde.
- OTA_PAL_USE_FILE_SYSTEM: se establece en 1 si PAL OTA utiliza la abstracción del sistema de archivos.

Cree e instale la aplicación con la cadena de herramientas que prefiera. Cuando se llame a `RunQualificationTest()`, se ejecutarán las pruebas PAL OTA. Los resultados de la prueba se envían al puerto serie.

Integración de las tareas OTA

- Añada un agente OTA a su demostración actual de MQTT.
- Realice las pruebas OTA de extremo a extremo (E2E) con. AWS IoT Esto verifica si la integración funciona según lo esperado.

> **Note**
>
> Para calificar oficialmente un dispositivo para FreeRTOS, debes validar el código fuente portado del dispositivo con los grupos de prueba OTA PAL y OTA E2E con. AWS IoT Device Tester Siga las instrucciones de Uso AWS IoT Device Tester para FreeRTOS de la Guía del usuario de FreeRTOS para configurar AWS IoT Device Tester la validación de puertos. Para probar el puerto de una biblioteca específica, debe estar habilitado el grupo de prueba correcto en el `device.json` archivo de la AWS IoT Device Tester `configs` carpeta.

Cargador de arranque del dispositivo con IoT

Debe proporcionar su propia aplicación de cargador de arranque seguro. Asegúrese de que el diseño y la implementación mitiguen adecuadamente las amenazas a la seguridad. A continuación se muestra el modelo de amenazas como referencia.

Modelado de amenazas para el cargador de arranque del dispositivo con IoT

Introducción

Como definición práctica, los AWS IoT dispositivos integrados a los que hace referencia este modelo de amenazas son productos basados en microcontroladores que interactúan con los servicios en la nube. Pueden implementarse en entornos industriales, comerciales o de consumo. Los dispositivos con IoT pueden recopilar datos acerca de un usuario, un paciente, una máquina o un entorno y pueden controlarlo todo, desde bombillas y cerraduras hasta maquinaria industrial.

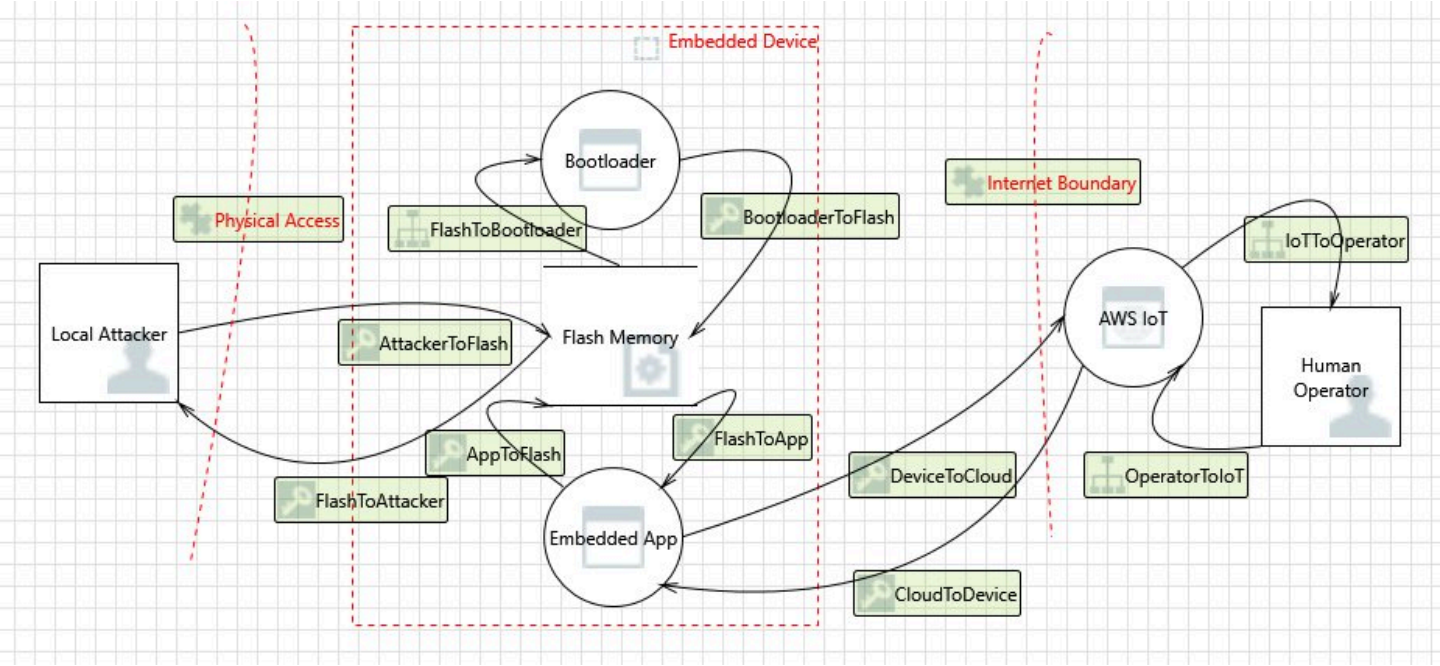
El modelado de amenazas es un enfoque de seguridad desde el punto de vista de un hipotético adversario. Al considerar los objetivos y métodos del adversario, se crea una lista de amenazas. Las amenazas son ataques contra un recurso o activo realizados por un adversario. La lista se prioriza y se utiliza para identificar y crear soluciones de mitigación. A la hora de elegir una solución de mitigación, el costo de implementarlas y mantenerlas debe equilibrarse con el valor de seguridad real que ofrecen. Existen varias metodologías de modelos de amenazas. Cada uno de ellos es capaz de respaldar el desarrollo de un producto seguro y exitoso. AWS IoT

Freertos ofrece actualizaciones de software OTA (over-the-air) para los AWS IoT dispositivos. La función actualizadora combina los servicios en la nube con las bibliotecas de software en el dispositivo y un cargador de arranque proporcionado por el socio. Este modelo de amenazas se centra específicamente en las amenazas contra el cargador de arranque.

Casos de uso del cargador de arranque

- Firmar y cifrar digitalmente el firmware antes de la implementación.
- Implementar nuevas imágenes de firmware en un único dispositivo, grupo de dispositivos o toda una flota.
- Verificar la autenticidad y la integridad del nuevo firmware una vez desplegado en los dispositivos.
- Los dispositivos solo ejecutan software sin modificar desde un origen de confianza.
- Los dispositivos son resistentes a errores en el software recibido mediante OTA.

Diagrama de flujo de datos



Amenazas

Algunos ataques tienen varios modelos de mitigación; por ejemplo, una red man-in-the-middle destinada a entregar una imagen de firmware maliciosa se mitiga verificando la confianza tanto en el certificado ofrecido por el servidor TLS como en el certificado del firmante de código de la nueva imagen de firmware. Para maximizar la seguridad del cargador de arranque, cualquier solución de mitigación no relacionada con el cargador de arranque se considera poco fiable. El cargador de arranque debe tener soluciones de mitigación intrínsecas para cada ataque. Disponer de soluciones de mitigación en capas se conoce como. defense-in-depth

Amenazas:

- Un atacante secuestra la conexión del dispositivo al servidor para entregar una imagen de firmware maliciosa.

  Ejemplo de mitigación

  - Al arrancar, el cargador de arranque verifica la firma criptográfica de la imagen mediante un certificado conocido. Si la verificación falla, el cargador de arranque restaura la imagen anterior.
- Un atacante aprovecha un desbordamiento del búfer para introducir un comportamiento malicioso en la imagen de firmware existente almacenada en flash.

  Ejemplos de mitigación

  - Al arrancar, el cargador de arranque verifica, tal y como se describió anteriormente. Cuando la verificación falla sin ninguna imagen anterior disponible, el cargador de arranque se detiene.
  - Al arrancar, el cargador de arranque verifica, tal y como se describió anteriormente. Cuando la verificación falla sin ninguna imagen anterior disponible, el cargador de arranque entra en modo solo OTA a prueba de errores.
- Un atacante arranca el dispositivo en una imagen almacenada anteriormente que es explotable.

  Ejemplos de mitigación

  - Los sectores flash que almacenan la última imagen se borran tras la instalación correcta y la prueba de una nueva imagen.
  - Con cada actualización correcta, se quema un fusible y cada imagen no se ejecuta a menos que se haya quemado el número correcto de fusibles.
- Una actualización OTA proporciona una imagen defectuosa o maliciosa que bloquea el dispositivo.

  Ejemplo de mitigación

  - El cargador de arranque inicia un temporizador de vigilancia de hardware que activa la reversión a la imagen anterior.
- Un atacante parchea el cargador de arranque para omitir la verificación de imágenes para que el dispositivo acepte imágenes sin firmar.

  Ejemplos de mitigación

  - El cargador de arranque está en ROM (memoria de solo lectura) y no se puede modificar.
  - El gestor de arranque está en OTP (one-time-programmable memoria) y no se puede modificar.

  - El gestor de arranque se encuentra en la zona segura de ARM TrustZone y no se puede modificar.
- Un atacante sustituye el certificado de verificación para que el dispositivo acepte imágenes maliciosas.

  Ejemplos de mitigación

  - El certificado se encuentra en un coprocesador criptográfico y no se puede modificar.
  - El certificado se encuentra en ROM (u OTP o zona segura) y no se puede modificar.

Modelado de amenazas adicionales

Este modelo de amenazas solo tiene en cuenta el cargador de arranque. Un modelado de amenazas adicionales podría mejorar la seguridad general. Un método recomendado consiste en enumerar los objetivos del adversario, los activos a los que se dirigen esos objetivos y los puntos de entrada a los activos. Se puede elaborar una lista de amenazas al tener en cuenta los ataques en los puntos de entrada para obtener control de los activos. A continuación se muestran listas de ejemplos de objetivos, activos y puntos de entrada para un dispositivo con IoT. Estas listas no son exhaustivas y están pensadas para impulsar la reflexión.

Objetivos del adversario

- Obtener dinero
- Arruinar reputaciones
- Falsificar datos
- Desviar recursos
- Espiar de forma remota a un objetivo
- Obtener acceso físico a un sitio
- Provocar el caos
- Infundir terror

Activos clave

- Claves privadas
- Certificado de cliente
- Certificados raíz de CA

- Tokens y credenciales de seguridad
- Información de identificación personal del cliente
- Implementaciones de secretos comerciales
- Datos del sensor
- Almacén de datos de análisis en la nube
- Infraestructura en la nube

Puntos de entrada

- Respuesta de DHCP
- Respuesta de DNS
- MQTT sobre TLS
- Respuesta HTTPS
- Imagen de software OTA
- Otros, según dicte la aplicación, por ejemplo, USB
- Acceso físico al bus
- Circuito integrado no limitado

Portabilidad de la biblioteca de interfaces móviles

FreeRTOS admite los comandos AT de una capa de abstracción móvil descargada de TCP. Para obtener más información, consulte la Biblioteca de interfaces móviles y Portabilidad de la biblioteca de interfaces móviles en freertos.org.

Requisitos previos

No existe una dependencia directa para la biblioteca de interfaces celulares. Sin embargo, en la pila de red FreeRTOS, Ethernet, Wi-Fi y móvil no pueden coexistir, por lo que los desarrolladores deben elegir una de ellas para integrarla con la Portabilidad de la interfaz de transporte de red.

> **Note**
>
> Si el módulo móvil admite la descarga de TLS o no admite los comandos AT, los desarrolladores pueden implementar su propia abstracción móvil para integrarla con la Portabilidad de la interfaz de transporte de red.

Migración de la versión 3 de MQTT a coreMQTT

En esta guía de migración se explica cómo migrar aplicaciones de MQTT a coreMQTT.

Migración de la versión 1 a la versión 3 para aplicaciones OTA

Esta guía le ayudará a migrar su aplicación de la versión 1 a la versión 3 de la biblioteca OTA.

> **Note**
>
> La versión 2 APIs de OTA es la misma que la versión 3 de OTA APIs, por lo que si su aplicación utiliza la versión 2 de la APIs , no es necesario realizar cambios en las llamadas a la API, pero le recomendamos que integre la versión 3 de la biblioteca.

Las demostraciones de la versión 3 de OTA están disponibles aquí:

- ota_demo_core_mqtt.
- ota_demo_core_http.
- ota_ble.

Resumen de cambios de API

Resumen de los cambios de API entre la versión 1 y la versión 3 de la biblioteca OTA

API de versión 1 de OTA	API de versión 3 de OTA	Descripción de los cambios
OTA_ AgentInit	OTA_Init	Los parámetros de entrada cambian, así como el valor devuelto por la función, debido a cambios en la implement ación de OTA v3. Consulte la sección correspondiente a OTA_init que aparece a continuación para obtener más información.
OTA_ AgentShutdown	OTA_Shutdown	Cambio en los parámetros de entrada, incluido un parámetro

API de versión 1 de OTA	API de versión 3 de OTA	Descripción de los cambios
		adicional para cancelar la suscripción opcional a los temas de MQTT. Consulte la sección correspondiente a OTA_Shutdown que aparece a continuación para obtener más información.
OTA_ GetAgentState	OTA_ GetState	El nombre de la API se cambia sin cambios en el parámetro de entrada. El valor devuelto es el mismo, pero se cambia el nombre de la enumeración y de los miembros. Consulte la sección correspondiente a OTA_ GetState que aparece a continuación para obtener más información.
n/a	OTA_ GetStatistics	Se agregó una nueva API que reemplaza a la APIs OTA_GetPacketsReceived, OTA_, GetPacketsQueued OTA_. GetPacketsProcessed GetPacketsDropped Consulte la sección relativa a la OTA_ que aparece a continuación para obtener más información. GetStatistics
OTA_ GetPacketsReceived	n/a	Esta API se ha eliminado de la versión 3 y se ha sustituido por OTA_. GetStatistics

API de versión 1 de OTA	API de versión 3 de OTA	Descripción de los cambios
OTA_ GetPacketsQueued	n/a	Esta API se ha eliminado de la versión 3 y se ha sustituido por OTA_. GetStatistics
OTA_ GetPacketsProcessed	n/a	Esta API se ha eliminado de la versión 3 y se ha sustituido por OTA_. GetStatistics
OTA_ GetPacketsDropped	n/a	Esta API se ha eliminado de la versión 3 y se ha sustituido por OTA_. GetStatistics
OTA_ ActivateNewImage	OTA_ ActivateNewImage	Los parámetros de entrada son los mismos, pero se cambia el nombre del código de error OTA devuelto y se añaden nuevos códigos de error en la versión 3 de la biblioteca OTA. Consulte la sección de ActivateNewImage OTA_ para obtener más información.
OTA_ SetImageState	OTA_ SetImageState	Los parámetros de entrada son los mismos y se les ha cambiado el nombre, pero se cambia el nombre del código de error OTA devuelto y se añaden nuevos códigos de error en la versión 3 de la biblioteca OTA. Consulte la sección de SetImageS tate OTA_ para obtener más información.

API de versión 1 de OTA	API de versión 3 de OTA	Descripción de los cambios
OTA_ GetImageState	OTA_ GetImageState	Los parámetros de entrada son los mismos, la enumeración de retorno cambia de nombre en la versión 3 de la biblioteca OTA. Consulte la sección de GetImageState OTA_ para obtener más información.
OTA_Suspend	OTA_Suspend	Los parámetros de entrada son los mismos, se cambia el nombre del código de error OTA devuelto y se añaden nuevos códigos de error en la versión 3 de la biblioteca OTA. Consulte la sección correspondiente a OTA_Suspend para obtener más información.
OTA_Resume	OTA_Resume	El parámetro de entrada para la conexión se elimina a medida que se gestiona la conexión en la demostración/aplicación OTA, se cambia el nombre del código de error OTA devuelto y se añaden nuevos códigos de error en la versión 3 de la biblioteca OTA. Consulte la sección correspondiente a OTA_Resume para obtener más información.

API de versión 1 de OTA	API de versión 3 de OTA	Descripción de los cambios
OTA_ CheckForUpdate	OTA_ CheckForUpdate	Los parámetros de entrada son los mismos, se cambia el nombre del código de error OTA devuelto y se añaden nuevos códigos de error en la versión 3 de la biblioteca OTA. Consulte la sección de CheckForUpdate OTA_ para obtener más información.
n/a	OTA_ EventProcessingTask	Se ha añadido una nueva API que es el bucle de eventos principal para gestionar los eventos de la actualiza ción OTA a la que debe llamar la tarea de la aplicació n. Consulte la sección de EventProcessingTask OTA_ para obtener más información.
n/a	OTA_ SignalEvent	Se ha añadido una nueva API que añade el evento al final de la cola de eventos OTA y que se utiliza en los módulos OTA internos para indicar la tarea del agente. Consulte la sección de SignalEvent OTA_ para obtener más información.
n/a	OTA_Err_strerror	Nueva API para la conversió n de código de error a cadena en caso de errores de OTA.

API de versión 1 de OTA	API de versión 3 de OTA	Descripción de los cambios
n/a	OTA_ _strerror JobParse	Nueva API para la conversió n de código de error a cadena en caso de errores de análisis de trabajos.
n/a	OTA_ OsStatus _strerror	Nueva API para la conversión de código de estado a cadena para obtener el estado del puerto OS OTA.
n/a	OTA_ PalStatus _strerror	Nueva API para la conversión de código de estado a cadena para obtener el estado del puerto PAL OTA.

Descripción de los cambios necesarios

OTA_Init

Al inicializar el agente OTA en la versión 1, se utiliza la API `OTA_AgentInit`, que toma como entrada los parámetros del contexto de la conexión, el nombre del objeto, la devolución de llamada completa y el tiempo de espera.

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

Esta API ahora ha cambiado a `OTA_Init` con los parámetros de los búferes necesarios para OTA, las interfaces OTA, el nombre del objeto y la devolución de llamada de la aplicación.

```
OtaErr_t OTA_Init( OtaAppBuffer_t * pOtaBuffer,
                   OtaInterfaces_t * pOtaInterfaces,
                   const uint8_t * pThingName,
                   OtaAppCallback OtaAppCallback );
```

Parámetros de entrada eliminados -

pvConnectionContext -

El contexto de conexión se elimina porque la versión 3 de la biblioteca OTA no requiere que se le pase el contexto de conexión a ella ni a la. MQTT/HTTP operations are handled by their respective interfaces in the OTA demo/application

xTicksToEspera -

El parámetro ticks to wait también se elimina cuando se crea la tarea en la demostración/aplicación OTA antes de llamar a OTA_init.

Parámetros de entrada con nombre cambiado -

xFunc -

Se cambia el nombre del parámetro a OtaAppCallback _t OtaAppCallback y su tipo.

Parámetros de entrada nuevos -

pOtaBuffer

La aplicación debe asignar los búferes y pasarlos a la biblioteca OTA mediante la estructura OtaAppBuffer _t durante la inicialización. Los búferes necesarios varían ligeramente según el protocolo utilizado para descargar el archivo. Para el protocolo MQTT, se requieren los búferes para el nombre de la secuencia y para el protocolo HTTP, los búferes para la URL prefirmada y el esquema de autorización.

Los búferes necesarios cuando se utiliza MQTT para la descarga de archivos -

```
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
};
```

Los búferes necesarios cuando se utiliza HTTP para la descarga de archivos -

```
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
};
```

Donde -

```
    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

El segundo parámetro de entrada de OTA_init es una referencia a las interfaces OTA del tipo _t. OtaInterfaces Este conjunto de interfaces debe transferirse a la biblioteca OTA e incluye en la interfaz del sistema operativo la interfaz MQTT, la interfaz HTTP y la interfaz de capa de abstracción de la plataforma.

Interfaz SO OTA

La interfaz funcional del sistema operativo OTA es un conjunto APIs que debe implementarse para que el dispositivo utilice la biblioteca OTA. Las implementaciones de funciones de esta interfaz se proporcionan a la biblioteca OTA de la aplicación de usuario. La biblioteca OTA llama a las implementaciones de funciones para realizar funcionalidades que normalmente proporciona un sistema operativo. Esto incluye la gestión de eventos, temporizadores y asignación de memoria. Las implementaciones de FreeRTOS y POSIX se proporcionan con la biblioteca OTA.

Ejemplo para FreeRTOS con el puerto FreeRTOS proporcionado -

```
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;
```

Ejemplo para Linux con el puerto POSIX proporcionado -

```
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;
```

Interfaz MQTT

La interfaz OTA MQTT es un conjunto APIs que debe implementarse en una biblioteca para permitir que la biblioteca OTA descargue un bloque de archivos del servicio de streaming.

Ejemplo de uso del agente CoreMQTT de la demostración OTA APIs sobre MQTT -

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

Interfaz HTTP

La interfaz HTTP OTA es un conjunto APIs que debe implementarse en una biblioteca para que la biblioteca OTA pueda descargar un bloque de archivos conectándose a una URL prefirmada y buscando bloques de datos. Es opcional, a menos que configure la biblioteca OTA para que descargue desde una URL prefirmada en lugar de hacerlo desde un servicio de streaming.

Ejemplo de uso del CoreHTTP APIs de la demostración OTA sobre HTTP:

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

Interfaz PAL OTA

La interfaz PAL OTA es un conjunto APIs que debe implementarse para que el dispositivo utilice la biblioteca OTA. La implementación específica del dispositivo para PAL OTA se proporciona a la biblioteca de la aplicación de usuario. La biblioteca utiliza estas funciones para almacenar, gestionar y autenticar las descargas.

```
  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;
```

Cambios en la devolución -

La devolución cambia del estado de agente OTA al código de error OTA. Consulte la AWS IoT Over-the-air actualización v3.0.0: OtaErr _t.

OTA_Shutdown

En la versión 1 de la biblioteca OTA, la API utilizada para cerrar el agente OTA era OTA_, AgentShutdown que ahora se ha cambiado a OTA_Shutdown junto con los cambios en los parámetros de entrada.

OTA Agent Shutdown (versión 1)

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

OTA Agent Shutdown (versión 3)

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

ticksToWait -

El número de tics que hay que esperar para que el agente OTA complete el proceso de cierre. Si se establece en cero, la función volverá inmediatamente sin esperar. El estado real se devuelve a la persona que llama. El agente no está inactivo durante este tiempo, sino que lo utiliza para el bucle ocupado.

Nuevo parámetro de entrada -

unsubscribeFlag -

Indicador para señalar si las operaciones de cancelación de suscripción deben realizarse desde los temas del trabajo cuando se llama al cierre. Si el indicador es 0, no se realizarán operaciones de cancelación de suscripción para los temas de trabajo. Si se debe cancelar la suscripción de la aplicación a los temas de trabajo, este indicador debe estar establecido en 1 al llamar a OTA_Shutdown.

Cambios en la devolución -

OtaState_t -

Se cambia el nombre de la enumeración del estado del agente OTA y sus miembros. Consulte la AWS IoT Over-the-air actualización v3.0.0.

OTA_ GetState

El nombre de la API se cambia de OTA_ a OTA_AgentGetState . GetState

OTA Agent Shutdown (versión 1)

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

OTA Agent Shutdown (versión 3)

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

Cambios en la devolución -

OtaState_t -

Se cambia el nombre de la enumeración del estado del agente OTA y sus miembros. Consulte la AWS IoT Over-the-air actualización v3.0.0.

OTA_ GetStatistics

Se ha añadido una nueva API única para las estadísticas. Sustituye a la APIs OTA_GetPacketsReceived, OTA_, GetPacketsQueued OTA_. GetPacketsProcessed GetPacketsDropped Además, en la versión 3 de la biblioteca OTA, los números de las estadísticas están relacionados únicamente con el trabajo actual.

Versión 1 de la biblioteca OTA

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

Versión 3 de la biblioteca OTA

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

pStatistics -

Parámetro de entrada/salida para datos estadísticos como paquetes recibidos, descartados, puestos en cola y procesados para el trabajo actual.

Parámetro de salida -

Código de error de OTA.

Ejemplo de uso -

```
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

Los parámetros de entrada son los mismos, pero se cambia el nombre del código de error OTA devuelto y se añaden nuevos códigos de error en la versión 3 de la biblioteca OTA.

Versión 1 de la biblioteca OTA

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

Versión 3 de la biblioteca OTA

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

Se cambia la enumeración del código de error OTA devuelto y se añaden nuevos códigos de error. Consulte la AWS IoT Over-the-air actualización v3.0.0: _t. OtaErr

Ejemplo de uso -

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

OTA_ SetImageState

Los parámetros de entrada son los mismos y se les ha cambiado el nombre, pero se cambia el nombre del código de error OTA devuelto y se añaden nuevos códigos de error en la versión 3 de la biblioteca OTA.

Versión 1 de la biblioteca OTA

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

Versión 3 de la biblioteca OTA

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

El nombre del parámetro de entrada pasa a OtaImageState ser _t. Consulte la AWS IoT Over-the-air actualización v3.0.0.

Se cambia la enumeración del código de error OTA devuelto y se añaden nuevos códigos de error. Consulte la AWS IoT Over-the-air actualización v3.0.0/_t. OtaErr

Ejemplo de uso -

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

OTA_ GetImageState

Los parámetros de entrada son los mismos, se cambia el nombre de la enumeración de retorno en la versión 3 de la biblioteca OTA.

Versión 1 de la biblioteca OTA

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

Versión 3 de la biblioteca OTA

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

El nombre de la enumeración de retorno pasa a ser _t. OtaImageState Consulte la AWS IoT Over-the-air actualización v3.0.0: _t. OtaImageState

Ejemplo de uso -

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

OTA_Suspend

Los parámetros de entrada son los mismos, se cambia el nombre del código de error OTA devuelto y se añaden nuevos códigos de error en la versión 3 de la biblioteca OTA.

Versión 1 de la biblioteca OTA

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

Versión 3 de la biblioteca OTA

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

Se cambia la enumeración del código de error OTA devuelto y se añaden nuevos códigos de error. Consulte la AWS IoT Over-the-air actualización v3.0.0: _t. OtaErr

Ejemplo de uso -

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

OTA_Resume

El parámetro de entrada para la conexión se elimina a medida que se gestiona la conexión en la demo/aplicación OTA, se cambia el nombre del código de error OTA devuelto y se añaden nuevos códigos de error en la versión 3 de la biblioteca OTA.

Versión 1 de la biblioteca OTA

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

Versión 3 de la biblioteca OTA

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

Se cambia la enumeración del código de error OTA devuelto y se añaden nuevos códigos de error. Consulte la AWS IoT Over-the-air actualización v3.0.0: _t. OtaErr

Ejemplo de uso -

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

OTA_ CheckForUpdate

Los parámetros de entrada son los mismos, se cambia el nombre del código de error OTA devuelto y se añaden nuevos códigos de error en la versión 3 de la biblioteca OTA.

Versión 1 de la biblioteca OTA

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

Versión 3 de la biblioteca OTA

```
OtaErr_t OTA_CheckForUpdate( void )
```

Se cambia la enumeración del código de error OTA devuelto y se añaden nuevos códigos de error. Consulte la AWS IoT Over-the-air actualización v3.0.0: _t. OtaErr

OTA_ EventProcessingTask

Se trata de una API nueva que es el bucle de eventos principal para gestionar los eventos de las actualizaciones OTA. Debe llamarla la tarea de la aplicación. Este bucle seguirá gestionando y ejecutando los eventos recibidos para la actualización OTA hasta que la aplicación finalice esta tarea.

Versión 3 de la biblioteca OTA

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

Ejemplo para FreeRTOS -

```
/* 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 );
}
```

Ejemplo para POSIX -

```
/* 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

Se trata de una nueva API que añade el evento al final de la cola de eventos y que se utiliza en los módulos OTA internos para indicar la tarea del agente.

Versión 3 de la biblioteca OTA

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

Ejemplo de uso -

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

Integración de la biblioteca OTA como un submódulo en su aplicación

Si desea integrar la biblioteca OTA en su propia aplicación, puede usar el comando del submódulo de Git. Los submódulos de Git le permiten mantener un repositorio de Git como un subdirectorio de otro repositorio de Git. La versión 3 de la biblioteca OTA se mantiene en el repositorio ota-for-aws-iot-embedded-sdk.

```
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
```

Para obtener más información, consulte Integración del agente OTA en su aplicación en la Guía del usuario de FreeRTOS.

Referencias

- OTAv1.
- OTAv3.

Migración de la versión 1 a la versión 3 para el puerto PAL OTA

La biblioteca de Over-the-air actualizaciones introdujo algunos cambios en la estructura de carpetas y en la ubicación de las configuraciones requeridas por la biblioteca y las aplicaciones de demostración. En el caso de las aplicaciones OTA diseñadas para funcionar con la versión 1.2.0 para migrar a la versión 3.0.0 de la biblioteca, debe actualizar las firmas de las funciones del puerto PAL e incluir archivos de configuración adicionales, tal y como se describe en esta guía de migración.

Modificaciones en PAL OTA

- El nombre del directorio de puertos PAL OTA se ha actualizado de `ota` a `ota_pal_for_aws`. Esta carpeta debe contener 2 archivos: `ota_pal.c` y `ota_pal.h`. El archivo de cabecera PAL `libraries/freertos_plus/aws/ota/src/aws_iot_ota_pal.h` se ha eliminado de la biblioteca OTA y debe definirse dentro del puerto.
- Los códigos de devolución (`OTA_Err_t`) se traducen en una enumeración `OTAMainStatus_t`. Consulte ota_platform_interface.h para ver los códigos de devolución traducidos. También se proporcionan macros auxiliares para combinar los códigos `OtaPalMainStatus` y `OtaPalSubStatus`, y extraer `OtaMainStatus` de `OtaPalStatus` y similares.
- Inicio de sesión en PAL
  - Se ha eliminado la macro `DEFINE_OTA_METHOD_NAME`.
  - Anterior: `OTA_LOG_L1( "[%s] Receive file created.\r\n", OTA_METHOD_NAME );`.
  - Actualizada: `LogInfo( "Receive file created."));` utiliceLogDebug, `LogWarn` y `LogError` para el registro correspondiente.
- Variable `cOTA_JSON_FileSignatureKey` cambiada a `OTA_JsonFileSignatureKey`.

Funciones

Las firmas de las funciones se definen en `ota_pal.h` y comienzan por el prefijo `otaPal` en lugar de `prvPAL`.

> **Note**
>
> El nombre exacto del PAL es técnicamente abierto, pero para que sea compatible con las pruebas de calificación, el nombre debe ajustarse a los que se especifican a continuación.

- Versión 1: `OTA_Err_t prvPAL_CreateFileForRx( OTA_FileContext_t * const *C* );`

  Versión 3: `OtaPalStatus_t otaPal_CreateFileForRx( OtaFileContext_t * const *pFileContext* );`

  Notas: Cree un nuevo archivo de recepción para los fragmentos de datos a medida que van llegando.

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

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

  Escribe un bloque de datos en el archivo especificado en el desplazamiento dado.

- Versión 1: `OTA_Err_t prvPAL_ActivateNewImage( void );`

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

  Notas: Active la imagen de MCU más reciente recibida a través de OTA.

- Versión 1: `OTA_Err_t prvPAL_ResetDevice( void );`

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

  Notas: Restablece el dispositivo.

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

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

  Notas: Autentica y cierra el archivo de recepción subyacente en el contexto OTA especificado.

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

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

  Notas: Detiene una transferencia OTA.

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

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

  Notas: Intenta establecer el estado de la imagen de actualización de la OTA.

- Versión 1: `OTA_PAL_ImageState_t prvPAL_GetPlatformImageState( void );`

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

  Notas: Obtiene el estado de la imagen de actualización OTA.

Data Types

- Versión 1: `OTA_PAL_ImageState_t`

  Archivo: `aws_iot_ota_agent.h`

  Versión 3: `OtaPalImageState_t`

  Archivo: `ota_private.h`

  Notas: El estado de la imagen establecido por la implementación de la plataforma.

- Versión 1: `OTA_Err_t`

  Archivo: `aws_iot_ota_agent.h`

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

  Archivo: `ota.h`, `ota_platform_interface.h`

Notas: v1: eran macros que definían un entero de 32 sin signo. v3: enumeración especializada que representa el tipo de error y está asociada a un código de error.

- Versión 1: `OTA_FileContext_t`

  Archivo: `aws_iot_ota_agent.h`

  Versión 3: `OtaFileContext_t`

  Archivo: `ota_private.h`

  Notas: v1: contiene una enumeración y búferes para los datos. v3: contiene variables de longitud de datos adicionales.
- Versión 1: `OTA_ImageState_t`

  Archivo: `aws_iot_ota_agent.h`

  Versión 3: `OtaImageState_t`

  Archivo: `ota_private.h`

  Notas: Estados de la imagen OTA

Cambios de configuración

Se cambió el nombre del archivo `aws_ota_agent_config.h` a `ota_config.h`, lo que cambia las protecciones de inclusión de `_AWS_OTA_AGENT_CONFIG_H_` a `OTA_CONFIG_H_`.

- El archivo `aws_ota_codesigner_certificate.h` se ha eliminado.
- Se ha incluido la nueva pila de registro para imprimir los mensajes de depuración:

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

- Se ha añadido la configuración de constantes:

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

Archivo nuevo: ota_demo_config.h contiene las configuraciones que requiere la demostración de OTA, como el certificado de firma de código y la versión de la aplicación.

- `signingcredentialSIGNING_CERTIFICATE_PEM`, que se definió en `demos/include/aws_ota_codesigner_certificate.h`, se ha trasladado a `ota_demo_config.h` como `otapalconfigCODE_SIGNING_CERTIFICATE` y se puede acceder a él desde los archivos PAL de la siguiente manera:

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

  El archivo `aws_ota_codesigner_certificate.h` se ha eliminado.

- Las macros `APP_VERSION_BUILD`, `APP_VERSION_MINOR` y `APP_VERSION_MAJOR` se han agregado a `ota_demo_config.h`. Se han eliminado los archivos antiguos que contenían la información de la versión, por ejemplo`tests/include/aws_application_version.h`, `libraries/c_sdk/standard/common/include/iot_appversion32.h`, `demos/demo_runner/aws_demo_version.c`.

Cambios en las pruebas de PAL OTA

- Se ha eliminado el grupo de pruebas “Full_OTA_Agent” junto con todos los archivos relacionados. Anteriormente, este grupo de pruebas era obligatorio para la calificación. Estas pruebas eran para la biblioteca OTA y no específicas para el puerto PAL OTA. La biblioteca OTA ahora tiene una cobertura de pruebas completa alojada en el repositorio OTA, por lo que este grupo de pruebas ya no es necesario.
- Se han eliminado los grupos de pruebas “Full_OTA_CBOR” y “Quarantine_OTA_CBOR”, así como todos los archivos relacionados. Estas pruebas no formaban parte de las pruebas de calificación. Las funcionalidades que cubrían estas pruebas se están probando ahora en el repositorio OTA.
- Se han trasladado los archivos de prueba del directorio de la biblioteca al directorio `tests/integration_tests/ota_pal`.
- Se han actualizado las pruebas de calificación PAL OTA para utilizar la versión 3.0.0 de la API de la biblioteca OTA.
- Se ha actualizado la forma en que las pruebas PAL OTA acceden al certificado de firma de código para las pruebas. Anteriormente, había un archivo de encabezado dedicado a la credencial de firma de código. Este ya no es el caso de la nueva versión de la biblioteca. El código de prueba espera que esta variable esté definida en `ota_pal.c`. El valor se asigna a una macro que se define en el archivo de configuración OTA específico de la plataforma.

Lista de comprobación

Utilice esta lista de verificación para asegurarse de seguir los pasos necesarios para la migración:

- Actualice el nombre de la carpeta del puerto PAL OTA de `ota` a `ota_pal_for_aws`.
- Añada el archivo `ota_pal.h` con las funciones mencionadas anteriormente. Para ver un `ota_pal.h` archivo de ejemplo, consulte GitHub.
- Añada los archivos de configuración:
  - Cambie el nombre del archivo de `aws_ota_agent_config.h` a (o cree) `ota_config.h`.
    - Añada:

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

    - Incluya:

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

  - Copie los archivos anteriores a la carpeta `aws_test config` y sustituya los que se incluyen en `ota_demo_config.h` por `aws_test_ota_config.h`.
  - Añada un archivo `ota_demo_config.h`.
  - Añada un archivo `aws_test_ota_config.h`.
- Realice los siguientes cambios en `ota_pal.c`:
  - Actualice las inclusiones con los nombres de archivo de la biblioteca OTA más recientes.
  - Elimine la macro `DEFINE_OTA_METHOD_NAME`.
  - Actualice las firmas de las funciones PAL OTA.
  - Actualice el nombre de la variable de contexto del archivo de `C` a `pFileContext`.
  - Actualice la estructura `OTA_FileContext_t` y todas las variables relacionadas.
  - Actualice `cOTA_JSON_FileSignatureKey` a `OTA_JsonFileSignatureKey`.
  - Actualice los tipos `OTA_PAL_ImageState_t` y `Ota_ImageState_t`.
  - Actualice el tipo y los valores de error.
  - Actualice las macros de impresión para usar la pila de registro.
  - Actualice `signingcredentialSIGNING_CERTIFICATE_PEM` para que sea `otapalconfigCODE_SIGNING_CERTIFICATE`.
  - Actualice los comentarios de las funciones `otaPal_CheckFileSignature` y `otaPal_ReadAndAssumeCertificate`.
- Actualice el archivo `CMakeLists.txt`.
- Actualice los proyectos del IDE.

Historial del documento

En la siguiente tabla se describe el historial de la documentación de la Guía de portabilidad de FreeRTOS y de la Guía de calificación de FreeRTOS.

Fecha	Versiones de la documentación	Historial de cambios	Versión de FreeRTOS
Mayo de 2022	Guía de portabilidad de FreeRTOS Guía de calificación de FreeRTOS	• Se han actualizado las pruebas existentes, se han añadido nuevas pruebas y se han eliminado las pruebas redundantes basadas en las bibliotecas Long Term Support (LTS) de FreeRTOS. Para obtener más información, consulte Pruebas de integración de bibliotecas Freertos a partir de 202205.00. GitHub • Actualizado Diagrama de flujo de portabilidad de FreeRTOS. • Se ha añadido una nueva Portabilidad de la interfaz de transporte de red.	202012.04-LTS 202112.00

Fecha	Versiones de la documentación	Historial de cambios	Versión de FreeRTOS
		• Portabilidad de la biblioteca de actualizaciones AWS IoT over-the-air (OTA) ahora se requiere para la calificación. • Se ha eliminado la guía de portabilidad de abstracción de Wi-Fi y TLS, ya que ya no es necesaria. • Consulte los cambios más recientes para obtener más información sobre la calificación de FreeRTOS.	

Fecha	Versiones de la documentación	Historial de cambios	Versión de FreeRTOS
Julio de 2021	202107.00 (Guía de portabilidad) 202107.00 (Guía de calificación)	• Versión 202107.00 • Se ha cambiado Portabilidad de la biblioteca de actualizaciones AWS IoT over-the-air (OTA) • Se ha agregado Migración de la versión 1 a la versión 3 para aplicaciones OTA • Se ha agregado Migración de la versión 1 a la versión 3 para el puerto PAL OTA	202107,00
Diciembre de 2020	202012.00 (Guía de portabilidad) 202012.00 (Guía de calificación)	• Versión 202012.00 • Se ha agregado Configuración de la biblioteca coreHTTP • Se ha agregado Portabilidad de la biblioteca de interfaces móviles	202012,00

Fecha	Versiones de la documentación	Historial de cambios	Versión de FreeRTOS
Noviembre de 2020	202011.00 (Guía de portabilidad) 202011.00 (Guía de cualificación)	• Versión 202011.00 • Se ha agregado Configuración de la biblioteca coreMQTT	202011,00
Julio de 2020	202007.00 (Guía de portabilidad) 202007.00 (Guía de calificación)	• Versión 202007.00	202007,00
18 de febrero de 2020	202002.00 (Guía de portabilidad) 202002.00 (Guía de cualificación)	• Versión 202002.00 • Amazon FreeRTOS ahora es FreeRTOS	202002.00
17 de diciembre de 2019	201912.00 (Guía de portabilidad) 201912.00 (Guía de cualificación)	• Versión 201912.00 • Se ha añadido la portabilidad de las bibliotecas de E/S comunes.	201912.00
29 de octubre de 2019	201910.00 (Guía de portabilidad) 201910.00 (Guía de cualificación)	• Versión 201910.00 • Se ha actualiza do la información de portabilidad del generador de números aleatorios.	201910.00

Fecha	Versiones de la documentación	Historial de cambios	Versión de FreeRTOS
26 de agosto de 2019	201908.00 (Guía de portabilidad) 201908.00 (Guía de cualificación)	• Versión 201908.00 • Se ha añadido Configuración de la biblioteca de cliente HTTPS para pruebas Se ha actualizado Migración de la PKCS11 biblioteca principal	201908.00
17 de junio de 2019	201906.00 (Guía de portabilidad) 201906.00 (Guía de calificación)	• Versión 201906.00 • Estructura de directorios actualizada	201906.00 Principal
21 de mayo de 2019	1.4.8 (Guía de portabilidad) 1.4.8 (Guía de cualificación)	• Documentación de portabilidad trasladada a la Guía de portabilidad de FreeRTOS • Documentación de calificación trasladada a la Guía de calificación de FreeRTOS	1.4.8

Fecha	Versiones de la documentación	Historial de cambios	Versión de FreeRTOS
25 de febrero de 2019	1.1.6	• Eliminadas las instrucciones de descarga y configuración del apéndice de plantillas de la Guía de introducción (página 84)	1.4.5 1.4.6 1.4.7
27 de diciembre de 2018	1.1.5	• Apéndice actualizado de la lista de verificación para la calificación con CMake requisitos (página 70)	1.4.5 1.4.6
12 de diciembre de 2018	1.1.4	• Añadidas instrucciones de portabilidad lwIP al apéndice sobre portabilidad TCP/IP (página 31)	1.4.5

Fecha	Versiones de la documentación	Historial de cambios	Versión de FreeRTOS
26 de noviembre de 2018	1.1.3	• Se ha añadido el apéndice de portabilidad de Bluetooth de bajo consumo (página 52) • Se agregó información AWS IoT sobre las pruebas de Device Tester for FreeRTOS en todo el documento • Se agregó CMake un enlace a la información para incluirlo en el apéndice de la consola FreeRTOS (página 85)	1.4.4

Fecha	Versiones de la documentación	Historial de cambios	Versión de FreeRTOS
7 de noviembre de 2018	1.1.2	• Actualizadas las instrucciones de portabilidad de la interfaz PAL PKCS # 11 en el apéndice de portabilidad de PKCS # 11 (página 38) • Actualizada la ruta a `CertificateConfigurator.html` (página 76) • Actualizado el apéndice de plantillas de la Guía de introducción (página 80)	1.4.3

Fecha	Versiones de la documentación	Historial de cambios	Versión de FreeRTOS
8 de octubre de 2018	1.1.1	• Añadida una nueva columna "Obligatorio para AFQP" a la tabla de configuración de prueba `aws_test_runner_config.h` (página 16) • Actualizada la ruta del directorio del módulo Unity en la sección Creación del proyecto de prueba (página 14) • Actualizada la sección «Orden de portabilidad recomendado» (página 22) • Actualizado el certificado de cliente y los nombres de variables clave en el apéndice de TLS apéndice, Configuración de la prueba (página 40) • Cambio en las rutas de archivo del apéndice de portabilidad de	1.4.2

Fecha	Versiones de la documentación	Historial de cambios	Versión de FreeRTOS
		sockets seguros, Configuración de prueba (página 34); apéndice de portabilidad de TLS, Configuración de prueba (página 40) y el apéndice Configuración del servidor TLS (página 57)	
27 de agosto de 2018	1.1.0	• Añadido apéndice de portabilidad de actualizaciones OTA (página 47) • Añadido apéndice de portabilidad de bootloader (página 51)	1.4.0 1.4.1

Fecha	Versiones de la documentación	Historial de cambios	Versión de FreeRTOS
9 de agosto de 2018	1.0.1	• Actualizada la sección «Orden de portabilidad recomenda do» (página 22) • Actualizado el apéndice de portabilidad de PKCS # 11 (página 36) • Cambio en las rutas de archivo del apéndice de portabilidad TLS, Configuración de prueba (página 40) y el apéndice Configuración del servidor TLS, paso 9 (página 51) • Corregidos hipervínculos en el apéndice de portabilidad MQTT, Requisitos previos (página 45) • Se agregaron instrucciones de AWS CLI configura ción a los ejemplos del apéndice Instrucciones para	1.3.1 1.3.2

Fecha	Versiones de la documentación	Historial de cambios	Versión de FreeRTOS
		crear un BYOC (página 57)	
31 de julio de 2018	1.0.0	Versión inicial de la Guía del programa de calificación de FreeRTOS	1.3.0

Las traducciones son generadas a través de traducción automática. En caso de conflicto entre la traducción y la version original de inglés, prevalecerá la version en inglés.