



Portierungshandbuch

Kostenlos RTOS



Kostenlos RTOS: Portierungshandbuch

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Die Handelsmarken und Handelsaufmachung von Amazon dürfen nicht in einer Weise in Verbindung mit nicht von Amazon stammenden Produkten oder Services verwendet werden, durch die Kunden irregeführt werden könnten oder Amazon in schlechtem Licht dargestellt oder diskreditiert werden könnte. Alle anderen Marken, die nicht im Besitz von Amazon sind, gehören den jeweiligen Besitzern, die möglicherweise mit Amazon verbunden sind oder von Amazon gesponsert werden.

Table of Contents

FreeRTOS	1
Was ist FreeRTOS	1
FreeRTOS-RTOS-S-S-	1
Portierung – Häufig gestellte Fragen	1
FreeRTOS für die Portierung herunterladen	3
Richten Sie Ihren Arbeitsbereich und Ihr Projekt für die Portierung ein	4
Portierung der FreeRTOS-Bibliotheken	5
Portierungsflussdiagramm	5
FreeRTOS-Kernel	7
Voraussetzungen	7
Konfigurieren des FreeRTOS-Kernels	7
Testen	8
Implementierung der Bibliotheksprotokollmakros	8
Testen	8
TCP/IP	9
Portierung von FreeRTOS+TCP	9
Testen	10
Kern-PKCS11	11
Wann sollte ein vollständiges PKCS #11 -Modul implementiert werden	12
Wann verwende ich FreeRTOS CorePKCS11	12
Portierung von CorePKCS11	12
Testen	13
Netzwerk-Transport-Schnittstelle	19
TLS	19
BIS	19
Voraussetzungen	19
Portierung	19
Testen	21
Kern-MQTT	22
Voraussetzungen	23
Testen	23
Referenz-MQTT-Demo erstellen	23
Kern-HTTP	24
Testen	24

Over-the-Air(OTA)-Updates	25
Voraussetzungen	25
Plattform-Portierung	26
E2E- und PAL-Tests	27
IoT-Geräte-Bootloader	34
Zelling Interface	39
Voraussetzungen	39
Migration von MQTT Version 3 zu CoreMQTT	40
Migrieren von der Version 1 zu Version 3 für OTA-Anwendungen	41
Zusammenfassung der API-Änderungen	41
Beschreibung der erforderlichen Änderungen	46
OTA_Init	46
OTA_Shutdown	51
OTA_GetState	52
OTA_GetStatistics	52
OTA_ActivateNewImage	53
OTA_SetImageState	54
OTA_GetImageState	54
OTA_aussetzen	55
OTA_Lebenslauf	55
OTA_CheckForUpdate	56
OTA_EventProcessingTask	56
OTA_SignalEvent	58
Integration der OTA-Bibliothek als Submodul in Ihre Anwendung	58
Referenzen	59
Migration von Version 1 auf Version 3 für den OTATAS-PALS-Anschluss	60
Änderungen an OTA PAL	60
Funktionen	60
Datentypen	62
Konfigurationsänderungen	63
Änderungen an den OTA-PAL-Tests	65
Checkliste	65
Dokumentverlauf	67
.....	lxxviii

FreeRTOS

Was ist FreeRTOS

FreeRTOS wurde in Zusammenarbeit mit den weltweit führenden Chip-Unternehmen über einen Zeitraum von 20 Jahren entwickelt und wird jetzt alle 170 Sekunden heruntergeladen. Es ist ein marktführendes Echtzeit-Betriebssystem (RTOS) für Mikrocontroller und kleine Mikroprozessoren. FreeRTOS wird unter der MIT-Open-Source-Lizenz kostenlos vertrieben und enthält einen Kernel und eine wachsende Anzahl von Bibliotheken, die für den Einsatz in allen Industriesektoren geeignet sind. FreeRTOS wurde mit einem Schwerpunkt auf Zuverlässigkeit und Benutzerfreundlichkeit entwickelt. FreeRTOS umfasst Bibliotheken für Konnektivität, Sicherheit und over-the-air (OTA) -Updates sowie Demoanwendungen, die FreeRTOS-Funktionen auf [qualifizierten Boards](#) demonstrieren.

Weitere Informationen finden Sie unter [FreeRTOS-TOS-TOS-S-TOS-S-TOS-S-TOS-S-](#)

Portierung von FreeRTOS auf Ihr IoT-Board

Sie müssen die FreeRTOS-Softwarebibliotheken auf der Grundlage der Funktionen und Ihrer Anwendung auf Ihr Mikrocontroller-basiertes Board portieren.

So portierst du FreeRTOS auf dein Gerät

1. Laden Sie die aktuelle Version von FreeRTOS [FreeRTOS für die Portierung herunterladen](#)
2. Folgen Sie den Anweisungen unter [Richten Sie Ihren Arbeitsbereich und Ihr Projekt für die Portierung ein](#), um die Dateien und Ordner in Ihrem FreeRTOS-Download für die Portierung und das Testen zu konfigurieren.
3. Folgen Sie den Anweisungen unter [Portierung der FreeRTOS-Bibliotheken](#), um die FreeRTOS-Bibliotheken auf Ihr Gerät zu portieren. Jedes Portierungsthema enthält auch Anweisungen zum Testen der Ports.

Portierung – Häufig gestellte Fragen

Was ist ein FreeRTOS-Port?

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

ermöglicht es den APIs, auf dem Board zu arbeiten, und implementiert die erforderliche Integration mit den Gerätetreibern und BSPs, die vom Plattformhersteller 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-Portierungshandbuchs nicht beantwortet werden, [sehen 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 GitHub ([FreeRTOS-LTS](#)) oder ([FreeRTOS](#)).

Note

Wir empfehlen, das Repository zu klonen. Durch das Klonen können Sie Updates für den Haupt-Branch leichter abrufen, sobald 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 der Kombination entsprechen, die in der `manifest.yml` Datei im FreeRTOS- oder FreeRTOS-LTS-Repository aufgeführt ist.

Nachdem Sie FreeRTOS heruntergeladen oder geklont haben, können Sie mit der Portierung der FreeRTOS-Bibliotheken auf Ihr Board beginnen. Weitere Informationen finden Sie unter [Richten Sie Ihren Arbeitsbereich und Ihr Projekt für die Portierung ein](#) und dann unter [Portierung der FreeRTOS-Bibliotheken](#).

Richten Sie Ihren Arbeitsbereich und Ihr Projekt für die Portierung ein

Führen Sie die unten genannten Schritte durch, um Ihren -Ink.

- 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 boardspezifischen Treiber in Ihr Projekt auf.

Sobald dein Workspace eingerichtet ist, kannst du mit der Portierung einzelner FreeRTOS-Bibliotheken beginnen.

Portierung der FreeRTOS-Bibliotheken

Bevor Sie mit der Portierung beginnen, folgen Sie den Anweisungen unter. [Richten Sie Ihren Arbeitsbereich und Ihr Projekt für die Portierung ein](#)

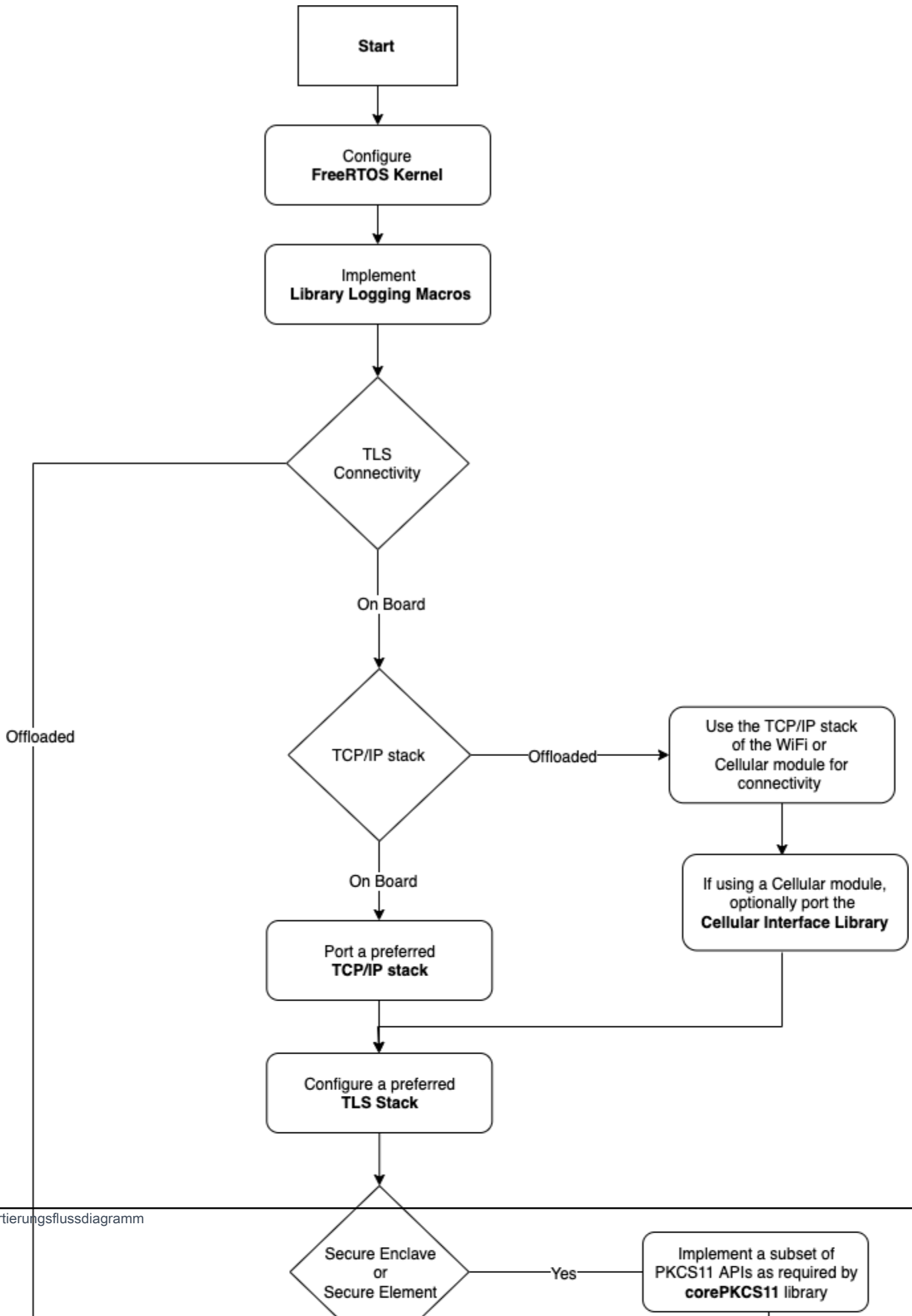
Das [Flussdiagramm zur FreeRTOS-Portierung](#) beschreibt die Bibliotheken, die für die Portierung benötigt werden.

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

1. [Konfigurieren eines FreeRTOS-Kernel-Ports](#)
2. [Implementierung der Bibliotheksprotokollmakros](#)
3. [Portieren eines TCP/IP-Stacks](#)
4. [Portierung der Netzwerktransportschnittstelle](#)
5. [Portierung der CorePKCS11-Bibliothek](#)
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.



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-Kernel-Ports](#).

FreeRTOS verwendet den FreeRTOS-Kernel für Multitasking und Intertask-Kommunikation. Weitere Informationen finden Sie in den [Grundlagen des FreeRTOS-Kernels](#) im FreeRTOS-Benutzerhandbuch und auf [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-Kernel-Ports unterstützt. Änderungen an diesen Ports werden im Programm 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-Kernel-Port 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 [Richten Sie Ihren Arbeitsbereich und Ihr Projekt für die Portierung ein](#).

Konfigurieren des FreeRTOS-Kernels

Der FreeRTOS-Kernel wird mithilfe einer Konfigurationsdatei namens `angepasstFreeRTOSConfig.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 Ausgabekonzole zu protokollieren.
- Stellen Sie sicher, dass die Nachricht wie erwartet an die Konsole ausgegeben wird.

Implementierung der Bibliotheksprotokollmakros

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 angegeben werden. Die Empfehlungen lauten:

- Makros sollten C89 Style Logging unterstützen.
- Die Protokollierung sollte threadsicher sein. Protokollzeilen aus mehreren Aufgaben dürfen sich nicht überschneiden.
- Logging-APIs dürfen nicht blockieren und müssen Anwendungsaufgaben von der Blockierung 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 überschneiden.
- Führen Sie einen Test durch, um sicherzustellen, dass die Logging-APIs I/O nicht blockieren.

- Testen Sie Logging-Makros mit verschiedenen Standards, z. B. C89, C99 Style Logging.
- Testen Sie die Protokollierungsmakros, indem Sie verschiedene Protokollebenen festlegen Debug, z. B. InfoError,, undWarning.

Portieren eines TCP/IP-Stacks

Dieser Abschnitt enthält Anweisungen zum Portieren und Testen von integrierten TCP/IP-Stacks. Wenn Ihre Plattform die TCP/IP- und TLS-Funktionalität auf einen separaten Netzwerkprozessor oder ein separates Netzwerkmodul auslagert, können Sie diesen Abschnitt zur Portierung überspringen und [Portierung der Netzwerktransportschnittstelle](#).

[Kostenloses RTOS+TCP](#) ist ein nativer TCP/IP-Stack für den FreeRTOS-Kernel. FreeRTOS+TCP wird vom FreeRTOS Engineering-Team entwickelt und verwaltet und ist der empfohlene TCP/IP-Stack für die Verwendung mit FreeRTOS. Weitere Informationen finden Sie unter [Portierung von FreeRTOS+TCP](#). Alternativ können Sie den TCP/IP-Stack eines Drittanbieters verwenden [Lwip](#). Die in diesem Abschnitt bereitgestellte Testanweisung 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 [Richten Sie Ihren Arbeitsbereich und Ihr Projekt für die Portierung ein](#).

- 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 die [GitHub](#) Verzeichnis, um zu sehen, ob ein Port zu Ihrem Board bereits 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`. Besuchen Sie die [GitHub](#) Repository für eine Vorlage.

Nachdem Sie einen Port erstellt haben oder wenn ein Port bereits vorhanden ist, erstellen Sie `FreeRTOSIPConfig.h`, und bearbeiten Sie die Konfigurationsoptionen so, dass 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, gehen Sie zum Testen wie folgt vor:

- Stellen Sie eine Implementierung bereit für `connect/disconnect/send/receive` APIs bei Transportschnittstellentests.
- Richten Sie einen Echo-Server im Klartext-TCP-Verbindungsmodus ein und führen Sie Transportschnittstellentests durch.

Note

Um ein Gerät offiziell für FreeRTOS zu qualifizieren, müssen Sie, falls Ihre Architektur die Portierung eines TCP/IP-Softwarestacks erfordert, den portierten Quellcode des Geräts anhand von Transportschnittstellentests im Klartext-TCP-Verbindungsmodus mit AWS IoT Device Tester. Folgen Sie den Anweisungen in [Verwenden AWS IoT Device Tester für](#)

[FreeRTOS](#) in der [FreeRTOS Benutzerhandbuch](#) aufstellen. AWS IoT Device Tester für die Portvalidierung. 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 CorePKCS11-Bibliothek

Der Public Key Cryptography Standard #11 definiert eine plattformunabhängige API zur Verwaltung und Verwendung kryptografischer Token. [STCK. 11](#) bezieht sich auf den Standard und die durch ihn definierten APIs. Die kryptografische PKCS #11 -API abstrahiert die Speicherung von Schlüsseln, ruft Eigenschaften für kryptografische Objekte ab und definiert 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 zu unterstützenden PKCS #11 -APIs aufgeführt.

Anwendungsfälle

Anwendungsfall	Erforderliche PKCS #11 -API-Familie
Alle	Sitzung initialisieren, abschließen, öffnen/schließen, GetSlotList, Einloggen
Bereitstellung	GenerateKeyPair, CreateObject, DestroyObject, InitToken, GetTokenInfo
TLS	Zufällig, Zeichen, 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 kryptografische Hardware enthält Komponenten mit Funktionen, die den Export kryptografischer 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 verwende ich FreeRTOS CorePKCS11

Die CorePKCS11-Bibliothek enthält eine softwarebasierte Implementierung der PKCS #11 - Schnittstelle (API), die die kryptografische Funktionalität von verwendet [TLS einbetten](#). Dies ist für Rapid-Prototyping- und Evaluierungsszenarien vorgesehen, in denen die Hardware nicht über eine spezielle kryptografische Hardware verfügt. In diesem Fall müssen Sie nur CorePKCS11 PAL implementieren, damit die softwarebasierte CorePKCS11-Implementierung mit Ihrer Hardwareplattform funktioniert.

Portierung von CorePKCS11

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 CorePKCS11-Bibliothek stellen Geräten Anmeldeinformationen zur Verfügung und programmieren das Gerät anschließend mit einer neuen Anwendung neu, die über die CorePKCS11-Schnittstelle auf diese Anmeldeinformationen zugreift. Die CorePKCS11 PAL-Ports müssen einen Speicherort für Folgendes bereitstellen:

- Das Geräteclient-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 einen sicheren Bootloader und over-the-air (OTA) - Aktualisierungen
- Ein Just-In-Time-Bereitstellungszertifikat

Einschließen [die Header-Datei](#) und implementieren Sie die definierten PAL-APIs.

PAL-APIs

Funktion	Beschreibung
PKCS11_PAL_Initialize	Initialisiert die PAL-Ebene. Wird von der CorePKCS11-Bibliothek 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 FreeRTOS CorePKCS11-Bibliothek verwenden oder die erforderliche Teilmenge der PKCS11-APIs implementieren, müssen Sie die FreeRTOS-PKCS11-Tests bestehen. Diese testen, ob die erforderlichen Funktionen für FreeRTOS-Bibliotheken erwartungsgemäß funktionieren.

Darüber hinaus beschreibt dieser Abschnitt, wie Sie die FreeRTOS SDK-11 Tests mit den Qualifizierungstests lokal ausführen können.

Voraussetzungen

Um die FreeRTOS PKCS11-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`

(Siehe [README.md](#) Datei für die Integrationstests der FreeRTOS-Bibliotheken für PKCS #11 auf GitHub.)

Tests portieren

- Fügt hinzu [FreeRTOS-Bibliotheken — Integrationstests](#) als Submodul in Ihr Projekt. Das Submodul kann in einem beliebigen Verzeichnis des Projekts platziert werden, sofern es erstellt werden kann.
- Kopieren `config_template/`
`test_execution_config_template.h` und `config_template/`
`test_param_config_template.h` zu einem Projektspeicherort im Build-Pfad und benennen sie um `intest_execution_config.h` und `intest_param_config.h`.
- Fügen Sie relevante Dateien in das Build-System ein. Wenn Sie verwenden `CMake`, `qualification_test.cmake` und `src/pkcs11_tests.cmake` kann verwendet werden, um relevante Dateien einzuschließen.
- Implementieren `UNITY_OUTPUT_CHARS` sodass Testausgabeprotokolle und Geräteprotokolle nicht ineinander überlappen.
- Integrieren Sie das MbedTLS, das das Ergebnis des Cryptoki-Vorgangs verifiziert.
- Rufen Sie `anRunQualificationTest()` aus der Anwendung.

Tests konfigurieren

Die PKCS11-Testsuite muss entsprechend der PKCS11-Implementierung konfiguriert werden. Darüber hinaus beschreibt die folgende Tabelle die für PKCS11-Tests in `test_param_config.h` Header-Datei.

PKCS11-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 Referenzen.
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, um das JITP zu aktivieren <code>codeverify</code> testen.
PKCS11_TEST_LABEL_CODE_VERIFICATION_KEY	Die Bezeichnung des in JITP verwendeten Code-Bestätigungsschlüssels <code>codeverify</code> testen.
PKCS11_TEST_LABEL_JITP_CERTIFICATE	Die Bezeichnung des in JITP verwendeten JITP-Zertifikats <code>codeverify</code> testen.
PKCS11_TEST_LABEL_ROOT_CERTIFICATE	Die Bezeichnung des in JITP verwendeten Stammzertifikats <code>codeverify</code> testen.

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

- Mindestens eine der folgenden Tastenfunktionskonfigurationen:
 - PKCS11_TEST_RSA_KEY_SUPPORT
 - PKCS11_TEST_EC_KEY_SUPPORT
- Darüber hinaus beschreibt dieser Abschnitt mindestens eine der wichtigsten Bereitstellungskonfigurationen:
 - PKCS11_TEST_IMPORT_PRIVATE_KEY_SUPPORT
 - PKCS11_TEST_GENERATE_KEYPAIR_SUPPORT
 - PKCS11_TEST_PREPROVISIONED_SUPPORT

Der Test der vorab bereitgestellten Geräte-Anmeldeinformationen muss unter den folgenden Bedingungen ausgeführt werden:

- PKCS11_TEST_PREPROVISIONED_SUPPORT muss aktiviert und andere Bereitstellungsmechanismen deaktiviert sein.

- Entweder nur eine Schlüsselfunktion `PKCS11_TEST_RSA_KEY_SUPPORT` oder `PKCS11_TEST_EC_KEY_SUPPORT`, ist aktiviert.
- Richten Sie die vorab bereitgestellten Tastenbeschriftungen entsprechend Ihrer Schlüsselfunktion ein, einschließlich `PKCS11_TEST_LABEL_DEVICE_PRIVATE_KEY_FOR_TLS`, `PKCS11_TEST_LABEL_DEVICE_PRIVATE_KEY_FOR_TLS`. Diese Anmeldeinformationen müssen vorhanden sein, bevor der Test ausgeführt wird.

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

Note

Die Objekte mit Beschriftungen `PKCS11_TEST_LABEL_DEVICE_PRIVATE_KEY_FOR_TLS`, `PKCS11_TEST_LABEL_DEVICE_PRIVATE_KEY_FOR_TLS` während des Tests zerstört, wenn entweder `PKCS11_TEST_GENERATE_KEYPAIR_SUPPORT` oder `PKCS11_TEST_GENERATE_KEYPAIR_SUPPORT` aktiviert.

Ausführen von Tests

In diesem Abschnitt wird beschrieben, wie Sie die PKCS11-Schnittstelle mit den Qualifikationstests lokal testen können. Darüber hinaus können Sie IDT verwenden, um die Ausführung zu automatisieren. Siehe [AWS IoT Device Tester für FreeRTOS](#) in der FreeRTOS-Benutzerhandbuch für Einzelheiten.

Darüber hinaus beschreibt die folgende Anleitung, wie die Tests durchgeführt werden:

- Öffnen Sie `test_execution_config.h` und definieren Sie `CORE_PKCS11_TEST_ENABLED` bis 1.
- Erstellen Sie die Anwendung und flashen Sie sie auf Ihr Gerät, um sie auszuführen. Die Testergebnisse werden an die serielle Schnittstelle ausgegeben.

Darüber hinaus beschreibt dieser Abschnitt 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 validierenAWS IoT Device Tester. Folgen Sie den Anweisungen in [BenutzenAWS IoT Device Testerfür FreeRTOS](#) im FreeRTOS User Guide zur EinrichtungAWS IoT Device Testerfür die Portvalidierung. Um den Port einer bestimmten

Bibliothek zu testen, muss die richtige Testgruppe in der `device.json` Datei im `AWS IoT Device Tester configs` Ordner.

Portierung der Netzwerktransportschnittstelle

Integration der TLS-Bibliothek

Verwenden Sie für Transport Layer Security (TLS) -Authentifizierung Ihren bevorzugten TLS Stack. Wir empfehlen die Verwendung [TLS einbetten](#) weil es mit FreeRTOS-Bibliotheken getestet wurde. Ein Beispiel dafür finden Sie hier [GitHub](#) Endlager.

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, die unterstützt werden von AWS IoT](#).

Portierung der Network Transport Interface-Bibliothek

Sie müssen eine Netzwerktransportschnittstelle implementieren, um sie verwenden zu können [CoreMQTT](#) und [CoreHTTP](#). Die Netzwerktransportschnittstelle enthält Funktionszeiger und Kontextdaten, die zum Senden und Empfangen von Daten über eine einzelne Netzwerkverbindung erforderlich sind. Siehe [Transport Interface](#) für weitere Informationen. 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

- Hinzufügen [FreeRTOS-Bibliotheken — Integrationstests](#) als Submodul in Ihr Projekt. Es spielt keine Rolle, wo sich das Submodul im Projekt befindet, solange es gebaut werden kann.

- Kopieren `config_template/test_execution_config_template.h` und `config_template/test_param_config_template.h` in einen Projektspeicherort im Build-Pfad und benennen sie um `intest_execution_config.h` und `intest_param_config.h`.
- Nehmen Sie relevante Dateien in das Build-System auf. Wenn Sie verwenden `Make`, `qualification_test.cmake` und `src/transport_interface_tests.cmake` werden verwendet, um relevante Dateien einzuschließen.
- Implementieren Sie die folgenden Funktionen an einem geeigneten Projektstandort:
 - `INnetwork connect function`: Die Signatur ist definiert durch `NetworkConnectFunc` in `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.
 - `INnetwork disconnect function`: Die Signatur ist definiert durch `NetworkDisconnectFunc` in `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()`: Das ist definiert in `src/transport_interface/transport_interface_tests.h`. Die Implementierung muss genau den gleichen Namen und die gleiche Signatur haben, wie sie in `transport_interface_tests.h` definiert sind. Diese Funktion nimmt einen Zeiger auf ein `TransportInterfaceTestParam` Struktur. Es wird die Felder in der `TransportInterfaceTestParam` Struktur, die vom Transportschnittstellentest verwendet wird.
- Implementieren `UNITY_OUTPUT_CHARS` sodass sich Testausgabeprotokolle nicht mit Geräteprotokollen überschneiden.
- Rufen Sie `runQualificationTest()` aus der Anwendung. 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 `test_param_config.h` auf 1 gesetzt, 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 des

Echo-Servers und als Client-Zertifikat beim Aufbau einer TLS-Verbindung mit dem Echo-Server. 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_P` Root-CA des Echo-Servers, Client-Zertifikat und privater 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 Qualifikationstests lokal testen können. Weitere Informationen finden Sie unter [transport_interface](#) Abschnitt der FreeRTOS-Libraries-Integrationstests zu GitHub.

Alternativ können Sie auch IDT verwenden, um die Ausführung zu automatisieren. Siehe [AWS IoT Device Tester für FreeRTOS](#) in der FreeRTOS Benutzerhandbuch Weitere Informationen.

Den Test aktivieren

Öffnen Sie `test_execution_config.h` und definieren Sie `TRANSPORT_INTERFACE_TEST_ENABLED` bis 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. Wenn Sie noch keine haben, [FreeRTOS-Bibliotheken — Integrationstests](#) GitHub Das Repository hat eine Echo-Server-Implementierung.

Konfiguration des Projekts zum Testen

Im `test_param_config.h`,
aktualisieren Sie `ECHO_SERVER_ENDPOINT` und `ECHO_SERVER_PORT` zum Endpunkt- und Server-Setup im vorherigen Schritt.

Anmeldeinformationen für die Einrichtung (auf dem Gerät generierter Schlüssel)

- Eingestellt `ECHO_SERVER_ROOT_CA` zum Serverzertifikat des Echoservers.

- Einstellen `FORCE_GENERATE_NEW_KEY_PAIR` auf 1, um ein key pair zu generieren und den öffentlichen Schlüssel zu erhalten.
- Stellen Sie ein `FORCE_GENERATE_NEW_KEY_PAIR` nach der Schlüsselgenerierung zurück auf 0.
- Verwenden Sie den öffentlichen Schlüssel sowie den Serverschlüssel und das Zertifikat, um das Client-Zertifikat zu generieren.
- Setzen Sie `TRANSPORT_CLIENT_CERTIFICATE` zum generierten Client-Zertifikat.

Anmeldeinformationen einrichten (Schlüssel importieren)

- Einstellen `ECHO_SERVER_ROOT_CA` zum Serverzertifikat des Echoservers.
- Einstellen `TRANSPORT_CLIENT_CERTIFICATE` zum vorgenerierten Client-Zertifikat.
- Setzen Sie `TRANSPORT_CLIENT_PRIVATE_KEY` zum vorgenerierten privaten Client-Schlüssel.

Erstellen und flashen Sie die Anwendung

Erstellen und flashen Sie die Anwendung mit der Toolchain Ihrer Wahl.

Wenn `runQualificationTest()` wird aufgerufen, 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 validieren mit AWS IoT Device Tester. Folgen Sie den Anweisungen unter [Verwenden AWS IoT Device Tester für FreeRTOS](#) in der FreeRTOS Benutzerhandbuch einrichten AWS IoT Device Tester für die Portvalidierung. Um den Port einer bestimmten Bibliothek zu testen, muss die richtige Testgruppe in der `device.json` Datei im AWS IoT Device Tester configs Ordner.

Konfiguration der CoreMQTT-Bibliothek

Geräte an der Grenze können das MQTT-Protokoll verwenden, um mit der AWS-Cloud zu kommunizieren. AWS IoT hostet einen MQTT-Broker, der Nachrichten an verbundene Geräte an der Grenze sendet und von ihnen 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 alle MQTT-Tests bestehen, um sich zu qualifizieren. Weitere Informationen finden Sie unter [CoreMQTT Library](#) im FreeRTOS-Benutzerhandbuch.

Voraussetzungen

Um die CoreMQTT-Bibliothekstests einzurichten, benötigen Sie einen Netzwerktransportschnittstellenport. Weitere Informationen [Portierung der Netzwerktransportschnittstelle](#) finden Sie unter.

Testen

Führen Sie CoreMQTT-Integrationstests durch:

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

Referenz-MQTT-Demo erstellen

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-Verbindung	Ja
TLS	TLS-Unterstützung SupportAWS IoT Cipher Suites	Eine empfohlene Verschlüsselungssuite

Testfall	Testfälle	Erforderliche Tests
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 Publish	Ja
MQTT	MQTT ClientPuback QoS 1	Ja
MQTT	MQTT Nein Ack PingResp	Ja

Konfiguration der CoreHTTP-Bibliothek

Geräte am Edge können das HTTP-Protokoll verwenden, um mit der AWS Cloud zu kommunizieren. 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 mithilfe von Code Signing für AWS IoT.

[Weitere Informationen finden Sie unter Over-the-Air-Updates für FreeRTOS im FreeRTOS-Benutzerhandbuch zusammen mit der O Update-Dokumentation.AWS IoT ver-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. [Richten Sie Ihren Arbeitsbereich und Ihr Projekt für die Portierung ein](#)
- 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
<code>otaPal_Abort</code>	Stoppt ein OTA-Update.
<code>otaPal_CreateFileForRx</code>	Erstellt eine Datei zum Speichern der empfangenen Datenblöcke.
<code>otaPal_CloseFile</code>	Schließt die angegebene Datei. Dadurch wird die Datei möglicherweise authentifiziert, wenn Sie Speicher verwenden, der kryptografischen Schutz implementiert.
<code>otaPal_WriteBlock</code>	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 .
<code>otaPal_ActivateNewImage</code>	Aktiviert oder startet das neue Firmware-Abbild. Bei einigen Anschlüssen kehrt diese Funktion nicht zurück, wenn das Gerät programmgesteuert synchron zurückgesetzt wird.

Funktionsname	Beschreibung
<code>otaPal_SetPlatformImageState</code>	Erfüllt die Anforderungen der Plattform, um das neueste OTA-Firmware-Image (oder -Bundle) zu akzeptieren oder abzulehnen. Informationen zur Implementierung dieser Funktion finden Sie in der Dokumentation zu Ihrem Motherboard (Plattform) und zur Architektur.
<code>otaPal_GetPlatformImageState</code>	Ruft den Status des OTA-Updates auf.

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

Funktionsname	Beschreibung
<code>otaPal_CheckFileSignature</code>	Überprüft die Signatur der angegebenen Datei.
<code>otaPal_ReadAndAssumeCertificate</code>	Liest das angegebene Signaturgeberzertifikat aus dem Dateisystem und gibt es an den Aufrufer zurück.
<code>otaPal_ResetDevice</code>	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 durch 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
OTA E2E Greater Version	Happy Path-Test für regelmäßige OTA-Updates. Es erstellt ein Update mit einer neueren Version, die das Gerät erfolgreich aktualisiert.
OTA E2E Back To Back Downloads	Dieser Test erstellt 3 aufeinanderfolgende OTA-Updates. Es wird erwartet, dass das Gerät dreimal hintereinander aktualisiert wird.
OTA E2E Rollback If Unable To Connect After Update	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.
OTA E2E Same Version	Dieser Test bestätigt, dass das Gerät die eingehende Firmware ablehnt, wenn die Version gleich bleibt.
OTA E2E Unsigned Image	Dieser Test überprüft, ob das Gerät ein Update ablehnt, wenn das Image nicht signiert ist.
OTA E2E Untrusted Certificate	Dieser Test überprüft, ob das Gerät ein Update ablehnt, wenn die Firmware mit einem nicht vertrauenswürdigen Zertifikat signiert ist.
OTA E2E Previous Version	Dieser Test überprüft, ob das Gerät eine ältere Update-Version ablehnt.
OTA E2E Incorrect Signing Algorithm	Verschiedene Geräte unterstützen unterschiedliche Signier- und Hash-Algorithmen. Dieser Test bestätigt, dass das Gerät das OTA-Update

Testfall	Beschreibung
	e nicht bestanden hat, wenn es mit einem nicht unterstützten Algorithmus erstellt wurde.
OTA2E2EDisconnectResume	<p>Dies ist der Happy-Path-Test für die Suspend- und Resume-Funktion. Dieser Test erstellt ein 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.</p>
OTA2E2EDisconnectCancelUpdate	<p>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 OTA2E2EDisconnectResume 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.</p>

Testfall	Beschreibung
OTA2E2PresignedUrlExpired	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 fortzusetzen.
OTA2E2UpdatesCancel1st	Dieser Test erstellt zwei OTA-Updates hintereinander. 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.
OTA2E2CancelThenUpdate	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.
OTA2E2ImageCrashed	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 `definiertota_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 aufgerufene `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 End-to-End-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 `configs` aktiviert sein.

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 Gegners. Unter Berücksichtigung der Ziele und Methoden des Gegners wird eine Bedrohungsliste

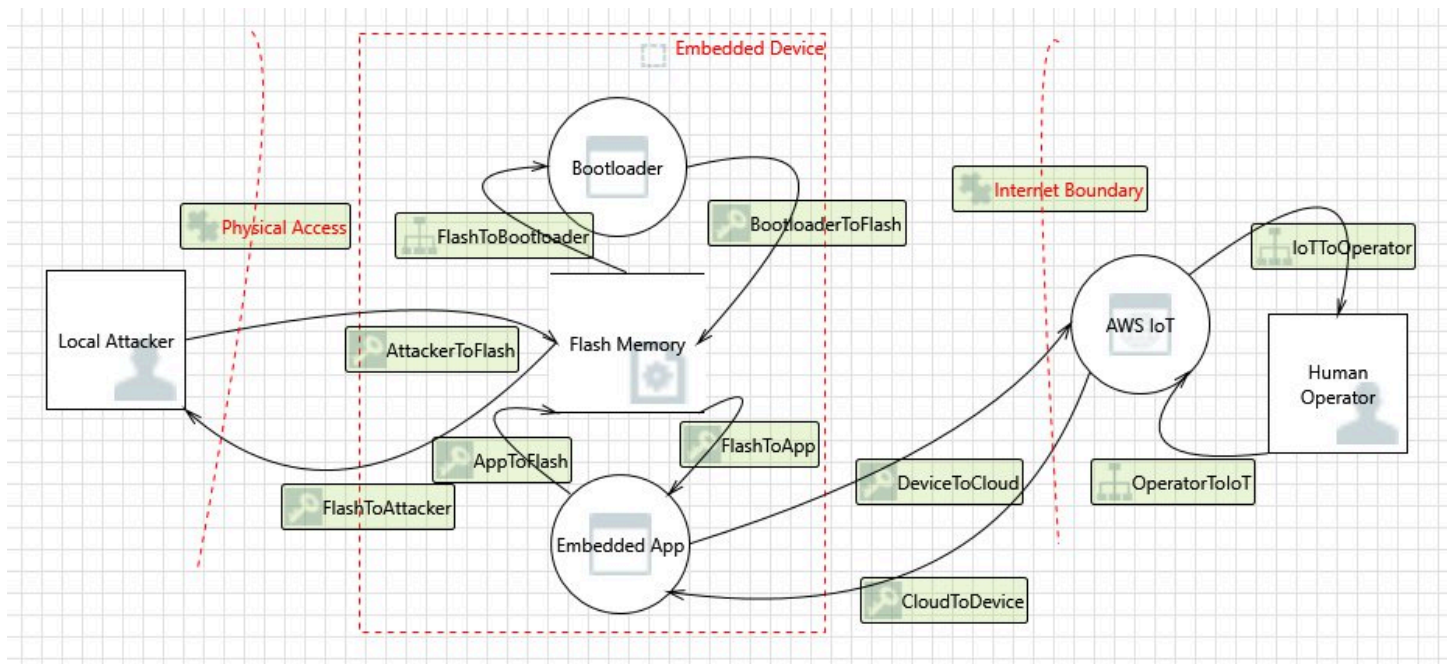
erstellt. Bedrohungen sind Angriffe auf eine Ressource oder eine Komponente durch einen Gegner. Die Liste wird priorisiert und zur Identifizierung und Erstellung von Lösungen zur Risikominderung verwendet. Bei der Auswahl einer Risikominderungslösung sollten die Kosten für deren Implementierung und Wartung 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 bösesartiges 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-Offload-Mobilfunkabstraktionsschicht. Weitere Informationen finden Sie in der [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 koexistieren, sodass Entwickler eines davon auswählen müssen, um es in den zu integrieren [Portierung der Netzwerktransportschnittstelle](#).

Note

Wenn das Mobilfunkmodul TLS-Offload unterstützen kann oder AT-Befehle nicht unterstützt, können Entwickler ihre eigene Mobilfunkabstraktion implementieren, um es in das zu integrieren [Portierung der Netzwerktransportschnittstelle](#).

Migration von MQTT Version 3 zu CoreMQTT

In diesem [Migrationsleitfaden](#) wird erklärt, wie Sie Anwendungen von MQTT zu CoreMQTT migrieren.

Migrieren von der Version 1 zu Version 3 für OTA-Anwendungen

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

Note

Die OTA-APIs der Version 2 sind dieselben wie die OTA v3-APIs. Wenn Ihre Anwendung also Version 2 der APIs verwendet, sind für API-Aufrufe keine Änderungen erforderlich. Wir empfehlen jedoch, Version 3 der Bibliothek zu integrieren.

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 OTA-Bibliotheksversion 1 und Version 3

OTA-Version 1-APIline-1-	OTA-Version 3	Beschreibung der Änderungen
OTA_AgentInit	OTA_Init	Die Eingabeparameter werden ebenso geändert wie der von der Funktion zurückgegebene Wert aufgrund von Änderungen in der Implementierung in OTA v3. Einzelheiten finden Sie im Abschnitt für OTA_Init weiter unten.
OTA_AgentShutdown	OTA_Shutdown	Änderung der Eingabeparameter einschließlich eines

OTA-Version 1-APLine-1-	OTA-Version 3	Beschreibung der Änderungen
		zusätzlichen Parameters für eine optionale Abmeldung von MQTT-Themen. Einzelheiten finden Sie im Abschnitt für OTA_Shutdown weiter unten.
OTA_GetAgentState	OTA_GetState	Der API-Name wird geändert, ohne dass der Eingabeparameter geändert wird. Der Rückgabewert ist derselbe, aber das Enum und die Mitglieder wurden umbenannt. Einzelheiten finden Sie im Abschnitt für OTA_GetState weiter unten.
–	OTA_GetStatistics	Es wurde eine neue API hinzugefügt, die die APIs OTA_GetPacketsReceivedGetPacketsQueued, OTA_GetPacketsProcessed, OTA_ ersetztGetPacketsDropped. Einzelheiten finden Sie im Abschnitt für OTA_GetStatistics weiter unten.
OTA_GetPacketsReceived	–	Diese API wurde aus Version 3 entfernt und durch OTA_ ersetztGetStatistics.
OTA_GetPacketsQueued	–	Diese API wurde aus Version 3 entfernt und durch OTA_ ersetztGetStatistics.

OTA-Version 1-APIline-1-	OTA-Version 3	Beschreibung der Änderungen
OTA_GetPacketsProcessed	–	Diese API wurde aus Version 3 entfernt und durch OTA_ersetztGetStatistics.
OTA_GetPacketsDropped	–	Diese API wurde aus Version 3 entfernt und durch OTA_ersetztGetStatistics.
OTA_ActivateNewImage	OTA_ActivateNewImage	Die Eingabeparameter sind dieselben, aber der OTA-Rückgabe-Fehlercode wird umbenannt und in Version 3 der OTA-Bibliothek werden neue Fehlercodes hinzugefügt. Einzelheiten finden Sie im Abschnitt für OTA_ActivateNewImage .
OTA_SetImageState	OTA_SetImageState	Die Eingabeparameter sind dieselben und wurden umbenannt, der OTA-Rückgabe-Fehlercode wird umbenannt und in Version 3 der OTA-Bibliothek werden neue Fehlercodes hinzugefügt. Einzelheiten finden Sie im Abschnitt für OTA_SetImageState .

OTA-Version 1-APIline-1-	OTA-Version 3	Beschreibung der Änderungen
OTA_GetImageState	OTA_GetImageState	Die Eingabeparameter sind dieselben, das Rückgabe-Enum wurde in Version 3 der OTA-Bibliothek umbenannt . Einzelheiten finden Sie im Abschnitt für OTA_GetImageState .
OTA_aussetzen	OTA_aussetzen	Die Eingabeparameter sind dieselben, der OTA-Rückgabe-Fehlercode wird umbenannt und in Version 3 der OTA-Bibliothek werden neue Fehlercodes hinzugefügt. Einzelheiten finden Sie im Abschnitt für OTA_Suspend.
OTA_Lebenslauf	OTA_Lebenslauf	Der Eingabeparameter für die Verbindung wird entfernt, wenn die Verbindung in der OTA-Demo/Anwendung behandelt wird, der OTA-Rückgabe-Fehlercode wird umbenannt und in Version 3 der OTA-Bibliothek werden neue Fehlercodes hinzugefügt. Einzelheiten finden Sie im Abschnitt für OTA_Resume.

OTA-Version 1-APIline-1-	OTA-Version 3	Beschreibung der Änderungen
OTA_CheckForUpdate	OTA_CheckForUpdate	Die Eingabeparameter sind dieselben, der OTA-Rückgabe-Fehlercode wird umbenannt und in Version 3 der OTA-Bibliothek werden neue Fehlercodes hinzugefügt. Einzelheiten finden Sie im Abschnitt für OTA_CheckForUpdate .
–	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	Eine neue API wurde hinzugefügt und fügt das Ereignis an den Ende der OTA-Ereigniswarteschlange hinzu. Es wird von internen OTA-Modulen verwendet, um die Agentenaufgabe zu signalisieren. Einzelheiten finden Sie im Abschnitt für OTA_SignalEvent .
–	OTA_ERR_STR-Fehler	Neue API für die Konvertierung von Fehlercode in Zeichenketten bei OTA-Fehlern.

OTA-Version 1-APline-1-	OTA-Version 3	Beschreibung der Änderungen
–	OTA_JobParse _str-Fehler	Neue API für die Konvertierung von Fehlercode in Zeichenketten bei Job Parsing-Fehlern.
–	OTA_OsStatus _str-Fehler	Neue API für die Konvertierung von Statuscode in Zeichenketten für den OTA OS-Portstatus.
–	OTA_PalStatus _str-Fehler	Neue API für die Konvertierung von Statuscode in Zeichenketten für den OTA-PAL-Portstatus.

Beschreibung der erforderlichen Änderungen

OTA_Init

Bei der Initialisierung des OTA-Agenten in v1 wird die `OTA_AgentInit` API verwendet, die Parameter für den Verbindungskontext, den Dingnamen, den vollständigen Callback und den Timeout als Eingabe verwendet.

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

Diese API wurde jetzt auf `OTA_Init` Parameter für die Puffer geändert, die für OTA, OTA-Schnittstellen, Dingnamen und 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 die MQTT/HTTP-Operationen von ihren jeweiligen Schnittstellen in der OTA-Demo/Anwendung abgewickelt werden.

xTicksToWarte -

Der Parameter ticks to wait 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 in OtaAppCallback und sein Typ wird in OtaAppCallback_t geändert.

Neue Eingabeparameter -

pOtaBuffer

Die Anwendung muss die Puffer zuweisen und sie während der Initialisierung mithilfe der OtaAppBuffer_t-Struktur an die OTA-Bibliothek übergeben. Die erforderlichen Puffer unterscheiden sich geringfügig, je nachdem, welches Protokoll zum Herunterladen der Datei verwendet wird. 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.

Erforderliche Puffer, 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
};
```

Erforderliche Puffer, 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 eine Referenz auf die OTA-Schnittstellen für den Typ `OtaInterfaces_t`. Dieser Satz von Schnittstellen muss an die OTA-Bibliothek übergeben

werden und umfasst in der Betriebssystemschnittstelle die MQTT-Schnittstelle, die HTTP-Schnittstelle und die Schnittstelle zur Plattformabstraktionsschicht.

OTA-Betriebssysteme-

Die funktionale Schnittstelle des OTA-Betriebssystems besteht aus einer Reihe von 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. Dazu gehören die Verwaltung von Ereignissen, Timern und Speicherzuweisung. 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 mit dem bereitgestellten POSIX-Port -

```
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 APIs, die in einer Bibliothek implementiert werden müssen, damit die OTA-Bibliothek einen Dateiblock vom Streaming-Dienst herunterladen kann.

Beispiel für die Verwendung der CoreMQTT Agent-APIs aus der [OTA über 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 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 abrufen. Dies ist optional, sofern Sie die OTA-Bibliothek nicht so konfigurieren, dass sie von einer vorsignierten URL statt von einem Streaming-Dienst heruntergeladen wird.

Beispiel für die Verwendung der 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

Die OTA-PAL-Schnittstelle besteht aus einer Reihe von APIs, die implementiert werden müssen, damit das Gerät die OTA-Bibliothek verwenden kann. Die gerätespezifische Implementierung für den OTA-PAL wird der Bibliothek in der Benutzeranwendung zur Verfügung gestellt. Diese Funktionen werden von der Bibliothek verwendet, um Downloads zu speichern, zu verwalten und zu authentifizieren.

```
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 wird vom OTA-Agentenstatus in den OTA-Fehlercode geändert. Bitte beachten Sie [AWS IoTOver-the-air Update v3.0.0: OtaErr_t](#).

OTA_Shutdown

In der Version 1 der OTA-Bibliothek war die API, die zum Herunterfahren des OTA-Agenten verwendet wurde, `OTA_AgentShutdown` was jetzt 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, bis der OTA-Agent den Shutdown-Vorgang abgeschlossen hat. Wenn dieser Wert auf Null gesetzt ist, kehrt die Funktion sofort zurück, ohne zu warten. Der tatsächliche Status wird an den Anrufer zurückgegeben. Der Agent schläft für diese Zeit nicht, sondern wird für geschäftiges Looping verwendet.

Neuer Eingabeparameter -

Abmeldekennzeichen -

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

Änderungen im Gegenzug -

OtaState_t -

Das Enum für den OTA-Agentenstatus und seine Mitglieder werden umbenannt. Bitte beachten Sie [AWS IoTOver-the-air Update v3.0.0](#).

OTA_GetState

Der API-Name wurde von OTA_AgentGetState in OTA_ geändertGetState.

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 -

Das Enum für den OTA-Agentenstatus und seine Mitglieder werden umbenannt. Bitte beachten Sie [AWS IoTOver-the-air Update v3.0.0](#).

OTA_GetStatistics

Neue Single-API für Statistiken hinzugefügt. Sie ersetzt die APIs OTA_GetPacketsReceived, OTA_GetPacketsQueued, OTA_GetPacketsProcessed, OTA_GetPacketsDropped. In der Version 3 der OTA-Bibliothek beziehen sich die Statistiknummern außerdem nur auf den aktuellen Job.

OTA-Bibliotheken Version 1

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


OTA-Bibliotheken Version 3

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

Statistiken -

Eingabe-/Ausgabeparameter für Statistikdaten wie empfangene, verworfene, in die Warteschlange gestellte und verarbeitete Pakete für den aktuellen Job.

Ausgangsparameter -

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 OTA-Rückgabe-Fehlercode wird umbenannt und in der Version 3 der OTA-Bibliothek werden neue Fehlercodes hinzugefügt.

OTA-Bibliotheken Version 1

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

OTA-Bibliotheken Version 3

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

Das Enum des OTA-Fehlercodes für die Rückgabe wird geändert und neue Fehlercodes werden hinzugefügt. Bitte beachten Sie [AWS IoTOver-the-air Update v3.0.0: OtaErr_t](#).

Anwendungsbeispiel -

```
OtaErr_t otaErr = OtaErrNone;
```

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

OTA_SetImageState

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

OTA-Bibliotheken Version 1

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

OTA-Bibliotheken Version 3

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

Der Eingabeparameter wird in `OtaImageState_t` umbenannt. Bitte beachten Sie [AWS IoTOver-the-air Update v3.0.0](#).

Das Enum des OTA-Fehlercodes für die Rückgabe wird geändert und neue Fehlercodes werden hinzugefügt. Bitte beachten Sie [AWS IoTOver-the-air Update v3.0.0/ OtaErr_t](#).

Anwendungsbeispiel -

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

OTA_GetImageState

Die Eingabeparameter sind dieselben, das Rückgabe-Enum wurde in der Version 3 der OTA-Bibliothek umbenannt.

OTA-Bibliotheken Version 1

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

OTA-Bibliotheken Version 3

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

Das Rückgabenum wird in OtaImageState_t umbenannt. Bitte beachten Sie [AWS IoTOver-the-air Update v3.0.0: OtaImageState_t](#).

Anwendungsbeispiel -

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

OTA_aussetzen

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

OTA-Bibliotheken Version 1

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

OTA-Bibliotheken Version 3

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

Das Enum des OTA-Fehlercodes für die Rückgabe wird geändert und neue Fehlercodes werden hinzugefügt. Bitte beachten Sie [AWS IoTOver-the-air Update v3.0.0: OtaErr_t](#).

Anwendungsbeispiel -

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

OTA_Lebenslauf

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

OTA-Bibliotheken Version 1

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

OTA-Bibliotheken Version 3

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

Das Enum des OTA-Fehlercodes für die Rückgabe wird geändert und neue Fehlercodes werden hinzugefügt. Bitte beachten Sie [AWS IoTOver-the-air Update v3.0.0: OtaErr_t](#).

Anwendungsbeispiel -

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

OTA_CheckForUpdate

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

OTA-Bibliotheken Version 1

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

OTA-Bibliotheken Version 3

```
OtaErr_t OTA_CheckForUpdate( void )
```

Das Enum des OTA-Fehlercodes für die Rückgabe wird geändert und neue Fehlercodes werden hinzugefügt. Bitte beachten Sie [AWS IoTOver-the-air Update v3.0.0: OtaErr_t](#).

OTA_EventProcessingTask

Dies ist eine neue API und die Hauptereignisschleife zur Verarbeitung von Ereignissen für OTA-Updates. Es 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-Bibliotheken 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 Agententasks zu signalisieren.

OTA-Bibliotheken Version 3

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

Anwendungsbeispiel -

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

Integration der 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. Git-Submodule ermöglichen es dir, ein Git-Repository als Unterverzeichnis eines anderen Git-Repositorys zu behalten. Die Version 3 der OTA-Bibliothek wird im Repository [ota-for-aws-iot-embedded-sdk](https://github.com/aws/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

- [OTA V1.](#)
- [OTA V3.](#)

Migration von Version 1 auf Version 3 für den OTATAS-PALS-Anschluss

Die Over-the-air Updates Library führte einige Änderungen in der Ordnerstruktur und der Platzierung der Konfigurationen ein, die für die Bibliothek und die Demoanwendungen erforderlich sind. Für OTA-Anwendungen, die für die Verwendung mit Version 1.2.0 entwickelt wurden, um auf Version 3.0.0 der Bibliothek zu migrieren, 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-Portverzeichnisses wurde von `ota` bis `aktualisiertota_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 ein Enum übersetzt `OTAMainStatus_t`. Übersetzte Rückgabecodes finden Sie auf [ota_platform_interface.h](#). [Hilfsmakros](#) werden auch zum Kombinieren `OtaPalMainStatus` und `OtaPalSubStatus` Codieren und Extrahieren `OtaMainStatus` von `OtaPalStatus` und ähnlichem bereitgestellt.
- Im PAL einloggen
 - Das `DEFINE_OTA_METHOD_NAME` Makro wurde entfernt.
 - Früher: `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 zu `OTA_JsonFileSignatureKey`.

Funktionen

Die Funktionssignaturen sind in `definiertota_pal.h` und beginnen mit dem Präfix `otaPal` anstelle von `privPAL`.

Note

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

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

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

Hinweise: Erstellen Sie eine neue Empfangsdatei für die Datenblöcke, sobald sie eingehen.

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

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

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

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

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

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

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

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

Hinweise: Setzen Sie das Gerät zurück.

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

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

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

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

Variante 3:OtaPalStatus_t otaPal_Abort(OtaFileContext_t * const *pFileContext*);

Hinweise: Stoppen Sie eine OTA-Übertragung.

- Variante 1:OTA_Err_t prvPAL_SetPlatformImageState(OTA_ImageState_t *eState*);

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

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

- Variante 1:OTA_PAL_ImageState_t prvPAL_GetPlatformImageState(void);

Variante 3:OtaPalImageState_t otaPal_GetPlatformImageState(OtaFileContext_t * const *pFileContext*);

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

Datentypen

- Variante 1:OTA_PAL_ImageState_t

Datei:aws_iot_ota_agent.h

Variante 3:OtaPalImageState_t

Datei:ota_private.h

Hinweise: Der durch die Plattformimplementierung festgelegte Image-Status.

- Variante 1:OTA_Err_t

Datei:aws_iot_ota_agent.h

Variante 3:OtaErr_t OtaPalStatus_t (combination of OtaPalMainStatus_t and OtaPalSubStatus_t)

Datei:ota.h,ota_platform_interface.h

Anmerkungen: v1: Dies waren Makros, die eine Ganzzahl von 32 ohne Vorzeichen definierten. v3: Spezialisiertes Enum, das die Art des Fehlers darstellt und mit einem Fehlercode verknüpft ist.

- Variante 1:OTA_FileContext_t

Datei:aws_iot_ota_agent.h

Variante 3:otaFileContext_t

Datei:ota_private.h

Hinweise: v1: Enthält eine Aufzählung und Puffer für die Daten. v3: Enthält zusätzliche Datenlängenvariablen.

- Variante 1:OTA_ImageState_t

Datei:aws_iot_ota_agent.h

Variante 3:otaImageState_t

Datei:ota_private.h

Hinweise: OTA-Bildzustände

Konfigurationsänderungen

Die Dateiaws_ota_agent_config.h wurde umbenannt [ota_config.h](#), wodurch sich die Include-Guards von_AWS_OTA_AGENT_CONFIG_H_ zu ändernOTA_CONFIG_H_.

- Die Dateiaws_ota_codesigner_certificate.h wurde gelöscht.
- Der neue Logging-Stack zum Drucken von Debug-Meldungen 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` das in `demo/include/aws_ota_codesigner_certificate.h` definiert wurde, wurde verschoben in `ota_demo_config.h` als `otaconfigCODE_SIGNING_CERTIFICATE` und kann von den PAL-Dateien aus aufgerufen werden als:

```

static const char codeSigningCertificatePEM[] = otaconfigCODE_SIGNING_CERTIFICATE;

```

Die Datei `aws_ota_codesigner_certificate.h` wurde gelöscht.

- Die Makros `APP_VERSION_BUILD`, `APP_VERSION_MINOR`, `APP_VERSION_MAJOR` wurden hinzugefügt in `ota_demo_config.h`. Die alten Dateien mit den Versionsinformationen wurden entfernt, zum Beispiel `tests/include/aws_application_version.h` in `demo/demo_runner/aws_demo_version.c`. `libraries/c_sdk/standard/common/include/iot_appversion32.h`

Änderungen an den OTA-PAL-Tests

- Die Testgruppe „Full_OTA_Agent“ wurde zusammen mit allen zugehörigen Dateien entfernt. Diese Testgruppe war zuvor für die Qualifikation 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 benötigt wird.
- Die Testgruppen „Full_OTA_CBOR“ und „Quarantine_OTA_CBOR“ sowie alle zugehörigen Dateien wurden entfernt. Diese Tests waren nicht Teil der Qualifikationstests. Die von diesen Tests abgedeckten Funktionen werden jetzt im OTA-Repository getestet.
- Die Testdateien wurden aus dem Bibliotheksverzeichnis in `dastests/integration_tests/ota_pal` Verzeichnis verschoben.
- Die OTA-PAL-Qualifikationstests wurden aktualisiert, um Version 3.0.0 der OTA-Bibliotheks-API zu verwenden.
- Es wurde aktualisiert, wie die OTA-PAL-Tests für Tests auf das Codesignaturzertifikat zugreifen. Bisher gab es eine dedizierte 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 `ota_pal.c` definiert ist. 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-Portordners 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` in (oder erstellen Sie sie) `ota_config.h`.
 - Hinzufügen:

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

- Schließen Sie ein:

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

- Kopieren Sie die obigen Dateien in den `aws_test_config` Ordner und ersetzen Sie alle enthaltenen Dateien `ota_demo_config.h` durch `aws_test_ota_config.h`.
- Füge eine `ota_demo_config.h` Datei hinzu.
- Füge 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_bispFileContext`.
 - Aktualisieren Sie die `OTA_FileContext_t` Struktur und alle zugehörigen Variablen.
 - Aktualisieren Sie die `OTA_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 Logging-Stack zu verwenden.
 - Aktualisiere dass `signingcredentialSIGNING_CERTIFICATE_PEM` to `beotapalconfigCODE_SIGNING_CERTIFICATE`.
 - Update `otaPal_CheckFileSignature` und `otaPal_ReadAndAssumeCertificate` Funktionskommentare.
- Aktualisiere die [CMakeLists.txt](#) Datei.
- Aktualisieren Sie die IDE-Projekte.

Dokumentverlauf

Die folgende Tabelle beschreibt die Dokumentation für diese Version des Leitfadens für diese Version des Leitfadens für diese Version des Leitfadens für diese Version von FreeRTOS.

Date (Datum)	Version der Dokumentation	Verlauf ändern	FreeRTOS-Version
Mai 2022	Anleitung zur FreeRTOS-Portierung FreeRTOS-Qualifikationsleitfaden	<ul style="list-style-type: none"> 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 neues hinzugefügt Portierung der Netzwerktransportchnittstelle. Portierung der AWS IoT over-the-air 	202012.04-LTS 2021 12.00

Date (Datum)	Version der Dokumentation	Verlauf ändern	FreeRTOS-Version
		<p>(OTA-) Update-Bibliothek ist jetzt für die Qualifikation erforderlich.</p> <ul style="list-style-type: none">• Die Anleitung zur Portierung von WLAN und TLS Abstraktion wurde entfernt, da sie nicht mehr benötigt werden.• Weitere Informationen zur FreeRTOS-Qualifikation finden Sie unter Aktuelle Änderungen.	

Date (Datum)	Version der Dokumentation	Verlauf ändern	FreeRTOS-Version
Juli 2021	<p>202107.00 (Portierungsleitfaden)</p> <p>202107.00 (Qualifikationsleitfaden)</p>	<ul style="list-style-type: none"> • Veröffentlichung 2021/07/00 • Geändert Portierung der AWS IoT over-the-air (OTA-) Update-Bibliothek • Migrieren von der Version 1 zu Version 3 für OTA-Anwendungen hinzugefügt • Migration von Version 1 auf Version 3 für den OTATAS-PALS-Anschluss hinzugefügt 	202107.00
Dezember 2020	<p>202012.00 (Portierungsleitfaden)</p> <p>202012.00 (Qualifikationsleitfaden)</p>	<ul style="list-style-type: none"> • Veröffentlichung 2020/12.00 • Konfiguration der CoreHTTP-Bibliothek hinzugefügt • Portierung der Cellular Interface-Bibliothek hinzugefügt 	202012,00

Date (Datum)	Version der Dokumentation	Verlauf ändern	FreeRTOS-Version
November 2020	202011.00 (Portierungsleitfaden) 202011.00 (Qualifikationsleitfaden)	<ul style="list-style-type: none"> • Veröffentlichung 202011.00 • Konfiguration der CoreMQTT-Bibliothek hinzugefügt 	202011.00
Juli 2020	202007.00 (Portierungsleitfaden) 202007.00 (Qualifikationsleitfaden)	<ul style="list-style-type: none"> • Veröffentlichung 2020-07-00 	202007.00
18. Februar 2020	202002.00 (Portierungshandbuch) 202002.00 (Qualifizierungshandbuch)	<ul style="list-style-type: none"> • Version 202002.00 • Amazon FreeRTOS ist jetzt FreeRTOS 	202002.00
17. Dezember 2019	201912.00 (Portierungshandbuch) 201912.00 (Qualifizierungshandbuch)	<ul style="list-style-type: none"> • Release 201912.00 • Portierung der gängigen I/O-Bibliotheken hinzugefügt. 	201912.00
29. Oktober 2019	201910.00 (Portierungshandbuch) 201910.00 (Qualifizierungshandbuch)	<ul style="list-style-type: none"> • Veröffentlichung 201910.00 • Aktualisierte Portierungsinformationen für den Zufallszahlengenerator. 	201910.00

Date (Datum)	Version der Dokumentation	Verlauf ändern	FreeRTOS-Version
26. August 2019	201908.00 (Portierungshandbuch) 201908.00 (Qualifizierungshandbuch)	<ul style="list-style-type: none"> • Version 201908.00 • Konfiguration der HTTPS-Clientbibliothek für Tests hinzugefügt <p>Portierung der CorePKCS11-Bibliothek aktualisiert</p>	201908.00
17. Juni 2019	201906.00 (Portierungsleitfaden) 201906.00 (Qualifikationsleitfaden)	<ul style="list-style-type: none"> • Veröffentlichung • Aktualisierte Verzeichnisstruktur 	201906.00 Major
21. Mai 2019	1.4.8 (Portierungshandbuch) 1.4.8 (Qualifikationshandbuch)	<ul style="list-style-type: none"> • Die Portierungsdokumentation wurde in den FreeRTOS Porting Guide verschoben • Die Qualifikationsdokumentation wurde in den FreeRTOS Qualification Guide verschoben 	1.4.8

Date (Datum)	Version der Dokumentation	Verlauf ändern	FreeRTOS-Version
25. Februar 2019	1.1.6	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• Aktualisierte Checkliste für den Qualifikations-Anhang mit CMake Anforderung (Seite 70)	1.4.5 1.4.6
12. Dezember 2018	1.1.4	<ul style="list-style-type: none">• TCP/IP-Portierungsanweisungen und lwIP Portierungsanhang hinzugefügt (Seite 31)	1.4.5

Date (Datum)	Version der Dokumentation	Verlauf ändern	FreeRTOS-Version
26. November 2018	1.1.3	<ul style="list-style-type: none">• Anhang zur Bluetooth Low Energy-Portierung hinzugefügt (Seite 52)• Im gesamten Dokument wurden Informationen zum AWS IoT Gerätetester für FreeRTOS hinzugefügt• CMake-Link zu den Informationen für die Auflistung im Anhang der FreeRTOS-Konsole hinzugefügt (Seite 85)	1.4.4

Date (Datum)	Version der Dokumentation	Verlauf ändern	FreeRTOS-Version
7. November 2018	1.1.2	<ul style="list-style-type: none">• 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-Version
8. Oktober 2018	1.1.1	<ul style="list-style-type: none"> • 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 Schlüsselvariablenamen 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-Version
		Setup (Seite 40) und TLS-Server-Setup-Anhang (Seite 57) geändert	
27. August 2018	1.1.0	<ul style="list-style-type: none">• 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-Version
9. August 2018	1.0.1	<ul style="list-style-type: none"> • 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) • AWS CLI Config-Anweisungen um Beispiele in Anweisungen zum Erstellen eines BYOC-Anhangs hinzugefügt (S. 57) 	1.3.1 1.3.2
31. Juli 2018	1.0.0	Erste Version des Leitfadens für das FreeRTOS-Qualifizierungsprogramm	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.