aws

Portierungshandbuch

FreeRTOS

FreeRTOS: Portierungshandbuch

Table of Contents

FreeRTOS RTOS-Portierung

Was ist FreeRTOS

FreeRTOS wurde in Zusammenarbeit mit den weltweit führenden Chipherstellern über einen Zeitraum von 20 Jahren entwickelt und wird nun alle 170 Sekunden heruntergeladen. Es ist ein marktführendes Echtzeitbetriebssystem (RTOS) für Mikrocontroller und kleine Mikroprozessoren. FreeRTOS wird unter der MIT-Open-Source-Lizenz kostenlos vertrieben und umfasst einen Kernel und eine wachsende Anzahl von Bibliotheken, die für den Einsatz in allen Branchen geeignet sind. FreeRTOS wurde mit einem Schwerpunkt auf Zuverlässigkeit und Benutzerfreundlichkeit entwickelt. FreeRTOS umfasst Bibliotheken für Konnektivitäts-, Sicherheits- und over-the-air (OTA-) Updates sowie Demo-Anwendungen, die FreeRTOS-Funktionen auf qualifizierten Boards demonstrieren.

Weitere Informationen finden Sie auf FreeRTOS.org.

Portierung von FreeRTOS auf Ihr IoT-Board

Sie müssen die FreeRTOS-Softwarebibliotheken je nach ihren Funktionen und Ihrer Anwendung auf Ihr Mikrocontroller-basiertes Board portieren.

Um FreeRTOS auf dein Gerät zu portieren

1. Folgen Sie den Anweisungen unterFreeRTOS für die Portierung herunterladen, um die neueste Version von FreeRTOS für die Portierung herunterzuladen.
2. Folgen Sie den Anweisungen unterDeinen Workspace und dein Projekt für die Portierung einrichten, um die Dateien und Ordner in Ihrem FreeRTOS-Download zum Portieren und Testen zu konfigurieren.
3. Folgen Sie den Anweisungen unterPortierung der FreeRTOS-Bibliotheken, um die FreeRTOS-Bibliotheken auf Ihr Gerät zu portieren. Jedes Portierungsthema enthält auch Anweisungen zum Testen der Ports.

Portierung FAQs

Was ist ein FreeRTOS-Port?

Ein FreeRTOS-Port ist eine plattformspezifische Implementierung APIs für die erforderlichen FreeRTOS-Bibliotheken und den FreeRTOS-Kernel, den Ihre Plattform unterstützt. Der Port

ermöglicht die Arbeit APIs auf dem Board und implementiert die erforderliche Integration mit den Gerätetreibern, BSPs die vom Plattformanbieter bereitgestellt werden. Ihr Port sollte auch alle Konfigurationsanpassungen (z.B. Taktfrequenz, Stapelgröße, Heap-Größe) beinhalten, die von der Karte benötigt werden.

Wenn Sie Fragen zur Portierung haben, die auf dieser Seite oder im Rest des FreeRTOS Porting Guide nicht beantwortet werden, schauen Sie sich bitte die verfügbaren FreeRTOS-Supportoptionen an.

FreeRTOS für die Portierung herunterladen

Laden Sie die neueste Version von FreeRTOS oder Long Term Support (LTS) von freertos.org herunter oder klonen Sie sie von (FreeRTOS-LTS) oder GitHub (FreeRTOS).

> **Note**
>
> Wir empfehlen, das Repository zu klonen. Das Klonen erleichtert es Ihnen, Updates für den Hauptzweig abzurufen, wenn sie in das Repository übertragen werden.

Alternativ können Sie die einzelnen Bibliotheken aus dem FreeRTOS- oder FreeRTOS-LTS-Repository submodulieren. Stellen Sie jedoch sicher, dass die Bibliotheksversionen mit der Kombination übereinstimmen, die in der `manifest.yml` Datei im FreeRTOS- oder FreeRTOS-LTS-Repository aufgeführt ist.

Nachdem du FreeRTOS heruntergeladen oder geklont hast, kannst du damit beginnen, die FreeRTOS-Bibliotheken auf dein Board zu portieren. Weitere Informationen finden Sie unter Deinen Workspace und dein Projekt für die Portierung einrichten und dann unter Portierung der FreeRTOS-Bibliotheken.

Deinen Workspace und dein Projekt für die Portierung einrichten

Gehen Sie wie folgt vor, um Ihren Workspace und Ihr Projekt einzurichten:

- Verwenden Sie eine Projektstruktur und ein Build-System Ihrer Wahl, um die FreeRTOS-Bibliotheken zu importieren.
- Erstellen Sie ein Projekt mit einer integrierten Entwicklungsumgebung (IDE) und einer Toolchain, die von Ihrem Board unterstützt werden.
- Nehmen Sie die Board Support Packages (BSP) und die Board-spezifischen Treiber in Ihr Projekt auf.

Sobald Ihr Workspace eingerichtet ist, können Sie mit der Portierung einzelner FreeRTOS-Bibliotheken beginnen.

Portierung der FreeRTOS-Bibliotheken

Bevor Sie mit der Portierung beginnen, folgen Sie den Anweisungen unter. Deinen Workspace und dein Projekt für die Portierung einrichten

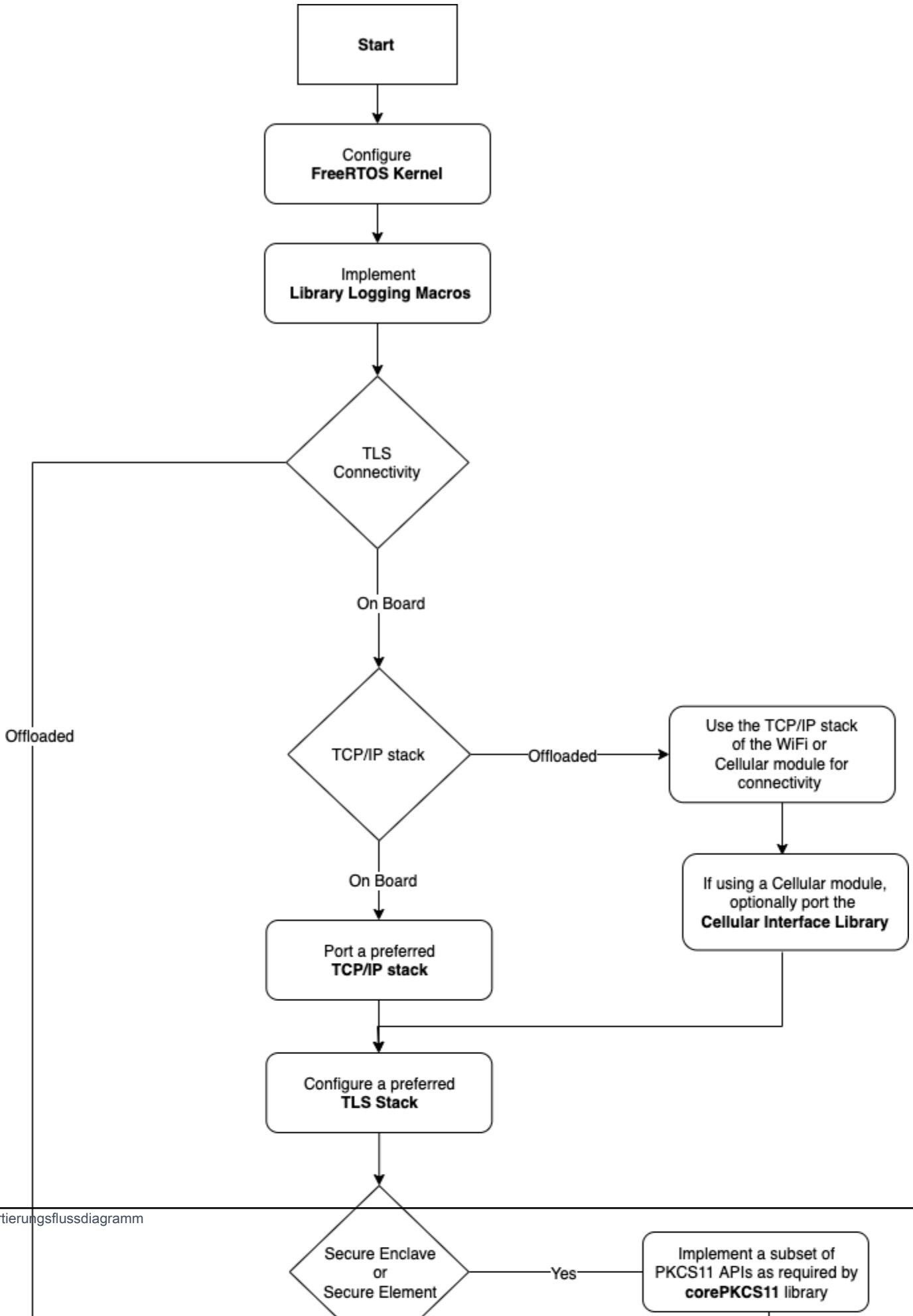
Das Flussdiagramm zur FreeRTOS-Portierung beschreibt die Bibliotheken, die für die Portierung benötigt werden.

Folgen Sie den Anweisungen in den folgenden Themen, um FreeRTOS auf Ihr Gerät zu portieren.

1. Konfigurieren eines FreeRTOS-Kernel-Ports
2. Implementierung der Bibliotheksprotokollierungsmakros
3. Portieren eines TCP/IP-Stacks
4. Portierung der Netzwerktransportschnittstelle
5. Portierung der PKCS11 Kernbibliothek
6. Konfiguration der CoreMQTT-Bibliothek
7. Konfiguration der CoreHTTP-Bibliothek
8. Portierung der AWS IoT over-the-air (OTA-) Update-Bibliothek
9. Portierung der Cellular Interface-Bibliothek

Flussdiagramm zur FreeRTOS-Portierung

Verwenden Sie das folgende Portierungs-Flussdiagramm als visuelle Hilfe, wenn Sie FreeRTOS auf Ihr Board portieren.


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

Konfigurieren eines FreeRTOS-Kernel-Ports

Dieser Abschnitt enthält Anweisungen zur Integration eines Ports des FreeRTOS-Kernels in ein FreeRTOS-Port-Testing-Projekt. Eine Liste der verfügbaren Kernel-Ports finden Sie unter FreeRTOS-Kernelports.

FreeRTOS verwendet den FreeRTOS-Kernel für Multitasking und Intertask-Kommunikation. Weitere Informationen finden Sie in den FreeRTOS-Kernel-Grundlagen im FreeRTOS-Benutzerhandbuch und unter FreeRTOS.org.

> **Note**
>
> Die Portierung des FreeRTOS-Kernels auf eine neue Architektur ist in dieser Dokumentation nicht enthalten. Bei Interesse wenden Sie sich an das FreeRTOS Engineering-Team.
> Für das FreeRTOS-Qualifizierungsprogramm werden nur bestehende FreeRTOS-Kernelports unterstützt. Änderungen an diesen Ports werden innerhalb des Programms nicht akzeptiert. Weitere Informationen finden Sie in der FreeRTOS-Kernel-Port-Richtlinie.

Voraussetzungen

Um den FreeRTOS-Kernel für die Portierung einzurichten, benötigen Sie Folgendes:

- Ein offizieller FreeRTOS-Kernelport oder von FreeRTOS unterstützte Ports für die Zielplattform.
- Ein IDE-Projekt mit den richtigen FreeRTOS Kernel-Port-Dateien für die Zielplattform und den Compiler. Weitere Informationen zum Einrichten eines Testprojekts finden Sie unter Deinen Workspace und dein Projekt für die Portierung einrichten.

Konfigurieren des FreeRTOS-Kernels

Der FreeRTOS-Kernel wird mithilfe einer Konfigurationsdatei namens angepasst. `FreeRTOSConfig.h` Diese Datei spezifiziert anwendungsspezifische Konfigurationseinstellungen für den Kernel. Eine Beschreibung der einzelnen Konfigurationsoptionen finden Sie unter Anpassung auf FreeRTOS.org.

Um den FreeRTOS-Kernel so zu konfigurieren, dass er mit Ihrem Gerät funktioniert`FreeRTOSConfig.h`, fügen Sie alle zusätzlichen FreeRTOS-Konfigurationen hinzu und ändern Sie sie.

Eine Beschreibung der einzelnen Konfigurationsoptionen finden Sie unter Anpassungskonfigurationen auf FreeRTOS.org.

Testen

- Führen Sie eine einfache FreeRTOS-Aufgabe aus, um eine Nachricht an der seriellen Ausgabekonsole zu protokollieren.
- Stellen Sie sicher, dass die Nachricht wie erwartet auf der Konsole ausgegeben wird.

Implementierung der Bibliotheksprotokollierungsmakros

Die FreeRTOS-Bibliotheken verwenden die folgenden Logging-Makros, die in aufsteigender Reihenfolge ihrer Ausführlichkeit aufgeführt sind.

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

Eine Definition für alle Makros muss bereitgestellt werden. Die Empfehlungen lauten wie folgt:

- Makros sollten die Protokollierung C89 im Stil unterstützen.
- Die Protokollierung sollte threadsicher sein. Protokollzeilen von mehreren Aufgaben dürfen nicht miteinander verschachtelt werden.
- APIs Die Protokollierung darf nicht blockieren und muss Anwendungsaufgaben vom Blockieren bei I/O befreien.

Einzelheiten zur Implementierung finden Sie in der Logging-Funktionalität auf FreeRTOS.org. In diesem Beispiel sehen Sie eine Implementierung.

Testen

- Führen Sie einen Test mit mehreren Aufgaben durch, um sicherzustellen, dass sich die Protokolle nicht ineinander überlagern.
- Führen Sie einen Test durch, um sicherzustellen, dass die Protokollierung APIs keine I/O blockiert.

- Testen Sie Makros für die Protokollierung anhand verschiedener Standards, z. B. der Protokollierung `C89,C99` im Stil.
- Testen Sie Protokollierungsmakros, indem Sie verschiedene Protokollebenen festlegen, z. B.`Debug`, `InfoError`, und. `Warning`

Portieren eines TCP/IP-Stacks

Dieser Abschnitt enthält Anweisungen zum Portieren und Testen der integrierten TCP/IP stacks. If your platform offloads TCP/IP und TLS-Funktionalität auf einen separaten Netzwerkprozessor oder ein separates Netzwerkmodul. Sie können diesen Abschnitt zur Portierung überspringen und besuchenPortierung der Netzwerktransportschnittstelle.

FreeRTOS+TCP ist ein nativer TCP/IP stack for the FreeRTOS kernel. FreeRTOS+TCP is developed and maintained by the FreeRTOS engineering team and is the recommended TCP/IP Stack zur Verwendung mit FreeRTOS. Weitere Informationen finden Sie unter Portierung von FreeRTOS+TCP. Alternativ können Sie den TCP/IP-Stack LwIP eines Drittanbieters verwenden. Die Testanweisung in diesem Abschnitt verwendet die Transportschnittstellentests für TCP-Klartext und ist nicht vom spezifisch implementierten TCP/IP-Stack abhängig.

Portierung von FreeRTOS+TCP

FreeRTOS+TCP ist ein nativer TCP/IP-Stack für den FreeRTOS-Kernel. Weitere Informationen finden Sie unter FreeRTOS.org.

Voraussetzungen

Um die FreeRTOS+TCP-Bibliothek zu portieren, benötigen Sie Folgendes:

- Ein IDE-Projekt, das die vom Hersteller bereitgestellten Ethernet- oder Wi-Fi-Treiber enthält.

  Weitere Informationen zum Einrichten eines Testprojekts finden Sie unter Deinen Workspace und dein Projekt für die Portierung einrichten.
- Eine validierte Konfiguration des FreeRTOS-Kernels.

  Informationen zur Konfiguration des FreeRTOS-Kernels für Ihre Plattform finden Sie unter Konfigurieren eines FreeRTOS-Kernel-Ports.

Portierung

Bevor Sie mit der Portierung der FreeRTOS+TCP-Bibliothek beginnen, überprüfen Sie im GitHubVerzeichnis, ob bereits ein Port zu Ihrem Board existiert.

Wenn kein Port vorhanden ist, führen Sie die folgenden Schritte aus:

1. Folgen Sie den Anweisungen in Portierung von FreeRTOS+TCP zu einem anderen Microcontroller auf FreeRTOS.org, um FreeRTOS+TCP auf Ihr Gerät zu portieren.
2. Falls erforderlich, folgen Sie den Anweisungen in Portierung von FreeRTOS+TCP auf einen neuen Embedded C Compiler auf FreeRTOS.org, um FreeRTOS+TCP auf einen neuen Compiler zu portieren.
3. Implementieren Sie einen neuen Port, der die vom Hersteller bereitgestellten Ethernet- oder Wi-Fi-Treiber verwendet, in einer Datei namens. `NetworkInterface.c` Eine Vorlage finden Sie im GitHubRepository.

Nachdem Sie einen Port erstellt haben oder falls bereits ein Port vorhanden ist`FreeRTOSIPConfig.h`, erstellen und bearbeiten Sie die Konfigurationsoptionen, sodass sie für Ihre Plattform korrekt sind. Weitere Informationen zu den Konfigurationsoptionen finden Sie in FreeRTOS+TCP-Konfiguration unter FreeRTOS.org.

Testen

Unabhängig davon, ob Sie die FreeRTOS+TCP-Bibliothek oder eine Bibliothek eines Drittanbieters verwenden, folgen Sie zum Testen den folgenden Schritten:

- Stellen Sie eine Implementierung für `connect/disconnect/send/receive` APIs In-Transport-Schnittstellentests bereit.
- Richten Sie einen Echoserver im TCP-Verbindungsmodus im Klartext-Modus ein und führen Sie Transportschnittstellentests durch.

> **Note**
>
> Um ein Gerät offiziell für FreeRTOS zu qualifizieren, müssen Sie, wenn Ihre Architektur die Portierung eines TCP/IP-Softwarestacks erfordert, den portierten Quellcode des Geräts anhand von Transportschnittstellentests im Klartext-TCP-Verbindungsmodus mit validieren. AWS IoT Device Tester Folgen Sie den Anweisungen unter Using AWS IoT Device Tester for

FreeRTOS im FreeRTOS User Guide, um die Port-Validierung einzurichten AWS IoT Device Tester . Um den Port einer bestimmten Bibliothek zu testen, muss die korrekte Testgruppe in der Datei `device.json` im Ordner `configs` in Device Tester aktiviert sein.

Portierung der PKCS11 Kernbibliothek

Der Public Key Cryptography Standard #11 definiert eine plattformunabhängige API zur Verwaltung und Verwendung kryptografischer Token. PKCS 11 bezieht sich auf den Standard und die durch ihn definierten. APIs Die kryptografische API PKCS #11 abstrahiert die Speicherung von Schlüsseln, das Abrufen/Setzen von Eigenschaften für kryptografische Objekte und die Sitzungssemantik. Sie wird häufig zur Manipulation gängiger kryptografischer Objekte verwendet. Seine Funktionen ermöglichen es Anwendungssoftware, kryptografische Objekte zu verwenden, zu erstellen, zu ändern und zu löschen, ohne dass diese Objekte dem Speicher der Anwendung zugänglich gemacht werden.

FreeRTOS-Bibliotheken und Referenzintegrationen verwenden eine Teilmenge des Schnittstellenstandards PCKS #11, wobei der Schwerpunkt auf Operationen liegt, die asymmetrische Schlüssel, Zufallszahlengenerierung und Hashing beinhalten. In der folgenden Tabelle sind die Anwendungsfälle und die für die Unterstützung erforderlichen PKCS #11 aufgeführt. APIs

Anwendungsfälle

Anwendungsfall	Erforderliche PKCS #11 -API-Familie
Alle	Sitzung initialisieren, abschließen, Sitzung öffnen/schließen, Anmelden GetSlotList
Bereitstellung	GenerateKeyPair, CreateObject, DestroyObject, InitToken, GetTokenInfo
TLS	Zufällig, Signieren, FindObject GetAttributeValue
FreeRTOS+TCP	Zufällig, Digest
OTA	Verifizieren, Digest, FindObject GetAttributeValue

Wann sollte ein vollständiges PKCS #11 -Modul implementiert werden

Die Speicherung privater Schlüssel in einem universellen Flash-Speicher kann in Evaluierungs- und Rapid-Prototyping-Szenarien nützlich sein. Wir empfehlen die Verwendung spezieller kryptografischer Hardware, um die Gefahr von Datendiebstahl und Geräteduplizierung in Produktionsszenarien zu verringern. Die kryptographische Hardware enthält Komponenten mit Funktionen, die den Export kryptographischer Geheimschlüssel verhindern. Um dies zu unterstützen, müssen Sie eine Teilmenge von PKCS #11 implementieren, die für die Arbeit mit FreeRTOS-Bibliotheken erforderlich ist, wie in der obigen Tabelle definiert.

Wann sollte FreeRTOS Core verwendet werden PKCS11

Die PKCS11 Kernbibliothek enthält eine softwarebasierte Implementierung der PKCS #11 - Schnittstelle (API), die die von Mbed TLS bereitgestellten kryptografischen Funktionen verwendet. Dies ist für schnelle Prototyping- und Evaluierungsszenarien vorgesehen, in denen die Hardware nicht über eine spezielle kryptografische Hardware verfügt. In diesem Fall müssen Sie nur PKCS11 Core-PAL implementieren, damit die PKCS11 softwarebasierte Kernimplementierung mit Ihrer Hardwareplattform funktioniert.

Core portieren PKCS11

Sie benötigen Implementierungen zum Lesen und Schreiben kryptografischer Objekte in nichtflüchtigen Speicher (NVM), z. B. im integrierten Flash-Speicher. Kryptografische Objekte müssen in einem NVM-Abschnitt gespeichert werden, der nicht initialisiert und bei einer Neuprogrammierung des Geräts nicht gelöscht wird. Benutzer der PKCS11 Kernbibliothek stellen den Geräten Anmeldeinformationen zur Verfügung und programmieren das Gerät anschließend mit einer neuen Anwendung neu, die über die Kernschnittstelle auf diese Anmeldeinformationen zugreift. PKCS11 Die wichtigsten PKCS11 PAL-Ports müssen einen Speicherort für Folgendes bereitstellen:

- Das Geräte-Client-Zertifikat
- Der private Schlüssel des Geräteclients
- Der öffentliche Schlüssel des Geräteclients
- Eine vertrauenswürdige Root-CA
- Ein öffentlicher Schlüssel zur Codeverifizierung (oder ein Zertifikat, das den öffentlichen Schlüssel zur Codeverifizierung enthält) für sichere Bootloader- und (OTA-) Updates over-the-air
- Ein Bereitstellungszertifikat Just-In-Time

Fügen Sie die Header-Datei hinzu und implementieren Sie die APIs definierte PAL.

PAL APIs

Funktion	Beschreibung
PKCS11_PAL_initialisieren	Initialisiert die PAL-Ebene. Wird von der PKCS11 Kernbibliothek zu Beginn ihrer Initialisierungssequenz aufgerufen.
PKCS11_PAL_ SaveObject	Schreibt Daten in den nichtflüchtigen Speicher.
PKCS11_PAL_ FindObject	Verwendet einen PKCS # 11 `CKA_LABEL` , um nach einem entsprechenden PKCS #11-Objekt im nichtflüchtigen Speicher zu suchen, und gibt das Handle dieses Objekts zurück, falls es existiert.
PKCS11_PAL_ GetObjectValue	Liefert den Wert eines Objekts entsprechend dem Handle.
PKCS11_PAL_ GetObjectValueCleanup	Bereinigung für den `PKCS11_PAL_GetObjectValue` -Aufruf. Kann verwendet werden, um den in einem `PKCS11_PAL_GetObjectValue` -Aufruf zugewiesenen Speicherplatz freizugeben.

Testen

Wenn Sie die PKCS11 FreeRTOS-Kernbibliothek verwenden oder die erforderliche Teilmenge davon implementieren PKCS11 APIs, müssen Sie die FreeRTOS-Tests bestehen. PKCS11 Diese testen, ob die erforderlichen Funktionen für FreeRTOS-Bibliotheken erwartungsgemäß funktionieren.

In diesem Abschnitt wird auch beschrieben, wie Sie die PKCS11 FreeRTOS-Tests mit den Qualifikationstests lokal ausführen können.

Voraussetzungen

Um die PKCS11 FreeRTOS-Tests einzurichten, muss Folgendes implementiert werden.

- Ein unterstützter Port von. PKCS11 APIs
- Eine Implementierung von Funktionen der FreeRTOS-Plattform für Qualifizierungstests, die Folgendes umfassen:
  - `FRTest_ThreadCreate`
  - `FRTest_ThreadTimedJoin`
  - `FRTest_MemoryAlloc`
  - `FRTest_MemoryFree`

(In der Datei README.md finden Sie die Integrationstests für PKCS #11 von FreeRTOS Libraries.) GitHub

Tests portieren

- Fügen Sie FreeRTOS-Libraries-Integration-Testses als Submodul zu Ihrem Projekt hinzu. Das Submodul kann in einem beliebigen Verzeichnis des Projekts platziert werden, sofern es erstellt werden kann.
- Kopieren Sie `config_template/test_execution_config_template.h` und `config_template/test_param_config_template.h` an einen Projektspeicherort im Build-Pfad und benennen Sie sie in `test_execution_config.h` und `test_param_config.h` um.
- Fügen Sie relevante Dateien in das Build-System ein. Falls verwendet`CMake`, `qualification_test.cmake` und `src/pkcs11_tests.cmake` kann verwendet werden, um relevante Dateien einzubeziehen.
- Implementieren Sie `UNITY_OUTPUT_CHAR` es so, dass sich Testausgabeprotokolle und Geräteprotokolle nicht überschneiden.
- Integrieren Sie das MbedTLS, das das Ergebnis des Cryptoki-Vorgangs verifiziert.
- Rufen Sie `RunQualificationTest()` von der Anwendung aus an.

Tests konfigurieren

Die PKCS11 Testsuite muss entsprechend der PKCS11 Implementierung konfiguriert werden. In der folgenden Tabelle ist die Konfiguration aufgeführt, die für PKCS11 Tests in der `test_param_config.h` Header-Datei erforderlich ist.

PKSC11 Testkonfigurationen

Konfiguration	Beschreibung
PKCS11_TEST_RSA_KEY_SUPPORT	Die Portierung unterstützt RSA-Schlüsselfunktionen.
PKCS11_TEST_EC_KEY_SUPPORT	Die Portierung unterstützt EC-Schlüsselfunktionen.
PKCS11_TEST_IMPORT_PRIVATE_KEY_SUPPORT	Die Portierung unterstützt den Import des privaten Schlüssels. Der Import von RSA- und EC-Schlüsseln wird im Test validiert, wenn die unterstützenden Schlüsselfunktionen aktiviert sind.
PKCS11_TEST_GENERATE_KEYPAIR_SUPPORT	Die Portierung unterstützt die Generierung von Schlüsselpaaren. Die Generierung von EC-Schlüsselpaaren wird im Test validiert, wenn die unterstützenden Tastenfunktionen aktiviert sind.
PKCS11_TEST_PREPROVISIONED_SUPPORT	Für die Portierung wurden vorab Anmeldeinformationen bereitgestellt. `PKCS11_TEST_LABEL_DEVICE_PRIVATE_KEY_FOR_TLS` , `PKCS11_TEST_LABEL_DEVICE_PUBLIC_KEY_FOR_TLS` und`PKCS11_TEST_LABEL_DEVICE_CERTIFICATE_FOR_TLS` , sind Beispiele für die Anmeldeinformationen.
PKCS11_TEST_LABEL_DEVICE_PRIVATE_KEY_FOR_TLS	Die Bezeichnung des privaten Schlüssels, der im Test verwendet wurde.
PKCS11_TEST_LABEL_DEVICE_PUBLIC_KEY_FOR_TLS	Die Bezeichnung des öffentlichen Schlüssels, der im Test verwendet wurde.
PKCS11_TEST_LABEL_DEVICE_CERTIFICATE_FOR_TLS	Die Bezeichnung des im Test verwendeten Zertifikats.

Konfiguration	Beschreibung
PKCS11_TEST_JITP_CODEVERIFY _ROOT_CERT_SUPPORTED	Die Portierung unterstützt Speicher für JITP. Stellen Sie diesen Wert auf 1 ein, um den `codeverify` JITP-Test zu aktivieren.
PKCS11_TEST_LABEL_CODE_VERI FICATION_KEY	Die Bezeichnung des Code-Bestätigungss chlüssels, der im JITP-Test verwendet wird. `codeverify`
PKCS11_TEST_LABEL_JITP_CERTIFICATE	Die Bezeichnung des JITP-Zertifikats, das im JITP-Test verwendet wird. `codeverify`
PKCS11_TEST_LABEL_ROOT_CERTIFICATE	Die Bezeichnung des im JITP-Test verwendet en Stammzertifikats. `codeverify`

FreeRTOS-Bibliotheken und Referenzintegrationen müssen mindestens eine Tastenfunktionskonfiguration wie RSA- oder Elliptic Curve-Tasten und einen von der unterstützten Schlüsselbereitstellungsmechanismus unterstützen. PKCS11 APIs Der Test muss die folgenden Konfigurationen ermöglichen:

- Mindestens eine der folgenden Konfigurationen der wichtigsten Funktionen:
  - PKCS11_TEST_RSA_KEY_SUPPORT
  - PKCS11_TEST_EC_KEY_SUPPORT
- Mindestens eine der folgenden wichtigen Bereitstellungskonfigurationen:
  - PKCS11_TEST_IMPORT_PRIVATE_KEY_SUPPORT
  - PKCS11_TEST_GENERATE_KEYPAIR-UNTERSTÜTZUNG
  - PKCS11_TEST_VORAB BEREITGESTELLTE_UNTERSTÜTZUNG

Der Test der vorab bereitgestellten Geräteanmeldedaten muss unter den folgenden Bedingungen ausgeführt werden:

- `PKCS11_TEST_PREPROVISIONED_SUPPORT`muss aktiviert und andere Bereitstellungsmechanismen deaktiviert sein.
- Nur eine Schlüsselfunktion, entweder `PKCS11_TEST_RSA_KEY_SUPPORT` oder`PKCS11_TEST_EC_KEY_SUPPORT`, ist aktiviert.

- Richten Sie die vorab bereitgestellten Tastenbezeichnungen entsprechend Ihrer Tastenfunktion ein, einschließlich`PKCS11_TEST_LABEL_DEVICE_PRIVATE_KEY_FOR_TLS`, und`PKCS11_TEST_LABEL_DEVICE_PUBLIC_KEY_FOR_TLS`. `PKCS11_TEST_LABEL_DEVICE_CERTIFICATE_FOR_TLS` Diese Anmeldeinformationen müssen vorhanden sein, bevor der Test ausgeführt werden kann.

Der Test muss möglicherweise mehrmals mit unterschiedlichen Konfigurationen ausgeführt werden, wenn die Implementierung vorab bereitgestellte Anmeldeinformationen und andere Bereitstellungsmechanismen unterstützt.

> **Note**
>
> Die Objekte mit Labels `PKCS11_TEST_LABEL_DEVICE_PUBLIC_KEY_FOR_TLS` und `PKCS11_TEST_LABEL_DEVICE_CERTIFICATE_FOR_TLS` werden während des Tests zerstört`PKCS11_TEST_LABEL_DEVICE_PRIVATE_KEY_FOR_TLS`, wenn entweder `PKCS11_TEST_GENERATE_KEYPAIR_SUPPORT` oder `PKCS11_TEST_GENERATE_KEYPAIR_SUPPORT` aktiviert ist.

Ausführen von Tests

In diesem Abschnitt wird beschrieben, wie Sie die PKCS11 Schnittstelle mit den Qualifikationstests lokal testen können. Alternativ können Sie IDT auch verwenden, um die Ausführung zu automatisieren. Einzelheiten finden Sie unter AWS IoT Device Tester für FreeRTOS im FreeRTOS-Benutzerhandbuch.

Die folgenden Anweisungen beschreiben, wie die Tests ausgeführt werden:

- Öffnen `test_execution_config.h` und definieren Sie CORE_ PKCS11 _TEST_ENABLED auf 1.
- Erstellen Sie die Anwendung und flashen Sie sie auf Ihr Gerät, um sie auszuführen. Die Testergebnisse werden an die serielle Schnittstelle ausgegeben.

Das Folgende ist ein Beispiel für das ausgegebene Testergebnis.

```
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
```

Der Test ist abgeschlossen, wenn alle Tests erfolgreich absolviert wurden.

> **Note**
>
> Um ein Gerät offiziell für FreeRTOS zu qualifizieren, müssen Sie den portierten Quellcode des Geräts mit validieren. AWS IoT Device Tester Folgen Sie den Anweisungen unter Using AWS IoT Device Tester for FreeRTOS im FreeRTOS User Guide, um die Port-Validierung einzurichten AWS IoT Device Tester . Um den Port einer bestimmten Bibliothek zu testen, muss die richtige Testgruppe in der `device.json` Datei im Ordner aktiviert sein. AWS IoT Device Tester `configs`

Portierung der Netzwerktransportschnittstelle

Integration der TLS-Bibliothek

Verwenden Sie für die TLS-Authentifizierung (Transport Layer Security) Ihren bevorzugten TLS-Stack. Wir empfehlen die Verwendung von Mbed TLS, da es mit FreeRTOS-Bibliotheken getestet wurde. Ein Beispiel dafür finden Sie in diesem Repository. GitHub

Unabhängig von der von Ihrem Gerät verwendeten TLS-Implementierung müssen Sie die zugrunde liegenden Transport-Hooks für den TLS-Stack mit dem TCP/IP-Stack implementieren. Sie müssen die TLS-Verschlüsselungssammlungen unterstützen, die von unterstützt werden. AWS IoT

Portierung der Network Transport Interface-Bibliothek

Sie müssen eine Netzwerktransportschnittstelle implementieren, um CoreMQTT und CoreHTTP verwenden zu können. Die Netzwerktransportschnittstelle enthält Funktionszeiger und Kontextdaten, die zum Senden und Empfangen von Daten über eine einzelne Netzwerkverbindung erforderlich sind. Weitere Informationen finden Sie unter Transportschnittstelle. FreeRTOS bietet eine Reihe von integrierten Netzwerk-Transport-Schnittstellentests, um diese Implementierungen zu validieren. Im folgenden Abschnitt erfahren Sie, wie Sie Ihr Projekt für die Ausführung dieser Tests einrichten.

Voraussetzungen

Um diesen Test zu portieren, benötigen Sie Folgendes:

- Ein Projekt mit einem Build-System, das FreeRTOS mit einem validierten FreeRTOS-Kernelport erstellen kann.
- Funktionierende Implementierung von Netzwerktreibern.

Portierung

- Fügen Sie FreeRTOS-Libraries-Integration-Testses Ihrem Projekt als Submodul hinzu. Es spielt keine Rolle, wo sich das Submodul im Projekt befindet, solange es erstellt werden kann.
- Kopieren Sie `config_template/test_execution_config_template.h` und `config_template/test_param_config_template.h` an einen Projektspeicherort im Buildpfad und benennen Sie sie in `test_execution_config.h` und `test_param_config.h` um.

- Fügen Sie relevante Dateien in das Build-System ein. Falls verwendet`CMake`, `qualification_test.cmake` und `src/transport_interface_tests.cmake` werden verwendet, um relevante Dateien einzubeziehen.
- Implementieren Sie die folgenden Funktionen an einem geeigneten Projektstandort:
- • A`network connect function`: Die Signatur ist durch `NetworkConnectFunc` in definiert`src/common/network_connection.h`. Diese Funktion verwendet einen Zeiger auf den Netzwerkkontext, einen Zeiger auf Hostinformationen und einen Zeiger auf Netzwerkanmeldedaten. Sie stellt mit den angegebenen Netzwerkanmeldedaten eine Verbindung zu dem in den Host-Informationen angegebenen Server her.
  - A`network disconnect function`: Die Signatur ist durch `NetworkDisconnectFunc` in definiert`src/common/network_connection.h`. Diese Funktion nimmt einen Zeiger auf einen Netzwerkkontext auf. Sie trennt eine zuvor hergestellte Verbindung, die im Netzwerkkontext gespeichert ist.
  - `setupTransportInterfaceTestParam()`: Dies ist in `src/transport_interface/transport_interface_tests.h` definiert. Die Implementierung muss genau den gleichen Namen und die gleiche Signatur haben wie in definiert`transport_interface_tests.h`. Diese Funktion nimmt einen Zeiger auf eine TransportInterfaceTestParamStruktur auf. Sie füllt die Felder in der TransportInterfaceTestParamStruktur auf, die vom Transportschnittstellentest verwendet wird.
- Implementieren Sie UNITY_OUTPUT_CHAR, sodass sich die Testausgabeprotokolle nicht mit den Geräteprotokollen überschneiden.
- `runQualificationTest()`Rufen Sie von der Anwendung aus an. Die Gerätehardware muss ordnungsgemäß initialisiert und das Netzwerk muss vor dem Anruf verbunden sein.

Verwaltung von Anmeldeinformationen (auf dem Gerät generierter Schlüssel)

Wenn FORCE_GENERATE_NEW_KEY_PAIR in auf 1 gesetzt **`test_param_config.h`** ist, generiert die Geräteanwendung ein neues key pair auf dem Gerät und gibt den öffentlichen Schlüssel aus. Die Geräteanwendung verwendet ECHO_SERVER_ROOT_CA und TRANSPORT_CLIENT_CERTIFICATE als Root-CA und Client-Zertifikat des Echoservers, wenn eine TLS-Verbindung mit dem Echo-Server hergestellt wird. IDT legt diese Parameter während des Qualifikationslaufs fest.

Verwaltung von Anmeldeinformationen (Schlüssel importieren)

Die Geräteanwendung verwendet ECHO_SERVER_ROOT_CA, TRANSPORT_CLIENT_CERTIFICATE und TRANSPORT_CLIENT_PRIVATE_KEY als Stammzertifizierungsstelle für den Echoserver, als Client-Zertifikat und **`test_param_config.h`** als privaten Client-Schlüssel, wenn eine TLS-Verbindung mit dem Echo-Server hergestellt wird. IDT legt diese Parameter während des Qualifikationslaufs fest.

Testen

In diesem Abschnitt wird beschrieben, wie Sie die Transportschnittstelle mit den Qualifizierungstests lokal testen können. Zusätzliche Details finden Sie in der Datei README.md, die im Abschnitt transport_interface von on bereitgestellt wird. FreeRTOS-Libraries-Integration-Tests GitHub

Alternativ können Sie IDT auch verwenden, um die Ausführung zu automatisieren. Einzelheiten finden Sie unter AWS IoT Device Tester für FreeRTOS im FreeRTOS-Benutzerhandbuch.

Aktivieren Sie den Test

Öffnen `test_execution_config.h` und definieren Sie TRANSPORT_INTERFACE_TEST_ENABLED auf 1.

Richten Sie den Echo-Server zum Testen ein

Für lokale Tests ist ein Echoserver erforderlich, auf den von dem Gerät aus zugegriffen werden kann, auf dem die Tests ausgeführt werden. Der Echoserver muss TLS unterstützen, wenn die Implementierung der Transportschnittstelle TLS unterstützt. Falls Sie noch keinen haben, verfügt das FreeRTOS-Libraries-Integration-Tests GitHub Repository über eine Echo-Server-Implementierung.

Konfiguration des Projekts zum Testen

Aktualisieren Sie `test_param_config.h` unter ECHO_SERVER_ENDPOINT und ECHO_SERVER_PORT auf das Endpunkt- und Server-Setup im vorherigen Schritt.

Anmeldeinformationen (auf dem Gerät generierter Schlüssel)

- Setzen Sie ECHO_SERVER_ROOT_CA auf das Serverzertifikat des Echoservers.
- Setzen Sie FORCE_GENERATE_NEW_KEY_PAIR auf 1, um ein key pair zu generieren und den öffentlichen Schlüssel abzurufen.
- Setzen Sie FORCE_GENERATE_NEW_KEY_PAIR nach der Schlüsselgenerierung wieder auf 0.

- Verwenden Sie den öffentlichen Schlüssel sowie den Serverschlüssel und das Zertifikat, um das Client-Zertifikat zu generieren.
- Setzen Sie TRANSPORT_CLIENT_CERTIFICATE auf das generierte Client-Zertifikat.

Anmeldeinformationen einrichten (Schlüssel importieren)

- Setzen Sie ECHO_SERVER_ROOT_CA auf das Serverzertifikat des Echoservers.
- Setzen Sie TRANSPORT_CLIENT_CERTIFICATE auf das vorgenerierte Client-Zertifikat.
- Setzen Sie TRANSPORT_CLIENT_PRIVATE_KEY auf den vorgenerierten privaten Client-Schlüssel.

Erstellen und flashen Sie die Anwendung

Erstellen und flashen Sie die Anwendung mit der Toolchain Ihrer Wahl. Wenn aufgerufen `runQualificationTest()` wird, werden die Tests der Transportschnittstelle ausgeführt. Die Testergebnisse werden an die serielle Schnittstelle ausgegeben.

> **Note**
>
> Um ein Gerät offiziell für FreeRTOS zu qualifizieren, müssen Sie den portierten Quellcode des Geräts anhand von OTA PAL- und OTA E2E-Testgruppen mit validieren. AWS IoT Device Tester Folgen Sie den Anweisungen unter Using AWS IoT Device Tester for FreeRTOS im FreeRTOS User Guide, um die Port-Validierung einzurichten AWS IoT Device Tester . Um den Port einer bestimmten Bibliothek zu testen, muss die richtige Testgruppe in der `device.json` Datei im Ordner aktiviert sein. AWS IoT Device Tester `configs`

Konfiguration der CoreMQTT-Bibliothek

Geräte am Edge können das MQTT-Protokoll verwenden, um mit der Cloud zu kommunizieren. AWS AWS IoT hostet einen MQTT-Broker, der Nachrichten an und von verbundenen Geräten am Edge sendet und empfängt.

Die CoreMQTT-Bibliothek implementiert das MQTT-Protokoll für Geräte, auf denen FreeRTOS ausgeführt wird. Die CoreMQTT-Bibliothek muss nicht portiert werden, aber das Testprojekt Ihres Geräts muss zur Qualifizierung alle MQTT-Tests bestehen. Weitere Informationen finden Sie in der CoreMQTT-Bibliothek im FreeRTOS-Benutzerhandbuch.

Voraussetzungen

Um die CoreMQTT-Bibliothekstests einzurichten, benötigen Sie einen Netzwerk-Transport-Schnittstellenport. Weitere Informationen hierzu finden Sie unter Portierung der Netzwerktransportschnittstelle.

Testen

Führen Sie die CoreMQTT-Integrationstests aus:

- Registrieren Sie Ihr Client-Zertifikat beim MQTT-Broker.
- Stellen Sie den Broker-Endpunkt ein `config` und führen Sie die Integrationstests aus.

Erstellen Sie eine Referenz-MQTT-Demo

Wir empfehlen, den CoreMQTT-Agenten zu verwenden, um die Thread-Sicherheit für alle MQTT-Operationen zu gewährleisten. Der Benutzer benötigt außerdem Aufgaben zum Veröffentlichen und Abonnieren sowie Device Advisor-Tests, um zu überprüfen, ob die Anwendung TLS, MQTT und andere FreeRTOS-Bibliotheken effektiv integriert.

Um ein Gerät offiziell für FreeRTOS zu qualifizieren, validieren Sie Ihr Integrationsprojekt mit AWS IoT Device Tester MQTT-Testfällen. Anweisungen zum Einrichten und Testen finden Sie im AWS IoT Device Advisor-Workflow. Die vorgeschriebenen Testfälle für TLS und MQTT sind unten aufgeführt:

TLS-Testfälle

Testfall	Testfälle	Erforderliche Tests
TLS	TLS Connect	Ja
TLS	TLS-Unterstützung Support AWS IoT Cipher Suites	Eine empfohlene Verschlüsselungssuite
TLS	Zertifikat für unsicheren TLS-Server	Ja
TLS	TLS Falscher Betreffname Serverzertifikat	Ja

MQTT-Testfälle

Testfall	Testfälle	Erforderliche Tests
MQTT	MQTT-Verbindung	Ja
MQTT	MQTT Connect Jitter versucht es erneut	Ja, ohne Warnungen
MQTT	MQTT Abonnieren	Ja
MQTT	MQTT Veröffentlichen	Ja
MQTT	MQTT ClientPuback QoS 1	Ja
MQTT	MQTT kein Ack PingResp	Ja

Konfiguration der CoreHTTP-Bibliothek

Geräte am Edge können das HTTP-Protokoll verwenden, um mit der Cloud zu kommunizieren. AWS AWS IoT Dienste hosten einen HTTP-Server, der Nachrichten an und von verbundenen Geräten am Edge sendet und empfängt.

Testen

Gehen Sie zum Testen wie folgt vor:

- Richten Sie die PKI für die gegenseitige TLS-Authentifizierung mit AWS oder einem HTTP-Server ein.
- Führen Sie CoreHTTP-Integrationstests durch.

Portierung der AWS IoT over-the-air (OTA-) Update-Bibliothek

Mit FreeRTOS over-the-air (OTA) -Updates können Sie Folgendes tun:

- Bereitstellen neuer Firmware-Images auf einem einzelnen Gerät, einer Gruppe von Geräten oder Ihrer gesamten Flotte

- Stellen Sie Firmware auf Geräten bereit, wenn diese zu Gruppen hinzugefügt, zurückgesetzt oder erneut bereitgestellt werden.
- Überprüfen Sie die Authentizität und Integrität der neuen Firmware, nachdem sie auf Geräten bereitgestellt wurde.
- Überwachen des Fortschritts einer Bereitstellung
- Debuggen einer fehlgeschlagenen Bereitstellung
- Signieren Sie Firmware digital mit Code Signing für AWS IoT.

Weitere Informationen finden Sie unter _Over-the-AirFreeRTOS-Updates im FreeRTOS-Benutzerhandbuch zusammen mit der Update-Dokumentation.AWS IoT Over-the-air

Sie können die OTA-Update-Bibliothek verwenden, um OTA-Funktionen in Ihre FreeRTOS-Anwendungen zu integrieren. Weitere Informationen finden Sie unter FreeRTOS OTA Update Library im FreeRTOS User Guide.

FreeRTOS-Geräte müssen die Überprüfung der kryptografischen Codesignatur auf den OTA-Firmware-Images, die sie erhalten, erzwingen. Wir empfehlen die folgenden Algorithmen:

- Elliptischer Kurven-Digital-Signatur-Algorithmus (ECDSA)
- NIST P256-Kurve
- SHA-256-Hash

Voraussetzungen

- Füllen Sie die Anweisungen unter aus. Deinen Workspace und dein Projekt für die Portierung einrichten
- Erstellen Sie einen Netzwerk-Transport-Schnittstellen-Port.

  Weitere Informationen finden Sie unter Portierung der Netzwerktransportschnittstelle.
- Integrieren Sie die CoreMQTT-Bibliothek. Siehe CoreMQTT-Bibliothek im FreeRTOS-Benutzerhandbuch.
- Erstellen Sie einen Bootloader, der OTA-Updates unterstützen kann.

Plattform-Portierung

Sie müssen eine Implementierung des OTA Portable Abstraction Layer (PAL) bereitstellen, um die OTA-Bibliothek auf ein neues Gerät zu portieren. Die PAL APIs sind in der Datei ota_platform_interface.h definiert, für die implementierungsspezifische Details angegeben werden müssen.

Funktionsname	Beschreibung
`otaPal_Abort`	Stoppt ein OTA-Update.
`otaPal_CreateFileForRx`	Erstellt eine Datei zum Speichern der empfangenen Datenblöcke.
`otaPal_CloseFile`	Schließt die angegebene Datei. Dadurch wird die Datei möglicherweise authentifiziert, wenn Sie Speicher verwenden, der kryptografischen Schutz implementiert.
`otaPal_WriteBlock`	Schreibt einen Datenblock in die spezifizierte Datei mit dem angegebenen Offset. Bei Erfolg gibt die Funktion die Anzahl der geschriebenen Byte zurück. Andernfalls gibt die Funktion einen negativen Fehlercode zurück. Die Blockgröße entspricht immer einer Zweierpotenz und wird ausgerichtet. Weitere Informationen finden Sie unter Konfiguration der OTA-Bibliothek.
`otaPal_ActivateNewImage`	Aktiviert oder startet das neue Firmware-Abbild. Bei einigen Anschlüssen kehrt diese Funktion nicht zurück, wenn das Gerät programmg esteuert synchron zurückgesetzt wird.
`otaPal_SetPlatformImageState`	Erfüllt die Anforderungen der Plattform, um das neueste OTA-Firmware-Image (oder -Bundle) zu akzeptieren oder abzulehnen. Informationen zur Implementierung dieser Funktion finden Sie

Funktionsname	Beschreibung
	in der Dokumentation zu Ihrem Motherboard (Plattform) und zur Architektur.
`otaPal_GetPlatformImageState`	Ruft den Status des OTA-Updates auf.

Implementieren Sie die Funktionen in dieser Tabelle, wenn Ihr Gerät über eine integrierte Unterstützung verfügt.

Funktionsname	Beschreibung
`otaPal_CheckFileSignature`	Überprüft die Signatur der angegebenen Datei.
`otaPal_ReadAndAssumeCertificate`	Liest das angegebene Signaturgeberzertifikat aus dem Dateisystem und gibt es an den Aufrufer zurück.
`otaPal_ResetDevice`	Setzt das Gerät zurück.

> **Note**
>
> Stellen Sie sicher, dass Sie einen Bootloader haben, der OTA-Updates unterstützt. Anweisungen zum Erstellen Ihres AWS IoT Geräte-Bootloaders finden Sie unter. IoT-Geräte-Bootloader

E2E- und PAL-Tests

Führen Sie OTA PAL- und E2E-Tests durch.

E2E-Tests

Der OTA-End-to-End-Test (E2E) wird verwendet, um die OTA-Fähigkeit eines Geräts zu überprüfen und Szenarien aus der Realität zu simulieren. Dieser Test beinhaltet die Fehlerbehandlung.

Voraussetzungen

Um diesen Test zu portieren, benötigen Sie Folgendes:

- Ein Projekt, in das AWS eine OTA-Bibliothek integriert ist. Weitere Informationen finden Sie im Portierungsleitfaden für die OTA-Bibliothek.
- Portieren Sie die Demo-Anwendung mithilfe der OTA-Bibliothek, mit AWS IoT Core der Sie interagieren und die OTA-Updates durchführen möchten. Siehe Portierung der OTA-Demoanwendung.
- Richten Sie das IDT-Tool ein. Dadurch wird die OTA E2E-Hostanwendung ausgeführt, um das Gerät mit unterschiedlichen Konfigurationen zu erstellen, zu flashen und zu überwachen, und die Integration der OTA-Bibliothek validiert.

Portierung der OTA-Demoanwendung

Der OTA E2E-Test muss über eine OTA-Demoanwendung verfügen, um die OTA-Bibliotheksintegration zu validieren. Die Demo-Anwendung muss in der Lage sein, OTA-Firmware-Updates durchzuführen. Sie finden die FreeRTOS OTA-Demoanwendung im GitHubFreeRTOS-Repository. Wir empfehlen Ihnen, die Demo-Anwendung als Referenz zu verwenden und sie gemäß Ihren Spezifikationen zu modifizieren.

Schritte zur Portierung

1. Initialisieren Sie den OTA-Agent.
2. Implementieren Sie die Callback-Funktion der OTA-Anwendung.
3. Erstellen Sie die Aufgabe zur Verarbeitung von Ereignissen für den OTA-Agenten.
4. Starten Sie den OTA-Agenten.
5. Überwachen Sie die Statistiken des OTA-Agenten.
6. Fahren Sie den OTA-Agenten herunter.

Eine ausführliche Anleitung finden Sie unter FreeRTOS OTA over MQTT — Einstiegspunkt der Demo.

Konfiguration

Für die Interaktion sind die folgenden Konfigurationen erforderlich: AWS IoT Core

- AWS IoT Core Kundenanmeldedaten
  - Richten Sie DemoConfigRoot_CA_PEM in den Amazon Trust Services-Endpunkten ein. `Ota_Over_Mqtt_Demo/demo_config.h` AWS Weitere Informationen finden Sie unter Serverauthentifizierung.
  - Richten Sie DemoConfigClient_Certificate_PEM und DemoConfigClient_Private_Key_PEM mit Ihren Client-Anmeldeinformationen ein. `Ota_Over_Mqtt_Demo/demo_config.h` AWS IoT Weitere AWS Informationen zu Client-Zertifikaten und privaten Schlüsseln finden Sie in den Details zur Client-Authentifizierung.
- Anwendungsversion
- OTA-Steuerprotokoll
- OTA-Datenprotokoll
- Anmeldeinformationen für die Codesignatur
- Andere OTA-Bibliothekskonfigurationen

Sie finden die obigen Informationen in `demo_config.h` und `ota_config.h` in FreeRTOS OTA-Demoanwendungen. Weitere Informationen finden Sie unter FreeRTOS OTA über MQTT — Gerät einrichten.

Verifizierung erstellen

Führen Sie die Demo-Anwendung aus, um den OTA-Job auszuführen. Wenn der Vorgang erfolgreich abgeschlossen wurde, können Sie die OTA E2E-Tests weiter ausführen.

Die FreeRTOS OTA-Demo bietet detaillierte Informationen zur Einrichtung eines OTA-Clients und eines AWS IoT Core OTA-Jobs im FreeRTOS-Windows-Simulator. AWS OTA unterstützt sowohl MQTT- als auch HTTP-Protokolle. Weitere Informationen finden Sie in den folgenden Beispielen:

- OTA über MQTT-Demo auf Windows Simulator
- OTA-über-HTTP-Demo auf Windows Simulator

Tests mit dem IDT-Tool ausführen

Um die OTA E2E-Tests auszuführen, müssen Sie AWS IoT Device Tester (IDT) verwenden, um die Ausführung zu automatisieren. Weitere Informationen finden Sie unter AWS IoT Device Tester für FreeRTOS im FreeRTOS-Benutzerhandbuch.

E2E-Testfälle

Testfall	Beschreibung
`OTAE2EGreaterVersion`	Happy Path-Test für regelmäßige OTA-Updates. Es erstellt ein Update mit einer neueren Version, die das Gerät erfolgreich aktualisiert.
`OTAE2EBackToBackDownloads`	Dieser Test erstellt 3 aufeinanderfolgende OTA-Updates. Es wird erwartet, dass das Gerät dreimal hintereinander aktualisiert wird.
`OTAE2ERollbackIfUnableToConnectAfterUpdate`	Mit diesem Test wird überprüft, ob das Gerät auf die vorherige Firmware zurückgesetzt wird, falls es mit der neuen Firmware keine Verbindung zum Netzwerk herstellen kann.
`OTAE2ESameVersion`	Dieser Test bestätigt, dass das Gerät die eingehende Firmware ablehnt, wenn die Version gleich bleibt.
`OTAE2EUnsignedImage`	Dieser Test überprüft, ob das Gerät ein Update ablehnt, wenn das Image nicht signiert ist.
`OTAE2EUntrustedCertificate`	Dieser Test überprüft, ob das Gerät ein Update ablehnt, wenn die Firmware mit einem nicht vertrauenswürdigen Zertifikat signiert ist.
`OTAE2EPreviousVersion`	Dieser Test überprüft, ob das Gerät eine ältere Update-Version ablehnt.
`OTAE2EIncorrectSigningAlgorithm`	Verschiedene Geräte unterstützen unterschiedliche Signier- und Hash-Algorithmen. Dieser Test bestätigt, dass das Gerät das OTA-Update nicht bestanden hat, wenn es mit einem nicht unterstützten Algorithmus erstellt wurde.
`OTAE2EDisconnectResume`	Dies ist der Happy-Path-Test für die Suspend- und Resume-Funktion. Dieser Test erstellt ein

Testfall	Beschreibung
	OTA-Update und startet das Update. Anschließend wird AWS IoT Core mit derselben Client-ID (Dingname) und denselben Anmeldeinformationen eine Verbindung hergestellt. AWS IoT Core trennt dann die Verbindung zum Gerät. Es wird erwartet, dass das Gerät erkennt, dass die Verbindung unterbrochen wurde AWS IoT Core, und sich nach einer gewissen Zeit in einen angehaltenen Zustand versetzt und versucht, erneut eine Verbindung herzustellen AWS IoT Core und den Download fortzusetzen.
`OTAE2EDisconnectCancelUpdate`	Bei diesem Test wird geprüft, ob das Gerät sich selbst wiederherstellen kann, wenn der OTA-Job abgebrochen wird, während er sich im angehaltenen Zustand befindet. Es macht dasselbe wie der `OTAE2EDisconnectResume` Test, außer dass nach dem Herstellen einer Verbindung zum Gerät AWS IoT Core, wodurch die Verbindung zum Gerät getrennt wird, das OTA-Update abgebrochen wird. Ein neues Update wird erstellt. Es wird erwartet, dass das Gerät erneut eine Verbindung mit dem AWS IoT Core herstellt, das aktuelle Update abbricht, in den Wartestatus zurückkehrt und das nächste Update akzeptiert und abschließt.

Testfall	Beschreibung
`OTAE2EPresignedUrlExpired`	Wenn ein OTA-Update erstellt wird, können Sie die Lebensdauer der vorsignierten S3-URL konfigurieren. Mit diesem Test wird überprüft , ob das Gerät in der Lage ist, einen OTA auszuführen, auch wenn der Download nach Ablauf der URL nicht abgeschlossen werden kann. Es wird erwartet, dass das Gerät ein neues Auftragsdokument anfordert, das eine neue URL enthält, um den Download fortzuset zen.
`OTAE2E2UpdatesCancel1st`	Dieser Test erstellt zwei OTA-Updates hinterein ander. Wenn das Gerät meldet, dass es das erste Update herunterlädt, bricht der Test Force das erste Update ab. Es wird erwartet, dass das Gerät das aktuelle Update abbricht und das zweite Update aufnimmt und abschließt.
`OTAE2ECancelThenUpdate`	Bei diesem Test werden zwei OTA-Updates hintereinander erstellt. Wenn das Gerät meldet, dass es das erste Update herunterlädt, bricht der Test Force das erste Update ab. Es wird erwartet, dass das Gerät das aktuelle Update abbricht und das zweite Update aufnimmt und es dann abschließt.
`OTAE2EImageCrashed`	Dieser Test überprüft, ob das Gerät ein Update ablehnen kann, wenn das Image abstürzt.

PAL-Tests

Voraussetzungen

Um die Network Transport Interface-Tests zu portieren, benötigen Sie Folgendes:

- Ein Projekt, das FreeRTOS mit einem gültigen FreeRTOS-Kernelport erstellen kann.

- Eine funktionierende Implementierung von OTA PAL.

Portierung

- Fügen Sie FreeRTOS-Libraries-Integration-Tests als Submodul zu Ihrem Projekt hinzu. Der Standort des Submoduls im Projekt muss dort sein, wo es gebaut werden kann.
- Kopieren Sie `config_template/test_execution_config_template.h` und `config_template/test_param_config_template.h` an eine Stelle im Buildpfad und benennen Sie sie in `test_execution_config.h` und `test_param_config.h` um.
- Fügen Sie relevante Dateien in das Build-System ein. Falls verwendet`CMake`, `qualification_test.cmake` und `src/ota_pal_tests.cmake` kann verwendet werden, um relevante Dateien einzubeziehen.
- Konfigurieren Sie den Test, indem Sie die folgenden Funktionen implementieren:
  - `SetupOtaPalTestParam()`: definiert in`src/ota/ota_pal_test.h`. Die Implementierung muss denselben Namen und dieselbe Signatur haben wie in definiert`ota_pal_test.h`. Derzeit müssen Sie diese Funktion nicht konfigurieren.
- Implementieren Sie UNITY_OUTPUT_CHAR, sodass sich Testausgabeprotokolle nicht mit Geräteprotokollen überschneiden.
- `RunQualificationTest()`Rufen Sie von der Anwendung aus an. Die Gerätehardware muss ordnungsgemäß initialisiert sein und das Netzwerk muss vor dem Anruf verbunden sein.

Testen

In diesem Abschnitt werden die lokalen Tests der OTA-PAL-Qualifikationstests beschrieben.

Aktivieren Sie den Test

Öffnen `test_execution_config.h` und definieren Sie OTA_PAL_TEST_ENABLED auf 1.

Aktualisieren `test_param_config.h` Sie unter die folgenden Optionen:

- OTA_PAL_TEST_CERT_TYPE: Wählen Sie den verwendeten Zertifikatstyp aus.
- OTA_PAL_CERTIFICATE_FILE: Pfad zum Gerätezertifikat, falls zutreffend.
- OTA_PAL_FIRMWARE_FILE: Name der Firmware-Datei, falls zutreffend.
- OTA_PAL_USE_FILE_SYSTEM: Auf 1 gesetzt, wenn das OTA-PAL die Dateisystemabstraktion verwendet.

Erstellen und flashen Sie die Anwendung mit einer Toolchain Ihrer Wahl. Wenn der aufgerufen `RunQualificationTest()` wird, werden die OTA-PAL-Tests ausgeführt. Die Testergebnisse werden an die serielle Schnittstelle ausgegeben.

Integration von OTA-Aufgaben

- Fügen Sie Ihrer aktuellen MQTT-Demo einen OTA-Agenten hinzu.
- Führen Sie OTA-Ende-zu-Ende-Tests (E2E) mit aus. AWS IoT Dadurch wird überprüft, ob die Integration wie erwartet funktioniert.

> **Note**
>
> Um ein Gerät offiziell für FreeRTOS zu qualifizieren, müssen Sie den portierten Quellcode des Geräts anhand von OTA PAL- und OTA E2E-Testgruppen mit validieren. AWS IoT Device Tester Folgen Sie den Anweisungen unter Using AWS IoT Device Tester for FreeRTOS im FreeRTOS User Guide, um die Port-Validierung einzurichten AWS IoT Device Tester . Um den Port einer bestimmten Bibliothek zu testen, muss die richtige Testgruppe in der `device.json` Datei im Ordner aktiviert sein. AWS IoT Device Tester `configs`

IoT-Geräte-Bootloader

Sie müssen Ihre eigene sichere Bootloader-Anwendung bereitstellen. Stellen Sie sicher, dass das Design und die Implementierung eine angemessene Abwehr von Sicherheitsbedrohungen bieten. Im Folgenden finden Sie die Bedrohungsmodellierung als Referenz.

Modellierung von Bedrohungen für IoT-Geräte-Bootloader

Hintergrund

Als funktionierende Definition gilt, dass es sich bei den eingebetteten AWS IoT Geräten, auf die sich dieses Bedrohungsmodell bezieht, um Produkte auf Mikrocontrollerbasis handelt, die mit Cloud-Diensten interagieren. Sie können in Verbraucher-,kommerziellen oder Industrieumgebungen eingesetzt werden. IoT-Geräte können Daten über einen Benutzer, einen Patienten, eine Maschine oder eine Umgebung erfassen und von Glühbirnen und Türschlössern bis hin zu Fabrikmaschinen alles steuern.

Die Modellierung von Bedrohungen ist ein Sicherheitsansatz aus der Sicht eines hypothetischen Gegner. Unter Berücksichtigung der Ziele und Methoden des Gegners wird eine Bedrohungsliste

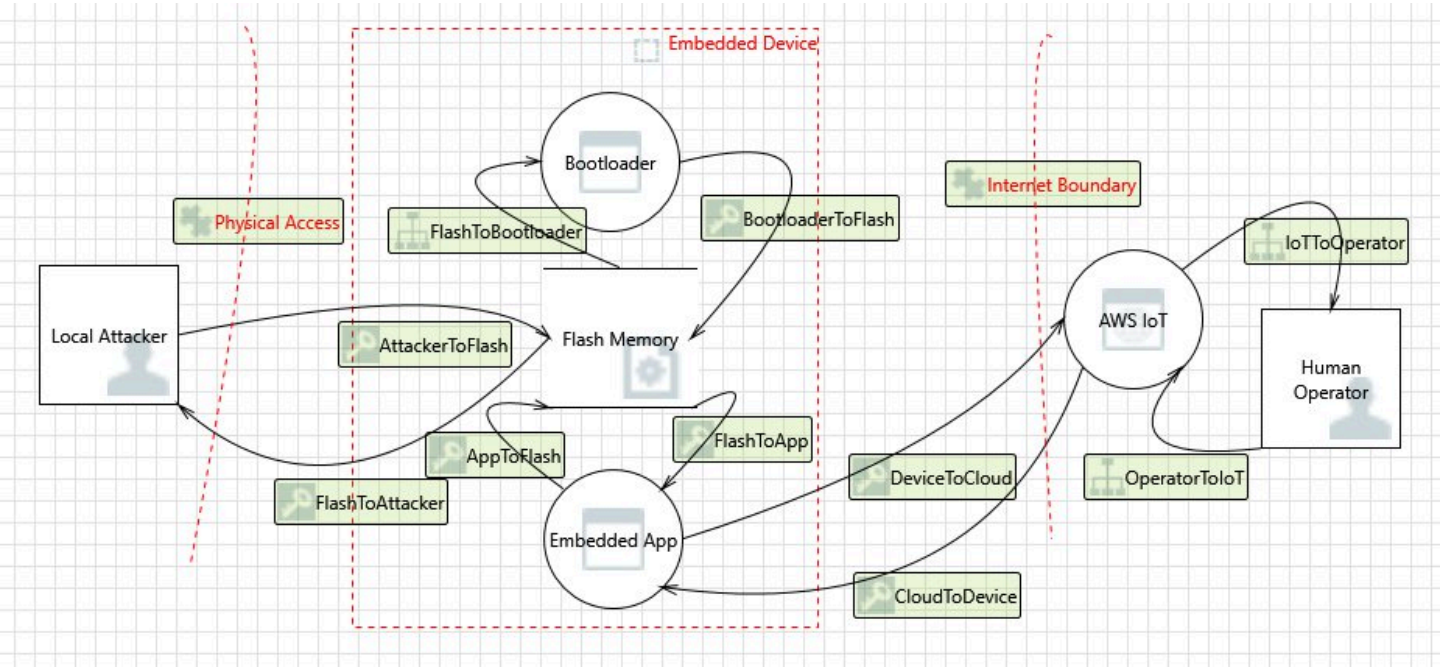
erstellt. Bedrohungen sind Angriffe auf eine Ressource oder eine Komponente durch einen Gegner. Die Liste wird priorisiert und zur Identifizierung und Erstellung von Abwehrlösungen verwendet. Bei der Auswahl einer Risikominderungslösung sollten die Kosten für deren Implementierung und Wartung gegen den tatsächlichen Sicherheitswert, den sie bietet, abgewogen werden. Es gibt verschiedene Methoden für Bedrohungsmodelle. Jede dieser Lösungen ist in der Lage, die Entwicklung eines sicheren und erfolgreichen AWS IoT Produkts zu unterstützen.

FreeRTOS bietet OTA (over-the-air) -Softwareupdates für Geräte an AWS IoT . Die Update-Funktion kombiniert Cloud-Services mit Softwarebibliotheken auf dem Gerät und einem vom Partner bereitgestellten Bootloader. Dieses Bedrohungsmodell konzentriert sich speziell auf Bedrohungen für den Bootloader.

Bootloader-Anwendungsfälle

- Digitales Signieren und Verschlüsseln der Firmware vor der Bereitstellung
- Bereitstellen neuer Firmware-Images auf einem einzelnen Gerät, einer Gruppe von Geräten oder einer gesamten Flotte
- Überprüfen der Authentizität und Integrität der neuen Firmware nach der Bereitstellung auf Geräten
- Geräte führen nur unveränderte Software aus einer vertrauenswürdigen Quelle aus
- Geräte sind widerstandsfähig gegenüber fehlerhafter Software, die sie über OTA empfangen

Datenflussdiagramm



Bedrohungen

Bei einigen Angriffen gibt es mehrere Abhilfemaßnahmen. Ein Netzwerk, das ein bösartiges Firmware-Image bereitstellen man-in-the-middle soll, wird beispielsweise dadurch entschärft, dass sowohl das vom TLS-Server angebotene Zertifikat als auch das Codesignerzertifikat des neuen Firmware-Images als vertrauenswürdig eingestuft werden. Um die Sicherheit des Bootloaders zu maximieren, werden alle Lösungen zur Abwehr von Bootloadern als unzuverlässig angesehen. Der Bootloader sollte für jeden Angriff über eigene Abwehrlösungen verfügen. Mehrschichtige Abwehrlösungen sind bekannt als. defense-in-depth

Bedrohungen:

- Ein Angreifer manipuliert die Verbindung des Geräts mit dem Server, um ein schädliches Software-Image bereitzustellen.

  Beispiel für Abwehrmaßnahmen

  - Beim Systemstart überprüft der Bootloader die kryptografische Signatur des Images anhand eines bekannten Zertifikats. Wenn die Verifizierung fehlschlägt, setzt der Bootloader zum vorherigen Image zurück.
- Ein Angreifer nutzt einen Pufferüberlauf, um schädliches Verhalten im vorhandenen Firmware-Image zu erzeugen, das in Flash gespeichert ist.

  Beispiele für Abwehrmaßnahmen

  - Beim Systemstart führt der Bootloader eine Überprüfung wie zuvor beschrieben aus. Wenn die Verifizierung fehlschlägt und kein vorheriges Image verfügbar ist, wird der Bootloader angehalten.
  - Beim Systemstart führt der Bootloader eine Überprüfung wie zuvor beschrieben aus. Wenn die Überprüfung fehlschlägt und kein vorheriges Image verfügbar ist, wechselt der Bootloader in den ausfallsicheren Modus „Nur OTA“.
- Ein Angreifer startet das Gerät mit einem zuvor gespeicherten ausnutzbaren Image.

  Beispiele für Abwehrmaßnahmen

  - Flash-Sektoren, in denen das letzte Image gespeichert ist, werden nach erfolgreicher Installation und erfolgreichen Tests eines neuen Images gelöscht.
  - Bei jedem erfolgreichen Upgrade brennt eine Sicherung durch und Images werden erst ausgeführt, wenn die richtige Anzahl an Sicherungen durchgebrannt ist.

- Ein OTA-Update stellt ein fehlerhaftes oder schädliches Image bereit, das einen Brick des Geräts verursacht.

  Beispiel für Abwehrmaßnahmen

  - Der Bootloader startet einen Hardware-Watchdog-Timer, der ein Rollback zum vorherigen Image auslöst.
- Ein Angreifer patcht den Bootloader, um die Image-Verifizierung zu umgehen, damit das Gerät nicht signierte Images akzeptiert.

  Beispiele für Abwehrmaßnahmen

  - Der Bootloader befindet sich in ROM (Festwertspeicher) und kann nicht geändert werden.
  - Der Bootloader befindet sich im OTP (one-time-programmable Speicher) und kann nicht geändert werden.
  - Der Bootloader befindet sich in der sicheren Zone von ARM und kann nicht TrustZone geändert werden.
- Ein Angreifer ersetzt das Verifizierungszertifikat, damit das Gerät schädliche Images akzeptiert.

  Beispiele für Abwehrmaßnahmen

  - Das Zertifikat befindet sich in einem kryptografischen Co-Prozessor und kann nicht geändert werden.
  - Das Zertifikat befindet sich in ROM (oder OTP oder einer sicheren Zone) und kann nicht geändert werden.

Zusätzliche Modellierung von Bedrohungen

Dieses Bedrohungsmodell betrachtet nur den Bootloader. Eine zusätzliche Modellierung von Bedrohungen könnte die allgemeine Sicherheit verbessern. Eine empfohlene Methode ist die Auflistung der Ziele des Gegners, der für diese Ziele anvisierten Komponenten und der Eintrittspunkte in diese Komponenten. Um eine Liste der Bedrohungen zu erstellen, sollten Sie mögliche Angriffe auf die Eintrittspunkte betrachten, durch die die Kontrolle der Komponenten erzielt werden soll. Im Folgenden finden Sie Beispiele für Ziele, Komponenten und Eintrittspunkte für ein IoT-Gerät. Diese Listen sind nicht vollständig und sollen zu weiteren Überlegungen anregen.

Ziele des Gegners

- Erpressen von Geld

- Rufschädigung
- Fälschen von Daten
- Umleiten von Ressourcen
- Ausspionieren eines Ziels
- Erlangen von physischem Zugriff auf eine Website
- Verursachen von Schäden
- Auslösen von Panik

Wichtige Komponenten

- Private Schlüssel
- Clientzertifikat
- CA-Stammzertifikate
- Sicherheitsanmeldeinformationen und -token
- Personenbezogene Daten des Kunden
- Implementierungen von Geschäftsgeheimnissen
- Sensordaten
- Cloud-Analyse-Datenspeicher
- Cloud-Infrastruktur

Eintrittspunkte

- DHCP-Antwort
- DNS-Antwort
- MQTT über TLS
- HTTPS-Antwort
- OTA-Software-Image
- Andere, wie von der Anwendung vorgegeben, z. B. USB
- Physischer Zugriff auf den Bus
- Offengelegter integrierter Schaltkreis

Portierung der Cellular Interface-Bibliothek

FreeRTOS unterstützt die AT-Befehle einer TCP-ausgelagerten zellulären Abstraktionsschicht. Weitere Informationen finden Sie unter Cellular Interface Library und Porting the Cellular Interface Library auf freertos.org.

Voraussetzungen

Es besteht keine direkte Abhängigkeit von der Cellular Interface-Bibliothek. Im FreeRTOS-Netzwerkstack können Ethernet, WLAN und Mobilfunk jedoch nicht nebeneinander existieren, sodass Entwickler eines davon auswählen müssen, um es in das zu integrieren. Portierung der Netzwerktransportschnittstelle

> **Note**
>
> Wenn das Mobilfunkmodul in der Lage ist, TLS-Offload zu unterstützen oder AT-Befehle nicht unterstützt, können Entwickler ihre eigene Mobilfunkabstraktion implementieren, um sie in das zu integrieren. Portierung der Netzwerktransportschnittstelle

Migration von MQTT Version 3 zu CoreMQTT

Dieser Migrationsleitfaden erklärt, wie Sie Anwendungen von MQTT zu CoreMQTT migrieren.

Migration von Version 1 zu Version 3 für OTA-Anwendungen

Dieses Handbuch hilft Ihnen bei der Migration Ihrer Anwendung von der OTA-Bibliothek Version 1 auf Version 3.

> **Note**
>
> Die OTA-Version 2 APIs entspricht OTA v3. Wenn Ihre Anwendung also Version 2 von verwendet APIs, sind keine Änderungen für API-Aufrufe erforderlich. Wir empfehlen jedoch, Version 3 der Bibliothek zu integrieren. APIs

Demos für OTA-Version 3 sind hier verfügbar:

- ota_demo_core_mqtt.
- ota_demo_core_http.
- ota_ble.

Zusammenfassung der API-Änderungen

Zusammenfassung der API-Änderungen zwischen Version 1 und Version 3 der OTA-Bibliothek

OTA-API, Version 1	API der OTA-Version 3	Beschreibung der Änderungen
OTA_ AgentInit	OTA_Init	Die Eingabeparameter sowie der von der Funktion zurückgegebene Wert werden aufgrund von Änderungen in der Implementierung in OTA v3 geändert. Einzelhei ten finden Sie im Abschnitt zu OTA_Init weiter unten.
OTA_ AgentShutdown	OTA_Shutdown	Änderung der Eingabepa rameter, einschließlich eines zusätzlichen Parameters für eine optionale Abmeldung

OTA-API, Version 1	API der OTA-Version 3	Beschreibung der Änderungen
		von MQTT-Themen. Einzelheiten finden Sie im Abschnitt zu OTA_Shutdown weiter unten.
OTA_ GetAgentState	OTA_ GetState	Der API-Name wird ohne Änderungen am Eingabeparameter geändert. Der Rückgabewert ist derselbe, aber die Enumeration und die Mitglieder wurden umbenannt. Einzelheiten finden Sie im Abschnitt zu OTA_ GetState weiter unten.
–	OTA_ GetStatistics	Neue API hinzugefügt, die OTA_, OTA_, APIs OTA_GetPacketsReceived, GetPacketsQueued OTA_ ersetzt. GetPacketsProcessed GetPacketsDropped Einzelheiten finden Sie im Abschnitt für OTA_ weiter unten. GetStatistics
OTA_ GetPacketsReceived	–	Diese API wurde aus Version 3 entfernt und durch OTA_ ersetzt. GetStatistics
OTA_ GetPacketsQueued	–	Diese API wurde aus Version 3 entfernt und durch OTA_ ersetzt. GetStatistics
OTA_ GetPacketsProcessed	–	Diese API wurde aus Version 3 entfernt und durch OTA_ ersetzt. GetStatistics

OTA-API, Version 1	API der OTA-Version 3	Beschreibung der Änderungen
OTA_ GetPacketsDropped	–	Diese API wurde aus Version 3 entfernt und durch OTA_ ersetzt. GetStatistics
OTA_ ActivateNewImage	OTA_ ActivateNewImage	Die Eingabeparameter sind dieselben, aber der Rückgabe-OTA-Fehlercode wurde umbenannt und neue Fehlercodes wurden in Version 3 der OTA-Bibliothek hinzugefügt. Einzelheiten finden Sie im Abschnitt für OTA_ActivateNewImage .
OTA_ SetImageState	OTA_ SetImageState	Die Eingabeparameter sind identisch und wurden umbenannt, der Rückgabe-OTA-Fehlercode wurde umbenannt und neue Fehlercodes wurden in Version 3 der OTA-Bibliothek hinzugefügt. Einzelheiten finden Sie im Abschnitt für OTA_SetImageState .
OTA_ GetImageState	OTA_ GetImageState	Die Eingabeparameter sind dieselben, die Return-Enumeration wurde in Version 3 der OTA-Bibliothek umbenannt. Einzelheiten finden Sie im Abschnitt für OTA_GetImageState .

OTA-API, Version 1	API der OTA-Version 3	Beschreibung der Änderungen
OTA_Suspend	OTA_suspendieren	Die Eingabeparameter sind dieselben, der Rückgabe-OTA-Fehlercode wurde umbenannt und neue Fehlercodes wurden in Version 3 der OTA-Bibliothek hinzugefügt. Einzelheiten finden Sie im Abschnitt für OTA_Suspend.
OTA_Resume	OTA_Lebenslauf	Der Eingabeparameter für die Verbindung wird entfernt, da die Verbindung in der OTA-Demo/Anwendung verarbeitet wird, der OTA-Rückgabefehler code wird umbenannt und neue Fehlercodes werden in Version 3 der OTA-Bibli othek hinzugefügt. Einzelhei ten finden Sie im Abschnitt für OTA_Resume.
OTA_ CheckForUpdate	OTA_ CheckForUpdate	Die Eingabeparameter sind dieselben, der Rückgabe-OTA-Fehlercode wurde umbenannt und neue Fehlercodes wurden in Version 3 der OTA-Bibliothek hinzugefügt. Einzelheiten finden Sie im Abschnitt für OTA_CheckForUpdate .

OTA-API, Version 1	API der OTA-Version 3	Beschreibung der Änderungen
–	OTA_ EventProcessingTask	Es wurde eine neue API hinzugefügt. Sie ist die Hauptereignisschleife zur Verarbeitung von Ereignissen für das OTA-Update und muss von der Anwendungsaufgabe aufgerufen werden. Einzelheiten finden Sie im Abschnitt für OTA_EventProcessingTask .
–	OTA_ SignalEvent	Es wurde eine neue API hinzugefügt, die das Ereignis am Ende der OTA-Ereignis warteschlange hinzufügt . Es wird von internen OTA-Modulen verwendet, um die Agentenaufgabe zu signalisieren. Einzelheiten finden Sie im Abschnitt für OTA_SignalEvent .
–	OTA_Err_StrError	Neue API für die Konvertierung von Fehlercode in Zeichenfolge bei OTA-Fehlern.
–	OTA_ _strerror JobParse	Neue API für die Konvertierung von Fehlercode in Zeichenketten bei Job Parsing-Fehlern.
–	OTA_ _strerror OsStatus	Neue API für die Konvertierung von Statuscode in Zeichenfolge für den Portstatus des OTA-Betriebssystems.

OTA-API, Version 1	API der OTA-Version 3	Beschreibung der Änderungen
–	OTA_ _strerror PalStatus	Neue API für die Konvertierung von Statuscode in Zeichenfolge für den OTA-PAL-Port-Status.

Beschreibung der erforderlichen Änderungen

OTA_Init

Bei der Initialisierung des OTA-Agenten in Version 1 wird die `OTA_AgentInit` API verwendet, die Parameter für Verbindungskontext, Dingname, vollständigen Callback und Timeout als Eingabe verwendet.

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

Diese API enthält jetzt Parameter für die Puffer, `OTA_Init` die für OTA-, OTA-Schnittstellen, den Dingnamen und den Anwendungs-Callback erforderlich sind.

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

Eingabeparameter wurden entfernt -

pvConnectionContext -

Der Verbindungskontext wird entfernt, da die OTA-Bibliothek Version 3 nicht erfordert, dass der Verbindungskontext an sie übergeben wird und dieMQTT/HTTP operations are handled by their respective interfaces in the OTA demo/application.

xTicksToWarte -

Der Ticks to Wait-Parameter wird ebenfalls entfernt, da die Aufgabe in der OTA-Demo/Anwendung erstellt wird, bevor OTA_Init aufgerufen wird.

Umbenannte Eingabeparameter -

xFunc -

Der Parameter wird umbenannt OtaAppCallback und sein Typ wird in _t geändert. OtaAppCallback

Neue Eingabeparameter -

pOtaBuffer

Die Anwendung muss die Puffer zuweisen und sie bei der Initialisierung mithilfe der OtaAppBuffer _t-Struktur an die OTA-Bibliothek übergeben. Die benötigten Puffer unterscheiden sich je nach dem Protokoll, das zum Herunterladen der Datei verwendet wird, geringfügig. Für das MQTT-Protokoll sind die Puffer für den Streamnamen erforderlich und für das HTTP-Protokoll sind die Puffer für die vorsignierte URL und das Autorisierungsschema erforderlich.

Puffer, die benötigt werden, wenn MQTT für den Dateidownload verwendet wird -

```
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
};
```

Puffer, die erforderlich sind, wenn HTTP für den Dateidownload verwendet wird -

```
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
};
```

Wo -

```
    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

Der zweite Eingabeparameter für OTA_Init ist ein Verweis auf die OTA-Schnittstellen für den Typ _t. OtaInterfaces Dieser Satz von Schnittstellen muss an die OTA-Bibliothek übergeben werden und umfasst in der Betriebssystemschnittstelle die MQTT-Schnittstelle, die HTTP-Schnittstelle und die Plattformabstraktionsschichtschnittstelle.

OTA OS-Schnittstelle

Die OTA OS-Funktionsschnittstelle besteht aus einer Reihe von Schnittstellen APIs , die implementiert werden müssen, damit das Gerät die OTA-Bibliothek verwenden kann. Die Funktionsimplementierungen für diese Schnittstelle werden der OTA-Bibliothek in der Benutzeranwendung zur Verfügung gestellt. Die OTA-Bibliothek ruft die Funktionsimplementierungen auf, um Funktionen auszuführen, die normalerweise von einem Betriebssystem bereitgestellt werden. Dies beinhaltet die Verwaltung von Ereignissen, Timern und Speicherzuweisungen. Die Implementierungen für FreeRTOS und POSIX werden mit der OTA-Bibliothek bereitgestellt.

Beispiel für FreeRTOS mit dem bereitgestellten FreeRTOS-Port -

```
  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;
```

Beispiel für Linux, das den bereitgestellten POSIX-Port verwendet -

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

MQTT-Schnittstelle

Die OTA-MQTT-Schnittstelle besteht aus einer Reihe von Schnittstellen APIs , die in einer Bibliothek implementiert werden müssen, damit die OTA-Bibliothek einen Dateiblock vom Streaming-Dienst herunterladen kann.

Beispiel mit dem CoreMQTT-Agenten APIs aus der OTA-over-MQTT-Demo -

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

HTTP-Schnittstelle

Die OTA-HTTP-Schnittstelle besteht aus einer Reihe von Schnittstellen APIs , die in einer Bibliothek implementiert werden müssen, damit die OTA-Bibliothek einen Dateiblock herunterladen kann, indem sie eine Verbindung zu einer vorsignierten URL herstellt und Datenblöcke abruft. Dies ist optional, es sei denn, Sie konfigurieren die OTA-Bibliothek so, dass sie von einer vorsignierten URL statt von einem Streaming-Dienst heruntergeladen wird.

Beispiel für die Verwendung von CoreHTTP APIs aus der OTA-über-HTTP-Demo -

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

OTA PAL-Schnittstelle

Bei der OTA-PAL-Schnittstelle handelt es sich um eine Reihe von Schnittstellen APIs , die implementiert werden müssen, damit das Gerät die OTA-Bibliothek verwenden kann. Die gerätespezifische Implementierung für das OTA-PAL wird der Bibliothek in der Benutzeranwendung zur Verfügung gestellt. Diese Funktionen werden von der Bibliothek zum Speichern, Verwalten und Authentifizieren von Downloads verwendet.

```
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;
```

Änderungen im Gegenzug -

Die Rückgabe wurde vom Status des OTA-Agenten in den OTA-Fehlercode geändert. Bitte beziehen Sie sich auf AWS IoT Over-the-air Update v3.0.0: OtaErr _t.

OTA_Shutdown

In der OTA-Bibliothek, Version 1, war die API, die zum Herunterfahren des OTA-Agenten verwendet wurde, OTA_AgentShutdown , was nun zusammen mit Änderungen der Eingabeparameter in OTA_Shutdown geändert wurde.

Herunterfahren des OTA-Agenten (Version 1)

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

Herunterfahren des OTA-Agenten (Version 3)

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

ticksToWait -

Die Anzahl der Ticks, die darauf warten sollen, dass der OTA-Agent den Shutdown-Vorgang abgeschlossen hat. Wenn dieser Wert auf Null gesetzt ist, kehrt die Funktion sofort zurück, ohne zu warten. Der aktuelle Status wird an den Aufrufer zurückgegeben. Der Agent befindet sich während dieser Zeit nicht im Ruhezustand, sondern wird für eine intensive Looping-Funktion verwendet.

Neuer Eingabeparameter -

UnsubscribeFlag -

Markierung, die angibt, ob Abmeldevorgänge von den Job-Themen aus ausgeführt werden sollen, wenn Shutdown aufgerufen wird. Wenn das Kennzeichen 0 ist, werden Abmeldevorgänge für Jobthemen nicht aufgerufen. Wenn die Bewerbung von den Jobthemen abgemeldet werden muss, muss dieses Flag beim Aufrufen von OTA_Shutdown auf 1 gesetzt werden.

Änderungen im Gegenzug -

OtaState_t -

Die Enumeration für den Status des OTA-Agenten und seine Mitglieder wurden umbenannt. Weitere Informationen finden Sie unter AWS IoT Over-the-air Update v3.0.0.

OTA_ GetState

Der API-Name wurde von OTA_ in OTA_ AgentGetState geändert. GetState

Herunterfahren des OTA-Agenten (Version 1)

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

Herunterfahren des OTA-Agenten (Version 3)

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

Änderungen im Gegenzug -

OtaState_t -

Die Enumeration für den Status des OTA-Agenten und seine Mitglieder wurden umbenannt. Weitere Informationen finden Sie unter AWS IoT Over-the-air Update v3.0.0.

OTA_ GetStatistics

Neue einzelne API für Statistiken hinzugefügt. Sie ersetzt OTA_, APIs OTA_GetPacketsReceived, OTA_, OTA_GetPacketsQueued. GetPacketsProcessed GetPacketsDropped Außerdem beziehen sich die Statistiknummern in der OTA-Bibliothek, Version 3, nur auf den aktuellen Job.

OTA-Bibliothek, Version 1

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

OTA-Bibliothek, Version 3

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

pStatistik -

Eingabe-/Ausgabeparameter für Statistikdaten wie Pakete, die für den aktuellen Job empfangen, gelöscht, in die Warteschlange gestellt und verarbeitet wurden.

Ausgabeparameter -

OTA-Fehlercode.

Anwendungsbeispiel -

```
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

Die Eingabeparameter sind dieselben, aber der Rückgabe-OTA-Fehlercode wurde umbenannt und neue Fehlercodes wurden in Version 3 der OTA-Bibliothek hinzugefügt.

OTA-Bibliothek, Version 1

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

OTA-Bibliothek, Version 3

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

Die Aufzählung des Rückgabe-OTA-Fehlercodes wurde geändert und neue Fehlercodes wurden hinzugefügt. Bitte beachten Sie AWS IoT Over-the-air Update v3.0.0: _t. OtaErr

Anwendungsbeispiel -

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

OTA_ SetImageState

Die Eingabeparameter sind identisch und wurden umbenannt, der Rückgabe-OTA-Fehlercode wurde umbenannt und neue Fehlercodes wurden in Version 3 der OTA-Bibliothek hinzugefügt.

OTA-Bibliothek, Version 1

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

OTA-Bibliothek, Version 3

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

Der Eingabeparameter wurde in OtaImageState _t umbenannt. Bitte beziehen Sie sich auf AWS IoT Over-the-air Update v3.0.0.

Die Aufzählung des Rückgabe-OTA-Fehlercodes wurde geändert und neue Fehlercodes wurden hinzugefügt. Bitte beachten Sie AWS IoT Over-the-air Update v3.0.0/_t. OtaErr

Anwendungsbeispiel -

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

OTA_ GetImageState

Die Eingabeparameter sind identisch, die Return-Enumeration wurde in Version 3 der OTA-Bibliothek umbenannt.

OTA-Bibliothek, Version 1

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

OTA-Bibliothek, Version 3

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

Die Rückgabe-Enumeration wurde in OtaImageState _t umbenannt. Bitte beziehen Sie sich auf AWS IoT Over-the-air Update v3.0.0: _t. OtaImageState

Anwendungsbeispiel -

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

OTA_Suspend

Die Eingabeparameter sind dieselben, der Rückgabe-OTA-Fehlercode wurde umbenannt und neue Fehlercodes wurden in Version 3 der OTA-Bibliothek hinzugefügt.

OTA-Bibliothek, Version 1

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

OTA-Bibliothek, Version 3

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

Die Aufzählung des Rückgabe-OTA-Fehlercodes wurde geändert und neue Fehlercodes wurden hinzugefügt. Bitte beachten Sie AWS IoT Over-the-air Update v3.0.0: _t. OtaErr

Anwendungsbeispiel -

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

OTA_Resume

Der Eingabeparameter für die Verbindung wird entfernt, da die Verbindung in der OTA-Demo/Anwendung verarbeitet wird, der OTA-Rückgabe-Fehlercode wird umbenannt und neue Fehlercodes werden in Version 3 der OTA-Bibliothek hinzugefügt.

OTA-Bibliothek, Version 1

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

OTA-Bibliothek, Version 3

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

Die Aufzählung des Rückgabe-OTA-Fehlercodes wurde geändert und neue Fehlercodes wurden hinzugefügt. Bitte beachten Sie AWS IoT Over-the-air Update v3.0.0: _t. OtaErr

Anwendungsbeispiel -

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

OTA_ CheckForUpdate

Die Eingabeparameter sind dieselben, der Rückgabe-OTA-Fehlercode wurde umbenannt und neue Fehlercodes wurden in Version 3 der OTA-Bibliothek hinzugefügt.

OTA-Bibliothek, Version 1

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

OTA-Bibliothek, Version 3

```
OtaErr_t OTA_CheckForUpdate( void )
```

Die Aufzählung des Rückgabe-OTA-Fehlercodes wurde geändert und neue Fehlercodes wurden hinzugefügt. Bitte beachten Sie AWS IoT Over-the-air Update v3.0.0: _t. OtaErr

OTA_ EventProcessingTask

Dies ist eine neue API und die Hauptereignisschleife zur Verarbeitung von Ereignissen für OTA-Updates. Sie muss von der Anwendungsaufgabe aufgerufen werden. Diese Schleife verarbeitet und führt weiterhin Ereignisse aus, die für das OTA-Update empfangen wurden, bis diese Aufgabe von der Anwendung beendet wird.

OTA-Bibliothek, Version 3

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

Beispiel für 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 );
}
```

Beispiel für 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

Dies ist eine neue API, die das Ereignis am Ende der Ereigniswarteschlange hinzufügt und auch von internen OTA-Modulen verwendet wird, um die Agentenaufgabe zu signalisieren.

OTA-Bibliothek, Version 3

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

Anwendungsbeispiel -

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

Integrieren Sie die OTA-Bibliothek als Submodul in Ihre Anwendung

Wenn Sie die OTA-Bibliothek in Ihre eigene Anwendung integrieren möchten, können Sie den Befehl git submodule verwenden. Mit Git-Submodulen können Sie ein Git-Repository als Unterverzeichnis eines anderen Git-Repositorys verwalten. Die Version 3 der OTA-Bibliothek wird im Repository ota-for-aws-iot-embedded-sdk verwaltet.

```
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
```

Weitere Informationen finden Sie unter Integrieren des OTA-Agenten in Ihre Anwendung im FreeRTOS-Benutzerhandbuch.

Referenzen

- OTAv1.
- OTAv3.

Migration von Version 1 zu Version 3 für den OTA-PAL-Port

Mit der Over-the-air Updates-Bibliothek wurden einige Änderungen an der Ordnerstruktur und der Platzierung der Konfigurationen vorgenommen, die für die Bibliothek und die Demo-Anwendungen erforderlich sind. Damit OTA-Anwendungen, die für die Verwendung mit Version 1.2.0 entwickelt wurden, auf Version 3.0.0 der Bibliothek migrieren können, müssen Sie die Funktionssignaturen des PAL-Ports aktualisieren und zusätzliche Konfigurationsdateien hinzufügen, wie in diesem Migrationshandbuch beschrieben.

Änderungen an OTA PAL

- Der Name des OTA-PAL-Port-Verzeichnisses wurde von `ota` bis aktualisiert`ota_pal_for_aws`. Dieser Ordner muss 2 Dateien enthalten: `ota_pal.c` und`ota_pal.h`. Die PAL-Header-Datei `libraries/freertos_plus/aws/ota/src/aws_iot_ota_pal.h` wurde aus der OTA-Bibliothek gelöscht und muss innerhalb des Ports definiert werden.
- Die Rückgabecodes (`OTA_Err_t`) werden in eine `OTAMainStatus_t` Aufzählung übersetzt. Übersetzte Rückgabecodes finden Sie auf ota_platform_interface.h. Es stehen auch Hilfsmakros zum Kombinieren `OtaPalMainStatus` und `OtaPalSubStatus` Codieren sowie `OtaMainStatus` Extrahieren `OtaPalStatus` und Ähnlichem zur Verfügung.
- Im PAL einloggen
  - Das `DEFINE_OTA_METHOD_NAME` Makro wurde entfernt.
  - Vorhin:`OTA_LOG_L1( "[%s] Receive file created.\r\n", OTA_METHOD_NAME );`.
  - Aktualisiert: `LogInfo(( "Receive file created."));` Verwenden Sie`LogDebug`, `LogWarn` und `LogError` für das entsprechende Protokoll.
- Die Variable `cOTA_JSON_FileSignatureKey` wurde geändert in`OTA_JsonFileSignatureKey`.

Funktionen

Die Funktionssignaturen sind in definiert `ota_pal.h` und beginnen mit dem Präfix `otaPal` anstelle von`prvPAL`.

> **Note**
>
> Der genaue Name des PAL ist technisch gesehen offen, aber um mit den Qualifikationstests kompatibel zu sein, sollte der Name den unten angegebenen entsprechen.

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

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

  Hinweise: Erstellen Sie eine neue Empfangsdatei für die eingehenden Datenblöcke.

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

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

  Hinweise: Schreibt einen Datenblock am angegebenen Offset in die angegebene Datei.

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

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

  Hinweise: Aktiviert das neueste MCU-Image, das über OTA empfangen wurde.

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

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

  Hinweise: Setzen Sie das Gerät zurück.

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

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

Hinweise: Authentifizieren und schließen Sie die zugrunde liegende Empfangsdatei im angegebenen OTA-Kontext.

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

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

  Hinweise: Stoppen Sie eine OTA-Übertragung.

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

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

  Hinweise: Versuch, den Status des OTA-Update-Images festzulegen.

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

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

  Hinweise: Rufen Sie den Status des OTA-Update-Images ab.

Datentypen

- Version 1: `OTA_PAL_ImageState_t`

  Datei: `aws_iot_ota_agent.h`

  Version 3: `OtaPalImageState_t`

  Datei: `ota_private.h`

  Hinweise: Der durch die Plattformimplementierung festgelegte Image-Status.

- Version 1: `OTA_Err_t`

  Datei: `aws_iot_ota_agent.h`

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

Datei:`ota.h`, `ota_platform_interface.h`

Hinweise: v1: Dies waren Makros, die eine 32-Ganzzahl ohne Vorzeichen definierten. v3: Spezialisierte Aufzählung, die den Fehlertyp darstellt und mit einem Fehlercode verknüpft ist.

- Version 1: `OTA_FileContext_t`

  Datei: `aws_iot_ota_agent.h`

  Version 3: `OtaFileContext_t`

  Datei: `ota_private.h`

  Hinweise: v1: Enthält eine Aufzählung und Puffer für die Daten. v3: Enthält zusätzliche Variablen mit Datenlänge.

- Version 1: `OTA_ImageState_t`

  Datei: `aws_iot_ota_agent.h`

  Version 3: `OtaImageState_t`

  Datei: `ota_private.h`

  Hinweise: Status des OTA-Images

Konfigurationsänderungen

Die Datei `aws_ota_agent_config.h` wurde umbenannt, `ota_config.h`wodurch sich der Include-Schutz von `_AWS_OTA_AGENT_CONFIG_H_` zu ändert`OTA_CONFIG_H_`.

- Die Datei `aws_ota_codesigner_certificate.h` wurde gelöscht.
- Der neue Logging-Stack zum Drucken von Debug-Nachrichten wurde hinzugefügt:

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

- Die konstante Konfiguration wurde hinzugefügt:

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

Neue Datei: **ota_demo_config.h**enthält die Konfigurationen, die für die OTA-Demo erforderlich sind, z. B. das Codesignaturzertifikat und die Anwendungsversion.

- `signingcredentialSIGNING_CERTIFICATE_PEM`was in definiert wurde, `demos/include/aws_ota_codesigner_certificate.h` wurde nach `ota_demo_config.h` as verschoben `otapalconfigCODE_SIGNING_CERTIFICATE` und kann über die PAL-Dateien wie folgt abgerufen werden:

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

  Die Datei `aws_ota_codesigner_certificate.h` wurde gelöscht.

- Die Makros `APP_VERSION_BUILDAPP_VERSION_MINOR`,, `APP_VERSION_MAJOR` wurden hinzugefügt`ota_demo_config.h`. Die alten Dateien, die die Versionsinformationen enthielten, wurden entfernt, zum Beispiel `tests/include/aws_application_version.hlibraries/c_sdk/standard/common/include/iot_appversion32.h`,`demos/demo_runner/aws_demo_version.c`.

Änderungen an den OTA-PAL-Tests

- Die Testgruppe „Full_OTA_Agent“ wurde zusammen mit allen zugehörigen Dateien entfernt. Diese Testgruppe war zuvor für die Qualifizierung erforderlich. Diese Tests betrafen die OTA-Bibliothek und waren nicht spezifisch für den OTA-PAL-Port. Die OTA-Bibliothek verfügt jetzt über eine vollständige Testabdeckung, die im OTA-Repository gehostet wird, sodass diese Testgruppe nicht mehr erforderlich ist.
- Die Testgruppen „Full_OTA_CBOR“ und „Quarantine_OTA_CBOR“ sowie alle zugehörigen Dateien wurden entfernt. Diese Tests waren nicht Teil der Qualifikationstests. Die Funktionen, die diese Tests abdeckten, werden jetzt im OTA-Repository getestet.
- Die Testdateien wurden aus dem Bibliotheksverzeichnis in das `tests/integration_tests/ota_pal` Verzeichnis verschoben.
- Die OTA-PAL-Qualifikationstests wurden aktualisiert, sodass sie Version 3.0.0 der OTA-Bibliotheks-API verwenden.
- Die Art und Weise, wie die OTA-PAL-Tests auf das Codesignaturzertifikat für Tests zugreifen, wurde aktualisiert. Bisher gab es eine spezielle Header-Datei für die Codesignatur-Anmeldeinformationen. Dies ist bei der neuen Version der Bibliothek nicht mehr der Fall. Der Testcode erwartet, dass diese Variable in definiert ist`ota_pal.c`. Der Wert wird einem Makro zugewiesen, das in der plattformspezifischen OTA-Konfigurationsdatei definiert ist.

Checkliste

Verwenden Sie diese Checkliste, um sicherzustellen, dass Sie die für die Migration erforderlichen Schritte befolgen:

- Aktualisieren Sie den Namen des Ota-Pal-Port-Ordners von `ota` bis`ota_pal_for_aws`.
- Fügen Sie die Datei `ota_pal.h` mit den oben genannten Funktionen hinzu. Eine `ota_pal.h` Beispieldatei finden Sie unter GitHub.
- Fügen Sie die Konfigurationsdateien hinzu:
  - Ändern Sie den Namen der Datei von `aws_ota_agent_config.h` zu (oder erstellen)`ota_config.h`.
    - Fügen Sie hinzu:

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

    - Einschließen:

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

  - Kopieren Sie die obigen Dateien in den `aws_test config` Ordner und ersetzen Sie alle Includes von `ota_demo_config.h` durch`aws_test_ota_config.h`.
  - Fügt eine `ota_demo_config.h` Datei hinzu.
  - Fügt eine `aws_test_ota_config.h` Datei hinzu.
- Nehmen Sie die folgenden Änderungen an `ota_pal.c` vor:
  - Aktualisieren Sie die Includes mit den neuesten OTA-Bibliotheksdateinamen.
  - Entfernen Sie das `DEFINE_OTA_METHOD_NAME`-Makro.
  - Aktualisieren Sie die Signaturen der OTA-PAL-Funktionen.
  - Aktualisieren Sie den Namen der Dateikontextvariablen von `C` bis`pFileContext`.
  - Aktualisieren Sie die `OTA_FileContext_t` Struktur und alle zugehörigen Variablen.
  - Aktualisieren `cOTA_JSON_FileSignatureKey` auf`OTA_JsonFileSignatureKey`.
  - Aktualisieren Sie die `Ota_ImageState_t` Typen `OTA_PAL_ImageState_t` und.
  - Aktualisieren Sie den Fehlertyp und die Fehlerwerte.
  - Aktualisieren Sie die Druckmakros, um den Protokollierungsstapel zu verwenden.
  - Aktualisieren Sie das `signingcredential`SIGNING_CERTIFICATE_PEM zu sein. `otapalconfig`CODE_SIGNING_CERTIFICATE
  - Kommentare zum Update `otaPal_CheckFileSignature` und zur `otaPal_ReadAndAssumeCertificate` Funktion.
- Aktualisiere die `CMakeLists.txt`Datei.
- Aktualisieren Sie die IDE-Projekte.

Dokumentverlauf

Die folgende Tabelle beschreibt die Dokumentationshistorie für den FreeRTOS Porting Guide und den FreeRTOS Qualification Guide.

Date (Datum)	Version der Dokumentation	Verlauf ändern	FreeRTOS RTOS-Version
Mai 2022	Leitfaden zur FreeRTOS-Portierung FreeRTOS-Qualifizierungsleitfaden	• Bestehende Tests wurden aktualisiert, neue Tests hinzugefügt und redundante Tests entfernt, die auf FreeRTOS Long Term Support (LTS) -Bibliotheken basieren. Weitere Informationen finden Sie unter FreeRTOS Libraries Integration Tests 202205.00 auf. GitHub • Aktualisiert Flussdiagramm zur FreeRTOS-Portierung. • Ein neuer wurde hinzugefügt. Portierung der Netzwerktransportschnittstelle • Portierung der AWS IoT over-the-air	202012.04-LTS 202112,00

Date (Datum)	Version der Dokumentation	Verlauf ändern	FreeRTOS RTOS-Version
		(OTA-) Update-Bibliothekist jetzt für die Qualifikation erforderlich. • Die Anleitung zur Portierung von Wi-Fi und TLS-Abstraktion wurde entfernt, da sie nicht mehr benötigt wird. • Weitere Informationen zur FreeRTOS-Qualifizierung finden Sie unter Letzte Änderungen.	

Date (Datum)	Version der Dokumentation	Verlauf ändern	FreeRTOS RTOS-Version
Juli 2021	202107.00 (Portierungsleitfaden) 202107.00 (Qualifizierungsleitfaden)	• Veröffentlichung 202107.00 • Geändert Portierung der AWS IoT over-the-air (OTA-) Update-Bibliothek • Migration von Version 1 zu Version 3 für OTA-Anwendungen hinzugefügt • Migration von Version 1 zu Version 3 für den OTA-PAL-Port hinzugefügt	202107.00
Dezember 2020	202012.00 (Portierungsleitfaden) 202012.00 (Qualifizierungsleitfaden)	• Veröffentlichung 202012.00 • Konfiguration der CoreHTTP-Bibliothek hinzugefügt • Portierung der Cellular Interface-Bibliothek hinzugefügt	202012.00

Date (Datum)	Version der Dokumentation	Verlauf ändern	FreeRTOS RTOS-Version
November 2020	202011.00 (Portierungsleitfaden) 202011.00 (Qualifizierungsleitfaden)	• Veröffentlichung 202011.00 • Konfiguration der CoreMQTT-Bibliothek hinzugefügt	202011.00
Juli 2020	202007.00 (Portierungsleitfaden) 202007.00 (Qualifizierungsleitfaden)	• Veröffentlichung 202007.00	202007.00
18. Februar 2020	202002.00 (Portierungshandbuch) 202002.00 (Qualifizierungshandbuch)	• Version 202002.00 • Amazon FreeRTOS ist jetzt FreeRTOS	202002.00
17. Dezember 2019	201912.00 (Portierungshandbuch) 201912.00 (Qualifizierungshandbuch)	• Release 201912.00 • Portierung der gängigen I/O-Bibliotheken hinzugefügt.	201912.00
29. Oktober 2019	201910.00 (Portierungshandbuch) 201910.00 (Qualifizierungshandbuch)	• Veröffentlichung 201910.00 • Aktualisierte Portierungsinformationen für den Zufallszahlengenerator.	201910.00

Date (Datum)	Version der Dokumentation	Verlauf ändern	FreeRTOS RTOS-Version
26. August 2019	201908.00 (Portierungshandbuch) 201908.00 (Qualifizierungshandbuch)	• Version 201908.00 • Konfiguration der HTTPS-Clientbibliothek zum Testen hinzugefügt Portierung der PKCS11 Kernbibliothek aktualisiert	201908.00
17. Juni 2019	201906.00 (Portierungsleitfaden) 201906.00 (Qualifizierungsleitfaden)	• Veröffentlichung • Aktualisierte Verzeichnisstruktur	201906.00 Major
21. Mai 2019	1.4.8 (Portierungshandbuch) 1.4.8 (Qualifikationshandbuch)	• Die Portierungsdokumentation wurde in den FreeRTOS Porting Guide verschoben • Die Qualifizierungsdokumentation wurde in den FreeRTOS Qualification Guide verschoben	1.4.8

Date (Datum)	Version der Dokumentation	Verlauf ändern	FreeRTOS RTOS-Version
25. Februar 2019	1.1.6	• Download- und Konfigurationsanweisungen aus dem Anhang der Vorlage in den Ersten Schritten entfernt (Seite 84).	1.4.5 1.4.6 1.4.7
27. Dezember 2018	1.1.5	• Der Anhang zur Checkliste für die Qualifizierung mit den CMake Anforderungen wurde aktualisiert (Seite 70)	1.4.5 1.4.6
12. Dezember 2018	1.1.4	• TCP/IP-Portierungsanweisungen und lwIP Portierungsanhang hinzugefügt (Seite 31)	1.4.5

Date (Datum)	Version der Dokumentation	Verlauf ändern	FreeRTOS RTOS-Version
26. November 2018	1.1.3	• Anhang zur Bluetooth Low Energy-Portierung hinzugefügt (Seite 52) • AWS IoT Device Tester for FreeRTOS-Testinformationen im gesamten Dokument hinzugefügt • CMake Link zu Informationen zur Auflistung im Anhang zur FreeRTOS-Konsole hinzugefügt (Seite 85)	1.4.4

Date (Datum)	Version der Dokumentation	Verlauf ändern	FreeRTOS RTOS-Version
7. November 2018	1.1.2	• Portierungsanweisungen der PKCS #11-PAL-Schnittstelle im PKCS #11-Portierungsanhang aktualisiert (Seite 38) • Pfad zu `CertificateConfigurator.html` aktualisiert (Seite 76) • Vorlagenanhang im Handbuch "Erste Schritte" aktualisiert (Seite 80)	1.4.3

Date (Datum)	Version der Dokumentation	Verlauf ändern	FreeRTOS RTOS-Version
8. Oktober 2018	1.1.1	• Neue Spalte "Erforderlich für AFQP" zur `aws_test_runner_config.h` Testkonfigurationstabelle hinzugefügt (Seite 16) • Verzeichnispfad des Unity-Moduls im Abschnitt Erstellen des Testprojekts aktualisiert (Seite 14) • Tabelle "Empfohlene Portierungsreihenfolge" aktualisiert (Seite 22) • Client-Zertifikat und Schlüsselvariablennamen im TLS-Anhang, Testaufbau aktualisiert (Seite 40) • Dateipfade im Secure Sockets Portierungsanhang, Test-Setup (Seite 34); TLS Portierungsanhang, Test-	1.4.2

Date (Datum)	Version der Dokumentation	Verlauf ändern	FreeRTOS RTOS-Version
		Setup (Seite 40) und TLS-Server-Setup-Anhang (Seite 57) geändert	
27. August 2018	1.1.0	• OTA-Updates Portierungsanhang hinzugefügt (S. 47) • Bootloader-Portierungsanhang hinzugefügt (Seite 51)	1.4.0 1.4.1

Date (Datum)	Version der Dokumentation	Verlauf ändern	FreeRTOS RTOS-Version
9. August 2018	1.0.1	• Tabelle "Empfohlene Portierungsreihenfolge" aktualisiert (Seite 22) • PKCS #11 Portierungsanhang aktualisiert (Seite 36) • Dateipfade im TLS Portierungsanhang, Test-Setup (Seite 40) und TLS-Server-Setup-Anhang, Schritt 9 (Seite 51) geändert • Feste Hyperlinks in MQTT Portierungsanhang, Voraussetzungen (Seite 45) • Zu den Beispielen im Anhang Instructions to Create a BYOC (Seite 57) wurden AWS CLI Konfigurationsanweisungen hinzugefügt	1.3.1 1.3.2
31. Juli 2018	1.0.0	Erste Version des FreeRTOS Qualification Program Guide	1.3.0

Die vorliegende Übersetzung wurde maschinell erstellt. Im Falle eines Konflikts oder eines Widerspruchs zwischen dieser übersetzten Fassung und der englischen Fassung (einschließlich infolge von Verzögerungen bei der Übersetzung) ist die englische Fassung maßgeblich.