aws

Guia de portabilidade

FreeRTOS

FreeRTOS: Guia de portabilidade

Table of Contents

Portabilidade do FreeRTOS

O que é o FreeRTOS?

Desenvolvido em parceria com as principais empresas de chips do mundo ao longo de um período de 20 anos, e agora baixado a cada 170 segundos, o FreeRTOS é um sistema operacional em tempo real (RTOS) líder de mercado para microcontroladores e microprocessadores pequenos. Distribuído livremente sob a licença de código aberto do MIT, o FreeRTOS inclui um kernel e um conjunto crescente de bibliotecas adequadas para uso em todos os setores. O FreeRTOS foi desenvolvido com ênfase na confiabilidade e facilidade de uso. O FreeRTOS inclui bibliotecas para conectividade, segurança e atualizações sem fios, e aplicações de demonstração que mostram recursos do FreeRTOS em placas qualificadas.

Para obter mais informações, visite FreeRTOS.org.

Portabilidade do FreeRTOS para sua placa do IoT

Será necessário fazer a portabilidade das bibliotecas de software do FreeRTOS para sua placa baseada em microcontrolador com base em seus recursos e em sua aplicação.

Como fazer a portabilidade do FreeRTOS para seu dispositivo

1. Siga as instruções em Fazer download do FreeRTOS para portabilidade para fazer download da versão mais recente do FreeRTOS para portabilidade.
2. Siga as instruções em Configuração do seu espaço de trabalho e projeto para portabilidade para configurar os arquivos e as pastas no download do FreeRTOS para portabilidade e teste.
3. Siga as instruções em Portabilidade da bibliotecas do FreeRTOS para fazer a portabilidade das bibliotecas do FreeRTOS para o dispositivo. Cada tópico inclui instruções sobre como testar a portabilidade.

Perguntas frequentes sobre portabilidade

O que é uma porta do FreeRTOS?

> Uma porta do FreeRTOS é uma implementação específica da placa de APIs para as bibliotecas necessárias do FreeRTOS e o kernel do FreeRTOS ao qual sua plataforma oferece suporte. A

porta permite que as APIs funcionem na placa e implementa a integração necessária com os drivers de dispositivo e BSPs fornecidos pelo fornecedor da plataforma. A porta também deve incluir quaisquer ajustes de configuração (por exemplo, velocidade de clock, tamanho da pilha, tamanho do heap) exigidos pela placa.

Se você tiver dúvidas sobre a portabilidade que não foram respondidas nesta página ou no restante do Guia de Portabilidade do FreeRTOS, consulte as opções de suporte do FreeRTOS disponíveis.

Fazer download do FreeRTOS para portabilidade

Baixe a versão mais recente do FreeRTOS ou do Long Term Support (LTS) em freertos.org ou clone do GitHub (FreeRTOS-LTS) ou (FreeRTOS).

> **Note**
>
> Recomendamos que você clone o repositório. A clonagem facilita o recebimento de atualizações para a ramificação principal à medida que elas são enviadas ao repositório.

Como alternativa, transforme as bibliotecas individuais em submódulos do repositório FreeRTOS ou FreeRTOS-LTS. No entanto, verifique se as versões da biblioteca correspondem à combinação listada no arquivo `manifest.yml` no repositório FreeRTOS ou FreeRTOS-LTS.

Depois de baixar ou clonar o FreeRTOS, você pode começar a fazer a portabilidade das bibliotecas do FreeRTOS para sua placa. Para obter instruções, consulte Configuração do seu espaço de trabalho e projeto para portabilidade e Portabilidade da bibliotecas do FreeRTOS.

Configuração do seu espaço de trabalho e projeto para portabilidade

Siga as etapas abaixo para configurar o espaço de trabalho e o projeto do:

- Use uma estrutura de projeto e crie um sistema que preferir para importar as bibliotecas do FreeRTOS.
- Crie um projeto usando um ambiente de desenvolvimento integrado (IDE) e uma cadeia de ferramentas compatíveis com sua placa.
- Inclua os pacotes de suporte da placa (BSP) e os drivers específicos da placa em seu projeto.

Depois que o espaço de trabalho estiver configurado, será possível começar a fazer a portabilidade de bibliotecas individuais do FreeRTOS.

Portabilidade da bibliotecas do FreeRTOS

Antes de iniciar a portabilidade, siga as instruções em Configuração do seu espaço de trabalho e projeto para portabilidade.

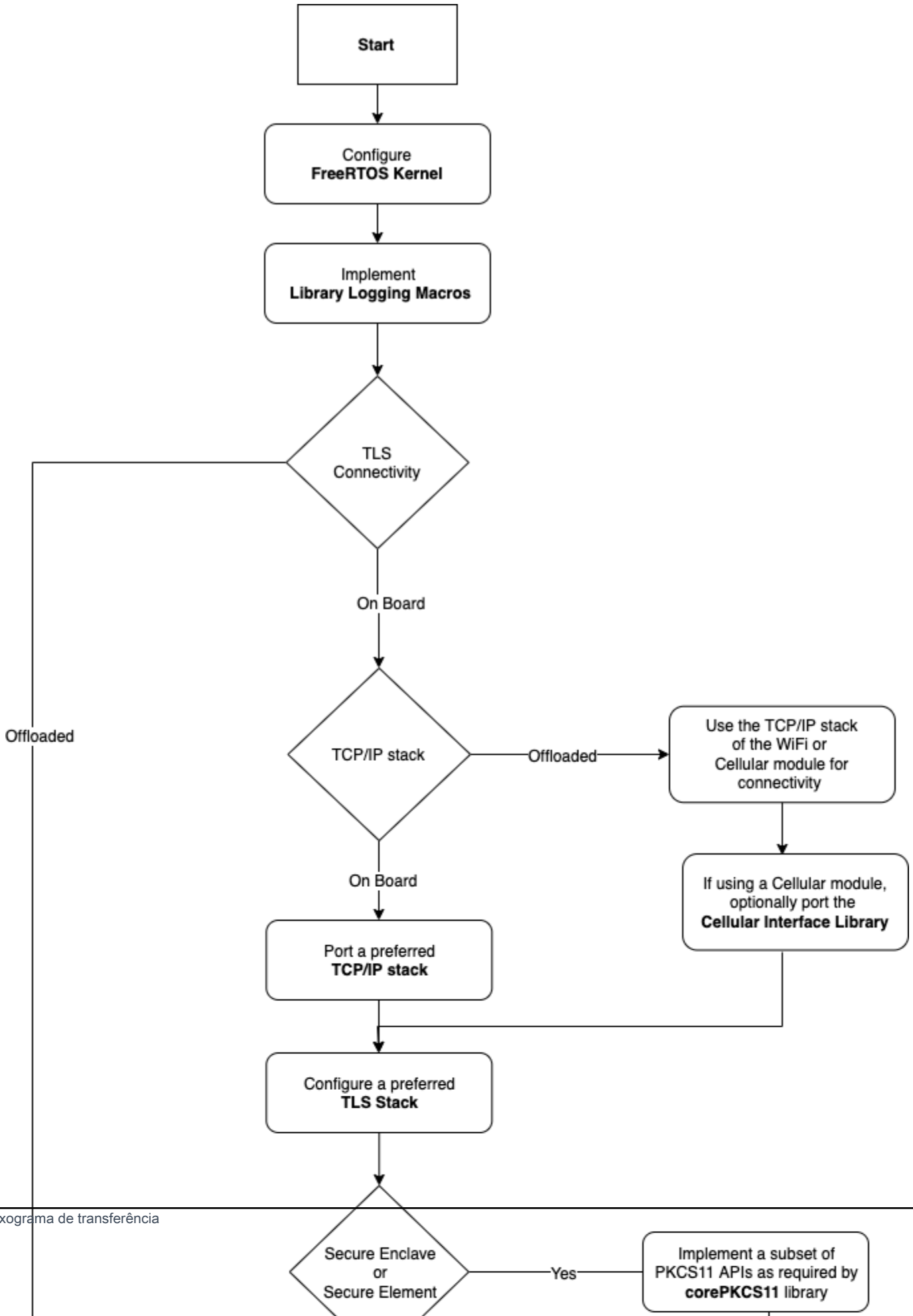
A Fluxograma de portabilidade do FreeRTOS descreve as bibliotecas necessárias para a portabilidade.

Para fazer a portabilidade do FreeRTOS para o dispositivo, siga as instruções nos tópicos a seguir.

1. Configuração de uma porta do kernel do FreeRTOS
2. Implementando as macros de registro em log da biblioteca
3. Transferência de pilha de TCP/IP
4. Fazer portabilidade da Network Transport Interface
5. Portabilidade da biblioteca corePKCS11
6. Configuração da biblioteca coreMQTT
7. Configuração da biblioteca coreHTTP
8. Portando a biblioteca de atualização AWS IoT over-the-air (OTA)
9. Portabilidade da biblioteca Cellular Interface

Fluxograma de portabilidade do FreeRTOS

Use o fluxograma de portabilidade abaixo como auxílio visual ao fazer a portabilidade do FreeRTOS para sua 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

Configuração de uma porta do kernel do FreeRTOS

Esta seção fornece instruções para integrar uma porta do kernel do FreeRTOS em um projeto de teste de porta do FreeRTOS. Para obter uma lista de portas disponíveis do kernel, consulte Portas do kernel do FreeRTOS.

O FreeRTOS usa o kernel do FreeRTOS para várias tarefas e comunicação entre tarefas. Para obter mais informações, consulte Princípios básicos do kernel do FreeRTOS no Guia do usuário do FreeRTOS e em FreeRTOS.org.

> **Note**
>
> A portabilidade do kernel do FreeRTOS para uma nova arquitetura não está incluída nessa documentação. Se você estiver interessado, entre em contato com a equipe de engenharia do FreeRTOS.
> Para o Programa de qualificação do FreeRTOS, apenas as portas existentes são compatíveis. Modificações nessas portas não são aceitas no programa. Revise a política de portas do kernel do FreeRTOS para obter mais informações.

Pré-requisitos

Para configurar o kernel do FreeRTOS para portabilidade, é necessário o seguinte:

- Uma porta oficial do kernel do FreeRTOS ou portas com suporte para a plataforma de destino.
- Um projeto do IDE que inclui os arquivos de porta corretos do kernel do FreeRTOS para a plataforma e o compilador de destino. Para obter informações sobre a configuração de um projeto de teste, consulte Configuração do seu espaço de trabalho e projeto para portabilidade.

Configuração do kernel do FreeRTOS

O kernel do FreeRTOS é personalizado usando um arquivo de configuração chamado `FreeRTOSConfig.h`. Esse arquivo de cabeçalho especifica as definições de configuração específicas da aplicação para o kernel. Para obter uma descrição de cada opção de configuração, consulte Personalização em FreeRTOS.org.

Para configurar o kernel do FreeRTOS para funcionar com seu dispositivo, inclua `FreeRTOSConfig.h` e modifique todas as configurações adicionais do FreeRTOS.

Para obter uma descrição de cada opção de configuração, consulte configurações de personalização em FreeRTOS.org.

Testar

- Execute uma tarefa simples do FreeRTOS para registrar uma mensagem no console de saída serial.
- Verifique se a mensagem é enviada para o console conforme o esperado.

Implementando as macros de registro em log da biblioteca

As bibliotecas do FreeRTOS usam as seguintes macros de registro em log, listadas em ordem crescente de detalhamento.

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

Uma definição para todas as macros deve ser fornecida. As recomendações são:

- As macros devem oferecer suporte ao registro em log de estilo `C89`.
- O registro em log deve ser seguro para thread. As linhas de log de várias tarefas não devem se intercalar umas com as outras.
- O registro em log não APIs deve ser bloqueado e deve liberar o bloqueio de tarefas do aplicativo na E/S.

Consulte a funcionalidade de registro em log em FreeRTOS.org para obter detalhes específicos de implementação. Você pode ver uma implementação neste exemplo.

Teste

- Execute um teste com várias tarefas para verificar se os registros não se intercalam.
- Execute um teste para verificar se o registro APIs não está bloqueado na E/S.
- Teste macros de registro com vários padrões, como registro em log de estilo `C89,C99`.

- Teste as macros de registro em log definindo diferentes níveis de logs, como `Debug`, `Info`, `Error` e `Warning`.

Transferência de pilha de TCP/IP

Esta seção fornece instruções para fazer a portabilidade e testar pilhas TCP/IP integradas. Se a plataforma descarregar a funcionalidade TCP/IP e TLS em um processador ou módulo de rede separado, você poderá ignorar essa seção de portabilidade e visitar Fazer portabilidade da Network Transport Interface.

O FreeRTOS+TCP é uma pilha TCP/IP nativa para o kernel do FreeRTOS. O FreeRTOS+TCP é desenvolvido e mantido pela equipe de engenharia do FreeRTOS, e a pilha TCP/IP é recomendada para usar com o FreeRTOS. Para obter mais informações, consulte Portabilidade do FreeRTOS +TCP. Como alternativa, você pode usar a pilha TCP/IP de terceiros lwIP. A instrução de teste fornecida nesta seção usa os testes da interface de transporte para texto simples TCP e não depende da pilha TCP/IP específica implementada.

Portabilidade do FreeRTOS+TCP

O FreeRTOS+TCP é uma pilha TCP/IP nativa para o kernel do FreeRTOS. Para obter mais informações, consulte FreeRTOS.org.

Pré-requisitos

Para fazer a portabilidade da biblioteca do FreeRTOS+TCP, é necessário o seguinte:

- Um projeto do IDE que inclui os drivers Ethernet ou Wi-Fi fornecidos pelo fornecedor.

  Para obter informações sobre a configuração de um projeto de teste, consulte Configuração do seu espaço de trabalho e projeto para portabilidade.
- Uma configuração validada do kernel do FreeRTOS.

  Para obter informações sobre como configurar o kernel do FreeRTOS para sua plataforma, consulte Configuração de uma porta do kernel do FreeRTOS.

Portabilidade

Antes de começar a fazer a transferência da biblioteca FreeRTOS+TCP, verifique o diretório do GitHub para ver se já existe uma porta para o dispositivo.

Se não existir uma porta, faça o seguinte:

1. Siga as instruções de Fazer a portabilidade do FreeRTOS+TCP para um microcontrolador diferente em FreeRTOS.org para fazer a portabilidade do FreeRTOS+TCP para seu dispositivo.
2. Se necessário, siga as instruções de Fazer a portabilidade do FreeRTOS+TCP para um novo compilador C incorporado em FreeRTOS.org para fazer a portabilidade do FreeRTOS+TCP para um novo compilador.
3. Implemente uma nova porta que use os drivers Ethernet ou Wi-Fi do fornecedor em um arquivo chamado `NetworkInterface.c`. Visite o repositório do GitHub para obter um modelo.

Depois de criar a porta, ou se já existir uma, crie `FreeRTOSIPConfig.h` e edite as opções de configuração para que estejam corretas para a plataforma. Para obter mais informações sobre as opções de configuração, consulte Configuração do FreeRTOS+TCP em FreeRTOS.org.

Testar

Se você usa a biblioteca FreeRTOS+TCP ou uma biblioteca de terceiros, siga as etapas abaixo para testar:

- Forneça uma implementação para APIs `connect/disconnect/send/receive` em testes de interface de transporte.
- Configure um servidor echo no modo de conexão TCP de texto simples e execute testes de interface de transporte.

> **Note**
>
> Para qualificar oficialmente um dispositivo para FreeRTOS, se sua arquitetura exigir a portabilidade de uma pilha de software TCP/IP, você precisará validar o código-fonte transferido do dispositivo nos testes de interface de transporte no modo de conexão TCP de texto simples no AWS IoT Device Tester. Siga as instruções em Usar o AWS IoT Device Tester para FreeRTOS no Guia do usuário do FreeRTOS para configurar o AWS IoT Device Tester para validação de porta. Para testar a portabilidade de uma biblioteca específica, o grupo de testes correto deve ser habilitado no arquivo `device.json` na pasta `configs` do Device Tester.

Portabilidade da biblioteca corePKCS11

O Padrão de criptografia de chave pública #11 define uma API independente de plataforma para gerenciar e usar tokens criptográficos. O PKCS 11 se refere ao padrão e APIs definida por ele. A API criptográfica de PKCS #11 abstrai armazenamento de chaves, propriedades de obtenção/definição para objetos criptográficos e semântica de sessão. É amplamente usado para manipular objetos criptográficos comuns. Suas funções permitem que o software da aplicação use, crie, modifique e exclua objetos criptográficos, sem expor esses objetos à memória da aplicação.

As bibliotecas e integrações de referência do FreeRTOS usam um subconjunto do padrão de interface PCKS #11, com foco nas operações que envolvem chaves assimétricas, geração de números aleatórios e hashing. A tabela abaixo lista os casos de uso e as APIs de PKCS #11 necessárias para oferecer suporte.

Casos de uso

Caso de uso	Família de API com PKCS #11 necessária
Todos	Initialize, Finalize, Open/Close Session, GetSlotList, Login
Provisionamento	GenerateKeyPair, CreateObject, DestroyObject, initToken, GetTokenInfo
TLS	Random, Sign, FindObject, GetAttributeValue
FreeRTOS+TCP	Random, Digest
OTA	Verificação, resumo, FindObject, GetAttributeValue

Quando implementar um módulo PKCS #11 completo

O armazenamento de chaves privadas na memória flash de uso geral pode ser conveniente em cenários de avaliação e de prototipagem rápida. Recomendamos o uso de hardware criptográfico dedicado para reduzir as ameaças de roubo de dados e duplicação de dispositivos em cenários de produção. O hardware criptográfico inclui componentes com recursos que impedem que as chaves secretas criptográficas sejam exportadas. Para oferecer suporte a isso, você precisará implementar

um subconjunto de PKCS #11 necessário para trabalhar com bibliotecas FreeRTOS, conforme definido na tabela acima.

Quando usar a corePKCS11 do FreeRTOS

A biblioteca corePKCS11 contém uma implementação baseada em software da interface PKCS #11 (API) que usa a funcionalidade criptográfica fornecida pelo TLS Mbed. Isso é fornecido para cenários rápidos de prototipagem e avaliação em que o hardware não tem um hardware criptográfico dedicado. Nesse caso, você só precisa implementar a PAL corePKCS11 para fazer com que a implementação baseada em software corePKCS11 funcione com sua plataforma de hardware.

Portabilidade da corePKCS11

Será necessário ter implementações para ler e gravar objetos criptográficos na memória não volátil (NVM), como a memória flash integrada. Os objetos criptográficos precisam ser armazenados em uma seção da NVM que não tenha sido inicializada e não seja apagada na reprogramação do dispositivo. Os usuários da biblioteca corePKCS11 provisionarão dispositivos com credenciais e reprogramarão o dispositivo com uma nova aplicação que acesse essas credenciais por meio da interface corePKCS11. As portas PAL corePKCS11 devem fornecer um local para armazenar:

- O certificado de cliente do dispositivo
- A chave privada de cliente do dispositivo
- A chave pública de cliente do dispositivo
- Uma CA raiz confiável
- Uma chave pública de verificação de código (ou um certificado que contém a chave pública de verificação de código) para atualizações seguras do carregador de inicializações e sem fios
- Um certificado de provisionamento just-in-time

Inclua o arquivo de cabeçalho e implemente as APIs PAL definidas.

APIs PAL

Função	Descrição
PKCS11_PAL_Initialize	Inicializa a camada PAL. Chamado pela biblioteca corePKCS11 no início da sequência de inicialização.

Função	Descrição
PKCS11_PAL_SaveObject	Grava dados no armazenamento não volátil.
PKCS11_PAL_FindObject	Usa um `CKA_LABEL` PKCS #11 para procurar um objeto PKCS #11 correspondente no armazenamento não volátil e retorna o identific ador desse objeto, se existir.
PKCS11_PAL_GetObjectValue	Recupera o valor de um objeto, considerando o identificador.
PKCS11_PAL_GetObjectValueCleanup	Limpeza da chamada `PKCS11_PA L_GetObjectValue` . Pode ser usada para liberar memória alocada em uma chamada `PKCS11_PAL_GetObjectValue` .

Testar

Se você usar a biblioteca corePKCS11 do FreeRTOS ou implementar o subconjunto necessário de APIs de PKCS11, deverá passar nos testes do PKCS11 do FreeRTOS. Eles testam se as funções necessárias para as bibliotecas FreeRTOS funcionam conforme o esperado.

Esta seção também descreve como você pode executar localmente os testes PKCS11 do FreeRTOS com os testes de qualificação.

Pré-requisitos

Para configurar os testes PKCS11 do FreeRTOS, deve-se implementar o seguinte.

- Uma porta compatível com as APIs PKCS11.
- Uma implementação das funções da plataforma de testes de qualificação do FreeRTOS, que incluem o seguinte:
  - `FRTest_ThreadCreate`
  - `FRTest_ThreadTimedJoin`
  - `FRTest_MemoryAlloc`
  - `FRTest_MemoryFree`

(Consulte o arquivo README.md para os testes de integração de bibliotecas FreeRTOS para PKCS #11 no GitHub.)

Testes de portabilidade

- Adicione FreeRTOS-Libraries-Integration-Tests como um submódulo em seu projeto. O submódulo pode ser colocado em qualquer diretório do projeto, desde que possa ser compilado.
- Copie `config_template/test_execution_config_template.h` e `config_template/test_param_config_template.h` para um local do projeto no caminho de compilação e renomeie-os para `test_execution_config.h` e `test_param_config.h`.
- Inclua os arquivos relevantes no sistema de compilação. Se estiver usando `CMake`, `qualification_test.cmake` e `src/pkcs11_tests.cmake` podem ser usados para incluir os arquivos relevantes.
- Implemente `UNITY_OUTPUT_CHAR` para que os logs de saída do teste e logs do dispositivo não se intercalem.
- Integre o MbedTLS, que verifica o resultado da operação cryptoki.
- Chame `RunQualificationTest()` da aplicação.

Configuração de testes

O conjunto de testes PKCS11 deve ser configurado de acordo com a implementação do PKCS11. A tabela a seguir lista a configuração exigida pelos testes PKCS11 no arquivo de cabeçalho `test_param_config.h`.

Configurações de teste PKSC11

Configuração	Descrição
PKCS11_TEST_RSA_KEY_SUPPORT	A portabilidade oferece suporte às funções de chave do RSA.
PKCS11_TEST_EC_KEY_SUPPORT	A portabilidade oferece suporte às funções de chave de EC.
PKCS11_TEST_IMPORT_PRIVATE_KEY_SUPPORT	A portabilidade oferece suporte à importaçã o da chave privada. A importação de chaves

Configuração	Descrição
	RSA e EC é validada no teste se o suporte às funções de chave estiver habilitado.
PKCS11_TEST_GENERATE_KEYPAIR_SUPPORT	A portabilidade oferece suporte às geração de pares de chaves. A geração de pares de chaves EC é validada no teste se o suporte às funções de chave estiver habilitado.
PKCS11_TEST_PREPROVISIONED_SUPPORT	A portabilidade tem credenciais pré-provisionadas. `PKCS11_TEST_LABEL_DEVICE_PRIVATE_KEY_FOR_TLS` , `PKCS11_TEST_LABEL_DEVICE_PUBLIC_KEY_FOR_TLS` e `PKCS11_TEST_LABEL_DEVICE_CERTIFICATE_FOR_TLS` , são exemplos das credenciais.
PKCS11_TEST_LABEL_DEVICE_PRIVATE_KEY_FOR_TLS	O rótulo da chave privada usada no teste.
PKCS11_TEST_LABEL_DEVICE_PUBLIC_KEY_FOR_TLS	O rótulo da chave pública usada no teste.
PKCS11_TEST_LABEL_DEVICE_CERTIFICATE_FOR_TLS	O rótulo do certificado usado no teste.
PKCS11_TEST_JITP_CODEVERIFY_ROOT_CERT_SUPPORTED	A portabilidade oferece suporte ao armazenamento de JITP. Defina isso como 1 para ativar o teste `codeverify` de JITP.
PKCS11_TEST_LABEL_CODE_VERIFICATION_KEY	O rótulo da chave de verificação de código usada no teste `codeverify` de JITP.
PKCS11_TEST_LABEL_JITP_CERTIFICATE	O rótulo do certificado JITP usado no teste `codeverify` de JITP.
PKCS11_TEST_LABEL_ROOT_CERTIFICATE	O rótulo do certificado raiz usado no teste `codeverify` de JITP.

As bibliotecas e integrações de referência do FreeRTOS devem oferecer suporte a no mínimo uma configuração de função de chave, como chaves de Curva elíptica ou RSA, e um mecanismo de provisionamento de teclas compatível com as APIs de PKCS11. O teste deve habilitar as seguintes configurações:

- Pelo menos uma das configurações de função de chave a seguir:
  - PKCS11_TEST_RSA_KEY_SUPPORT
  - PKCS11_TEST_EC_KEY_SUPPORT
- Pelo menos uma das configurações de provisionamento de chave a seguir:
  - PKCS11_TEST_IMPORT_PRIVATE_KEY_SUPPORT
  - PKCS11_TEST_GENERATE_KEYPAIR_SUPPORT
  - PKCS11_TEST_PREPROVISIONED_SUPPORT

O teste de credencial do dispositivo pré-provisionado deve ser executado sob as seguintes condições:

- `PKCS11_TEST_PREPROVISIONED_SUPPORT` devem estar habilitados e outros mecanismos de provisionamento desativados.
- Ter somente uma função de chave, `PKCS11_TEST_RSA_KEY_SUPPORT` ou `PKCS11_TEST_EC_KEY_SUPPORT` está ativada.
- Configure os rótulos de chave pré-provisionados de acordo com a função de chave, incluindo `PKCS11_TEST_LABEL_DEVICE_PRIVATE_KEY_FOR_TLS`, `PKCS11_TEST_LABEL_DEVICE_PUBLIC_KEY_FOR_TLS` e `PKCS11_TEST_LABEL_DEVICE_CERTIFICATE_FOR_TLS`. Essas credenciais devem existir antes de executar o teste.

É aconselhável executar o teste várias vezes em configurações diferentes, se a implementação oferecer suporte a credenciais pré-provisionadas e outros mecanismos de provisionamento.

> **Note**
>
> Os objetos com rótulos `PKCS11_TEST_LABEL_DEVICE_PRIVATE_KEY_FOR_TLS`, `PKCS11_TEST_LABEL_DEVICE_PUBLIC_KEY_FOR_TLS` e `PKCS11_TEST_LABEL_DEVICE_CERTIFICATE_FOR_TLS` serão destruídos

durante o teste se `PKCS11_TEST_GENERATE_KEYPAIR_SUPPORT` ou `PKCS11_TEST_GENERATE_KEYPAIR_SUPPORT` estiverem habilitados.

Execução de testes

Esta seção descreve como você pode testar localmente a interface de PKCS11 com os testes de qualificação. Como alternativa, você também pode usar o IDT para automatizar a execução. Consulte o AWS IoT Device Tester para o FreeRTOS no Guia do usuário do FreeRTOS para obter detalhes.

As instruções a seguir descrevem como executar os testes:

- Abra `test_execution_config.h` e defina CORE_PKCS11_TEST_ENABLED como 1.
- Compile e atualize a aplicação em seu dispositivo para ser executada. Os resultados do teste são enviados para a porta serial.

Confira a seguir um exemplo do resultado do teste de saída.

```
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
```

O teste será concluído quando todos os testes forem aprovados.

> **Note**
>
> Para qualificar oficialmente um dispositivo para o FreeRTOS, é necessário validar o código-fonte transferido do dispositivo com o AWS IoT Device Tester. Siga as instruções em Usar o AWS IoT Device Tester para FreeRTOS no Guia do usuário do FreeRTOS para configurar o AWS IoT Device Tester para validação de porta. Para testar a portabilidade de uma biblioteca específica, o grupo de testes correto deve ser habilitado no arquivo `device.json` na pasta do AWS IoT Device Tester `configs`.

Fazer portabilidade da Network Transport Interface

Integração da biblioteca TLS

Para autenticação Transport Layer Security (TLS), use a pilha TLS de sua preferência. Recomendamos usar o Mbed TLS porque ele é testado com bibliotecas FreeRTOS. Você pode encontrar um exemplo disso neste GitHubrepositório.

Independentemente da implementação de TLS usada pelo seu dispositivo, você deve implementar os ganchos de transporte subjacentes para TLS pilha com pilha. TCP/IP Eles devem oferecer suporte aos pacotes de criptografia TLS que são compatíveis com o AWS IoT.

Fazer portabilidade da biblioteca Network Transport Interface

Você deve implementar uma interface de transporte de rede para usar coreMQTT e coreHTTP. A interface de transporte de rede contém ponteiros de função e dados de contexto necessários para

enviar e receber dados em uma única conexão de rede. Consulte Interface de transporte para obter mais detalhes. O FreeRTOS fornece um conjunto de testes integrados de interface de transporte de rede para validar essas implementações. A seção a seguir orienta você sobre como configurar seu projeto para executar esses testes.

Pré-requisitos

Para fazer a portabilidade desse teste, é necessário:

- Um projeto com um sistema de compilação que pode criar o FreeRTOS com uma porta validada do kernel do FreeRTOS.
- Implementação funcional de drivers de rede.

Portabilidade

- Adicione FreeRTOS-Libraries-Integration-Testscomo um submódulo ao seu projeto. Não importa onde o submódulo será colocado no projeto, desde que ele possa ser compilado.
- Copie `config_template/test_execution_config_template.h` e `config_template/test_param_config_template.h` para um local do projeto no caminho de compilação e renomeie-os para `test_execution_config.h` e `test_param_config.h`.
- Inclua os arquivos relevantes no sistema de compilação. Se estiver usando `CMake`, `qualification_test.cmake` e `src/transport_interface_tests.cmake` são usados para incluir os arquivos relevantes.
- Implemente as seguintes funções em um local apropriado do projeto:
- • Rnetwork connect function: a assinatura é definida por `NetworkConnectFunc` em `src/common/network_connection.h`. Essa função usa um ponteiro para o contexto da rede, um ponteiro para as informações do host e um ponteiro para as credenciais da rede. Ela estabelece uma conexão com o servidor especificado nas informações do host com as credenciais de rede fornecidas.
  - Rnetwork disconnect function: a assinatura é definida por `NetworkDisconnectFunc` em `src/common/network_connection.h`. Essa função usa um ponteiro para um contexto de rede. Ela desconecta uma conexão previamente estabelecida armazenada no contexto da rede.
  - `setupTransportInterfaceTestParam()`: esta API é definida em `src/transport_interface/transport_interface_tests.h`. A implementação deve ter

exatamente o mesmo nome e assinatura definidos em `transport_interface_tests.h`. Essa função recebe um ponteiro para uma TransportInterfaceTestParamestrutura. Ele preencherá os campos na TransportInterfaceTestParamestrutura usada pelo teste da interface de transporte.

- Implemente UNITY_OUTPUT_CHAR para que os logs de saída do teste não intercalem com os logs do dispositivo.
- Chame `runQualificationTest()` da aplicação. O hardware do dispositivo deve ser inicializado corretamente e a rede deve estar conectada antes da chamada.

Gerenciamento de credenciais (chave gerada no dispositivo)

Quando FORCE_GENERATE_NEW_KEY_PAIR em `test_param_config.h` é definido como 1, a aplicação do dispositivo gera um novo par de chaves no dispositivo e gera a chave pública. A aplicação do dispositivo usa ECHO_SERVER_ROOT_CA e TRANSPORT_CLIENT_CERTIFICATE como certificado da CA raiz do servidor echo e do cliente ao estabelecer uma conexão TLS com o servidor echo. O IDT define esses parâmetros durante a execução da qualificação.

Gerenciamento de credenciais (chave de importação)

A aplicação do dispositivo usa ECHO_SERVER_ROOT_CA, TRANSPORT_CLIENT_CERTIFICATE e TRANSPORT_CLIENT_PRIVATE_KEY em `test_param_config.h` como certificado da CA raiz do servidor echo, certificado de cliente e chave privada de cliente ao estabelecer uma conexão TLS com o servidor echo. O IDT define esses parâmetros durante a execução da qualificação.

Teste

Esta seção descreve como você pode testar localmente a interface de transporte com os testes de qualificação. Detalhes adicionais podem ser encontrados no arquivo README.md fornecido na seção transport_interface do on. FreeRTOS-Libraries-Integration-Tests GitHub

Como alternativa, você também pode usar o IDT para automatizar a execução. Consulte o AWS IoT Device Tester para o FreeRTOS no Guia do usuário do FreeRTOS para obter detalhes.

Habilitação do teste

Abra `test_execution_config.h` e defina TRANSPORT_INTERFACE_TEST_ENABLED como 1.

Configuração do servidor echo para testes

É necessário ter um servidor echo acessível a partir do dispositivo que executa os testes para fazer testes locais. O servidor echo deve oferecer suporte ao TLS se a implementação da interface de transporte oferecer suporte ao TLS. Se você ainda não tiver um, o FreeRTOS-Libraries-Integration-Tests GitHub repositório tem uma implementação de servidor de eco.

Configuração do projeto para teste

Em `test_param_config.h`, atualize ECHO_SERVER_ENDPOINT e ECHO_SERVER_PORT para a configuração do endpoint e do servidor na etapa anterior.

Credenciais de configuração (chave gerada no dispositivo)

- Defina ECHO_SERVER_ROOT_CA como o certificado do servidor echo.
- Defina FORCE_GENERATE_NEW_KEY_PAIR como 1 para gerar um par de chaves e obter a chave pública.
- Defina FORCE_GENERATE_NEW_KEY_PAIR como 0 novamente após a geração da chave.
- Use a chave pública e do servidor e o certificado para gerar o certificado do cliente.
- Defina TRANSPORT_CLIENT_CERTIFICATE como o certificado de cliente gerado.

Credenciais de configuração (chave de importação)

- Defina ECHO_SERVER_ROOT_CA como o certificado do servidor echo.
- Defina TRANSPORT_CLIENT_CERTIFICATE como o certificado de cliente pré-gerado.
- Defina TRANSPORT_CLIENT_PRIVATE_KEY como a chave privada pré-gerada do cliente.

Compilação e instalação da aplicação

Compile e instale a aplicação usando a cadeia de ferramentas de sua escolha. Quando `runQualificationTest()` for invocado, os testes da interface de transporte serão executados. Os resultados do teste são emitidos para a porta serial.

> **Note**
>
> Para qualificar oficialmente um dispositivo para FreeRTOS, você deve validar o código-fonte portado do dispositivo em relação aos grupos de teste OTA PAL e OTA E2E com. AWS IoT

Device Tester Siga as instruções em Usando o FreeRTOS no Guia do Usuário do FreeRTOS AWS IoT Device Tester para configurar a validação de portas. AWS IoT Device Tester Para testar a porta de uma biblioteca específica, o grupo de teste correto deve estar habilitado no `device.json` arquivo na AWS IoT Device Tester `configs` pasta.

Configuração da biblioteca coreMQTT

Os dispositivos na borda podem usar o protocolo MQTT para se comunicar com a Nuvem AWS. O AWS IoT hospeda um agente MQTT que recebe e envia mensagens para dispositivos conectados na borda.

A biblioteca coreMQTT implementa o protocolo MQTT para dispositivos que executam o . A biblioteca coreMQTT não precisa ser transferida, mas o projeto de teste do dispositivo deve passar em todos os testes MQTT para qualificação. Para obter mais informações, consulte Biblioteca coreMQTT no Guia do usuário do FreeRTOS.

Pré-requisitos

Para configurar os testes da biblioteca coreMQTT, você precisa de uma porta de interface de transporte de rede. Para saber mais, consulte Fazer portabilidade da Network Transport Interface.

Testar

Execute testes de integração da coreMQTT:

- Registre seu certificado de cliente no agente MQTT.
- Configure o endpoint do agente em `config` e execute os testes de integração.

Criação de demonstração de referência do MQTT

Recomendamos usar o agente coreMQTT para lidar com a segurança de thread em todas as operações do MQTT. O usuário também precisará publicar e assinar tarefas e testes do Device Advisor para validar se a aplicação integra TLS, MQTT e outras bibliotecas FreeRTOS de forma eficaz.

Para qualificar oficialmente um dispositivo para FreeRTOS, valide seu projeto de integração com os casos de teste da MQTT do AWS IoT Device Tester. Consulte o fluxo de trabalho do AWS IoT

Device Advisor para obter instruções de configuração e teste. Os casos de teste obrigatórios para TLS e MQTT estão listados abaixo:

Casos de teste do TLS

Caso de teste	Casos de teste	Testes necessários
TLS	Conexão TLS	Sim
TLS	Pacotes de criptografia de suporte ao TLS do AWS IoT	Um pacote de criptografia recomendado
TLS	Certificado de servidor TLS desprotegido	Sim
TLS	TLS: certificado do servidor de nome de assunto incorreto	Sim

Casos de teste do MQTT

Caso de teste	Casos de teste	Testes necessários
MQTT	MQTT Connect	Sim
MQTT	Novas tentativas de jitter de conexão MQTT	Sim, sem avisos
MQTT	MQTT Subscribe	Sim
MQTT	MQTT Publish	Sim
MQTT	QoS 1 ClientPuback de MQTT	Sim
MQTT	No Ack PingResp de MQTT	Sim

Configuração da biblioteca coreHTTP

Os dispositivos na borda podem usar o protocolo HTTP para se comunicar com a nuvem da AWS. Os serviços do AWS IoT hospedam um servidor HTTP que envia e recebe mensagens para dispositivos conectados na borda.

Testar

Siga as etapas abaixo para testar:

- Configure a PKI para autenticação mútua TLS na AWS ou um servidor HTTP.
- Execute testes de integração da coreHTTP.

Portando a biblioteca de atualização AWS IoT over-the-air (OTA)

Com as atualizações do over-the-air FreeRTOS (OTA), você pode fazer o seguinte:

- Implante novas imagens de firmware em um único dispositivo, um grupo de dispositivos ou toda a sua frota.
- Implantar firmware em dispositivos conforme eles são adicionados a grupos, redefinidos ou provisionados novamente.
- Verifique a autenticidade e a integridade do novo firmware depois da sua implantação nos dispositivos.
- Monitore o progresso de uma implantação.
- Depure uma implantação com falha.
- Assine digitalmente o firmware usando a Assinatura de Código para AWS IoT.

Para obter mais informações, consulte as Atualizações do _Over-the-AirFreeRTOS no Guia do Usuário do FreeRTOS junto com a documentação da atualização.AWS IoT Over-the-air

Você pode usar a biblioteca de atualização OTA para integrar a funcionalidade OTA nas aplicações do FreeRTOS. Para obter mais informações, consulte Biblioteca de atualização OTA do FreeRTOS no Guia do usuário do FreeRTOS.

Os dispositivos do FreeRTOS devem impor a verificação criptográfica de assinatura de código nas imagens de firmware OTA que recebem. Recomendamos os seguintes algoritmos:

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

Pré-requisitos

- Complete as instruções em Configuração do seu espaço de trabalho e projeto para portabilidade.
- Crie uma porta de interface de transporte de rede.

  Para mais informações, consulte Fazer portabilidade da Network Transport Interface.
- Integre a biblioteca coreMQTT. Consulte Biblioteca coreMQTT no Guia do usuário do FreeRTOS.
- Crie um carregador de inicialização que ofereça suporte a atualizações OTA.

Portabilidade de plataforma

Você deve fornecer uma implementação da camada de abstração portável (PAL) OTA para transferir a biblioteca OTA para um novo dispositivo. Os PAL APIs são definidos no arquivo ota_platform_interface.h para o qual detalhes específicos da implementação devem ser fornecidos.

Nome da função	Description
`otaPal_Abort`	Interrompe uma atualização OTA.
`otaPal_CreateFileForRx`	Cria um arquivo para armazenar os blocos de dados recebidos.
`otaPal_CloseFile`	Fecha o arquivo especificado. Isso pode autenticar o arquivo se você usar armazenam ento que implementa a proteção criptográfica.
`otaPal_WriteBlock`	Grava um bloco de dados no arquivo especific ado no deslocamento determinado. A função retornará o número de bytes gravados quando tiver êxito. Caso contrário, a função retornará um código de erro negativo. O tamanho do bloco sempre será uma potência de dois e

Nome da função	Description
	será alinhado. Para obter mais informações, consulte Configuração da biblioteca OTA.
`otaPal_ActivateNewImage`	Ativa ou inicia a nova imagem de firmware. Em algumas portas, se o dispositivo for redefinid o programaticamente de forma síncrona, essa função poderá não retornar.
`otaPal_SetPlatformImageState`	Faz o que é exigido pela plataforma para aceitar ou rejeitar a imagem de firmware (ou pacote) OTA mais recente. Para implementar essa função, consulte a documentação para obter os detalhes e a arquitetura da sua placa (plataforma).
`otaPal_GetPlatformImageState`	Obtém o estado da imagem de atualização OTA.

Implemente as funções nesta tabela se o dispositivo tiver suporte integrado para elas.

Nome da função	Description
`otaPal_CheckFileSignature`	Verifica a assinatura do arquivo especificado.
`otaPal_ReadAndAssumeCertificate`	Lê o certificado do assinante especificado no sistema de arquivos e o retorna para o chamador.
`otaPal_ResetDevice`	Redefine o dispositivo.

> **Note**
>
> Certifique-se de que você tem um bootloader que pode oferecer suporte a atualizações OTA. Para obter instruções sobre como criar o carregador de inicialização do dispositivo do AWS IoT , consulte Bootloader de dispositivo de IoT.

Testes E2E e PAL

Execute testes PAL OTA e E2E.

Testes E2E

O teste de ponta a ponta (E2E) OTA é usado para verificar a capacidade OTA de um dispositivo e simular cenários reais. Esse teste incluirá tratamento de erros.

Pré-requisitos

Para fazer a portabilidade desse teste, é necessário:

- Um projeto com uma biblioteca AWS OTA integrada. Visite o Guia de portabilidade da biblioteca OTA para obter informações adicionais.
- Faça a portabilidade da aplicação de demonstração usando a biblioteca OTA para interagir com o AWS IoT Core e fazer as atualizações OTA. Consulte Portabilidade da aplicação de demonstração OTA.
- Configure a ferramenta do IDT. Isso executa a aplicação host E2E OTA para compilar, instalar e monitorar o dispositivo em diferentes configurações e valida a integração da biblioteca OTA.

Portabilidade da aplicação de demonstração OTA

O teste E2E OTA deve ter uma aplicação de demonstração OTA para validar a integração da biblioteca OTA. A aplicação de demonstração deve ter a capacidade de realizar atualizações de firmware OTA. Você pode encontrar o aplicativo de demonstração FreeRTOS OTA no repositório do FreeRTOS. GitHub Recomendamos que você use a aplicação de demonstração como referência e a modifique de acordo com suas especificações.

Etapas da portabilidade

1. Inicialize o agente OTA.

2. Implemente a função de retorno de chamada da aplicação OTA.
3. Crie a tarefa de processamento de eventos do agente OTA.
4. Inicie o agente OTA.
5. Monitore as estatísticas do agente OTA.
6. Encerre o agente OTA.

Visite OTA do FreeRTOS por MQTT - Ponto de entrada da demonstração para obter instruções detalhadas.

Configuração

As seguintes configurações são necessárias para interagir com AWS IoT Core:

- AWS IoT Core credenciais do cliente
  - Configure DemoConfigRoot_CA_PEM em `Ota_Over_Mqtt_Demo/demo_config.h` com os Endpoints dos serviços de confiança da Amazon. Consulte Autenticação do servidor da AWS para obter mais detalhes.
  - Configure DemoConfigClient_Certificate_PEM e DemoConfigClient_Private_Key_PEM com suas credenciais de cliente. `Ota_Over_Mqtt_Demo/demo_config.h` AWS IoT Consulte os detalhes da autenticação do cliente da AWS para saber mais sobre certificados e chaves privadas do cliente.
- Versão da aplicação
- Protocolo de controle OTA
- Protocolo de dados OTA
- Credenciais de assinatura de código
- Outras configurações da biblioteca OTA

Você pode encontrar as informações anteriores nas aplicações de demonstração OTA do FreeRTOS em `demo_config.h` e `ota_config.h`. Visite OTA do FreeRTOS por MQTT - Configuração do dispositivo para obter mais informações.

Verificação de compilação

Execute a aplicação de demonstração para executar o trabalho OTA. Quando isso for concluído com êxito, você poderá continuar executando os testes E2E OTA.

A demonstração do FreeRTOS OTA fornece informações detalhadas sobre como configurar um cliente OTA e AWS IoT Core uma tarefa OTA no simulador de janelas do FreeRTOS. AWS O OTA suporta os protocolos MQTT e HTTP. Consulte os exemplos a seguir para obter mais detalhes:

- Demonstração OTA por MQTT no simulador do Windows
- Demonstração OTA por HTTP no simulador do Windows

Execução de testes com a ferramenta do IDT

Para executar os testes OTA E2E, você deve usar AWS IoT Device Tester (IDT) para automatizar a execução. Consulte o AWS IoT Device Tester para o FreeRTOS no Guia do usuário do FreeRTOS para obter mais detalhes.

Casos de teste E2E

Caso de teste	Description
`OTAE2EGreaterVersion`	Teste "caminho feliz" para atualizações regulares do OTA. Ele cria uma atualização com uma versão mais recente, que o dispositivo atualiza com êxito.
`OTAE2EBackToBackDownloads`	Esse teste cria três atualizações OTA consecutivas. O esperado é que o dispositivo atualize três vezes consecutivas.
`OTAE2ERollbackIfUnableToConnectAfterUpdate`	Esse teste verifica se o dispositivo é revertido para o firmware anterior se não conseguir se conectar à rede com o novo firmware.
`OTAE2ESameVersion`	Esse teste confirma que o dispositivo rejeita o firmware de entrada se a versão permanecer a mesma.
`OTAE2EUnsignedImage`	Esse teste verifica se o dispositivo rejeita uma atualização se a imagem não estiver assinada.

Caso de teste	Description
`OTAE2EUntrustedCertificate`	Esse teste verifica se o dispositivo rejeita uma atualização se o firmware estiver assinado com um certificado não confiável.
`OTAE2EPreviousVersion`	Esse teste verifica se o dispositivo rejeita uma versão de atualização mais antiga.
`OTAE2EIncorrectSigningAlgorithm`	Dispositivos diferentes oferecem suporte a algoritmos de assinatura e hashing diferente s. Esse teste verifica se o dispositivo falha na atualização OTA, caso ele seja criado com um algoritmo não compatível.
`OTAE2EDisconnectResume`	Este é o teste de "caminho feliz" para o recurso de suspensão e retomada. Esse teste cria uma atualização OTA e inicia a atualização. Em seguida, ele se conecta AWS IoT Core com o mesmo ID de cliente (nome da coisa) e credenciais. AWS IoT Core em seguida, desconecta o dispositivo. Espera-se que o dispositivo detecte que está desconectado e AWS IoT Core, após um período de tempo, passe para um estado suspenso e tente se reconectar AWS IoT Core e retomar o download.

Caso de teste	Description
`OTAE2EDisconnectCancelUpdate`	Esse teste verifica se o dispositivo pode se recuperar, caso o trabalho OTA seja cancelado enquanto está em um estado suspenso. Ele faz a mesma coisa que o `OTAE2EDisconnectResume` teste, exceto que depois de se conectar ao dispositivo AWS IoT Core, que desconect a o dispositivo, ele cancela a atualização do OTA. Uma nova atualização é criada. Espera-se que o dispositivo se reconecte ao AWS IoT Core, aborte a atualização atual, volte ao estado de espera e aceite e conclua a próxima atualização.
`OTAE2EPresignedUrlExpired`	Quando uma atualização OTA é criada, você pode configurar a vida útil do URL pré-assin ado do S3. Esse teste verifica se o dispositivo é capaz de realizar um OTA, mesmo que não consiga concluir o download quando o URL expirar. O esperado é que o dispositivo solicite um novo documento de trabalho, que contém um novo URL para retomar o download.
`OTAE2E2UpdatesCancel1st`	Esse teste cria duas atualizações OTA consecutivas. Quando o dispositivo relata que está baixando a primeira atualização, o teste força o cancelamento da primeira atualizaç ão. O esperado é que o dispositivo anule a atualização atual, detecte a segunda atualizaç ão e a conclua.

Caso de teste	Description
`OTAE2ECancelThenUpdate`	Esse teste cria duas atualizações OTA consecutivas. Quando o dispositivo relata que está baixando a primeira atualização, o teste força o cancelamento da primeira atualizaç ão. O esperado é que o dispositivo anule a atualização atual e detecte a segunda atualizaç ão, depois a conclua.
`OTAE2EImageCrashed`	Esse teste verifica se o dispositivo é capaz de rejeitar uma atualização quando a imagem falha.

Testes PAL

Pré-requisitos

Para transferir os testes da interface de transporte de rede, será necessário:

- Um projeto que pode compilar o FreeRTOS com uma porta kernel válida do FreeRTOS.
- Uma implementação funcional do PAL OTA.

Portabilidade

- Adicione Freertos-Libraries-Integration-Tests como um submódulo em seu projeto. O submódulo deve ser localizado no projeto onde ele possa ser compilado.
- Copie `config_template/test_execution_config_template.h` e `config_template/test_param_config_template.h` para um local no caminho de compilação e renomeie-os para `test_execution_config.h` e `test_param_config.h`.
- Inclua os arquivos relevantes no sistema de compilação. Se estiver usando `CMake`, `qualification_test.cmake` e `src/ota_pal_tests.cmake` podem ser usados para incluir os arquivos relevantes.
- Configure o teste implementando as seguintes funções:

  - `SetupOtaPalTestParam()` definido em `src/ota/ota_pal_test.h`. A implementação deve ter exatamente o mesmo nome e assinatura definidos em `ota_pal_test.h`. No momento, não é necessário configurar essa função.
- Implemente UNITY_OUTPUT_CHAR para que os logs de saída do teste não intercalem com os logs do dispositivo.
- Chame `RunQualificationTest()` da aplicação. O hardware do dispositivo deve ser inicializado corretamente e a rede deve estar conectada antes da chamada.

Teste

Esta seção descreve os testes locais dos testes de qualificação PAL OTA.

Habilitação do teste

Abra `test_execution_config.h` e defina OTA_PAL_TEST_ENABLED como 1.

Em `test_param_config.h`, atualize as seguintes opções:

- OTA_PAL_TEST_CERT_TYPE: selecione o tipo de certificado usado.
- OTA_PAL_CERTIFICATE_FILE: caminho para o certificado do dispositivo, se aplicável.
- OTA_PAL_FIRMWARE_FILE: nome do arquivo de firmware, se aplicável.
- OTA_PAL_USE_FILE_SYSTEM: defina como 1 se o PAL OTA usar abstração do sistema de arquivos.

Compile e instale a aplicação usando uma cadeia de ferramentas de sua escolha. Quando `RunQualificationTest()` for chamado, os testes PAL OTA serão executados. Os resultados do teste são enviados para a porta serial.

Integração de tarefas OTA

- Adicione o agente OTA à demonstração atual do MQTT.
- Execute testes OTA End to End (E2E) com. AWS IoT Isso verifica se a integração está funcionando conforme esperado.

> **Note**
>
> Para qualificar oficialmente um dispositivo para FreeRTOS, você deve validar o código-fonte portado do dispositivo em relação aos grupos de teste OTA PAL e OTA E2E com. AWS IoT Device Tester Siga as instruções em Usando o FreeRTOS no Guia do Usuário do FreeRTOS AWS IoT Device Tester para configurar a validação de portas. AWS IoT Device Tester Para testar a porta de uma biblioteca específica, o grupo de teste correto deve estar habilitado no `device.json` arquivo na AWS IoT Device Tester `configs` pasta.

Bootloader de dispositivo de IoT

Você deve fornecer a própria aplicação de carregamento de inicialização seguro. Verifique se o design e a implementação fornecem uma mitigação adequada das ameaças à segurança. A seguir confira a modelagem de ameaças para sua referência.

Modelagem de ameaças para o bootloader de dispositivos de IoT

Contexto

Como definição prática, os AWS IoT dispositivos incorporados referenciados por esse modelo de ameaça são produtos baseados em microcontroladores que interagem com serviços em nuvem. Eles podem ser implantados em ambientes de consumo, comerciais ou industriais. Os dispositivos de IoT podem coletar dados sobre um usuário, um paciente, uma máquina ou um ambiente, além de poderem controlar qualquer coisa, desde lâmpadas e fechaduras de porta até máquinas de fábrica.

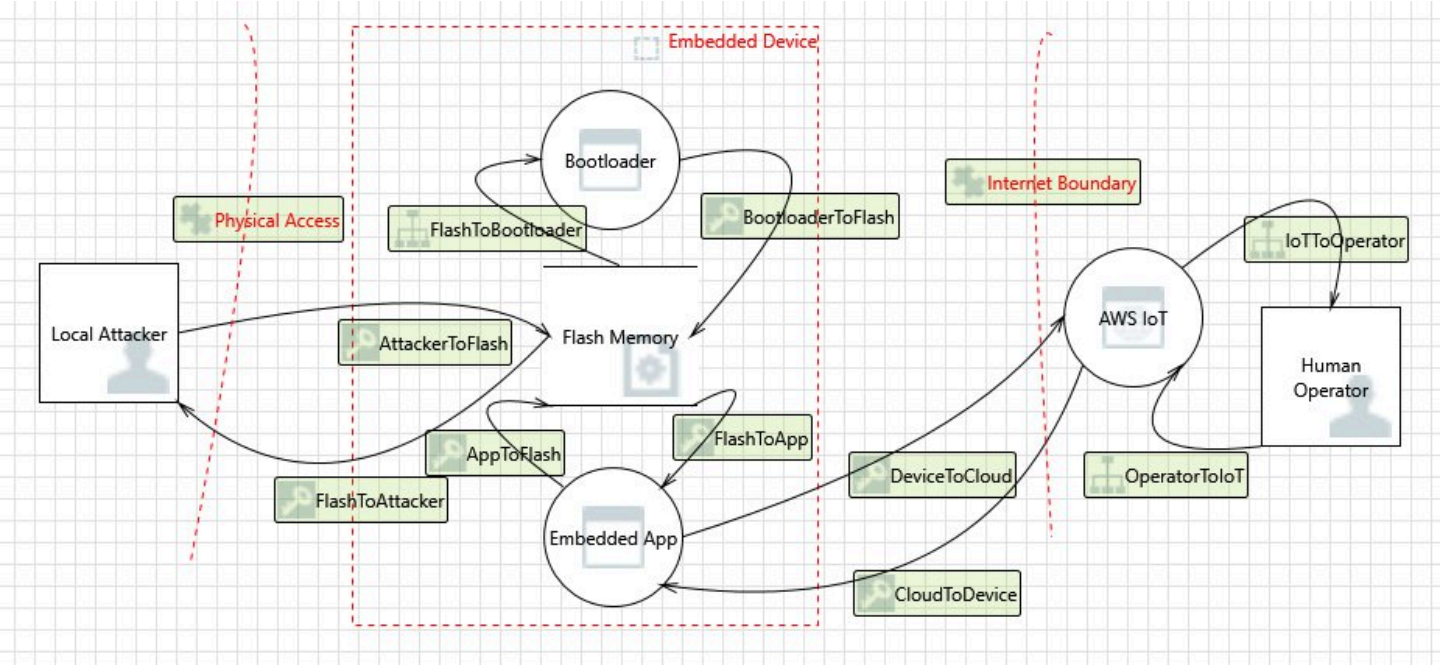
A modelagem de ameaças é uma abordagem da segurança do ponto de vista de um adversário hipotético. Considerando as metas e os métodos do adversário, uma lista de ameaças é criada. Ameaças são ataques contra um recurso ou ativo executado por um adversário. A lista é priorizada e usada para identificar ou criar soluções de mitigações. Ao escolher uma solução de mitigação, o custo de implementá-la e mantê-la deve ser equilibrado com o valor real de segurança que ela fornece. Existem várias metodologias de modelos de ameaças. Cada um é capaz de apoiar o desenvolvimento de um AWS IoT produto seguro e bem-sucedido.

O FreeRTOS oferece atualizações de software OTA over-the-air () para dispositivos. AWS IoT O recurso de atualização combina serviços em nuvem com bibliotecas de software no dispositivo e um bootloader fornecido pelo parceiro. Esse modelo de ameaça se concentra especificamente em ameaças contra o bootloader.

Casos de uso do bootloader

- Assine e criptografe digitalmente o firmware antes da implantação.
- Implante novas imagens de firmware em um único dispositivo, um grupo de dispositivos ou toda a frota.
- Verificar a autenticidade e a integridade do novo firmware depois de implantá-lo nos dispositivos.
- Os dispositivos só executam software não modificado de uma origem confiável.
- Os dispositivos são resilientes a software com falha recebido por meio do OTA.

Diagrama de fluxo de dados



Ameaças

Alguns ataques têm vários modelos de mitigação; por exemplo, uma rede man-in-the-middle destinada a fornecer uma imagem de firmware maliciosa é atenuada pela verificação da confiança no certificado oferecido pelo servidor TLS e no certificado do assinante do código da nova imagem do firmware. Para maximizar a segurança do carregador de inicialização, toda solução de mitigação que não sejam dele será consideradas não confiável. O carregador de inicialização deve ter soluções de mitigação intrínsecas para cada ataque. Ter soluções de mitigação em camadas é conhecido como. defense-in-depth

Ameaças:

- Um invasor sequestra a conexão do dispositivo com o servidor para entregar uma imagem de firmware mal-intencionada.

  Exemplo de atenuação

  - Na inicialização, o bootloader verifica a assinatura criptográfica da imagem usando um certificado conhecido. Se a verificação falhar, o bootloader reverterá para a imagem anterior.
- Um invasor explora um estouro de buffer para introduzir comportamento mal-intencionado na imagem de firmware existente armazenada em flash.

  Exemplos de atenuação

  - Na inicialização, o bootloader verifica, conforme descrito anteriormente. Quando a verificação falha sem nenhuma imagem anterior disponível, o bootloader é interrompido.
  - Na inicialização, o bootloader verifica, conforme descrito anteriormente. Quando a verificação falha sem nenhuma imagem anterior disponível, o carregador de inicialização entra em um modo somente OTA à prova de falhas.
- Um invasor inicializa o dispositivo em uma imagem armazenada anteriormente, que é explorável.

  Exemplos de atenuação

  - Os setores flash que armazenam a última imagem são apagados após a instalação e o teste bem-sucedidos de uma nova imagem.
  - Um fusível é queimado a cada atualização bem-sucedida, e cada imagem se recusa a ser executada, a menos que o número correto de fusíveis tenha sido queimado.
- Uma atualização OTA fornece uma imagem com falha ou mal-intencionada que bloqueia o dispositivo.

  Exemplo de atenuação

  - O bootloader inicia um temporizador de vigilância de hardware que aciona a reversão para a imagem anterior.
- Um invasor corrige o bootloader para ignorar a verificação de imagem para que o dispositivo aceite imagens não assinadas.

  Exemplos de atenuação

  - O bootloader está em ROM (memória somente leitura) e não pode ser modificado.

  - O bootloader está na OTP (one-time-programmable memória) e não pode ser modificado.
  - O bootloader está na zona segura do ARM TrustZone e não pode ser modificado.
- Um invasor substitui o certificado de verificação para que o dispositivo aceite imagens mal-intencionadas.

  Exemplos de atenuação

  - O certificado está em um coprocessador criptográfico e não pode ser modificado.
  - O certificado está em ROM (ou OTP ou zona segura) e não pode ser modificado.

Modelagem adicional de ameaças

Esse modelo de ameaça considera apenas o bootloader. Uma modelagem de ameaças mais ampla pode melhorar a segurança geral. Um método recomendado é listar as metas do adversário, os ativos visados por essas metas e os pontos de entrada dos ativos. Uma lista de ameaças pode ser feita considerando ataques aos pontos de entrada para ganhar controle dos ativos. Veja a seguir listas de exemplos de metas, ativos e pontos de entrada para um dispositivo IoT. Essas listas não são exaustivas e têm como objetivo estimular uma reflexão mais aprofundada.

Metas do adversário

- Extorquir dinheiro
- Arruinar reputações
- Falsificar dados
- Desviar recursos
- Espionar remotamente um alvo
- Obter acesso físico a um site
- Causar estragos
- Incutir terror

Principais ativos

- Chaves privadas
- Certificado do cliente
- Certificados CA raiz

- Credenciais e tokens de segurança
- Informações de identificação pessoal do cliente
- Implementações de segredos comerciais
- Dados do sensor
- Armazenamento de dados de análise na nuvem
- Infraestrutura de nuvem

Pontos de entrada

- Resposta DHCP
- Resposta DNS
- MQTT por TLS
- Resposta HTTPS
- Imagem de software OTA
- Outros, conforme determinado pela aplicação, por exemplo, USB
- Acesso físico ao barramento
- IC desanexado

Portabilidade da biblioteca Cellular Interface

O FreeRTOS oferece suporte aos comandos AT de uma camada de abstração de rede celular descarregada por TCP. Para obter mais informações, consulte a Biblioteca Cellular Interface e Fazer portabilidade da biblioteca Cellular Interface em freertos.org.

Pré-requisitos

Não há dependência direta da biblioteca Cellular Interface. No entanto, na pilha de rede do FreeRTOS, Ethernet, Wi-Fi e rede celular não podem coexistir, então os desenvolvedores devem escolher um deles para integrar com o Fazer portabilidade da Network Transport Interface.

> **Note**
>
> Se o módulo de rede celular for compatível com o descarregamento de TLS ou não for compatível com os comandos AT, os desenvolvedores poderão implementar a própria

abstração de rede celular para integração ao Fazer portabilidade da Network Transport Interface.

Migração do MQTT versão 3 para o coreMQTT

Este guia de migração explica como migrar aplicações do MQTT para o coreMQTT.

Como migrar da versão 1 para a versão 3 para as aplicações OTA

Este guia ajudará você a migrar sua aplicação da versão 1 da biblioteca OTA para a versão 3.

> **Note**
>
> O OTA versão 2 APIs é igual ao OTA v3 APIs, portanto, se seu aplicativo estiver usando a versão 2 do APIs , as alterações não serão necessárias para chamadas de API, mas recomendamos que você integre a versão 3 da biblioteca.

As demonstrações da versão 3 do OTA estão disponíveis aqui:

- ota_demo_core_mqtt.
- ota_demo_core_http.
- ota_ble.

Resumo das alterações da API

Resumo das alterações da API entre a versão 1 e a versão 3 da Biblioteca OTA

API OTA versão 1	API OTA versão 3	Descrição das alterações
OTA_ AgentInit	OTA_Init	Os parâmetros de entrada são alterados, assim como o valor retornado da função devido às mudanças na implementação no OTA v3. Consulte a seção OTA_Init abaixo para obter detalhes.
OTA_ AgentShutdown	OTA_Shutdown	Alteração nos parâmetro s de entrada, incluindo um parâmetro adicional para um cancelamento opcional

API OTA versão 1	API OTA versão 3	Descrição das alterações
		de assinatura dos tópicos do MQTT. Consulte a seção OTA_Shutdown abaixo para obter detalhes.
OTA_ GetAgentState	OTA_ GetState	O nome da API foi alterado sem alterações no parâmetro de entrada. O valor de retorno é o mesmo, mas o enum e os membros são renomeados. Consulte a seção OTA_ GetState abaixo para obter detalhes.
n/a	OTA_ GetStatistics	Foi adicionada uma nova API que substitui a APIs OTA_GetPacketsReceived, OTA_, OTA_, OTA_GetPacketsQueued. GetPacketsProcessed GetPacketsDropped Consulte a seção OTA_ GetStatistics abaixo para obter detalhes.
OTA_ GetPacketsReceived	n/a	Essa API foi removida da versão 3 e substituída pela OTA_GetStatistics.
OTA_ GetPacketsQueued	n/a	Essa API foi removida da versão 3 e substituída pela OTA_GetStatistics.
OTA_ GetPacketsProcessed	n/a	Essa API foi removida da versão 3 e substituída pela OTA_GetStatistics.

API OTA versão 1	API OTA versão 3	Descrição das alterações
OTA_ GetPacketsDropped	n/a	Essa API foi removida da versão 3 e substituída pela OTA_GetStatistics.
OTA_ ActivateNewImage	OTA_ ActivateNewImage	Os parâmetros de entrada são os mesmos, mas o código de erro OTA de retorno é renomeado e novos códigos de erro são adicionados na versão 3 da biblioteca OTA. Consulte a seção OTA_ ActivateNewImage para obter detalhes.
OTA_ SetImageState	OTA_ SetImageState	Os parâmetros de entrada são os mesmos e renomeados, o código de erro OTA de retorno é renomeado e novos códigos de erro são adicionados na versão 3 da biblioteca OTA. Consulte a seção OTA_ SetImageState para obter detalhes.
OTA_ GetImageState	OTA_ GetImageState	Os parâmetros de entrada são os mesmos. O enum de retorno é renomeado na versão 3 da biblioteca OTA. Consulte a seção OTA_ GetImageState para obter detalhes.

API OTA versão 1	API OTA versão 3	Descrição das alterações
OTA_Suspend	OTA_Suspend	Os parâmetros de entrada são os mesmos, o código de erro OTA de retorno é renomeado e novos códigos de erro são adicionados na versão 3 da biblioteca OTA. Consulte a seção OTA_Suspend para obter detalhes.
OTA_Resume	OTA_Resume	O parâmetro de entrada para a conexão é removido quando a conexão é tratada na demonstração/aplicação OTA, o código de erro OTA de retorno é renomeado e novos códigos de erro são adicionados na versão 3 da biblioteca OTA. Consulte a seção OTA_Resume para obter detalhes.
OTA_ CheckForUpdate	OTA_ CheckForUpdate	Os parâmetros de entrada são os mesmos, o código de erro OTA de retorno é renomeado e novos códigos de erro são adicionados na versão 3 da biblioteca OTA. Consulte a seção OTA_ CheckForUpdate para obter detalhes.

API OTA versão 1	API OTA versão 3	Descrição das alterações
n/a	OTA_ EventProcessingTask	Uma nova API foi adicionada e é o principal loop de eventos para lidar com eventos para atualização do OTA e deve ser chamada pela tarefa da aplicação. Consulte a seção OTA_ EventProcessingTask para obter detalhes.
n/a	OTA_ SignalEvent	Uma nova API foi adicionada e adiciona o evento ao final da fila de eventos OTA e é usada por módulos OTA internos para sinalizar a tarefa do atendente. Consulte a seção OTA_ SignalEvent para obter detalhes.
n/a	OTA_Err_strerror	Nova API para conversão de código de erro em string para erros OTA.
n/a	OTA_ _str error JobParse	Nova API para conversão de código de erro em string para erros de Job Parsing.
n/a	OTA_ _str error OsStatus	Nova API para conversão de código de status em string para status de porta do sistema operacional OTA.
n/a	OTA_ _str error PalStatus	Nova API para conversão de código de status em string para status de porta do sistema operacional PAL OTA.

Descrição das alterações necessárias

OTA_Init

Ao inicializar o atendente OTA na v1, a API `OTA_AgentInit` é usada, que usa parâmetros para contexto de conexão, nome da coisa, retorno de chamada completo e tempo limite como entrada.

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

Essa API agora foi alterada para `OTA_Init` com parâmetros para os buffers necessários para ota, interfaces ota, nome da coisa e retorno de chamada da aplicação.

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

Parâmetros de entrada removidos:

pvConnectionContext -

O contexto de conexão é removido porque a versão 3 da biblioteca OTA não exige que o contexto de conexão seja passado para ela e as MQTT/HTTP operações são tratadas por suas respectivas interfaces na demonstração/aplicativo OTA.

xTicksToEspere -

O parâmetro ticks to wait também é removido quando a tarefa é criada no OTA demo/application antes de chamar OTA_init.

Parâmetros de entrada renomeados:

xFunc:

O parâmetro é renomeado para OtaAppCallback e seu tipo é alterado para OtaAppCallback _t.

Novos parâmetros de entrada:

pOtaBuffer

O aplicativo deve alocar os buffers e passá-los para a biblioteca OTA usando a estrutura OtaAppBuffer _t durante a inicialização. Os buffers necessários diferem um pouco

dependendo do protocolo usado para baixar o arquivo. Para o protocolo MQTT, os buffers para o nome do fluxo são necessários e, para o protocolo HTTP, os buffers para URL pré-assinado e esquema de autorização são necessários.

Buffers necessários ao usar MQTT para baixar arquivos:

```
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
};
```

Buffers necessários ao usar HTTP para baixar arquivos:

```
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
};
```

Onde:

```
    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

O segundo parâmetro de entrada para OTA_init é uma referência às interfaces OTA para o tipo _t. OtaInterfaces Este conjunto de interfaces deve ser passado para a Biblioteca OTA e inclui na interface do sistema operacional a interface MQTT, a interface HTTP e a interface da camada de abstração da plataforma.

Interface do sistema operacional OTA

A interface funcional do sistema operacional OTA é um conjunto APIs que deve ser implementado para que o dispositivo use a biblioteca OTA. As implementações de funções para essa interface são fornecidas à biblioteca OTA na aplicação do usuário. A biblioteca OTA chama as implementações de funções para executar funcionalidades que normalmente são fornecidas por um sistema operacional. Isto inclui o gerenciamento de eventos, temporizadores e alocação de memória. As implementações para FreeRTOS e POSIX são fornecidas com a biblioteca OTA.

Exemplo para FreeRTOS usando a porta FreeRTOS fornecida:

```
    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;
```

Exemplo para Linux usando a porta POSIX fornecida:

```
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;
```

Interface MQTT

A interface OTA MQTT é um conjunto APIs que deve ser implementado em uma biblioteca para permitir que a biblioteca OTA baixe um bloco de arquivos do serviço de streaming.

Exemplo de uso do CoreMQTT Agent APIs da demonstração OTA over MQTT -

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

Interface HTTP

A interface HTTP OTA é um conjunto APIs que deve ser implementado em uma biblioteca para permitir que a biblioteca OTA baixe um bloco de arquivo conectando-se a um URL pré-assinado e buscando blocos de dados. É opcional, a menos que a biblioteca OTA seja configurada para fazer o download de um URL pré-assinado em vez de um serviço de streaming.

Exemplo usando o CoreHTTP APIs da demonstração OTA sobre HTTP -

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

Interface PAL OTA

A interface OTA PAL é um conjunto APIs que deve ser implementado para que o dispositivo use a biblioteca OTA. A implementação específica do dispositivo para o OTA PAL é fornecida à biblioteca na aplicação do usuário. Estas funções são usadas pela biblioteca para armazenar, gerenciar e autenticar downloads.

```
    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;
```

Alterações em retorno:

O retorno é alterado do estado do atendente OTA para o código de erro OTA. Consulte a AWS IoT Over-the-air Atualização v3.0.0: OtaErr _t.

OTA_Shutdown

Na versão 1 da Biblioteca OTA, a API usada para desligar o Agente OTA era OTA_, AgentShutdown que agora foi alterada para OTA_Shutdown junto com as alterações nos parâmetros de entrada.

Desligamento do atendente OTA (versão 1)

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

Desligamento do atendente OTA (versão 3)

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

ticksToWait -

O número de tiques que aguardam até que o atendente OTA conclua o processo de desligamento. Se for definido como zero, a função retornará imediatamente sem esperar. O

estado real é retornado ao chamador. O atendente não fica em repouso nesse tempo, mas é usado para fazer loops ocupados.

Novo parâmetro de entrada:

unsubscribeFlag:

Sinalize para indicar se as operações de cancelamento de assinatura devem ser realizadas nos tópicos do trabalho quando o encerramento é chamado. Se o sinalizador for 0, as operações de cancelamento de inscrição não serão chamadas para tópicos de trabalho. Se a aplicação precisar cancelar a assinatura dos tópicos do trabalho, esse sinalizador deverá ser definido como 1 ao chamar OTA_Shutdown.

Alterações em retorno:

OtaState_t -

O enum para o estado do atendente OTA e seus membros é renomeado. Consulte a AWS IoT Over-the-air Atualização v3.0.0.

OTA_ GetState

O nome da API foi alterado de OTA_ AgentGetState para OTA_. GetState

Desligamento do atendente OTA (versão 1)

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

Desligamento do atendente OTA (versão 3)

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

Alterações em retorno:

OtaState_t -

O enum para o estado do atendente OTA e seus membros é renomeado. Consulte a AWS IoT Over-the-air Atualização v3.0.0.

OTA_ GetStatistics

Nova API única adicionada para estatísticas. Ele substitui o APIs OTA_GetPacketsReceived, OTA_, OTA_, OTA_GetPacketsQueued. GetPacketsProcessed GetPacketsDropped Além disso, na versão 3 da Biblioteca OTA, os números das estatísticas estão relacionados apenas ao trabalho atual.

Biblioteca OTA versão 1

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

Biblioteca OTA versão 3

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

pStatistics:

O parâmetro de entrada/saída para dados estatísticos, como pacotes recebidos, descartados, colocados em fila e processados para o trabalho atual.

Parâmetro de saída:

Código de erro OTA.

Exemplo 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

Os parâmetros de entrada são os mesmos, mas o código de erro OTA de retorno é renomeado e novos códigos de erro são adicionados na versão 3 da biblioteca OTA.

Biblioteca OTA versão 1

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

Biblioteca OTA versão 3

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

A enumeração do código de erro OTA de retorno é alterada e novos códigos de erro são adicionados. Consulte a AWS IoT Over-the-air Atualização v3.0.0: OtaErr _t.

Exemplo de uso:

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

OTA_ SetImageState

Os parâmetros de entrada são os mesmos e renomeados, o código de erro OTA de retorno é renomeado e novos códigos de erro são adicionados na versão 3 da biblioteca OTA.

Biblioteca OTA versão 1

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

Biblioteca OTA versão 3

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

O parâmetro de entrada é renomeado para OtaImageState _t. Consulte a AWS IoT Over-the-air Atualização v3.0.0.

A enumeração do código de erro OTA de retorno é alterada e novos códigos de erro são adicionados. Consulte a AWS IoT Over-the-air Atualização v3.0.0/ OtaErr_t.

Exemplo de uso:

```
OtaErr_t otaErr = OtaErrNone;
otaErr = OTA_SetImageState( OtaImageStateAccepted );
```

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

OTA_ GetImageState

Os parâmetros de entrada são os mesmos. O enum de retorno é renomeado na versão 3 da biblioteca OTA.

Biblioteca OTA versão 1

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

Biblioteca OTA versão 3

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

O enum de retorno é renomeado para OtaImageState _t. Consulte a AWS IoT Over-the-air Atualização v3.0.0: OtaImageState _t.

Exemplo de uso:

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

OTA_Suspend

Os parâmetros de entrada são os mesmos, o código de erro OTA de retorno é renomeado e novos códigos de erro são adicionados na versão 3 da biblioteca OTA.

Biblioteca OTA versão 1

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

Biblioteca OTA versão 3

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

A enumeração do código de erro OTA de retorno é alterada e novos códigos de erro são adicionados. Consulte a AWS IoT Over-the-air Atualização v3.0.0: OtaErr _t.

Exemplo de uso:

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

OTA_Resume

O parâmetro de entrada para a conexão é removido quando a conexão é tratada na demonstração/aplicação OTA, o código de erro OTA de retorno é renomeado e novos códigos de erro são adicionados na versão 3 da biblioteca OTA.

Biblioteca OTA versão 1

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

Biblioteca OTA versão 3

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

A enumeração do código de erro OTA de retorno é alterada e novos códigos de erro são adicionados. Consulte a AWS IoT Over-the-air Atualização v3.0.0: OtaErr _t.

Exemplo de uso:

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

OTA_ CheckForUpdate

Os parâmetros de entrada são os mesmos, o código de erro OTA de retorno é renomeado e novos códigos de erro são adicionados na versão 3 da biblioteca OTA.

Biblioteca OTA versão 1

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

Biblioteca OTA versão 3

```
OtaErr_t OTA_CheckForUpdate( void )
```

A enumeração do código de erro OTA de retorno é alterada e novos códigos de erro são adicionados. Consulte a AWS IoT Over-the-air Atualização v3.0.0: OtaErr _t.

OTA_ EventProcessingTask

Esta é uma nova API e é o principal loop de eventos para lidar com eventos para atualizações do OTA. Ele deve ser chamado pela tarefa da aplicação. Esse loop continuará manipulando e executando eventos recebidos pela atualização OTA até que essa tarefa seja encerrada pela aplicação.

Biblioteca OTA versão 3

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

Exemplo 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 );
}
```

Exemplo 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

Esta é uma nova API que adiciona o evento ao final da fila de eventos e também é usada por módulos OTA internos para sinalizar a tarefa do atendente.

Biblioteca OTA versão 3

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

Exemplo de uso:

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

Como integrar a biblioteca OTA como um submódulo em sua aplicação

Se você quiser integrar a biblioteca OTA em sua própria aplicação, você pode usar o comando git submodule. Os submódulos Git permitem que você mantenha um repositório Git como um subdiretório de outro repositório Git. A versão 3 da biblioteca OTA é mantida no repositório 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 obter mais informações, consulte Como integrar o atendente OTA em sua aplicação no Manual do usuário do FreeRTOS.

Referências

- OTAv1.
- OTAv3.

Migrando da versão 1 para a versão 3 para a porta OTA PAL

A Biblioteca de atualizações sem fios introduziu algumas mudanças na estrutura de pastas e no posicionamento das configurações exigidas pela biblioteca e pelas aplicações de demonstração. Para que aplicações OTA projetadas para funcionar com a v1.2.0 migrem para a v3.0.0 da biblioteca, você deve atualizar as assinaturas da função da porta PAL e incluir arquivos de configuração adicionais, conforme descrito neste guia de migração.

Alterações em OTA PAL

- O nome do diretório da porta OTA PAL foi atualizado de `ota` para `ota_pal_for_aws`. Essa pasta deve conter dois arquivos: `ota_pal.c` e `ota_pal.h`. O arquivo de cabeçalho PAL `libraries/freertos_plus/aws/ota/src/aws_iot_ota_pal.h` foi excluído da biblioteca OTA e deve ser definido dentro da porta.
- Os códigos de retorno (`OTA_Err_t`) são convertidos em uma enumeração `OTAMainStatus_t`. Consulte ota_platform_interface.h para obter os códigos de retorno convertidos. Macros auxiliares também são fornecidas para combinar os códigos `OtaPalMainStatus` e `OtaPalSubStatus`. e extrair `OtaMainStatus` de `OtaPalStatus` e similares.
- Registro em log na PAL
  - Remova a macro `DEFINE_OTA_METHOD_NAME`.
  - Anteriormente: `OTA_LOG_L1( "[%s] Receive file created.\r\n", OTA_METHOD_NAME );`.
  - Atualizado: use `LogInfo(( "Receive file created."));`, `LogDebug`, `LogWarn` e `LogError` para o log apropriado.
- Variável `cOTA_JSON_FileSignatureKey` alterada para `OTA_JsonFileSignatureKey`.

Funções

As assinaturas da função são definidas em `ota_pal.h` e começam com o prefixo `otaPal` em vez de `prvPAL`.

> **Note**
>
> O nome exato do PAL é tecnicamente aberto, mas para ser compatível com os testes de qualificação, o nome deve estar de acordo com os especificados abaixo.

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

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

  Observações: cria um novo arquivo de recebimento para os blocos de dados à medida que eles chegam.

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

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

  Grava um bloco de dados no arquivo especificado no deslocamento determinado.

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

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

  Observações: ativa a imagem MCU mais recente recebida via OTA.

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

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

  Observações: reinicia o dispositivo.

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

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

Observações: autentica e fecha o arquivo de recebimento subjacente no contexto OTA especificado.

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

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

  Observações: interrompe uma transferência OTA.

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

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

  Observações: tenta definir o estado da imagem de atualização OTA.

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

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

  Observações: obtém o estado da imagem de atualização OTA.

Tipos de dados

- Versão 1: `OTA_PAL_ImageState_t`

  Arquivo , : `aws_iot_ota_agent.h`

  Versão 3: `OtaPalImageState_t`

  Arquivo , : `ota_private.h`

  Observações: o estado da imagem definido pela implementação da plataforma.

- Versão 1: `OTA_Err_t`

  Arquivo , : `aws_iot_ota_agent.h`

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

Arquivo `ota.h`, `ota_platform_interface.h`

Observações: v1: esses macros definiam um número inteiro de 32 não assinado. v3: enumeração especializada representando o tipo de erro e associada a um código de erro.

- Versão 1: `OTA_FileContext_t`

  Arquivo , : `aws_iot_ota_agent.h`

  Versão 3: `OtaFileContext_t`

  Arquivo , : `ota_private.h`

  Observações: v1: contém uma enumeração e buffers para os dados. v3: contém variáveis adicionais de comprimento de dados.
- Versão 1: `OTA_ImageState_t`

  Arquivo , : `aws_iot_ota_agent.h`

  Versão 3: `OtaImageState_t`

  Arquivo , : `ota_private.h`

  Observações: estados de imagem OTA

Alterações de configuração

O arquivo `aws_ota_agent_config.h` foi renomeado para `ota_config.h`, o que altera as guards incluídos de `_AWS_OTA_AGENT_CONFIG_H_` para `OTA_CONFIG_H_`.

- O arquivo `aws_ota_codesigner_certificate.h` foi excluído.
- Incluiu a nova pilha de registro em log para imprimir mensagens de depuração:

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

- Configuração constante adicionada:

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

Novo arquivo: ota_demo_config.h contém as configurações exigidas pela demonstração OTA, como o certificado de assinatura de código e a versão da aplicação.

- `signingcredentialSIGNING_CERTIFICATE_PEM` que foi definido em `demos/include/aws_ota_codesigner_certificate.h` foi movido para `ota_demo_config.h` as `otapalconfigCODE_SIGNING_CERTIFICATE` e pode ser acessado a partir dos arquivos PAL como:

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

  O arquivo `aws_ota_codesigner_certificate.h` foi excluído.

- As macros`APP_VERSION_BUILD`, `APP_VERSION_MINOR`, `APP_VERSION_MAJOR` foram adicionadas a `ota_demo_config.h`. Os arquivos antigos contendo as informações da versão foram removidos, por exemplo `tests/include/aws_application_version.h`, `libraries/c_sdk/standard/common/include/iot_appversion32.h`, `demos/demo_runner/aws_demo_version.c`.

Alterações nos testes OTA PAL

- O grupo de teste "FULL_OTA_Agent" foi removido junto com todos os arquivos relacionados. Esse grupo de teste era exigido anteriormente para qualificação. Esses testes foram para a biblioteca OTA e não específicos para a porta OTA PAL. A biblioteca OTA agora tem cobertura de teste completa hospedada no repositório OTA, portanto, esse grupo de teste não é mais necessário.
- Os grupos de teste "Full_OTA_CBOR" e "Quarantine_OTA_CBOR" fora removidos, bem como todos os arquivos relacionados. Esses testes não faziam parte dos testes de qualificação. As funcionalidades abordadas por esses testes agora estão sendo testadas no repositório OTA.
- Os arquivos de teste foram movidos do diretório da biblioteca para o diretório `tests/integration_tests/ota_pal`.
- Os testes de qualificação OTA PAL foram atualizados para usar a versão 3.0.0 da API da biblioteca OTA.
- A forma como os testes OTA PAL acessam o certificado de assinatura de código para testes foi atualizada. Anteriormente, havia um arquivo de cabeçalho dedicado para a credencial de assinatura de código. Esse não é mais o caso da nova versão da biblioteca. O código de teste espera que essa variável seja definida em `ota_pal.c`. O valor é atribuído a uma macro definida no arquivo de configuração OTA específico da plataforma.

Lista de verificação

Use essa lista de verificação para seguir as etapas necessárias para a migração:

- Atualize o nome da pasta da porta OTA PAL de `ota` para `ota_pal_for_aws`.
- Adicione o arquivo `ota_pal.h` com as funções mencionadas acima. Para ver um exemplo de arquivo `ota_pal.h`, consulte GitHub.
- Adicione os arquivos de configuração:
  - Altere o nome do arquivo de `aws_ota_agent_config.h` para (ou crie) `ota_config.h`.
    - Adicionar:

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

    - Inclusão:

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

  - Copie os arquivos acima para a pasta `aws_test config` e substitua as inclusões de `ota_demo_config.h` por `aws_test_ota_config.h`.
  - Adicione um arquivo `ota_demo_config.h`.
  - Adicione um arquivo `aws_test_ota_config.h`.
- Faça as seguintes alterações em `ota_pal.c`:
  - Atualize as inclusões com os nomes mais recentes dos arquivos da biblioteca OTA.
  - Remova a macro `DEFINE_OTA_METHOD_NAME`.
  - Atualize as assinaturas das funções OTA PAL.
  - Atualize o nome da variável de contexto do arquivo de `C` para `pFileContext`.
  - Atualize a estrutura `OTA_FileContext_t` e todas as variáveis relacionadas.
  - Atualize `cOTA_JSON_FileSignatureKey` para `OTA_JsonFileSignatureKey`.
  - Atualize os tipos `OTA_PAL_ImageState_t` e `Ota_ImageState_t`.
  - Atualize o tipo de erro e os valores.
  - Atualize as macros de impressão para usar a pilha de registro em log.
  - Atualize o `signingcredentialSIGNING_CERTIFICATE_PEM` para ser `otapalconfigCODE_SIGNING_CERTIFICATE`.
  - Atualize os comentários de função `otaPal_CheckFileSignature` e `otaPal_ReadAndAssumeCertificate`.
- Para atualizar o arquivo `CMakeLists.txt`.
- Atualize os projetos do IDE.

Histórico do documento

A tabela a seguir descreve o histórico da documentação do Guia de portabilidade do FreeRTOS e do Guia de qualificação do FreeRTOS.

Data	Versão da documentação	Histórico de alterações	Versão do FreeRTOS
Maio de 2022	Guia de portabilidade do FreeRTOS Guia de qualificação do FreeRTOS	• Atualizou os testes existentes, adicionou novos testes e removeu testes redundantes com base nas Bibliotecas de suporte de longo prazo (LTS) do FreeRTOS. Para obter mais informações, consulte Testes de integração das bibliotecas do FreeRTOS 202205.00 no GitHub. • Atualização de Fluxograma de portabilidade do FreeRTOS. • Adicionou um novo Fazer portabilidade da Network Transport Interface.	202012.04-LTS 202112.00

Data	Versão da documentação	Histórico de alterações	Versão do FreeRTOS
		• Agora, Portando a biblioteca de atualização AWS IoT over-the-air (OTA) é necessário para a qualificação. • Guia de portabilidade de abstração de Wi-Fi e TLS removido, pois ele não é mais necessário. • Consulte asúltimas mudanças para obter mais atualizações sobre a qualificação dos FreeRTOS.	

Data	Versão da documentação	Histórico de alterações	Versão do FreeRTOS
Julho de 2022	202107.00 (Guia de portabilidade) 202107.00 (Guia de qualificação)	• Versão 202107.00 • Portando a biblioteca de atualização AWS IoT over-the-air (OTA) alterado • adicionado Como migrar da versão 1 para a versão 3 para as aplicações OTA • adicionado Migrando da versão 1 para a versão 3 para a porta OTA PAL	202107.00
Dezembro de 2020	202012.00 (Guia de portabilidade) 202002.00 (Guia de qualificação)	• Versão 202012.00 • adicionado Configuração da biblioteca coreHTTP • adicionado Portabilidade da biblioteca Cellular Interface	202012.00
Novembro de 2020	202011.00 (Guia de portabilidade) 202011.00 (Guia de qualificação)	• Versão 202011.00 • adicionado Configuração da biblioteca coreMQTT	202011.00

Data	Versão da documentação	Histórico de alterações	Versão do FreeRTOS
Julho de 2020	202007.00 (Guia de portabilidade) 202007.00 (Guia de qualificação)	• Versão 202007.00	202007.00
18 de fevereiro de 2020	202002.00 (Guia de transferência) 202002.00 (Guia de qualificação)	• Versão 202002.00 • Agora, o Amazon FreeRTOS é FreeRTOS	202002.00
17 de dezembro de 2019	201912.00 (Guia de transferência) 201912.00 (Guia de qualificação)	• Versão 201912.00 • Portabilidade das bibliotecas de E/S comuns adicionada.	201912.00
29 de outubro de 2019	201910.00 (Guia de transferência) 201910.00 (Guia de qualificação)	• Versão 201910.00 • Atualização das informações sobre portabilidade do gerador de números aleatórios.	201910.00
26 de agosto de 2019	201908.00 (Guia de transferência) 201908.00 (Guia de qualificação)	• Versão 201908.00 • Configuração da biblioteca de cliente HTTPS para testes adicionada Atualização de Portabilidade da biblioteca corePKCS11	201908.00

Data	Versão da documentação	Histórico de alterações	Versão do FreeRTOS
17 de junho de 2019	201906.00 (Guia de portabilidade) 201906.00 (Guia de qualificação)	• Versão 201906.00 • Diretório estrutura do atualizado	201906.00 principal
21 de maio de 2019	1.4.8 (Guia de portabilidade) 1.4.8 (Guia de qualificação)	• Documentação de portabilidade movida para o Guia de portabilidade do FreeRTOS • Documentação de qualificação movida para o Guia de qualificação do FreeRTOS	1.4.8
25 de fevereiro de 2019	1.1.6	• Instruções de download e configuração removidas do Apêndice Modelo do guia de conceitos básicos (página 84)	1.4.5 1.4.6 1.4.7
27 de dezembro de 2018	1.1.5	• Apêndice Lista de verificação para qualificação atualizado com o requisito do CMake (página 70)	1.4.5 1.4.6

Data	Versão da documentação	Histórico de alterações	Versão do FreeRTOS
12 de dezembro de 2018	1.1.4	• Instruções de portabilidade lwIP adicionadas ao apêndice Portabilidade TCP/IP (página 31)	1.4.5
26 de novembro de 2018	1.1.3	• Inclusão do apêndice de portabilidade da biblioteca Bluetooth Low Energy (página 52) • AWS IoT Device Tester adicionado para informações de teste do FreeRTOS em todo o documento • Link do CMake adicionado em Informações para listagem no apêndice Console do FreeRTOS (página 85)	1.4.4

Data	Versão da documentação	Histórico de alterações	Versão do FreeRTOS
7 de novembro de 2018	1.1.2	• Instruções de portabilidade da interface PAL PKCS #11 atualizadas no apêndice de portabilidade PKCS #11 (página 38) • Caminho atualizado para `CertificateConfigurator.html` página 76) • Apêndice Modelo do guia de conceitos básicos atualizado (página 80)	1.4.3

Data	Versão da documentação	Histórico de alterações	Versão do FreeRTOS
8 de outubro de 2018	1.1.1	• Nova coluna "Necessário para AFQP" para `aws_test_runner_config.h` testar a tabela de configuração (página 16) • Caminho do diretório do módulo do Unity atualizado na seção Criar o projeto de teste (página 14) • Gráfico "Ordem de portabilidade recomendada" atualizado (página 22) • Certificado de cliente e nomes de variáveis de chave atualizados no apêndice TLS, Configuração de teste (página 40) • Caminhos de arquivo alterados no apêndice Portabilidade de Secure Sockets, Configuração de teste (página 34);	1.4.2

Data	Versão da documentação	Histórico de alterações	Versão do FreeRTOS
		apêndice Portabilidade do TLS, Configuração de teste (página 40); e apêndice Configuração do servidor TLS (página 57)	
27 de agosto de 2018	1.1.0	• Apêndice Portabilidade das atualizações OTA adicionado (página 47) • Apêndice Portabilidade do bootloader adicionado (página 51)	1.4.0 1.4.1

Data	Versão da documentação	Histórico de alterações	Versão do FreeRTOS
9 de agosto de 2018	1.0.1	• Gráfico "Ordem de portabilidade recomendada" atualizado (página 22) • Apêndice Portabilidade PKCS #11 atualizado (página 36) • Caminhos de arquivo alterados no apêndice Portabilidade do TLS, Configuração de teste (página 40) e apêndice Configuração do servidor TLS, etapa 9 (página 51) • Hiperlinks corrigidos no apêndice Portabilidade MQTT, Pré-requisitos (página 45) • Instruções de configuração da AWS CLI adicionadas aos exemplos no apêndice Instruções para criar um BYOC (página 57)	1.3.1 1.3.2

Data	Versão da documentação	Histórico de alterações	Versão do FreeRTOS
31 de julho de 2018	1.0.0	Versão inicial do Guia do programa de qualificação do FreeRTOS	1.3.0

As traduções são geradas por tradução automática. Em caso de conflito entre o conteúdo da tradução e da versão original em inglês, a versão em inglês prevalecerá.