aws

Benutzer-Leitfaden

FreeRTOS

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

FreeRTOS: Benutzer-Leitfaden

Copyright © 2026 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 Handelsmarken, die nicht Eigentum von Amazon sind, gehören den jeweiligen Besitzern, die möglicherweise zu Amazon gehören oder nicht, mit Amazon verbunden sind oder von Amazon gesponsert werden.

Table of Contents

Was ist FreeRTOS? 1
Laden Sie den FreeRTOS-Quellcode herunter 1
FreeRTOS-qualifizierte Hardwareplattformen 1
Weitere Ressourcen 2
FreeRTOS versionen 3
FreeRTOS Langzeitsupport 3
Erweiterter FreeRTOS-Wartungsplan 4
FreeRTOS-Architektur 4
Entwicklungs-Workflow 5
Grundlagen zum FreeRTOS-Kernel 6
Der FreeRTOS-Kernel-Scheduler 6
Zuweisung von Kernelspeicher 7
Anwendungsspeicher verwalten 8
Inter-Task-Koordination 8
Warteschlangen 9
Semaphoren und Mutexe 9
Direct-to-task Benachrichtigungen 10
Stream-Puffer 10
Nachrichtenpuffer 12
Unterstützung für symmetrisches Multiprocessing (SMP) 14
Software-Timer 14
Energiesparunterstützung 15
`FreeRTOSConfig.h` 15
AWS IoT Geräte-SDK für Embedded C 16
Verstehen Sie das FreeRTOS Common IO APIs 17
Bibliotheken 17
Common IO — einfach 17
Gemeinsames I/O — BLE 19
Gemeinsames I/O für Amazon Common Software 19
Was ist ACS? 20
Qualifizierungsprogramm 20
Erste Schritte mit FreeRTOS 21
Erste Schritte mit Quick Connect 21
Erkunden Sie die FreeRTOS-Bibliotheken 21

Entwickeln Sie ein sicheres und AWS IoT robustes Produkt ... 22
Entwickeln Sie Ihre AWS IoT Anwendung ... 22
AWS IoT Device Tester für FreeRTOS ... 23
FreeRTOS-Qualifizierungssuite ... 23
Verstehen Sie benutzerdefinierte Testsuiten ... 24
Unterstützte Versionen von IDT für FreeRTOS ... 25
Aktuelle Version von IDT für FreeRTOS ... 25
Frühere IDT-Versionen ... 28
Nicht unterstützte IDT-Versionen ... 34
Laden Sie IDT für FreeRTOS herunter ... 74
Laden Sie IDT manuell herunter ... 75
Laden Sie IDT programmgesteuert herunter ... 75
IDT mit FreeRTOS Qualification Suite 2.0 (FRQ 2.0) ... 81
Richten Sie die Voraussetzungen für die LTS-Qualifikation ein ... 82
Erster Test Ihres Mikrocontroller-Boards ... 90
Verwenden Sie die IDT-Benutzeroberfläche, um die FreeRTOS Qualification Suite auszuführen ... 107
Führen Sie die FreeRTOS Qualification 2.0 Suite aus ... 123
Sehen Sie sich das IDT kostenlos an RTOSresults ... 126
Interpretieren Sie die IDT for FreeRTOS-Ergebnisse ... 127
Sehen Sie sich das IDT kostenlos an RTOSlogs ... 130
IDT mit FreeRTOS Qualification Suite 1.0 (FRQ 1.0) ... 130
Richten Sie die 1.0-Qualifikationsvoraussetzungen ein ... 132
Erster Test Ihres Mikrocontroller-Boards ... 136
Verwenden Sie die IDT-Benutzeroberfläche, um die FreeRTOS Qualification Suite auszuführen ... 157
Führen Sie Bluetooth Low Energy-Tests durch ... 168
Führen Sie die FreeRTOS Qualification Suite aus ... 174
Sehen Sie sich die Ergebnisse von IDT for FreeRTOS an ... 180
Interpretieren Sie die IDT for FreeRTOS-Ergebnisse ... 181
Die IDT for FreeRTOS-Protokolle anzeigen ... 184
Entwickeln und betreiben Sie Ihre eigenen IDT-Testsuiten ... 185
Laden Sie die neueste Version von IDT für FreeRTOS herunter ... 185
Arbeitsablauf der Testsuite ... 186
Tutorial: Erstellen Sie die IDT-Testsuite als Beispiel und führen Sie sie aus ... 187
Tutorial: Entwickeln Sie eine einfache IDT-Testsuite ... 193

Test-Suite-Versionen 282
Beheben von Fehlern in der 283
Beheben Sie Fehler bei der Gerätekonfiguration 284
Beheben Sie Timeout-Fehler 299
Mobilfunkfunktion und Gebühren AWS 299
Richtlinie zur Generierung von Qualifikationsberichten 300
AWS Verwaltete Richtlinie für AWS IoT Device Tester 300
Verwaltete Richtlinie 301
Richtlinienaktualisierungen 308
Support-Richtlinie 310
Sicherheit in AWS 312
Identitäts- und Zugriffsverwaltung 312
Zielgruppe 313
Authentifizierung mit Identitäten 313
Verwalten des Zugriffs mit Richtlinien 315
So funktioniert FreeRTOS mit IAM 317
Beispiele für identitätsbasierte Richtlinien 322
Fehlerbehebung 326
Compliance-Validierung 328
Ausfallsicherheit 328
Sicherheit der Infrastruktur 328
Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys 330
Anhang 330
Archiv 337
Archiv des FreeRTOS-Benutzerhandbuches 337
Frühere Inhalte des FreeRTOS-Benutzerhandbuchs 337
Erste Schritte mit FreeRTOS 337
Over-the-Air Aktualisierungen 545
FreeRTOS-Bibliotheken 633
FreeRTOS RTOS-Demos 703
..... dcccxxxii

Was ist FreeRTOS?

FreeRTOS wurde in Zusammenarbeit mit den weltweit führenden Chipherstellern über einen Zeitraum von 15 Jahren entwickelt und wird nun alle 170 Sekunden heruntergeladen. Es ist ein marktführendes Echtzeitbetriebssystem (RTOS) für Mikrocontroller und kleine Mikroprozessoren. FreeRTOS wird unter der MIT-Open-Source-Lizenz kostenlos vertrieben und umfasst einen Kernel und eine wachsende Anzahl von Bibliotheken, die für den Einsatz in allen Branchen geeignet sind. FreeRTOS wurde mit einem Schwerpunkt auf Zuverlässigkeit und Benutzerfreundlichkeit entwickelt.

FreeRTOS enthält Bibliotheken für Konnektivitäts-, Sicherheits- und over-the-air (OTA-) Updates. FreeRTOS enthält auch Demo-Anwendungen, die FreeRTOS-Funktionen auf qualifizierten Boards zeigen.

FreeRTOS ist ein Open-Source-Projekt. Sie können den Quellcode herunterladen, Änderungen oder Verbesserungen beitragen oder Probleme auf der GitHub Website unter https://github.com/FreeRTOS/FreeRTOS melden.

Wir veröffentlichen FreeRTOS-Code unter der MIT-Open-Source-Lizenz, sodass Sie ihn in kommerziellen und persönlichen Projekten verwenden können.

Wir freuen uns auch über Beiträge zur FreeRTOS-Dokumentation (FreeRTOS User Guide, FreeRTOS Porting Guide und FreeRTOS Qualification Guide). Den Markdown-Quellcode der Dokumentation finden Sie unter. https://github.com/awsdocs/aws-freertos-docs Es ist unter der Creative Commons-Lizenz (CC BY-ND) veröffentlicht.

Laden Sie den FreeRTOS-Quellcode herunter

Laden Sie die neuesten FreeRTOS- und Long Term Support (LTS) -Pakete von der Downloads-Seite auf freertos.org herunter.

FreeRTOS-qualifizierte Hardwareplattformen

Die folgenden Hardwareplattformen sind für FreeRTOS qualifiziert:

- ATECC608Ein Zero Touch Provisioning Kit für AWS IoT
- Cypress CYW943907 AEVAL1 F-Entwicklungskit

- Cypress CYW954907 AEVAL1 F-Entwicklungskit
- Cypress CY8 CKIT-064S0S2-4343W Kit
- ESP32Espressif - C DevKit
- Espressif ESP-WROVER-KIT
- Espressif ESP-WROOM-32SE
- Espressif ESP32 -S2-Saola-1
- Infineon XMC4800 IoT-Konnektivitätskit
- Marvell Einsteigerpaket MW320 AWS IoT
- Marvell Starterpaket MW322 AWS IoT
- MediaTek MT7697Hx-Entwicklungskit
- Microchip Curiosity MZEF-Paket PIC32
- Nordisch und -DK RF52840
- NuMaker-IoT-M 488
- NXP LPC54018 IoT-Modul
- OPTIGA Trust X Sicherheitslösung
- Renesas RX65 N RSK IoT-Modul
- STMicroelectronicsSTM32L4 Discovery Kit IoT-Knoten
- Texas Instruments CC3220 SF-LAUNCHXL
- Microsoft Windows 7 oder höher, mit mindestens einer Dual Core- und einer kabelgebundenen Ethernet-Verbindung
- Xilinx MicroZed Avnet Industrielles IoT-Kit

Qualifizierte Geräte sind auch im Gerätekatalog der AWS -Partner aufgeführt.

Informationen zur Qualifizierung eines neuen Geräts finden Sie im FreeRTOS Qualification Guide.

Weitere Ressourcen

Diese Ressourcen könnten nützlich sein.

- Zusätzliche FreeRTOS-Dokumentation finden Sie unter freertos.org.

- Bei Fragen zu FreeRTOS für das FreeRTOS-Entwicklungsteam können Sie auf der FreeRTOS-Seite eine Ausgabe öffnen. GitHub
- Technische Fragen zu FreeRTOS finden Sie in den FreeRTOS Community-Foren.
- Weitere Informationen zum Anschließen von Geräten an AWS IoT finden Sie unter Device Provisioning im Developer Guide.AWS IoT Core
- Technischen Support für AWS finden Sie unter AWS Support.
- Wenn Sie Fragen zur AWS Abrechnung, zu Kontodiensten, Ereignissen, Missbrauch oder anderen Problemen mit haben AWS, besuchen Sie bitte die Kontaktseite.

FreeRTOS versionen

Einzelne Bibliotheken verwenden Versionsnummern im x.y.z-Stil, ähnlich wie bei der semantischen Versionierung. X ist die Hauptversionsnummer, y die Nebenversionsnummer, und ab 2022 ist z eine Patch-Nummer. Vor 2022 war z eine Point-Release-Nummer, weshalb die ersten LTS-Bibliotheken eine Patch-Nummer in der Form „x.y.z LTS Patch 2“ haben mussten.

Bibliothekspakete verwenden Versionsnummern mit Datumsstempeln im Format yyyymm.x. yyyy ist das Jahr, mm der Monat und x ist eine optionale Sequenznummer, die die Veröffentlichungsreihenfolge innerhalb des Monats angibt. Im Fall des LTS-Pakets ist x eine sequentielle Patch-Nummer für diese LTS-Version. Bei den einzelnen Bibliotheken, die in einem Paket enthalten sind, handelt es sich um die neueste Version der Bibliothek zu diesem Zeitpunkt. Für das LTS-Paket ist es die neueste Patch-Version der LTS-Bibliotheken, die ursprünglich an diesem Tag als LTS-Version veröffentlicht wurden.

FreeRTOS Langzeitsupport

FreeRTOS Long Term Support (LTS) -Versionen erhalten nach ihrer Veröffentlichung mindestens zwei Jahre lang Sicherheits- und kritische Bugfixes (falls erforderlich). Mit dieser fortlaufenden Wartung können Sie während eines gesamten Entwicklungs- und Bereitstellungszyklus Fehlerkorrekturen einbauen, ohne dass die Aktualisierung auf neue Hauptversionen der FreeRTOS-Bibliotheken kostspielig unterbrochen werden muss.

Mit FreeRTOS LTS erhalten Sie den kompletten Satz an Bibliotheken, die Sie für die Entwicklung sicherer vernetzter IoT- und Embedded-Produkte benötigen. LTS trägt dazu bei, die Wartungs- und Testkosten zu senken, die mit der Aktualisierung von Bibliotheken auf Ihren Geräten verbunden sind, die sich bereits in Produktion befinden.

FreeRTOS LTS umfasst den FreeRTOS-Kernel und die IoT-Bibliotheken: FreeRTOS+TCP, CoreMQTT, CoreHTTP, CorePKCS11, CoreJSON, OTA, Jobs und Device Shadow. AWS IoT AWS IoT AWS IoT Device Defender AWS IoT Weitere Informationen finden Sie in den FreeRTOS LTS-Bibliotheken.

Erweiterter FreeRTOS-Wartungsplan

AWS bietet auch den FreeRTOS Extended Maintenance Plan (EMP) an, der Sicherheitspatches und kritische Bugfixes für die von Ihnen gewählte FreeRTOS Long Term Support (LTS) -Version für bis zu zehn weitere Jahre bereitstellt. Mit FreeRTOS EMP können sich Ihre langlebigen FreeRTOS-Geräte auf eine Version verlassen, die über Jahre hinweg über stabile Funktionen verfügt und Sicherheitsupdates erhält. Sie erhalten rechtzeitig Benachrichtigungen über bevorstehende Patches für FreeRTOS-Bibliotheken, sodass Sie die Bereitstellung von Sicherheitspatches auf Ihren IoT-Geräten (Internet of Things) planen können.

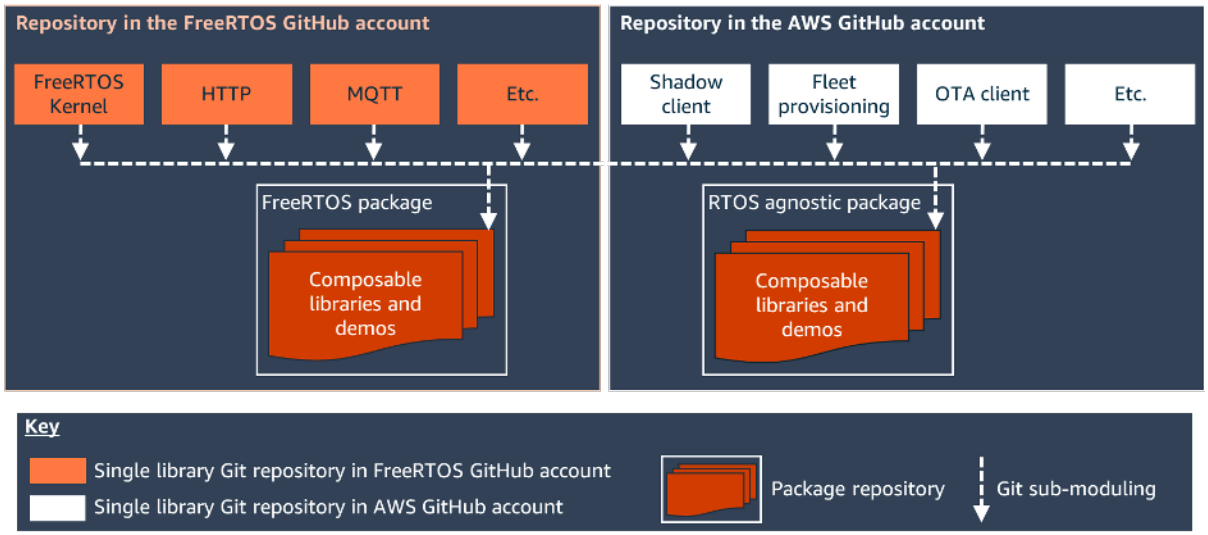
Weitere Informationen zu FreeRTOS EMP finden Sie auf der Seite Funktionen.

FreeRTOS-Architektur

FreeRTOS enthält zwei Arten von Repositorien, Repositorys für einzelne Bibliotheken und Paket-Repositorys. Jedes einzelne Bibliotheks-Repository enthält den Quellcode für eine Bibliothek ohne Build-Projekte oder Beispiele. Paket-Repositorys enthalten mehrere Bibliotheken und können vorkonfigurierte Projekte enthalten, die die Verwendung der Bibliothek demonstrieren.

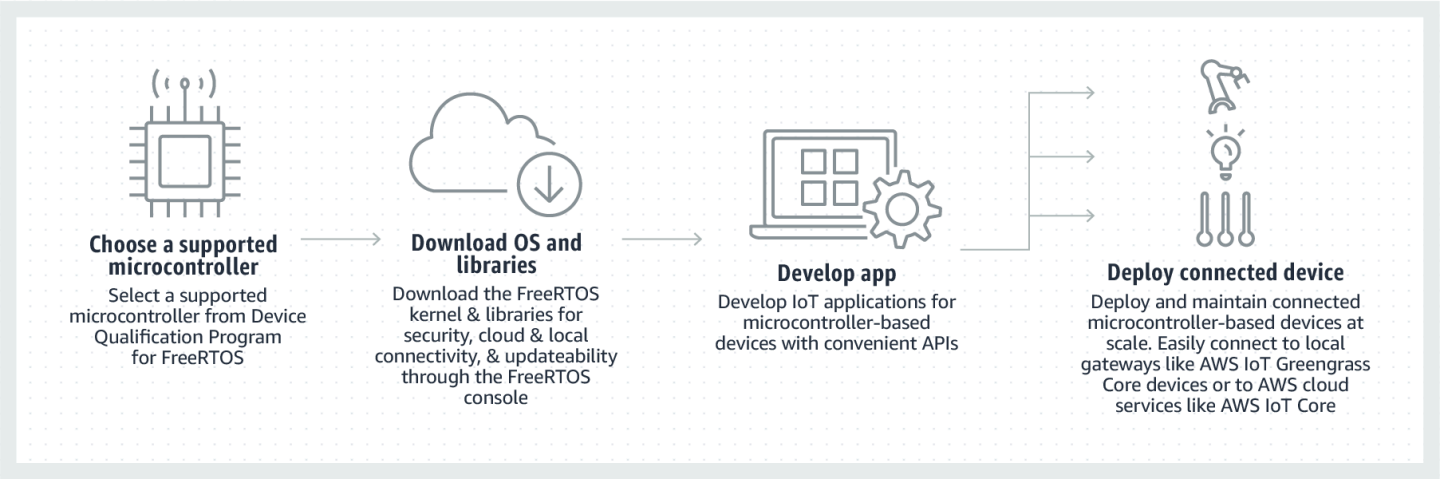
Paket-Repositorien enthalten zwar mehrere Bibliotheken, aber keine Kopien dieser Bibliotheken. Stattdessen verweisen Paket-Repositorys als Git-Submodule auf die Bibliotheken, die sie enthalten. Die Verwendung von Submodulen stellt sicher, dass es für jede einzelne Bibliothek eine einzige Informationsquelle gibt.

Die Git-Repositorys der einzelnen Bibliotheken sind auf zwei GitHub Organisationen aufgeteilt. Repositorys, die FreeRTOS-spezifische Bibliotheken (wie FreeRTOS+TCP) oder generische Bibliotheken (wie CoreMQTT, das Cloud-unabhängig ist, weil es mit jedem MQTT-Broker funktioniert) enthalten, befinden sich in der FreeRTOS-Organisation. GitHub Repositorys, die bestimmte Bibliotheken enthalten (wie den Update-Client), befinden sich in der Organisation. AWS IoT AWS IoT over-the-air AWS GitHub Das folgende Diagramm erklärt die Struktur.



Entwicklungs-Workflow

Sie beginnen mit der Entwicklung, indem Sie FreeRTOS herunterladen. Entpacken Sie das Paket und importieren Sie es in Ihre integrierte Entwicklungsumgebung. Anschließend können Sie eine Anwendung auf Ihrer ausgewählten Hardwareplattform entwickeln. Der für Ihr Gerät geeignete Entwicklungsvorgang hilft Ihnen bei der Herstellung und Bereitstellung dieser Geräte. Eingesetzte Geräte können mit dem AWS IoT Service oder AWS IoT Greengrass als Teil einer kompletten IoT-Lösung verbunden werden.



Grundlagen zum FreeRTOS-Kernel

Der FreeRTOS-Kernel ist ein Echtzeitbetriebssystem, das zahlreiche Architekturen unterstützt. Die Grundlagen sind ideal für den Aufbau eingebetteter Mikrocontroller-Anwendungen. Es bietet:

- Einen Multitasking-Scheduler.
- Mehrere Speicherzuweisungsoptionen (einschließlich der Möglichkeit, vollständig statisch zugeordnete Systeme zu erstellen).
- Primitiven für die Koordination zwischen den Tasks, einschließlich Task-Benachrichtigungen, Nachrichtenwarteschlangen, verschiedenen Arten von Semaphoren sowie Stream- und Nachrichtenpuffer.
- Support für symmetrisches Multiprocessing (SMP) auf Mehrkern-Mikrocontrollern.

Der FreeRTOS-Kernel führt keine nicht-deterministischen Operationen aus (z. B. das Durchlaufen einer verknüpften Liste, innerhalb eines kritischen Abschnitts oder Interrupts). Der FreeRTOS-Kernel enthält eine effiziente Software-Timer-Implementierung, die keine CPU-Zeit verbraucht – es sei denn, ein Timer benötigt ein Servicing. Blockierte Aufgaben erfordern keine zeitaufwändige regelmäßige Wartung. Direct-to-taskBenachrichtigungen ermöglichen eine schnelle Signalisierung von Aufgaben, praktisch ohne RAM-Overhead. Sie können in den meisten Szenarien zwischen Aufgaben und interrupt-to-task Signalisierung verwendet werden.

Der FreeRTOS-Kernel ist so konzipiert, dass er klein, einfach und leicht einzusetzen ist. Ein typisches binäres RTOS-Kernel-Image ist zwischen 4000 bis 9000 Byte groß.

Die meiste up-to-date Dokumentation zum FreeRTOS-Kernel finden Sie unter FreeRTOS.org. FreeRTOS.org bietet eine Reihe detaillierter Tutorials und Anleitungen zur Verwendung des FreeRTOS-Kernels, darunter eine FreeRTOS FreeRTOS Kernel-Schnellstartanleitung und die ausführlichere RTOS-Implementierung in der FreeRTOS-Dokumentation.

Der FreeRTOS-Kernel-Scheduler

Eine Embedded-Anwendung, die RTOS verwendet, kann als eine Reihe von unabhängigen Tasks strukturiert werden. Jeder Task wird in einem eigenen Kontext ausgeführt, ohne Abhängigkeit von anderen Tasks. In der Anwendung wird immer nur ein Task gleichzeitig ausgeführt. Der Echtzeit-RTOS-Scheduler bestimmt, wann die einzelnen Tasks ausgeführt werden sollen. Jeder Task ist

mit einem eigenen Stack ausgestattet. Wenn ein Task ausgelagert wird, damit ein anderer Task ausgeführt werden kann, wird der Ausführungskontext des Tasks im Task-Stack gespeichert, sodass er wiederhergestellt werden kann, wenn derselbe Task später wieder zur Ausführung geladen wird.

Um ein deterministisches Echtzeitverhalten zu ermöglichen, ermöglicht der FreeRTOS Task-Scheduler, dass Tasks strikte Prioritäten zugewiesen werden. RTOS stellt sicher, dass der ausführbare Task mit der höchsten Priorität Verarbeitungszeit erhält. Dies erfordert die Aufteilung der Verarbeitungszeit zwischen Tasks mit gleicher Priorität, wenn diese gleichzeitig ausgeführt werden können. FreeRTOS erstellt außerdem einen Leerlauf-Task, der nur ausgeführt wird, wenn keine anderen Tasks zur Ausführung bereit sind.

Zuweisung von Kernelspeicher

Der RTOS-Kernel benötigt RAM, wenn ein Task, eine Warteschlange oder ein anderes RTOS-Objekt erstellt wird. Das RAM kann folgendermaßen zugewiesen werden:

- Statisch zur Kompilierungszeit
- Dynamisch aus dem RTOS-Heap, über die RTOS-API-Objekterstellungsfunktionen

Bei der dynamischen Erstellung von RTOS-Objekten ist die Verwendung der Funktionen `malloc()` und `free()` aus der der Standard-C-Bibliothek aus verschiedenen Gründen nicht immer sinnvoll:

- Sie sind möglicherweise auf Embedded-Systemen nicht verfügbar
- Sie beanspruchen wertvollen Code-Speicherplatz
- Sie sind in der Regel nicht Thread-sicher
- Sie sind nicht deterministisch

Aus diesen Gründen befindet sich Speicherzuweisungs-API im portablen Layer von FreeRTOS. Der portable Layer befindet sich außerhalb der Quelldateien, die die Kern-RTOS-Funktionalität implementieren. So können Sie eine anwendungsspezifische Implementierung bereitstellen, die für das von Ihnen entwickelte Echtzeit-System geeignet ist. Wenn der RTOS-Kernel RAM benötigt, ruft er `pvPortMalloc()` statt `malloc()`() auf. Wenn RAM freigegeben wird, ruft der RTOS-Kernel `vPortFree()` statt `free()` auf.

Anwendungsspeicher verwalten

Wenn Anwendungen Speicher benötigen, können sie ihn aus dem FreeRTOS-Heap zuweisen. FreeRTOS bietet mehrere Heap-Management-Schemata an, die sich in ihrer Komplexität und ihren Funktionen unterscheiden. Sie können außerdem Ihre eigene Heap-Implementierung bereitstellen.

Der FreeRTOS-Kernel enthält fünf Heap-Implementierungen:

`heap_1`

Dies ist die einfachste Implementierung. Sie erlaubt nicht, dass Speicher freigegeben wird.

`heap_2`

Lässt die Freigabe des Speichers zu, lässt jedoch keine Zusammenführung benachbarter freier Blöcke zu.

`heap_3`

Wrapper für die Standardfunktionen `malloc()` und `free()` (aus Gründen der Thread-Sicherheit).

`heap_4`

Führt benachbarte freie Blöcke zusammen, um Fragmentierung zu vermeiden. Enthält eine Option zur absoluten Platzierung von Adressen.

`heap_5`

Ist ähnlich wie heap_4. Kann den Heap über mehrere, nicht benachbarte Speicherbereiche verteilen.

Inter-Task-Koordination

Dieser Abschnitt enthält Informationen über FreeRTOS-Primitiven.

Themen

- Warteschlangen
- Semaphoren und Mutexe
- Direct-to-task Benachrichtigungen
- Stream-Puffer

- Nachrichtenpuffer

Warteschlangen

Warteschlangen sind die primäre Form der Inter-Task-Kommunikation. Sie können verwendet werden, um Nachrichten zwischen Tasks und zwischen Interrupts und Tasks zu senden. In den meisten Fällen werden sie als Thread-sichere First In First Out (FIFO)-Puffer verwendet, wobei neue Daten an das Ende der Warteschlange gesendet werden. (Daten können auch an den Anfang der Warteschlange gesendet werden.) Nachrichten werden als Kopie über Warteschlangen gesendet. Das heißt, die tatsächlichen Daten (die ein Zeiger auf größere Puffer sein können) werden in die Warteschlange kopiert – und nicht nur ein Verweis auf die Daten.

In APIs der Warteschlange kann eine Blockzeit angegeben werden. Wenn ein Task versucht, aus einer leeren Warteschlange zu lesen, wird der Task in den Status "Blocked" versetzt, bis Daten in der Warteschlange verfügbar sind oder die Blockierungsdauer abgelaufen ist. Tasks im Status "Blocked" verbrauchen keine CPU-Zeit, sodass andere Tasks ausgeführt werden können. Ähnlich verhält es sich, wenn ein Task versucht, in eine volle Warteschlange zu schreiben. Dann wird der Task in den Status "Blocked" versetzt, bis Platz in der Warteschlange verfügbar wird oder die Blockierungsdauer verstrichen ist. Wenn sich mehr als ein blockierter Task in einer Warteschlange befindet, wird zuerst der Task mit der höchsten Priorität freigegeben.

Andere FreeRTOS-Primitive, wie direct-to-task Benachrichtigungen und Stream- und Nachrichtenpuffer, bieten in vielen gängigen Entwurfsszenarien einfache Alternativen zu Warteschlangen.

Semaphoren und Mutexe

Der FreeRTOS-Kernel bietet binäre Semaphoren, zählende Semaphoren und Mutexe für den gegenseitigen Ausschluss und für Synchronisationszwecke.

Binärsemaphore können nur zwei Werte haben. Sie sind eine gute Wahl bei der Implementierung der Synchronisation (entweder zwischen Tasks oder zwischen Tasks und einem Interrupt). Zählende Semaphoren haben mehr als zwei Werte. Sie ermöglichen es vielen Tasks, Ressourcen gemeinsam zu nutzen oder komplexere Synchronisationsoperationen durchzuführen.

Mutexe sind binäre Semaphoren, die einen Vererbungsmechanismus für Prioritäten beinhalten. Das bedeutet Folgendes: Wenn ein Task mit hoher Priorität blockiert ist, währen er versucht einen derzeit von einem Task mit niedrigerer Priorität gehaltenen Mutex zu erhalten, wird die Priorität des Tasks,

der das Token hält, vorübergehend auf die des blockierten Tasks erhöht. Dieser Mechanismus wurde entwickelt, um sicherzustellen, dass der Task mit höherer Priorität so kurz wie möglich im Status "Blocked" gehalten wird. So wird die aufgetretene Prioritätsumkehrung minimiert.

Direct-to-task Benachrichtigungen

Aufgabenbenachrichtigungen ermöglichen es Aufgaben, mit anderen Aufgaben zu interagieren und sich mit Interrupt-Serviceroutinen (ISRs) zu synchronisieren, ohne dass ein separates Kommunikationsobjekt wie eine Semaphore erforderlich ist. Jeder RTOS-Task hat einen 32-Bit-Benachrichtigungswert. Dieser wird verwendet, um den Inhalt der Benachrichtigung zu speichern (falls vorhanden). Eine RTOS-Task-Benachrichtigung ist ein Ereignis, das direkt an einen Task gesendet wird, der die Blockierung des empfangenden Tasks aufheben und optional den Benachrichtigungswert des empfangenden Tasks aktualisieren kann.

RTOS-Task-Benachrichtigungen können als schnellere und einfachere Alternative zu binären und zählenden Semaphoren und in einigen Fällen zu Warteschlangen verwendet werden. Task-Benachrichtigungen haben sowohl Geschwindigkeits- als auch RAM-Footprint-Vorteile gegenüber anderen FreeRTOS-Funktionen zur Ausführung gleichwertiger Funktionalitäten. Task-Benachrichtigungen können jedoch nur verwendet werden, wenn es nur einen Task als Empfänger des Ereignisses gibt.

Stream-Puffer

Stream-Puffer ermöglichen die Übergabe eines Bytestroms von einer Interrupt-Serviceroutine an einen Task oder von einem Task an einen anderen. Ein Bytestrom kann eine beliebige Länge haben und muss nicht unbedingt einen Anfang oder ein Ende haben. Es können beliebig viele Bytes auf einmal geschrieben und beliebig viele Bytes auf einmal gelesen werden. Die Stream-Pufferfunktionalität wird durch die Einbindung der Quelldatei `stream_buffer.c` in Ihr Projekt aktiviert.

Stream-Puffer gehen davon aus, dass es nur einen Task oder Interrupt gibt, der in den Puffer schreibt (der Writer) und nur einen Task oder Interrupt, der aus dem Puffer liest (der Reader). Writer und Reader können unterschiedliche Tasks oder Interrupt-Serviceroutinen sein. Es darf jedoch nicht mehrere Writer oder Reader geben.

Die Stream-Buffer-Implementierung verwendet direct-to-task Benachrichtigungen. Daher kann der Aufruf einer Stream-Puffer-API, die den aufrufenden Task in den Status "Blocked" versetzt, den Benachrichtigungsstatus und -wert des aufrufenden Tasks ändern.

Senden von Daten

`xStreamBufferSend()` wird verwendet, um Daten an einen Stream-Puffer in einen Task zu senden. `xStreamBufferSendFromISR()` wird verwendet, um Daten in einer Interrupt-Serviceroutine (ISR) an einen Stream-Puffer zu senden.

`xStreamBufferSend()` ermöglicht die Angabe einer Blockierungsdauer. Wenn `xStreamBufferSend()` mit einer Blockierungsdauer ungleich Null aufgerufen wird, um in einen Stream-Puffer zu schreiben und der Puffer voll ist, wird der Task in den Status "Blocked" versetzt, bis Platz verfügbar wird oder die Blockierungsdauer abläuft.

`sbSEND_COMPLETED()` und `sbSEND_COMPLETED_FROM_ISR()` sind Makros, die (intern von der FreeRTOS-API) aufgerufen werden, wenn Daten in einen Stream-Puffer geschrieben werden. Sie nehmen das Handle des Stream-Puffers entgegen, der aktualisiert wurde. Beide Makros prüfen, ob ein blockierter Task im Stream-Puffer auf Daten wartet. Wenn dies der Fall ist, wird Status "Blocked" für den Task entfernt.

Sie können dieses Standardverhalten ändern, indem Sie Ihre eigene Implementierung von `sbSEND_COMPLETED()` in `FreeRTOSConfig.h` bereitstellen. Dies ist hilfreich, wenn ein Stream-Puffer verwendet wird, um Daten zwischen den Kernen auf einem Mehrkern-Prozessor zu übertragen. In diesem Szenario kann `sbSEND_COMPLETED()` implementiert werden, um einen Interrupt im anderen CPU-Kern zu erzeugen. Die Serviceroutine des Interrupts kann dann die `xStreamBufferSendCompletedFromISR()`-API verwenden, um einen Task, der auf die Daten wartet, zu überprüfen und gegebenenfalls freizugeben.

Empfangen von Daten

`xStreamBufferReceive()` wird verwendet, um Daten aus einem Stream-Puffer in einen Task zu lesen. `xStreamBufferReceiveFromISR()` wird verwendet, um Daten aus einem Streambuffer in eine Interrupt-Serviceroutine (ISR) zu lesen.

`xStreamBufferReceive()` ermöglicht die Angabe einer Blockierungsdauer. Wenn `xStreamBufferReceive()` mit einer Blockierungsdauer ungleich Null zum Lesen aus einem Stream-Puffer aufgerufen wird und der Puffer leer ist, wird der Task in den Status "Blocked" versetzt, bis entweder eine bestimmte Datenmenge im Stream-Puffer verfügbar ist oder die Blockierungsdauer abläuft.

Die Datenmenge, die sich im Stream-Puffer befinden muss, bevor ein Task freigegeben wird, wird als Triggerlevel des Stream-Puffers bezeichnet. Ein mit einem Triggerlevel von 10 blockierter Task wird freigegeben, wenn mindestens 10 Bytes in den Puffer geschrieben

werden oder die Blockierungsdauer des Tasks abläuft. Wenn die Blockierungsdauer eines lesenden Tasks vor Erreichen des Triggerlevels abläuft, erhält der Task alle in den Puffer geschriebenen Daten. Das Triggerlevel eines Tasks muss auf einen Wert zwischen 1 und der Größe des Stream-Puffers festgelegt werden. Das Triggerlevel eines Stream-Puffers wird beim Aufruf von `xStreamBufferCreate()` festgelegt. Es kann durch Aufruf von `xStreamBufferSetTriggerLevel()` geändert werden.

`sbRECEIVE_COMPLETED()` und `sbRECEIVE_COMPLETED_FROM_ISR()` sind Makros, die (intern von der FreeRTOS-API) aufgerufen werden, wenn Daten aus einem Stream-Puffer gelesen werden. Die Makros prüfen, ob ein Task im Stream-Puffer blockiert ist, der darauf wartet, dass Platz im Puffer verfügbar wird. Wenn dies der Fall ist, wird der Task aus dem Zustand „Blocked“ entfernt. Sie können das Standardverhalten von `sbRECEIVE_COMPLETED()` ändern, indem Sie in `FreeRTOSConfig.h` eine alternative Implementierung bereitstellen.

Nachrichtenpuffer

Nachrichtenpuffer ermöglichen die Übergabe diskreter Nachrichten variabler Länge von einer Interrupt-Serviceroutine an einen Task oder von einem Task an einen anderen. Beispielsweise können Nachrichten der Länge 10, 20 und 123 Byte in denselben Nachrichtenpuffer geschrieben und aus aus demselben Nachrichtenpuffer gelesen werden. Eine 10-Byte-Nachricht kann nur als 10-Byte-Nachricht gelesen werden, nicht als einzelne Bytes. Nachrichtenpuffer bauen auf der Stream-Pufferimplementierung auf. Sie können die Nachrichtenpufferfunktionalität aktivieren, indem Sie die `stream_buffer.c`-Quelldatei in Ihr Projekt einfügen.

Nachrichtenpuffer gehen davon aus, dass es nur einen Task oder Interrupt gibt, der in den Puffer schreibt (der Writer) und nur einen Task oder Interrupt, der aus dem Puffer liest (der Reader). Writer und Reader können unterschiedliche Tasks oder Interrupt-Serviceroutinen sein. Es darf jedoch nicht mehrere Writer oder Reader geben.

Die Implementierung des Nachrichtenpuffers verwendet direct-to-task Benachrichtigungen. Daher kann der Aufruf einer Stream-Puffer-API, die den aufrufenden Task in den Status "Blocked" versetzt, den Benachrichtigungsstatus und -wert des aufrufenden Tasks ändern.

Um Nachrichtenpuffern die Verarbeitung von Nachrichten variabler Größe zu ermöglichen, wird die Länge jeder Nachricht vor der Nachricht selbst in den Nachrichtenpuffer geschrieben. Die Länge wird in einer Variablen vom Typ `size_t` gespeichert, die bei einer 32-Byte-Architektur typischerweise 4 Byte groß ist. Daher verbraucht das Schreiben einer 10-Byte-Nachricht in einen Nachrichtenpuffer tatsächlich 14 Byte Pufferspeicher. Ebenso verbraucht das Schreiben einer 100-Byte-Nachricht in einen Nachrichtenpuffer tatsächlich 104 Byte Pufferspeicher.

Senden von Daten

`xMessageBufferSend()` wird verwendet, um Daten von einem Task an einen Nachrichtenpuffer zu senden. `xMessageBufferSendFromISR()` wird verwendet, um Daten von einer Interrupt-Serviceroutine (ISR) an einen Nachrichtenpuffer zu senden.

`xMessageBufferSend()` ermöglicht die Angabe einer Blockierungsdauer. Wenn `xMessageBufferSend()` mit einer Blockierungsdauer ungleich Null für das Schreiben in den Nachrichtenpuffer aufgerufen wird und der Puffer voll ist, wird der Task in den Status "Blocked" versetzt, bis entweder Platz im Nachrichtenpuffer verfügbar wird oder die Blockierungsdauer abläuft.

`sbSEND_COMPLETED()` und `sbSEND_COMPLETED_FROM_ISR()` sind Makros, die (intern von der FreeRTOS-API) aufgerufen werden, wenn Daten in einen Stream-Puffer geschrieben werden. Sie nehmen einen einzigen Parameter entgegen, der das Handle des aktualisierten Stream-Puffers darstellt. Beide Makros prüfen, ob ein blockierter Task im Stream-Puffer auf Daten wartet. Wenn dies der Fall ist, entfernen sie den Status "Blocked" des Tasks.

Sie können dieses Standardverhalten ändern, indem Sie Ihre eigene Implementierung von `sbSEND_COMPLETED()` in `FreeRTOSConfig.h` bereitstellen. Dies ist hilfreich, wenn ein Stream-Puffer verwendet wird, um Daten zwischen den Kernen auf einem Mehrkern-Prozessor zu übertragen. In diesem Szenario kann `sbSEND_COMPLETED()` implementiert werden, um einen Interrupt im anderen CPU-Kern zu erzeugen. Die Serviceroutine des Interrupts kann dann die `xStreamBufferSendCompletedFromISR()`-API verwenden, um einen auf die Daten wartenden Task zu überprüfen und gegebenenfalls freizugeben.

Empfangen von Daten

`xMessageBufferReceive()` wird verwendet, um Daten aus einem Nachrichtenpuffer in einem Task zu lesen. `xMessageBufferReceiveFromISR()` wird verwendet, um Daten aus einem Nachrichtenpuffer in einer Interrupt-Serviceroutine (ISR) zu lesen. `xMessageBufferReceive()` ermöglicht die Angabe einer Blockierungsdauer. Wenn `xMessageBufferReceive()` mit einer Blockierungsdauer ungleich Null zum Lesen aus einem Nachrichtenpuffer aufgerufen wird und der Puffer leer ist, wird der Task in den Status "Blocked" versetzt, bis entweder Daten verfügbar werden oder die Blockierungsdauer abläuft.

`sbRECEIVE_COMPLETED()` und `sbRECEIVE_COMPLETED_FROM_ISR()` sind Makros, die (intern von der FreeRTOS-API) aufgerufen werden, wenn Daten aus einem Stream-Puffer gelesen werden. Die Makros prüfen, ob ein Task im Stream-Puffer blockiert ist, der darauf wartet, dass Platz im Puffer verfügbar wird. Wenn dies der Fall ist, wird der Task aus dem Zustand „Blocked“ entfernt. Sie können

das Standardverhalten von `sbRECEIVE_COMPLETED()` ändern, indem Sie in `FreeRTOSConfig.h` eine alternative Implementierung bereitstellen.

Unterstützung für symmetrisches Multiprocessing (SMP)

Die SMP-Unterstützung im FreeRTOS-Kernel ermöglicht es einer Instanz des FreeRTOS-Kernels, Aufgaben über mehrere identische Prozessorkerne hinweg zu planen. Die Kernarchitekturen müssen identisch sein und sich denselben Speicher teilen.

Die FreeRTOS-API bleibt zwischen Single-Core- und SMP-Versionen im Wesentlichen gleich, mit Ausnahme dieser zusätzlichen. APIs Daher sollte eine Anwendung, die für die FreeRTOS-Single-Core-Version geschrieben wurde, mit minimalem bis gar keinem Aufwand mit der SMP-Version kompiliert werden. Es kann jedoch einige funktionale Probleme geben, da einige Annahmen, die für Single-Core-Anwendungen zutrafen, für Multicore-Anwendungen möglicherweise nicht mehr zutreffen.

Eine gängige Annahme ist, dass eine Aufgabe mit niedrigerer Priorität nicht ausgeführt werden kann, während eine Aufgabe mit höherer Priorität ausgeführt wird. Dies galt zwar für ein Single-Core-System, gilt aber nicht mehr für Multi-Core-Systeme, da mehrere Aufgaben gleichzeitig ausgeführt werden können. Wenn sich die Anwendung auf relative Aufgabenprioritäten stützt, um sich gegenseitig auszuschließen, kann es in einer Multicore-Umgebung zu unerwarteten Ergebnissen kommen.

Eine weitere verbreitete Annahme ist, dass sie nicht gleichzeitig miteinander oder mit anderen Aufgaben ausgeführt werden ISRs kann. In einer Multicore-Umgebung ist dies nicht mehr der Fall. Der Anwendungsautor muss beim Zugriff auf Daten, die zwischen Aufgaben und ISRs gemeinsam genutzt werden, für einen ordnungsgemäßen gegenseitigen Ausschluss sorgen.

Software-Timer

Ein Software-Timer ermöglicht die Ausführung einer Funktion zu einem bestimmten Zeitpunkt. Die vom Timer ausgeführte Funktion wird als Callback-Funktion des Timers bezeichnet. Die Zeit zwischen dem Start eines Timers und der Ausführung seiner Callback-Funktion wird als Periode des Timers bezeichnet. Der FreeRTOS-Kernel bietet eine effiziente Software-Timer-Implementierung. Dies liegt an folgenden Dingen:

- Sie führt keine Timer-Callback-Funktionen aus einem Interrupt-Kontext aus
- Sie verbraucht keine Verarbeitungszeit, es sei denn, ein Timer ist tatsächlich abgelaufen

- Sie fügt dem Tick-Interrupt keinen Verarbeitungsaufwand hinzu
- Sie durchläuft keine Linklistenstrukturen, während Interrupts deaktiviert sind

Energiesparunterstützung

Wie die meisten Embedded-Betriebssysteme nutzt der FreeRTOS-Kernel einen Hardware-Timer, um periodische Tick-Interrupts zu erzeugen, die zur Zeitmessung verwendet werden. Die Energieeinsparung bei regulären Hardware-Timer-Implementierungen wird durch die Notwendigkeit begrenzt, den Energiesparzustand für die Verarbeitung der Tick-Interrupts periodisch zu verlassen und dann wieder in den Energiesparzustand zurückzukehren. Wenn die Frequenz des Tick-Interrupts zu hoch ist, überwiegt der Energie- und Zeitaufwand für das Wechseln in und aus dem Energiesparzustand für jeden Tick die möglichen Energiespargewinne in allen Stromsparmodi.

Um dieser Einschränkung entgegenzuwirken, bietet FreeRTOS einen Tick-freien Timer-Modus für Low-Power-Anwendungen. Der FreeRTOS-Tickless-Idle-Modus stoppt den periodischen Tick-Interrupt während der Leerlaufzeiten (Perioden, in denen es keine Anwendungs-Tasks gibt, die ausgeführt werden können) und nimmt eine korrigierende Anpassung des RTOS-Tick-Count-Wertes vor, wenn der Tick-Interrupt neu gestartet wird. Das Stoppen des Tick-Interrupts ermöglicht dem Mikrocontroller, im Stromsparzustand zu bleiben, bis entweder ein Interrupt eintritt oder der RTOS-Kernel einen Task in den Ready-Status überführt.

Konfiguriere den Kernel (Free RTOSConfig .h)

Sie können den FreeRTOS-Kernel für ein bestimmtes Board und eine bestimmte Anwendung über die `FreeRTOSConfig.h` Header-Datei konfigurieren. Jede Anwendung, die auf dem Kernel basiert, muss eine `FreeRTOSConfig.h` -Header-Datei im Präprozessorofad enthalten. `FreeRTOSConfig.h` ist anwendungsspezifisch und sollte in einem Anwendungsverzeichnis und nicht in einem der FreeRTOS-Kernel-Quellcode-Verzeichnisse gespeichert werden.

Die `FreeRTOSConfig.h` Dateien für die FreeRTOS-Demo- und Testanwendungen befinden sich unter *`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/FreeRTOSConfig.h` und. *`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_tests/config_files/FreeRTOSConfig.h`

Die Liste der verfügbaren Konfigurationsparameter, die in `FreeRTOSConfig.h` angegeben werden müssen, finden Sie unter FreeRTOS.org.

AWS IoT Geräte-SDK für Embedded C

> **Note**
>
> Dieses SDK ist für die Verwendung durch erfahrene Entwickler eingebetteter Software vorgesehen.

Das AWS IoT Device SDK for Embedded C (C-SDK) ist eine Sammlung von C-Quelldateien unter der MIT-Open-Source-Lizenz, die in eingebetteten Anwendungen verwendet werden können, um IoT-Geräte sicher zu AWS IoT Core verbinden. Es umfasst einen MQTT-Client, einen HTTP-Client, einen JSON-Parser und AWS IoT Device Shadow, AWS IoT Jobs, AWS IoT Fleet Provisioning und Bibliotheken. AWS IoT Device Defender Dieses SDK wird im Quellformat verteilt und kann zusammen mit dem Anwendungscode, weiteren Bibliotheken und einem Betriebssystem (BS) Ihrer Wahl in die Kunden-Firmware integriert werden.

Das AWS IoT Device SDK for Embedded C richtet sich im Allgemeinen an Geräte mit eingeschränkten Ressourcen, die eine optimierte Laufzeit in C-Sprache benötigen. Sie können das SDK auf jedem Betriebssystem verwenden und es auf jedem Prozessortyp hosten (z. B. MCUs und MPUs). Wenn Ihre Geräte jedoch über ausreichend Speicher und Verarbeitungsressourcen verfügen, empfehlen wir Ihnen, eines der AWS IoT Geräte höherer Ordnung zu verwenden SDKs.

Weitere Informationen finden Sie hier:

- AWS IoT Geräte-SDK für Embedded C
- AWS IoT Geräte-SDK für Embedded C auf GitHub
- AWS IoT Readme-Datei zum Geräte-SDK für Embedded C
- AWS IoT Geräte-SDK für eingebettete C-Beispiele

Verstehen Sie das FreeRTOS Common IO APIs

Common I/O APIs fungieren als Hardware-Abstraktionsschichten (HAL), die eine gemeinsame Schnittstelle zwischen Treibern und übergeordnetem Anwendungscode bereitstellen. FreeRTOS Common IO bietet eine Reihe von Standards APIs für den Zugriff auf gängige serielle Geräte auf unterstützten Referenzkarten. Implementierungen davon APIs sind nicht enthalten. Diese Common APIs kommunizieren und interagieren mit diesen Peripheriegeräten und ermöglichen es Ihrem Code, plattformübergreifend zu funktionieren. Ohne Common IO ist das Schreiben von Code für Low-Level-Geräte herstellerspezifisch.

> **Note**
>
> FreeRTOS benötigt keine Implementierungen des Common IO, APIs um zu funktionieren, aber es wird versuchen, das Common IO APIs als Schnittstelle zu den spezifischen Peripheriegeräten auf einer Mikrocontroller-basierten Platine zu verwenden, anstatt herstellerspezifisch. APIs

Im Allgemeinen sind Gerätetreiber unabhängig vom zugrunde liegenden Betriebssystem und spezifisch für eine bestimmte Hardwarekonfiguration. Die HAL abstrahiert die Details, wie ein bestimmter Treiber funktioniert, und stellt eine einheitliche API zur Steuerung solcher Geräte bereit. Sie können dasselbe verwenden, APIs um auf verschiedene Gerätetreiber für mehrere Mikrocontroller- (MCU) -basierte Referenzkarten zuzugreifen.

Bibliotheken

Derzeit bietet FreeRTOS zwei Common IO-Bibliotheken: Common IO — Basic und Common IO — BLE.

Common IO — einfach

Übersicht

Common I/O — Basic-Angebote, APIs die sich mit grundlegenden I/O-Peripheriegeräten und Funktionen befassen, die Sie möglicherweise auf MCU-basierten Boards finden. Das Common IO — Basic Repository ist verfügbar unter. GitHub

Unterstützte Peripheriegeräte

- ADC
- GPIO
- I2C
- PWM
- SPI
- UART
- Wachhund
- Flash
- RTC
- ABLEHNEN
- Setzt zurück
- I2S
- Leistungszähler
- Informationen zur Hardwareplattform

Unterstützte Features

- Synchrones Lesen/Schreiben

  Die Funktion kehrt erst zurück, wenn die angeforderte Datenmenge übertragen wurde.

- Asynchrones Lesen/Schreiben

  Die Funktion kehrt sofort zurück und die Datenübertragung erfolgt asynchron. Wenn die Aktion abgeschlossen ist, wird ein registriertes Benutzer-Callback aufgerufen.

Peripheriegeräte-spezifisch

- I2C

  Kombinieren Sie mehrere Operationen zu einer Transaktion. Wird verwendet, um Schreib- und Leseaktionen in einer Transaktion auszuführen.

- SPI

  Übertragen Sie Daten zwischen primär und sekundär, was bedeutet, dass das Schreiben und Lesen gleichzeitig erfolgt.

API-Referenz

Eine vollständige API-Referenz finden Sie in der Common IO — Basic API-Referenz.

Gemeinsames I/O — BLE

Übersicht

Common IO — BLE bietet Abstraktion vom Bluetooth Low Energy-Stack des Herstellers. Es bietet die folgenden Schnittstellen, über die das Gerät gesteuert und GAP- und GATT-Operationen ausgeführt werden können. Das Common IO - BLE-Repository ist verfügbar unter GitHub.

Bluetooth-Geräte-Manager:

Dies bietet eine Schnittstelle zur Steuerung des Bluetooth-Geräts, zur Durchführung von Geräteerkennungsvorgängen und anderen Aufgaben im Zusammenhang mit der Konnektivität.

BLE-Adaptermanager:

Dies bietet eine Schnittstelle für die GAP-API-Funktionen, die für BLE spezifisch sind.

Klassischer Bluetooth-Adaptermanager:

Dies bietet eine Schnittstelle zur Steuerung der BT Classic-Funktionen eines Geräts.

GATT-Server:

Dies bietet eine Schnittstelle zur Verwendung der Bluetooth-GATT-Serverfunktion.

GATT-Client:

Dies bietet eine Schnittstelle zur Verwendung der Bluetooth-Funktion des GATT-Clients.

A2DP-Verbindungsschnittstelle:

Dies bietet eine Schnittstelle für das A2DP-Quellprofil für das lokale Gerät.

API-Referenz

Eine vollständige API-Referenz finden Sie in der Common IO - BLE API-Referenz.

Gemeinsames I/O für Amazon Common Software

Die Common I/O APIs sind Teil der erforderlichen Implementierungen, die von Amazon Common Software for Devices benötigt werden, insbesondere zur Implementierung in einem Device Porting Kit (DPK) eines Anbieters.

Was ist ACS?

Amazon Common Software (ACS) for Devices ist eine Software, mit der Sie Amazon Device schneller SDKs auf Ihren Geräten integrieren können. ACS bietet eine einheitliche API-Integrationsschicht, vorab validierte und speichereffiziente Komponenten für allgemeine Funktionen wie Konnektivität, ein Device Porting Kit (DPK) und mehrstufige Testsuiten.

Qualifizierungsprogramm

Das Qualifizierungsprogramm für Amazon Common Software for Devices überprüft, ob ein Build des ACS DPK (Device Porting Kit), das auf einem bestimmten Mikrocontroller-basierten Entwicklungsboard ausgeführt wird, mit den veröffentlichten Best Practices des Programms kompatibel und robust genug ist, um die vom Qualifizierungsprogramm vorgeschriebenen ACS-vorgeschriebenen Tests zu bestehen.

Anbieter, die für dieses Programm qualifiziert sind, sind auf der Seite Anbieter von ACS-Chipsätzen aufgeführt.

Informationen zur Qualifizierung erhalten Sie bei ACS for Devices.

Erste Schritte mit FreeRTOS

Themen

- Erste Schritte mit Quick Connect
- Erkunden Sie die FreeRTOS-Bibliotheken
- Entwickeln Sie ein sicheres und AWS IoT robustes Produkt
- Entwickeln Sie Ihre AWS IoT Anwendung

Erste Schritte mit Quick Connect

Beginnen Sie mit AWS Quick Connect-Demos AWS IoT, um schnell auf Entdeckungsreise zu gehen. Quick Connect-Demos sind einfach einzurichten und mit einem von einem Partner bereitgestellten, FreeRTOS-qualifizierten Board zu verbinden. AWS IoT

Folgen Sie dem Tutorial AWS IoT Erste Schritte, um mehr über AWS IoT und die Konsole zu erfahren. AWS IoT Sie können den in den Quick Connect-Demos enthaltenen Demo-Quellcode ändern, indem Sie das Build-System und die Tools des ausgewählten Boards verwenden, um eine Verbindung zu Ihrem AWS Konto herzustellen. Der Datenfluss von der AWS IoT Konsole in deinem Konto ist jetzt sichtbar.

Erkunden Sie die FreeRTOS-Bibliotheken

Sobald Sie verstanden haben, wie ein IoT-Gerät und ein Gerät AWS IoT zusammenarbeiten, können Sie beginnen, die FreeRTOS-Bibliotheken und die Long-Term-Support-Bibliotheken (LTS) zu erkunden.

Einige häufig verwendete Bibliotheken für FreeRTOS-basierte AWS IoT Geräte sind:

- FreeRTOS RTOS-Kernel
- Kern-MQTT
- AWS IoT Over-the-Air (OTA)

Auf freertos.org finden Sie bibliotheksspezifische technische Dokumentationen und Demos.

Entwickeln Sie ein sicheres und AWS IoT robustes Produkt

Unter Ausgewählte AWS IoT FreeRTOS-Integrationen erfahren Sie mehr über bewährte Methoden, um IoT-Gerätesoftware sicherer und robuster zu machen. Diese FreeRTOS-IoT-Integrationen wurden für eine verbesserte Sicherheit entwickelt und verwenden eine Kombination aus FreeRTOS-Software und einem von Partnern bereitgestellten Board mit Hardware-Sicherheitsfunktionen. Verwenden Sie sie unverändert in der Produktion oder verwenden Sie sie als Modell für Ihre eigenen Designs.

Entwickeln Sie Ihre AWS IoT Anwendung

Gehen Sie wie folgt vor, um ein Anwendungsprojekt für Ihr AWS IoT Produkt zu erstellen:

1. Laden Sie die neueste Version von FreeRTOS oder Long Term Support (LTS) von freertos.org herunter oder klonen Sie sie aus dem FreeRTOS-LTS-Repository. GitHub Sie können die erforderlichen FreeRTOS-Bibliotheken auch aus der Toolchain des MCU-Anbieters in Ihr Projekt integrieren, sofern verfügbar.
2. Folgen Sie dem FreeRTOS-Portierungsleitfaden, um ein Projekt zu erstellen, die Entwicklungsumgebung einzurichten und FreeRTOS-Bibliotheken in Ihr Projekt zu integrieren. Verwenden Sie das FreeRTOS-Libraries-Integration-Tests-Repository GitHub , um die Portierung zu validieren.

AWS IoT Device Tester für FreeRTOS

Das IDT für FreeRTOS ist ein Tool zur Qualifizierung der Datendurchsatzrate mit dem FreeRTOS-Betriebssystem. Der Gerätetester (IDT) öffnet zunächst eine USB- oder UART-Verbindung zu einem Gerät. Anschließend blinkt ein Bild von FreeRTOS, das konfiguriert ist, um die Gerätefunktionalität unter verschiedenen Bedingungen zu testen. AWS IoT Device Tester Suiten sind erweiterbar und IDT wird für die Orchestrierung von Kundentests verwendet. AWS IoT

IDT for FreeRTOS läuft auf einem Host-Computer (Windows, macOS oder Linux), der mit dem getesteten Gerät verbunden ist. IDT konfiguriert und orchestriert Testfälle und aggregiert die Ergebnisse. Er stellt ebenfalls eine Befehlszeilenschnittstelle zur Verwaltung der Testausführung bereit.

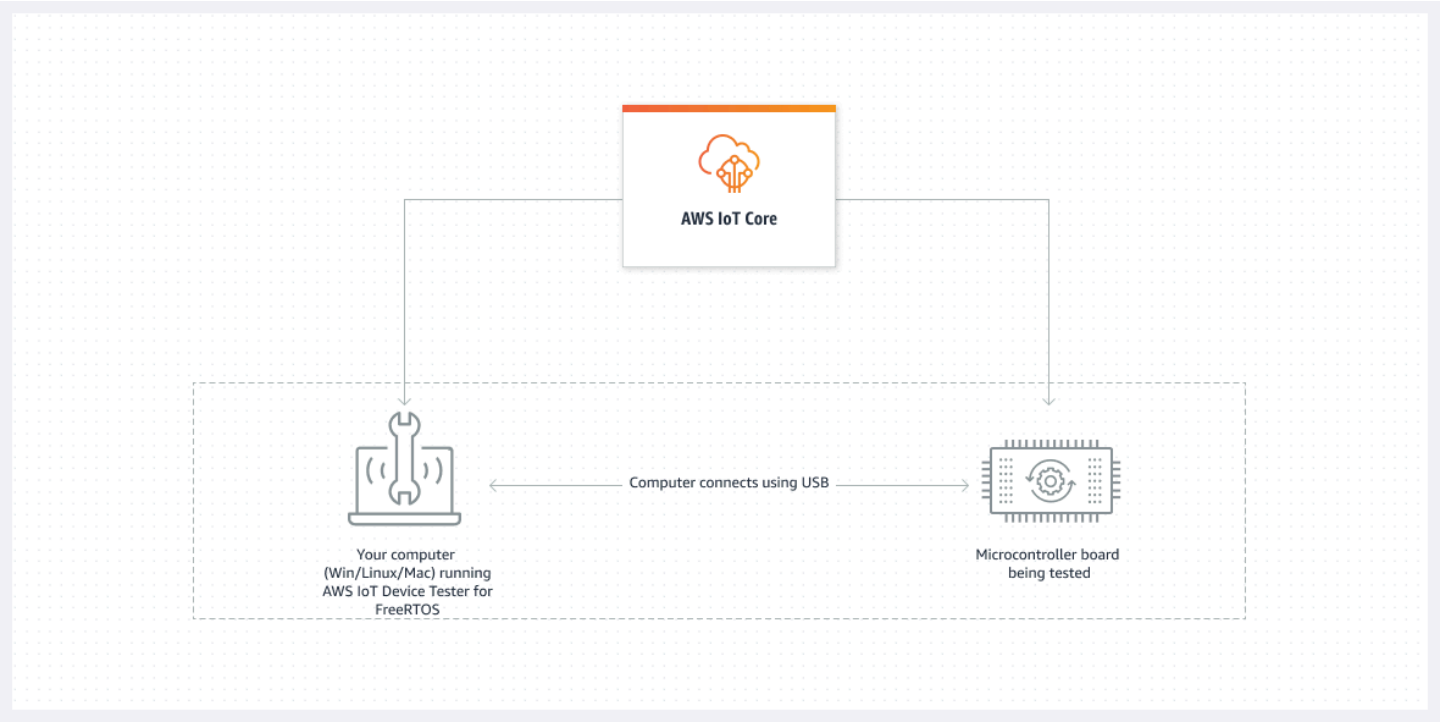
FreeRTOS-Qualifizierungssuite

IDT for FreeRTOS überprüft den Port von FreeRTOS auf Ihrem Mikrocontroller und ob er effektiv und zuverlässig und sicher kommunizieren kann. AWS IoT Insbesondere wird überprüft, ob die Portierungslayer-Schnittstellen für FreeRTOS-Bibliotheken korrekt implementiert sind. Es führt end-to-end auch Tests mit durch. AWS IoT Core Es überprüft beispielsweise, ob Ihr Board MQTT-Nachrichten senden und empfangen und korrekt verarbeiten kann.

Die FreeRTOS Qualification (FRQ) 2.0 Suite verwendet Testfälle von FreeRTOS-Libraries-Integration-Tests und Device Advisor, die im FreeRTOS Qualification Guide definiert sind.

IDT for FreeRTOS generiert Testberichte, die Sie an das AWS Partner Network (APN) senden können, um Ihre FreeRTOS-Geräte in den Partnergerätekatalog aufzunehmen. AWS Weitere Informationen finden Sie unter AWS Device Qualification Program.

Das folgende Diagramm zeigt die Einrichtung der Testinfrastruktur für die FreeRTOS-Qualifizierung.



IDT for FreeRTOS organisiert Testressourcen in Testsuiten und Testgruppen:

- Eine Testsuite besteht aus einer Reihe von Testgruppen, die verwendet werden, um zu überprüfen, ob ein Gerät mit bestimmten Versionen von FreeRTOS funktioniert.
- Eine Testgruppe besteht aus einzelnen Testfällen, die sich auf eine bestimmte Funktion beziehen, z. B. BLE- und MQTT-Messaging.

Weitere Informationen finden Sie unter Test-Suite-Versionen.

Verstehen Sie benutzerdefinierte Testsuiten

IDT for FreeRTOS kombiniert ein standardisiertes Konfigurations-Setup und ein standardisiertes Ergebnisformat mit einer Testsuite-Umgebung. In dieser Umgebung können Sie benutzerdefinierte Testsuiten für Ihre Geräte und Gerätesoftware entwickeln. Sie können benutzerdefinierte Tests für Ihre eigene interne Validierung hinzufügen oder sie Ihren Kunden zur Geräteverifizierung zur Verfügung stellen.

Wie Sie benutzerdefinierte Testsuiten konfigurieren, bestimmt die Einstellungskonfigurationen, die Sie Ihren Benutzern zur Ausführung Ihrer benutzerdefinierten Testsuiten zur Verfügung stellen müssen. Weitere Informationen finden Sie unter Entwickeln und betreiben Sie Ihre eigenen IDT-Testsuiten.

Unterstützte Versionen von AWS IoT Device Tester

Dieses Thema listet die unterstützten Versionen von AWS IoT Device Tester für FreeRTOS auf. Als bewährte Methode empfehlen wir, dass Sie die neueste Version von IDT for FreeRTOS verwenden, die Ihre Zielversion von FreeRTOS unterstützt. Jede Version von IDT for FreeRTOS hat eine oder mehrere entsprechende Versionen von FreeRTOS, die sie unterstützt. Wir empfehlen Ihnen, eine neue Version von IDT für FreeRTOS herunterzuladen, wenn eine neue Version von FreeRTOS veröffentlicht wird.

Durch das Herunterladen der Software stimmen Sie der im Download-Archiv enthaltenen AWS IoT Device Tester Lizenzvereinbarung zu.

> **Note**
>
> Wenn Sie AWS IoT Device Tester für FreeRTOS verwenden, empfehlen wir Ihnen, auf die neueste Patch-Version der neuesten FreeRTOS-LTS-Version zu aktualisieren.

> **Important**
>
> Ab Oktober 2022 generiert AWS IoT FreeRTOS Qualification (FRQ) 1.0 keine signierten Qualifikationsberichte. AWS IoT Device Tester Sie können neue AWS IoT FreeRTOS-Geräte nicht für die Aufnahme in den AWS Partnergerätekatalog über das Gerätequalifizierungsprogramm qualifizieren, wenn Sie IDT FRQ 1.0-Versionen verwenden.AWS Sie können FreeRTOS-Geräte zwar nicht mit IDT FRQ 1.0 qualifizieren, aber Sie können Ihre FreeRTOS-Geräte weiterhin mit FRQ 1.0 testen. Wir empfehlen Ihnen, IDT FRQ 2.0 zu verwenden, um FreeRTOS-Geräte zu qualifizieren und im Partnergerätekatalog aufzulisten.AWS

Aktuelle Version von AWS IoT Device Tester for freeRTOS

Verwenden Sie die folgenden Links, um die neuesten Versionen von IDT für FreeRTOS herunterzuladen.

Aktuelle Version von AWS IoT Device Tester for freeRTOS

AWS IoT Device Tester Version	Testsuite-Versionen	Unterstützte FreeRTOS-Versionen	Links herunterladen	Datum der Veröffentlichung	Versionshinweise
IDT v4.9.0	FRQ_2.5.0	• 202112,00 • 202212,00 • 202212,01 • Alle Patches von FreeRTOS 202210-LTS, die FreeRTOS LTS-Bibliotheken verwenden.	• Linux • macOS • Windows	2023.04.04	• Unterstützt Tests gegen FreeRTOS 202112, 202212, 202212.01 und alle Patches von FreeRTOS 202210-LTS, die FreeRTOS-Bibliotheken verwenden. Weitere Informationen finden Sie unter README.md. Sie müssen die Patch-Version für FreeRTOS-LTS in Ihre aufnehmen.

AWS IoT Device Tester Version	Testsuite-Versionen	Unterstützte FreeRTOS-Versionen	Links herunterladen	Datum der Veröffentlichung	Versionshinweise
					`manifest.yml` • Verbesserte Laufzeit von OTA E2E-Tests. • Beschränkt die Anzahl der aufgelisteten Geräte `device.json` auf 1. • Kleinere Fehlerbehebungen und Verbesserungen.

> **Note**
>
> Es wird nicht empfohlen, dass mehrere Benutzer IDT aus einem freigegebenen Speicherort ausführen, z. B. einem NFS-Verzeichnis oder einem freigegebenen Windows-Netzwerkordner. Diese Vorgehensweise kann zu Abstürzen oder Datenbeschädigungen führen. Es wird empfohlen, das IDT-Paket auf ein lokales Laufwerk zu extrahieren und die IDT-Binärdatei auf Ihrer lokalen Workstation auszuführen.

Frühere IDT-Versionen für FreeRTOS

Die folgenden früheren Versionen von IDT für FreeRTOS werden ebenfalls unterstützt.

Frühere Versionen von AWS IoT Device Tester for freeRTOS

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Links herunterladen	Datum der Veröffentlichung	Versionshinweise
IDT v4.8.1	FRQ_2.4.0	• 202112,00 • 202212,00 • 202212,01 • Alle Patches von FreeRTOS 202210-LTS, die FreeRTOS LTS-Bibliotheken verwenden.	• Linux • macOS • Windows	2023.01.23	• Weitere Informationen finden Sie in README.MD. Sie müssen die Patch-Version für FreeRTOS-LTS in Ihre aufnehmen. `manifest.yml` • Kleinere Fehlerbehebungen und Verbesserungen.
IDT v4.6.0	FRQ_2.3.0	• 202112,00 • 202212,00 • 202212,01	• Linux • macOS • Windows	2022.11.16	• Weitere Informationen finden Sie in

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Links herunterladen	Datum der Veröffentlichung	Versionshinweise
		• 202210-LTS, die FreeRTOS-LTS-Bibliotheken verwenden.			README.MD. Sie müssen die Patch-Version für FreeRTOS-LTS in Ihre aufnehmen. `manifest.yml` • Weitere Informationen darüber, was in der FreeRTOS 202210-LTS-Version enthalten ist, finden Sie in der Datei CHANGELOG.md unter. GitHub • Fügt die Möglichkeit hinzu, FreeRTOS über eine webbasier

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Links herunterladen	Datum der Veröffentlichung	Versionshinweise
					te Benutzeroberfläche zu konfigurieren und auszuführen AWS IoT Device Tester . Informationen zum Einstieg finden Sie unter Benutzeroberfläche für IDT für FreeRTOS Qualification Suite 2.0 (FRQ 2.0). • Fügt eine Option hinzu, um die modifizierten Kopien des Quellcodes beizubehalten, die zur Laufzeit

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Links herunterladen	Datum der Veröffentlichung	Versionshinweise
					für das Debuggen nach dem Test erstellt und verwendet wurden. Weitere Informationen finden Sie unter Konfiguration von Build-, Flash- und Testeinstellungen. • Fügt IDT Client SDK-Unterstützung für Java hinzu. Weitere Informationen zum IDT Client SDK finden Sie unter. Entwickeln und betreiben

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Links herunterladen	Datum der Veröffentlichung	Versionshinweise
					Sie Ihre eigenen IDT-Testsuiten

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Links herunterladen	Datum der Veröffentlichung	Versionshinweise
IDT v4.5.11	FRQ_2.2.0	• 202112,00 • 202212,00 • 202212,01 • 202210-LTS, die FreeRTOS-LTS-Bibliotheken verwenden.	• Linux • macOS • Windows	2022.10.14	• Weitere Informationen finden Sie in README.MD. Sie müssen die Patch-Version für FreeRTOS-LTS in Ihre aufnehmen. `manifest.yml` • Weitere Informationen darüber, was in der FreeRTOS 202210-LTS-Version enthalten ist, finden Sie in der Datei CHANGELOG.md unter. GitHub

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Links herunterladen	Datum der Veröffentlichung	Versionshinweise
					• Kleinere Fehlerbehebungen und Verbesserungen.

Weitere Informationen finden Sie unter Machen Sie sich mit den Support-Richtlinien vertraut für AWS IoT Device Tester.

Nicht unterstützte IDT-Versionen für FreeRTOS

In diesem Abschnitt werden nicht unterstützte Versionen von IDT für FreeRTOS aufgeführt. Nicht unterstützte Versionen erhalten keine Fehlerbehebungen oder Updates. Weitere Informationen finden Sie unter Machen Sie sich mit den Support-Richtlinien vertraut für AWS IoT Device Tester.

Die folgenden Versionen von IDT-FreeRTOS werden nicht mehr unterstützt.

Nicht unterstützte Versionen von AWS IoT Device Tester for FreeRTOS

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Datum der Veröffentlichung	Versionshinweise
IDT v4.5.10	FRQ_2.1.4	• 202112,00 • 202012-LTS, die FreeRTOS-LTS-Bibliotheken verwenden.	2022.09.02	• Weitere Informationen darüber, was in der FreeRTOS 202012-LTS-Version enthalten ist, finden Sie in der Datei

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Datum der Veröffentlichung	Versionshinweise
				CHANGELOG.md unter. GitHub • Es wurde ein Problem behoben, das die Testgruppe betraf. `OTA End to End` • `FullTransportInterfacePlaintext` Aus dem Status „In Qualifikationsläufen“ entfernt. Mit dem `-\-group-id` Flag kann weiterhin Klartext als Entwicklungstestgruppe ausgeführt werden. • Die Protokollierung und die Lesbarkeit der Konsolen- und Dateiausg

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Datum der Veröffentlichung	Versionshinweise
				abe wurden verbessert. • Kleinere Fehlerbeh ebungen und Verbesser ungen.

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Datum der Veröffentlichung	Versionshinweise
IDT v4.5.9	FRQ_2.1.3	• 202112,00 • 202012.04-LTS, die FreeRTOS-LTS-Bibliotheken verwenden.	2022.08.17	• Weitere Informationen darüber, was in der FreeRTOS 202012.04-LTS-Version enthalten ist, finden Sie in der Datei CHANGELOG.md unter. GitHub • Ein Problem, das die Testgruppe betraf, wurde behoben. `FreeRTOSIntegrity` • Die `FullCloudIoT` Testgruppe wurde aktualisiert, indem der Testfall „MQTT Connect Exponential Backoff Retries“ entfernt wurde.

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Datum der Veröffentlichung	Versionshinweise
				• Kleinere Fehlerbehebungen und Verbesserungen.

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Datum der Veröffentlichung	Versionshinweise
IDT v4.5.6	FRQ_2.1.2	• 202112,00 • 202012.04-LTS, die FreeRTOS-LTS-Bibliotheken verwenden.	2022.06.29	• Weitere Informationen darüber, was in der FreeRTOS 202012.04-LTS-Version enthalten ist, finden Sie in der Datei CHANGELOG.md unter. GitHub • Fügt eine neue Testgruppe hinzu, gegen die das Board getestet wird. `FullCloudIoT` AWS IoT Core Device Advisor • Es wurde ein Problem behoben, das die OTA E2E-Testfälle betraf. • Kleinere Fehlerbehebungen und

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Datum der Veröffentlichung	Versionshinweise
				Verbesserungen.

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Datum der Veröffentlichung	Versionshinweise
IDT v4.5.5	FRQ_2.1.1	• 202112,00 • 202012.04-LTS, die FreeRTOS-LTS-Bibliotheken verwenden.	2022.06.06	• Weitere Informationen darüber, was in der FreeRTOS 202012.04-LTS-Version enthalten ist, finden Sie in der Datei CHANGELOG.md unter. GitHub • Fügt eine neue Testgruppe hinzu, gegen die das Board getestet wird. `FullCloudIoT` AWS IoT Core Device Advisor • Es wurde ein Problem behoben, das die kostenlosen RTOSVersion und kostenlosen RTOSIntegrity Testfälle betraf.

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Datum der Veröffentlichung	Versionshinweise
				• Kleinere Fehlerbehebungen und Verbesserungen.

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Datum der Veröffentlichung	Versionshinweise
IDT v4.5.5	FRQ_2.1.0	• 202107,00 • 202112,00 • 202012.04-LTS, die FreeRTOS-LTS-Bibliotheken verwenden.	2022.05.31	• Weitere Informationen darüber, was in der FreeRTOS 202012.04-LTS-Version enthalten ist, finden Sie in der Datei CHANGELOG.md unter. GitHub • Fügt eine neue Testgruppe hinzu, gegen die das Board getestet wird. `FullCloudIoT` AWS IoT Core Device Advisor • Kleinere Fehlerbehebungen und Verbesserungen.

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Datum der Veröffentlichung	Versionshinweise
IDT v4.5.4	FRQ_2.0.0	• 202112,00 • 202012.04-LTS, die FreeRTOS-LTS-Bibliotheken verwenden.	2022.05.09	• Weitere Informationen darüber, was in der FreeRTOS 202012.04-LTS-Version enthalten ist, finden Sie in der Datei CHANGELOG.md unter. GitHub • Damit entfällt die Anforderung, Boards nur mit Versionen von Amazon FreeRTOS aus dem `aws/amazon-freertos` GitHub Repository zu qualifizieren. • Kleinere Fehlerbehebungen und Verbesserungen.

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Datum der Veröffentlichung	Versionshinweise
IDT v4.5.2	FRQ_1.6.2	202107,00	2022.01.25	• Weitere Informationen darüber, was in der Version FreeRTOS 202107.00 enthalten ist, finden Sie in der Datei CHANGELOG.md unter. GitHub • Implementiert den neuen IDT-Testorchestrator für die Konfiguration benutzerdefinierter Testsuiten. Weitere Informationen finden Sie unter Konfiguration des IDT-Testorchestrators. • Kleinere Fehlerbehebungen und Verbesserungen.

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Datum der Veröffentlichung	Versionshinweise
IDT v4.0.3	FRQ_1.5.1	202012,00	2021,07,30	• Support für die Qualifizierung von Geräten mit gesperrten Anmeldeinformationen auf einem Hardware-Sicherheitsmodul. • Kleinere Fehlerbehebungen und Verbesserungen.

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Datum der Veröffentlichung	Versionshinweise
IDT v4.3.0	FRQ_1.6.1	202107,00	2021.07.26	• Weitere Informationen darüber, was in der Version FreeRTOS 202107.00 enthalten ist, finden Sie in der Datei CHANGELOG.md unter. GitHub • Fügt die Möglichkeit hinzu, FreeRTOS über eine webbasierte Benutzeroberfläche zu konfigurieren und auszuführen AWS IoT Device Tester . Informationen zum Einstieg finden Sie unter Verwenden Sie die IDT-Benutzeroberfl

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Datum der Veröffentlichung	Versionshinweise
				äche, um die FreeRTOS Qualification Suite auszuführen.

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Datum der Veröffentlichung	Versionshinweise

IDT v4.1.0	FRQ_1.6.0	202107,00	2021.07.21	• Weitere Informationen darüber, was in der Version FreeRTOS 202107.00 enthalten ist, finden Sie in der Datei CHANGELOG.md unter. GitHub
				• Entfernt die folgenden Testfälle aus der OTA-Qualifizierung:
				  • OTA-Agent
				  • OTA Fehlender Dateiname
				  • Max. konfigurierte Anzahl von Blöcken für OTA
				• Entfernt die `Both` OTA-Dataplane-Testgruppe aus der

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Datum der Veröffentlichung	Versionshinweise
				OTA-Qualifikation. In der [device.json Datei] akzeptiert die `OTADataPlaneProtocol` Konfiguration jetzt nur `HTTP` oder `MQTT` als unterstützte Werte. • Implementiert die folgenden Änderungen an der `freertosFileConfiguration` Konfiguration in der [userdata.json Datei] für Änderungen am FreeRTOS-Quellcode: • Ändert den Dateinamen, der für `otaAgentT`

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Datum der Veröffentlichung	Versionshinweise
				estsConfig und otaAgentDemosConfig von bis aws_ota_agent_config.h angegeben ist. ota_config.h • Fügt eine neue otaDemosConfig optionale Konfiguration hinzu, um den Dateipfad zur neuen ota_demo_config.h Datei anzugeben. • Fügt ein neues Feld hinzutestStartDelayms , userdata.json

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Datum der Veröffentlichung	Versionshinweise
				um eine Verzögerung zwischen dem Zeitpunkt, zu dem ein Gerät geflasht wird, um eine FreeRTOS-Testgruppe auszuführen, und dem Zeitpunkt, zu dem es mit der Ausführung von Tests beginnt, anzugeben. Der Wert sollte in Millisekunden angegeben werden. Diese Verzögerung kann genutzt werden, um IDT die Möglichkeit zu geben, eine Verbindung herzustellen, sodass keine

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Datum der Veröffentlichung	Versionshinweise
				Testausgabe verpasst wird.

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Datum der Veröffentlichung	Versionshinweise
IDT v4.0.1	FRQ_1.4.1	202012,00	2021.01.19	• Weitere Informationen darüber, was in der Version FreeRTOS 202012.00 enthalten ist, finden Sie in der Datei CHANGELOG.md unter. GitHub • Führt zusätzliche OTA (O) E2E () -Testfälle ein. ver-the-air end-to-end • Unterstützt die Qualifizierung von Entwicklungsboards, auf denen FreeRTOS 202012.00 ausgeführt wird und die FreeRTOS LTS-Bibliotheken verwenden. • Fügt Unterstützung für

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Datum der Veröffentlichung	Versionshinweise
				die Qualifizierung von FreeRTOS-Entwicklungsboards mit Mobilfunkverbindung hinzu. • Behebt einen Fehler in der Echo-Serverkonfiguration. • Ermöglicht es Ihnen, Ihre eigenen benutzerdefinierten Testsuiten mit AWS IoT Device Tester for FreeRTOS zu entwickeln und auszuführen. Weitere Informationen finden Sie unter Entwickeln und betreiben Sie Ihre

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Datum der Veröffentlichung	Versionshinweise
				eigenen IDT-Testsuiten. • Stellt codesignierte IDT-Anwendungen bereit, sodass Sie keine Berechtigungen erteilen müssen, wenn Sie sie unter Windows oder macOS ausführen. • Die Logik zur Analyse der BLE-Testergebnisse wurde verfeinert.

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Datum der Veröffentlichung	Versionshinweise
IDT v3.4.0	FRQ_1.3.0	202011,01	2020.11.05	• Weitere Informationen finden Sie in der Datei CHANGELOG.md unter. GitHub • Ein Fehler wurde behoben, bei dem 'RSA' keine gültige Konfigurationsoption war. PKCS11 • Es wurde ein Fehler behoben, bei dem Amazon S3-Buckets nach OTA-Tests nicht korrekt bereinigt wurden. • Updates zur Unterstützung der neuen Testfälle innerhalb der

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Datum der Veröffentlichung	Versionshinweise
				FullMQTT-Testgruppe.

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Datum der Veröffentlichung	Versionshinweise
IDT v3.3.0	FRQ_1.2.0	202007,00	17.09.2020	• Weitere Informationen finden Sie in der Datei CHANGELOG.md unter. GitHub • Neue end-to-end Tests zur Validierung der Funktion zum Aussetzen und Wiederaufnehmen von Over-The-Air-Updates (OTA). • Es wurde ein Fehler behoben, der dazu führte, dass Benutzer in der Region eu-central-1 die Konfigurationsvalidierung für OTA-Tests nicht bestehen konnten.

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Datum der Veröffentlichung	Versionshinweise
				• Dem Befehl wurde ein `--update-idt` Parameter hinzugefügt. `run-suite` Sie können diese Option verwenden, um die Antwort für die IDT-Aktualisierung saufforderung festzulegen. • Dem `run-suite` Befehl wurde ein `--update-managed-policy` Parameter hinzugefügt. Sie können diese Option verwenden , um die Antwort auf die Aufforder ung zur Aktualisierung verwalteter

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Datum der Veröffentlichung	Versionshinweise
				Richtlinien festzulegen. • Interne Verbesserungen und Fehlerkorrekturen, darunter: • Für automatische Updates der Testsuite, Verbesserungen beim Upgrade der Konfigurationsdatei.

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Datum der Veröffentlichung	Versionshinweise
IDT v3.0.2	FRQ_1.0.1	202002.00		• Weitere Informationen finden Sie in der Datei CHANGELOG.md unter. GitHub • Fügt die automatische Aktualisierung der Testsuiten innerhalb von IDT hinzu. IDT kann jetzt die neuesten Testsuiten herunterladen, die für Ihre FreeRTOS-Version verfügbar sind. Mit dieser Funktion können Sie: • Die neuesten Testsuiten mit dem Befehl `upgrade-test-suite`

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Datum der Veröffentlichung	Versionshinweise
				herunterladen. • Die neuesten Testsuites herunterladen, indem Sie beim Starten von IDT ein Flag setzen. Verwenden Sie die `-u flag` Option '`flag`', um immer herunterzuladen, oder y 'n', um die bestehende Version zu verwenden. Wenn mehrere Test-Suite-Versionen verfügbar sind, wird die neueste Version verwendet,

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Datum der Veröffentlichung	Versionshinweise
				es sei denn, Sie geben beim Starten von IDT eine Test-Suite-ID an. • Verwenden Sie die neue `list-supported-versions` Option, um die FreeRTOS- und Testsuite-Versionen aufzulisten, die von der installierten Version von IDT unterstützt werden. • Listen Sie Testfälle in einer Gruppe auf und führen Sie einzelne Tests aus.

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Datum der Veröffentlichung	Versionshinweise
				Testsuites sind beginnend mit 1.0.0 im Format `major.minor.patch` versioniert. • Fügt den `list-supported-products` Befehl hinzu — Listet die FreeRTOS- und Testsuite-Versionen auf, die von der installierten Version von IDT unterstützt werden. • Fügt `list-test-cases` den Befehl hinzu — Listet die Testfälle auf, die in einer Testgruppe verfügbar sind. • Fügt die `test-id`

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Datum der Veröffentlichung	Versionshinweise
				Option für den `run-suite` Befehl hinzu — Verwenden Sie diese Option, um einzelne Testfälle in einer Testgruppe auszuführen.

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Datum der Veröffentlichung	Versionshinweise
IDT v1.7.1	FRQ_1.0.0	202002.00		• Weitere Informationen finden Sie in der Datei CHANGELOG.md unter. GitHub • Unterstützt die benutzerdefinierte Codesignaturmethode für over-the-air (OTA) end-to-end -Testfälle, sodass Sie Ihre eigenen Codesignaturbefehle und -Skripts verwenden können, um OTA-Payloads zu signieren. • Fügt vor Beginn der Tests eine Vorprüfung für serielle Ports hinzu. Tests werden schnell mit verbesser

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Datum der Veröffentlichung	Versionshinweise
				ter Fehlermeldungen fehlschlagen, wenn der serielle Port in der `device.json` -Datei falsch konfiguriert ist. • Es wurde eine AWS verwaltete Richtlinie hinzugefügt, für `AWSIoTDeviceTesterForFreeRTOSFullAccess` deren Ausführung AWS IoT Device Tester die erforderlichen Berechtigungen erforderlich sind. Wenn für neue Versionen zusätzliche Berechtig

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Datum der Veröffentlichung	Versionshinweise
				ungen erforderlich sind, fügen wir sie zu dieser verwalteten Richtlinie hinzu, sodass Sie Ihre IAM-Berechtigungen nicht manuell aktualisieren müssen. • Die Datei namens „`AFQ_Report.xml` “ im Ergebnisverzeichnis ist jetzt `FRQ_Report.xml` .

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Datum der Veröffentlichung	Versionshinweise
IDT v1.6.2	FRQ_1.0.0	202002.00		• Unterstützt optionale Tests für OTA über HTTPS, um Ihre FreeRTOS-Entwicklungsboards zu qualifizieren. • Unterstützt den AWS IoT ATS-Endpunkt beim Testen. • Unterstützt die Fähigkeit, Benutzer vor Beginn der Testsuite über die neueste IDT-Version zu informieren.

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Datum der Veröffentlichung	Versionshinweise
IDT v1.5.2	FRQ_1.0.0	201910.00		• Unterstützt die Qualifizierung von FreeRTOS-Geräten mit einem sicheren Element (integrierter Schlüssel). • Unterstützt konfigurierbare Echo-Server-Ports für Secure Sockets- und WLAN-Testgruppen. • Unterstützt das Timeout-Multiplikator-Flag, um die Timeouts zu erhöhen. Dies ist nützlich, wenn Sie Fehler im Zusammenhang mit Timeouts beheben möchten.

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Datum der Veröffentlichung	Versionshinweise
				• Fehlerbehebung für Protokollanalyse hinzugefügt. • Unterstützt IOT-ATS-Endpunkt beim Testen.
IDT v1.4.1	FRQ_1.0.0	201908.00		• Unterstützung für neue PKCS11 Bibliotheks- und Testfall-Updates hinzugefügt. • Einführung von verwertbaren Fehlercodes. Weitere Informationen finden Sie unter IDT-Fehlercodes • Aktualisierung der IAM-Richtlinie, die für die Ausführung von IDT verwendet wird.

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Datum der Veröffentlichung	Versionshinweise
IDT v1.3.2	FRQ_1.0.0	201906,00		• Hinzufügung der Unterstützung für das Testen von Bluetooth Low Energy (BLE). • Verbesserte Benutzererfahrung für Befehle der IDT-Befehlszeilenschnittstelle (CLI). • Aktualisierung der IAM-Richtlinie, die für die Ausführung von IDT verwendet wird.
IDT-FreeRTOS v1.2	FRQ_1.0.0	• FreeRTOS v1.4.8 • FreeRTOS v1.4.9		Unterstützung für das Testen von FreeRTOS-Geräten mit dem CMAKE-Build-System hinzugefügt.
IDT-FreeRTOS v1.1	FRQ_1.0.0			

AWS IoT Device Tester Version	Versionen der Testsuite	Unterstützte FreeRTOS-Versionen	Datum der Veröffentlichung	Versionshinweise
IDT-FreeRTOS v1.0	FRQ_1.0.0			

Laden Sie IDT für FreeRTOS herunter

In diesem Thema werden die Optionen zum Herunterladen von IDT für FreeRTOS beschrieben. Sie können entweder einen der folgenden Links zum Herunterladen von Software verwenden oder den Anweisungen folgen, um IDT programmgesteuert herunterzuladen.

> **Important**
>
> Stand Oktober 2022 generiert AWS IoT FreeRTOS Qualification (FRQ) 1.0 keine signierten Qualifikationsberichte. AWS IoT Device Tester Sie können neue AWS IoT FreeRTOS-Geräte nicht für die Aufnahme in den AWS Partnergerätekatalog über das AWS Gerätequalifizierungsprogramm qualifizieren, wenn Sie IDT FRQ 1.0-Versionen verwenden. Sie können FreeRTOS-Geräte zwar nicht mit IDT FRQ 1.0 qualifizieren, aber Sie können Ihre FreeRTOS-Geräte weiterhin mit FRQ 1.0 testen. Wir empfehlen Ihnen, IDT FRQ 2.0 zu verwenden, um FreeRTOS-Geräte zu qualifizieren und im Partnergerätekatalog aufzulisten.AWS

Themen

- Laden Sie IDT manuell herunter
- Laden Sie IDT programmgesteuert herunter

Durch das Herunterladen der Software stimmen Sie der im Download-Archiv enthaltenen AWS IoT Device Tester Lizenzvereinbarung zu.

> **Note**
>
> IDT unterstützt nicht die Ausführung durch mehrere Benutzer von einem gemeinsam genutzten Speicherort aus, z. B. einem NFS-Verzeichnis oder einem freigegebenen

Windows-Netzwerkordner. Es wird empfohlen, das IDT-Paket auf ein lokales Laufwerk zu extrahieren und die IDT-Binärdatei auf Ihrer lokalen Workstation auszuführen.

Laden Sie IDT manuell herunter

In diesem Thema werden die unterstützten Versionen von IDT für FreeRTOS aufgeführt. Als bewährte Methode empfehlen wir, dass Sie die neueste Version verwenden AWS IoT Device Tester , die Ihre Zielversion von FreeRTOS unterstützt. Bei neuen Versionen von FreeRTOS müssen Sie möglicherweise eine neue Version von herunterladen. AWS IoT Device Tester Sie erhalten eine Benachrichtigung, wenn Sie einen Testlauf starten, falls dieser nicht mit der von Ihnen verwendeten Version von FreeRTOS kompatibel AWS IoT Device Tester ist.

Siehe Unterstützte Versionen von AWS IoT Device Tester

Laden Sie IDT programmgesteuert herunter

IDT bietet eine API-Operation, mit der Sie eine URL abrufen können, über die Sie IDT programmgesteuert herunterladen können. Sie können diesen API-Vorgang auch verwenden, um zu überprüfen, ob Sie über die neueste Version von IDT verfügen. Dieser API-Vorgang hat den folgenden Endpunkt.

```
https://download.devicetester.iotdevicesecosystem.amazonaws.com/latestidt
```

Um diesen API-Vorgang aufzurufen, benötigen Sie die Berechtigung, die **`iot-device-tester:LatestIdt`** Aktion auszuführen. Fügen Sie Ihre AWS Signatur mit `iot-device-tester` als Dienstnamen hinzu

API-Anfrage

HostOs — Das Betriebssystem des Host-Computers. Wählen Sie aus den folgenden Optionen aus:

- `mac`
- `linux`
- `windows`

TestSuiteType — Der Typ der Testsuite. Wählen Sie die folgende Option:

`FR`— IDT für FreeRTOS

ProductVersion

(Optional) Die Version von FreeRTOS. Der Dienst gibt die neueste kompatible Version von IDT für diese Version von FreeRTOS zurück. Wenn Sie diese Option nicht angeben, gibt der Dienst die neueste Version von IDT zurück.

API-Antwort

Die API-Antwort hat das folgende Format. Das `DownloadURL` beinhaltet eine Zip-Datei.

```
{
    "Success": True or False,
    "Message": Message,
    "LatestBk": {
        "Version": The version of the IDT binary,
        "TestSuiteVersion": The version of the test suite,
        "DownloadURL": The URL to download the IDT Bundle, valid for one hour
    }
 }
```

Beispiele

Sie können die folgenden Beispiele verwenden, um IDT programmgesteuert herunterzuladen. In diesen Beispielen werden Anmeldeinformationen verwendet, die Sie in den Umgebungsvariablen `AWS_ACCESS_KEY_ID` und `AWS_SECRET_ACCESS_KEY` speichern. Speichern Sie Ihre Anmeldeinformationen nicht in Ihrem Code, um die besten Sicherheitsvorkehrungen zu befolgen.

Example

Beispiel: Herunterladen mit cURL Version 7.75.0 oder höher (Mac und Linux)

Wenn Sie die cURL-Version 7.75.0 oder höher haben, können Sie das `aws-sigv4` Flag verwenden, um die API-Anfrage zu signieren. In diesem Beispiel wird jq verwendet, um die Download-URL aus der Antwort zu analysieren.

> **Warning**
>
> Das **`aws-sigv4`** Flag erfordert, dass die Abfrageparameter der curl-GET-Anfrage in der Reihenfolge von HostOs/ProductVersion/TestSuiteTypeoder/angegeben werden. HostOs

TestSuiteType Bestellungen, die nicht konform sind, führen zu einem Fehler beim Abrufen nicht übereinstimmender Signaturen für den Canonical String vom API Gateway.
Wenn der optionale Parameter enthalten ProductVersionist, müssen Sie eine unterstützte Produktversion verwenden, wie unter Unterstützte Versionen von AWS IoT Device Tester für FreeRTOS dokumentiert.

- Ersetze es durch *us-west-2* dein. AWS-Region Eine Liste der Regionscodes finden Sie unter Regionale Endpunkte.
- *linux*Ersetzen Sie es durch das Betriebssystem Ihres Host-Computers.
- Ersetze es *202107.00* durch deine Version von FreeRTOS.

```
url=$(curl --request GET "https://
download.devicetester.iotdevicesecosystem.amazonaws.com/latestidt?
HostOs=linux&ProductVersion=202107.00&TestSuiteType=FR" \
--user $AWS_ACCESS_KEY_ID:$AWS_SECRET_ACCESS_KEY \
--aws-sigv4 "aws:amz:us-west-2:iot-device-tester" \
| jq -r '.LatestBk["DownloadURL"]')

curl $url --output devicetester.zip
```

Example

Beispiel: Herunterladen mit einer früheren Version von cURL (Mac und Linux)

Sie können den folgenden cURL-Befehl mit einer AWS Signatur verwenden, die Sie signieren und berechnen. Weitere Informationen zum Signieren und Berechnen einer AWS Signatur finden Sie unter AWS API-Anfragen signieren.

- *linux*Ersetzen Sie es durch das Betriebssystem Ihres Host-Computers.
- *Timestamp*Ersetzen Sie durch Datum und Uhrzeit, z. **20220210T004606Z** B.
- *Date*Durch das Datum ersetzen, z. **20220210** B.
- Ersetze *AWSRegion* durch dein AWS-Region. Eine Liste der Regionscodes finden Sie unter Regionale Endpunkte.
- *AWSSignature*Ersetzen Sie es durch die AWS Signatur, die Sie generieren.

```
curl --location --request GET 'https://
download.devicetester.iotdevicesecosystem.amazonaws.com/latestidt?
HostOs=linux&TestSuiteType=FR' \
--header 'X-Amz-Date: Timestamp \
--header 'Authorization: AWS4-HMAC-SHA256 Credential=$AWS_ACCESS_KEY_ID/Date/AWSRegion/
iot-device-tester/aws4_request, SignedHeaders=host;x-amz-date, Signature=AWSSignature'
```

Example

Beispiel: Herunterladen mit einem Python-Skript

In diesem Beispiel wird die Python-Anforderungsbibliothek verwendet. Dieses Beispiel ist an das Python-Beispiel angepasst, um eine AWS API-Anfrage zu signieren in der AWS Allgemeinen Referenz.

- Ersetze es *us-west-2* durch deine Region. Eine Liste der Regionalcodes finden Sie unter Regionale Endpunkte.
- *linux*Ersetzen Sie es durch das Betriebssystem Ihres Host-Computers.

```
# Copyright 2010-2022 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#
# This file is licensed under the Apache License, Version 2.0 (the "License").
# You may not use this file except in compliance with the License. A copy of the
#License is located at
#
# http://aws.amazon.com/apache2.0/
#
# This file is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS
# OF ANY KIND, either express or implied. See the License for the specific
# language governing permissions and limitations under the License.

# See: http://docs.aws.amazon.com/general/latest/gr/sigv4_signing.html
# This version makes a GET request and passes the signature
# in the Authorization header.
import sys, os, base64, datetime, hashlib, hmac
import requests # pip install requests
# ************* REQUEST VALUES *************
method = 'GET'
service = 'iot-device-tester'
host = 'download.devicetester.iotdevicesecosystem.amazonaws.com'
region = 'us-west-2'
```

```
endpoint = 'https://download.devicetester.iotdevicesecosystem.amazonaws.com/latestidt'
request_parameters = 'HostOs=linux&TestSuiteType=FR'

# Key derivation functions. See:
# http://docs.aws.amazon.com/general/latest/gr/signature-v4-examples.html#signature-v4-
examples-python
def sign(key, msg):
    return hmac.new(key, msg.encode('utf-8'), hashlib.sha256).digest()

def getSignatureKey(key, dateStamp, regionName, serviceName):
    kDate = sign(('AWS4' + key).encode('utf-8'), dateStamp)
    kRegion = sign(kDate, regionName)
    kService = sign(kRegion, serviceName)
    kSigning = sign(kService, 'aws4_request')
    return kSigning

# Read AWS access key from env. variables or configuration file. Best practice is NOT
# to embed credentials in code.
access_key = os.environ.get('AWS_ACCESS_KEY_ID')
secret_key = os.environ.get('AWS_SECRET_ACCESS_KEY')
if access_key is None or secret_key is None:
    print('No access key is available.')
    sys.exit()

# Create a date for headers and the credential string
t = datetime.datetime.utcnow()
amzdate = t.strftime('%Y%m%dT%H%M%SZ')
datestamp = t.strftime('%Y%m%d') # Date w/o time, used in credential scope

# ************* TASK 1: CREATE A CANONICAL REQUEST *************
# http://docs.aws.amazon.com/general/latest/gr/sigv4-create-canonical-request.html
# Step 1 is to define the verb (GET, POST, etc.)--already done.
# Step 2: Create canonical URI--the part of the URI from domain to query
# string (use '/' if no path)
canonical_uri = '/latestidt'
# Step 3: Create the canonical query string. In this example (a GET request),
# request parameters are in the query string. Query string values must
# be URL-encoded (space=%20). The parameters must be sorted by name.
# For this example, the query string is pre-formatted in the request_parameters
 variable.
canonical_querystring = request_parameters
# Step 4: Create the canonical headers and signed headers. Header names
# must be trimmed and lowercase, and sorted in code point order from
# low to high. Note that there is a trailing \n.
```

```
canonical_headers = 'host:' + host + '\n' + 'x-amz-date:' + amzdate + '\n'
# Step 5: Create the list of signed headers. This lists the headers
# in the canonical_headers list, delimited with ";" and in alpha order.
# Note: The request can include any headers; canonical_headers and
# signed_headers lists those that you want to be included in the
# hash of the request. "Host" and "x-amz-date" are always required.
signed_headers = 'host;x-amz-date'
# Step 6: Create payload hash (hash of the request body content). For GET
# requests, the payload is an empty string ("").
payload_hash = hashlib.sha256(('').encode('utf-8')).hexdigest()
# Step 7: Combine elements to create canonical request
canonical_request = method + '\n' + canonical_uri + '\n' + canonical_querystring + '\n'
 + canonical_headers + '\n' + signed_headers + '\n' + payload_hash

# ************* TASK 2: CREATE THE STRING TO SIGN*************
# Match the algorithm to the hashing algorithm you use, either SHA-1 or
# SHA-256 (recommended)
algorithm = 'AWS4-HMAC-SHA256'
credential_scope = datestamp + '/' + region + '/' + service + '/' + 'aws4_request'
string_to_sign = algorithm + '\n' +  amzdate + '\n' +  credential_scope + '\n' +
 hashlib.sha256(canonical_request.encode('utf-8')).hexdigest()
# ************* TASK 3: CALCULATE THE SIGNATURE *************
# Create the signing key using the function defined above.
signing_key = getSignatureKey(secret_key, datestamp, region, service)
# Sign the string_to_sign using the signing_key
signature = hmac.new(signing_key, (string_to_sign).encode('utf-8'),
 hashlib.sha256).hexdigest()

# ************* TASK 4: ADD SIGNING INFORMATION TO THE REQUEST *************
# The signing information can be either in a query string value or in
# a header named Authorization. This code shows how to use a header.
# Create authorization header and add to request headers
authorization_header = algorithm + ' ' + 'Credential=' + access_key + '/' +
 credential_scope + ', ' +  'SignedHeaders=' + signed_headers + ', ' + 'Signature=' +
 signature
# The request can include any headers, but MUST include "host", "x-amz-date",
# and (for this scenario) "Authorization". "host" and "x-amz-date" must
# be included in the canonical_headers and signed_headers, as noted
# earlier. Order here is not significant.
# Python note: The 'host' header is added automatically by the Python 'requests'
 library.
headers = {'x-amz-date':amzdate, 'Authorization':authorization_header}

# ************* SEND THE REQUEST *************
```

```
request_url = endpoint + '?' + canonical_querystring
print('\nBEGIN REQUEST++++++++++++++++++++++++++++++++++++')
print('Request URL = ' + request_url)
response = requests.get(request_url, headers=headers)
print('\nRESPONSE++++++++++++++++++++++++++++++++++++')
print('Response code: %d\n' % response.status_code)
print(response.text)

download_url = response.json()["LatestBk"]["DownloadURL"]
r = requests.get(download_url)
open('devicetester.zip', 'wb').write(r.content)
```

IDT mit FreeRTOS Qualification Suite 2.0 (FRQ 2.0)

Die FreeRTOS Qualification Suite 2.0 ist eine aktualisierte Version der FreeRTOS Qualification Suite. Wir empfehlen Entwicklern, FRQ 2.0 zu verwenden, da es aus relevanten Testfällen besteht, um Geräte zu qualifizieren, auf denen FreeRTOS Long Term Support (LTS) -Bibliotheken ausgeführt werden.

IDT for FreeRTOS überprüft den Port von FreeRTOS auf Ihrem Mikrocontroller und ob er effektiv mit ihm kommuniziert. AWS IoT Insbesondere überprüft es die Schnittstellen der Portierungsschicht zu den FreeRTOS-Bibliotheken und ob die FreeRTOS-Testrepositorien korrekt implementiert sind. Es führt auch Tests mit durch. end-to-end AWS IoT CoreDie von IDT für FreeRTOS ausgeführten Tests sind im FreeRTOS-Repository definiert. GitHub

IDT for FreeRTOS führt Tests als eingebettete Anwendungen aus, die auf dem zu testenden Mikrocontroller-Gerät blinken. Die Binär-Images der Anwendung enthalten FreeRTOS, die portierten FreeRTOS-Schnittstellen und Platinengerätetreiber. Der Zweck der Tests besteht darin, zu überprüfen, ob die portierten FreeRTOS-Schnittstellen auf Ihren Gerätetreibern korrekt funktionieren.

IDT for FreeRTOS generiert Testberichte, die Sie einreichen können, AWS IoT damit Ihre Hardware im AWS Partnergerätekatalog aufgeführt wird. Weitere Informationen finden Sie unter AWS Device Qualification Program.

IDT for FreeRTOS läuft auf einem Host-Computer (Windows, macOS oder Linux), der mit dem zu testenden Gerät verbunden ist. IDT konfiguriert und orchestriert Testfälle und aggregiert die Ergebnisse. Es bietet auch eine Befehlszeilenschnittstelle zur Verwaltung der Ausführung der Tests.

Um Ihr Gerät zu testen, erstellt IDT for FreeRTOS Ressourcen wie AWS IoT Dinge, FreeRTOS-Gruppen und Lambda-Funktionen. Um diese Ressourcen zu erstellen, verwendet IDT for FreeRTOS

die in der konfigurierten AWS Anmeldeinformationen, `config.json` um API-Aufrufe in Ihrem Namen durchzuführen. Diese Ressourcen werden zu verschiedenen Zeiten während eines Tests bereitgestellt.

Wenn Sie IDT for FreeRTOS auf Ihrem Host-Computer ausführen, werden die folgenden Schritte ausgeführt:

1. Laden und überprüfen Sie die Konfiguration Ihres Geräts und Ihrer Anmeldeinformationen.
2. Führen Sie ausgewählte Tests mit den erforderlichen lokalen und Cloud-Ressourcen durch.
3. Bereinigen Sie lokale und Cloud-Ressourcen.
4. Erstellen Sie Testberichte, die anzeigen, ob Ihr Board die für die Qualifikation erforderlichen Tests bestanden hat.

Richten Sie die Voraussetzungen für die LTS-Qualifikation ein

In diesem Abschnitt werden die Voraussetzungen für das Testen von Mikrocontrollern mit AWS IoT Device Tester beschrieben.

Bereiten Sie sich auf die FreeRTOS-Qualifikation vor

> **Note**
>
> AWS IoT Device Tester for FreeRTOS empfiehlt dringend, die neueste Patch-Version der neuesten FreeRTOS-LTS-Version zu verwenden.

IDT für FRQ 2.0 ist eine Qualifikation für FreeRTOS. Bevor Sie IDT FRQ 2.0 zur Qualifizierung ausführen, müssen Sie die Qualifizierung Ihres Boards im FreeRTOS Qualification Guide abschließen. Informationen zum Portieren, Testen und Einrichten von Bibliotheken finden Sie unter Portieren der `manifest.yml` FreeRTOS-Bibliotheken im FreeRTOS Porting Guide. FRQ 2.0 beinhaltet ein anderes Qualifizierungsverfahren. Einzelheiten finden Sie im FreeRTOS-Qualifikationsleitfaden unter Letzte Qualifikationsänderungen.

Das FreeRTOS-Libraries-Integration-Tests-Repository muss vorhanden sein, damit IDT ausgeführt werden kann. Informationen zum Klonen und Portieren dieses Repositorys in Ihr Quellprojekt finden Sie in der README.md. FreeRTOS-Libraries-Integration-Testsmuss die Datei enthalten, die `manifest.yml` sich im Stammverzeichnis Ihres Projekts befindet, damit IDT ausgeführt werden kann.

> **Note**
>
> IDT hängt von der Implementierung von im Test-Repository ab. `UNITY_OUTPUT_CHAR` Die Testausgabeprotokolle und die Geräteprotokolle dürfen nicht miteinander verschachtelt werden. Weitere Informationen finden Sie im Abschnitt Implementierung der Bibliotheksprotokollierungsmakros im FreeRTOS Porting Guide.

Laden Sie IDT für FreeRTOS herunter

Jede Version von FreeRTOS hat eine entsprechende Version von IDT für FreeRTOS zur Durchführung von Qualifizierungstests. Laden Sie die entsprechende Version von IDT für FreeRTOS von Unterstützte Versionen von AWS IoT Device Tester für FreeRTOS herunter.

Extrahieren Sie IDT for FreeRTOS an einen Speicherort im Dateisystem, für den Sie Lese- und Schreibberechtigungen haben. Da Microsoft Windows eine Zeichenbeschränkung für die Pfadlänge hat, extrahieren Sie IDT für FreeRTOS in ein Stammverzeichnis wie oder. `C:\` `D:\`

> **Note**
>
> IDT darf nicht von mehreren Benutzern von einem gemeinsam genutzten Speicherort aus ausgeführt werden, z. B. von einem NFS-Verzeichnis oder einem gemeinsam genutzten Windows-Netzwerkordner. Dies führt zu Abstürzen oder Datenbeschädigungen. Es wird empfohlen, das IDT-Paket in einem lokalen Laufwerk zu extrahieren.

Laden Sie Git herunter

IDT muss Git als Voraussetzung installiert haben, um die Integrität des Quellcodes sicherzustellen.

Folgen Sie den Anweisungen in der GitHubAnleitung, um Git zu installieren. Um die aktuell installierte Version von Git zu überprüfen, geben Sie den Befehl `git --version` am Terminal ein.

> **Warning**
>
> IDT verwendet Git, um sich an den Status eines Verzeichnisses „sauber“ oder „Dirty“ anzupassen. Wenn Git nicht installiert ist, schlagen die `FreeRTOSIntegrity` Testgruppen entweder fehl oder laufen nicht wie erwartet. Wenn IDT einen Fehler wie `git executable`

`not found` oder zurückgibt`git command not found`, installieren Sie Git oder installieren Sie es erneut und versuchen Sie es erneut.

Themen

- Erstelle ein Konto AWS
- AWS IoT Device Tester verwaltete Richtlinie
- (Optional) Installieren Sie den AWS Command Line Interface

Erstelle ein Konto AWS

> **Note**
>
> Die vollständige IDT-Qualifizierungssuite wird nur in den folgenden Fällen unterstützt AWS-Regionen
>
> - USA Ost (Nord-Virginia)
> - USA West (Oregon)
> - Asien-Pazifik (Tokio)
> - Europa (Irland)

Um Ihr Gerät zu testen, erstellt IDT for FreeRTOS Ressourcen wie AWS IoT Dinge, FreeRTOS-Gruppen und Lambda-Funktionen. Um diese Ressourcen zu erstellen, müssen Sie für IDT for FreeRTOS ein AWS Konto und eine IAM-Richtlinie erstellen und konfigurieren, die IDT for FreeRTOS die Erlaubnis erteilt, während der Ausführung von Tests in Ihrem Namen auf Ressourcen zuzugreifen.

In den folgenden Schritten erstellen und konfigurieren Sie Ihr Konto. AWS

1. Wenn Sie bereits ein AWS Konto haben, fahren Sie mit dem nächsten Schritt fort. Andernfalls erstellen Sie ein AWS Konto.
2. Folgen Sie den Schritten unter IAM-Rollen erstellen. Fügen Sie derzeit keine Berechtigungen oder Richtlinien hinzu.
3. Um OTA-Qualifizierungstests durchzuführen, fahren Sie mit Schritt 4 fort. Andernfalls fahren Sie mit Schritt 5 fort.
4. Hängen Sie die Inline-Richtlinie für OTA-IAM-Berechtigungen an Ihre IAM-Rolle an.

a.

> **Important**
>
> Die folgende Richtlinienvorlage erteilt die IDT-Berechtigung zum Erstellen von Rollen, Erstellen von Richtlinien und Zuweise von Richtlinien an Rollen. IDT for FreeRTOS verwendet diese Berechtigungen für Tests, die Rollen erstellen. Obwohl die Richtlinienvorlage dem Benutzer keine Administratorrechte gewährt, können die Berechtigungen verwendet werden, um Administratorzugriff auf Ihr Konto zu erhalten. AWS

b. Gehen Sie wie folgt vor, um Ihrer IAM-Rolle die erforderlichen Berechtigungen zuzuweisen:

i. Wählen Sie auf der Seite Berechtigungen die Option Berechtigungen hinzufügen aus.

ii. Wählen Sie Inline-Richtlinie erstellen aus.

iii. Wählen Sie die Registerkarte JSON aus und kopieren Sie die folgenden Berechtigungen in das JSON-Textfeld. Verwenden Sie die Vorlage unter Die meisten Regionen, wenn Sie sich nicht in der Region China befinden. Wenn Sie sich in der Region China befinden, verwenden Sie die Vorlage unter den Regionen Peking und Ningxia.

Most Regions

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iotdeviceadvisor:*",
            "Resource": [
                "arn:aws:iotdeviceadvisor:*:*:suiterun/*/*",
                "arn:aws:iotdeviceadvisor:*:*:suitedefinition/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::*:role/idt*",
            "Condition": {
```

```
                "StringEquals": {
                    "iam:PassedToService":
"iotdeviceadvisor.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "execute-api:Invoke*",
                "iam:ListRoles",
                "iot:Connect",
                "iot:CreateJob",
                "iot:DeleteJob",
                "iot:DescribeCertificate",
                "iot:DescribeEndpoint",
                "iot:DescribeJobExecution",
                "iot:DescribeJob",

                "iot:DescribeThing",
                "iot:GetPolicy",
                "iot:ListAttachedPolicies",
                "iot:ListCertificates",
                "iot:ListPrincipalPolicies",
                "iot:ListThingPrincipals",
                "iot:ListThings",
                "iot:Publish",
                "iot:UpdateThingShadow",
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:DescribeLogGroups",
                "logs:DescribeLogStreams",
                "logs:PutLogEvents",
                "logs:PutRetentionPolicy"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iotdeviceadvisor:*",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
```

```
            "Action": "logs:DeleteLogGroup",
            "Resource": "arn:aws:logs:*:*:log-group:/aws/iot/
deviceadvisor/*"
        },
        {
            "Effect": "Allow",
            "Action": "logs:GetLogEvents",
            "Resource": "arn:aws:logs:*:*:log-group:/aws/iot/
deviceadvisor/*:log-stream:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreatePolicy",
                "iam:DetachRolePolicy",
                "iam:DeleteRolePolicy",
                "iam:DeletePolicy",
                "iam:CreateRole",
                "iam:DeleteRole",
                "iam:AttachRolePolicy"
            ],
            "Resource": [
                "arn:aws:iam::*:policy/idt*",
                "arn:aws:iam::*:role/idt*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ssm:GetParameters"
            ],
            "Resource": [
                "arn:aws:ssm:*::parameter/aws/service/ami-amazon-
linux-latest/amzn2-ami-hvm-x86_64-gp2"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeInstances",
                "ec2:RunInstances",
                "ec2:CreateSecurityGroup",
                "ec2:CreateTags",
                "ec2:DeleteTags"
```

```
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateKeyPair",
                "ec2:DeleteKeyPair"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:key-pair/idt-ec2-ssh-key-*"
            ]
        },
        {
            "Effect": "Allow",
            "Condition": {
                "StringEqualsIgnoreCase": {
                    "aws:ResourceTag/Owner": "IoTDeviceTester"
                }
            },
            "Action": [
                "ec2:TerminateInstances",
                "ec2:DeleteSecurityGroup",
                "ec2:AuthorizeSecurityGroupIngress",
                "ec2:RevokeSecurityGroupIngress"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Beijing and Ningxia Regions

Die folgende Richtlinienvorlage kann in den Regionen Peking und Ningxia verwendet werden.

iv. Wählen Sie, wenn Sie fertig sind, Review policy (Richtlinie überprüfen) aus.

v. Geben Sie IDTFreeRTOSIAMPermissionsden Namen der Richtlinie ein.

vi. Wählen Sie Richtlinie erstellen aus.

5. Hängen Sie AWSIoTDeviceTesterForFreeRTOSFullAccess an Ihre IAM-Rolle an.

   a. Gehen Sie wie folgt vor, um Ihrer IAM-Rolle die erforderlichen Berechtigungen zuzuweisen:

      i. Wählen Sie auf der Seite Berechtigungen die Option Berechtigungen hinzufügen aus.

      ii. Wählen Sie Richtlinien anfügen.

      iii. Suchen Sie nach der AWSIoTDeviceTesterForFreeRTOSFullZugriffsrichtlinie. Markieren Sie das Kästchen.

   b. Wählen Sie Add permissions (Berechtigungen hinzufügen) aus.

6. Anmeldeinformationen für IDT exportieren. Weitere Informationen finden Sie unter Abrufen von IAM-Rollenanmeldedaten für den CLI-Zugriff.

AWS IoT Device Tester verwaltete Richtlinie

Die `AWSIoTDeviceTesterForFreeRTOSFullAccess` verwaltete Richtlinie enthält die folgenden AWS IoT Device Tester Berechtigungen für die Versionsprüfung, auto Aktualisierungsfunktionen und die Erfassung von Metriken.

- `iot-device-tester:SupportedVersion`

  Erteilt die AWS IoT Device Tester Berechtigung zum Abrufen der Liste der unterstützten Produkte, Testsuiten und IDT-Versionen.

- `iot-device-tester:LatestIdt`

  AWS IoT Device Tester Erteilt die Erlaubnis, die neueste IDT-Version abzurufen, die zum Herunterladen verfügbar ist.

- `iot-device-tester:CheckVersion`

  AWS IoT Device Tester Erteilt die Erlaubnis, die Versionskompatibilität für IDT, Testsuiten und Produkte zu überprüfen.

- `iot-device-tester:DownloadTestSuite`

  AWS IoT Device Tester Erteilt die Erlaubnis zum Herunterladen von Testsuite-Updates.

- `iot-device-tester:SendMetrics`

  AWS Erteilt die Erlaubnis, Metriken zur AWS IoT Device Tester internen Nutzung zu sammeln.

(Optional) Installieren Sie den AWS Command Line Interface

Möglicherweise ziehen Sie es vor, den zu verwenden AWS CLI , um einige Operationen auszuführen. Wenn Sie das nicht AWS CLI installiert haben, folgen Sie den Anweisungen unter Installieren von AWS CLI.

Konfigurieren Sie das AWS CLI für die AWS Region, die Sie verwenden möchten, indem Sie es aws configure von einer Befehlszeile aus ausführen. Informationen zu den AWS Regionen, die IDT für FreeRTOS unterstützen, finden Sie unter AWS Regionen und Endpunkte. Weitere Informationen finden Sie unter Schnellkonfiguration mitaws configure. aws configure

Erster Test Ihres Mikrocontroller-Boards

Sie können IDT for FreeRTOS verwenden, um Ihre Implementierung der FreeRTOS-Bibliotheken zu testen. Nachdem Sie die FreeRTOS-Bibliotheken für die Gerätetreiber Ihres Boards portiert haben, verwenden Sie AWS IoT Device Tester sie, um die Qualifizierungstests auf Ihrer Mikrocontroller-Karte durchzuführen.

Hinzufügen von Portierungsebenen für Bibliotheken

Informationen zur Portierung von FreeRTOS für Ihr Gerät finden Sie im FreeRTOS Porting Guide. Bei der Implementierung des FreeRTOS-Testrepositorys und der Portierung der FreeRTOS-Schichten müssen Sie Pfade zu jeder Bibliothek angeben, einschließlich des Test-Repositorys. `manifest.yml` Diese Datei befindet sich im Stammverzeichnis Ihres Quellcodes. Einzelheiten finden Sie in den Anweisungen zur Manifestdatei.

Konfigurieren Sie Ihre AWS Anmeldeinformationen für AWS IoT Device Tester die Kommunikation mit der AWS Cloud

Sie müssen Ihre AWS Anmeldeinformationen für AWS IoT Device Tester die Kommunikation mit der AWS Cloud konfigurieren. Weitere Informationen finden Sie unter AWS Zugangsdaten und Region für die Entwicklung einrichten. Gültige AWS Anmeldeinformationen sind in der *`devicetester_extract_location`*`/devicetester_freertos_`*`[win|mac|linux]`*`/configs/config.json` Konfigurationsdatei angegeben.

```
"auth": {
   "method": "environment"
```

```
}

"auth": {
    "method": "file",
    "credentials": {
        "profile": "<your-aws-profile>"
    }
}
```

Das `auth` Attribut der `config.json` Datei hat ein Methodenfeld, das die AWS Authentifizierung steuert, und kann entweder als Datei oder als Umgebung deklariert werden. Wenn Sie das Feld auf Umgebung setzen, werden Ihre AWS Anmeldeinformationen aus den Umgebungsvariablen Ihres Host-Computers abgerufen. Wenn Sie das Feld auf Datei setzen, wird ein bestimmtes Profil aus der `.aws/credentials` Konfigurationsdatei importiert.

Themen

- Erstellen Sie einen Gerätepool in IDT für FreeRTOS
- Konfiguration von Build-, Flash- und Testeinstellungen

Erstellen Sie einen Gerätepool in IDT für FreeRTOS

Zu testende Geräte werden in Gerätepools organisiert. Jeder Gerätepool besteht aus einem oder mehreren identischen Geräten. Sie können IDT for FreeRTOS so konfigurieren, dass ein einzelnes Gerät oder mehrere Geräte in einem Pool getestet werden. Um den Qualifizierungsprozess zu beschleunigen, kann IDT for FreeRTOS Geräte mit denselben Spezifikationen parallel testen. Er verwendet eine Round Robin-Methode, um auf jedem Gerät in einem Gerätepool eine andere Testgruppe auszuführen.

Die `device.json` Datei hat ein Array in der obersten Ebene. Jedes Array-Attribut ist ein neuer Gerätepool. Jeder Gerätepool hat ein Geräte-Array-Attribut, für das mehrere Geräte deklariert sind. In der Vorlage gibt es einen Gerätepool und nur ein Gerät in diesem Gerätepool. Sie können ein oder mehrere Geräte zu einem Gerätepool hinzufügen, indem Sie den Abschnitt `devices` der Vorlage `device.json` im Ordner `configs` bearbeiten.

> **Note**
>
> Alle Geräte im selben Pool müssen dieselbe technische Spezifikation und SKU haben. Um parallel Builds des Quellcodes für verschiedene Testgruppen zu ermöglichen, kopiert IDT

for FreeRTOS den Quellcode in einen Ergebnisordner innerhalb des extrahierten IDT for FreeRTOS-Ordners. Sie müssen in Ihrem Build- oder Flash-Befehl mithilfe der Variablen auf den Quellcodepfad verweisen. `testdata.sourcePath` IDT for FreeRTOS ersetzt diese Variable durch einen temporären Pfad des kopierten Quellcodes. Weitere Informationen finden Sie unter IDT für FreeRTOS-Variablen.

Die folgende `device.json` Beispieldatei wurde verwendet, um einen Gerätepool mit mehreren Geräten zu erstellen.

```
[
    {
        "id": "pool-id",
        "sku": "sku",
        "features": [
           {
              "name": "Wifi",
              "value": "Yes | No"
           },
           {
              "name": "Cellular",
              "value": "Yes | No"
           },
           {
              "name": "BLE",
              "value": "Yes | No"
         },
         {
            "name": "PKCS11",
            "value": "RSA | ECC | Both"
         },
         {
             "name": "OTA",
             "value": "Yes | No",
             "configs": [
             {
                 "name": "OTADataPlaneProtocol",
                 "value": "MQTT | HTTP | None"
             }
           ]
         },
         {
```

```
              "name": "KeyProvisioning",
              "value": "Onboard | Import | Both | No"
          }
        ],
        "devices": [
          {
            "id": "device-id",
            "connectivity": {
              "protocol": "uart",
              "serialPort": "/dev/tty*"
            },
            "secureElementConfig" : {
              "publicKeyAsciiHexFilePath": "absolute-path-to/public-key-txt-file:
 contains-the-hex-bytes-public-key-extracted-from-onboard-private-key",
              "publiDeviceCertificateArn": "arn:partition:iot:region:account-
id:resourcetype:resource:qualifier",
              "secureElementSerialNumber": "secure-element-serialNo-value",
              "preProvisioned"           : "Yes | No",
              "pkcs11JITPCodeVerifyRootCertSupport": "Yes | No"
            },
            "identifiers": [
              {
                "name": "serialNo",
                "value": "serialNo-value"
              }
            ]
          }
        ]
    }
]
```

Die folgenden Attribute werden in der Datei `device.json` verwendet:

`id`

Eine benutzerdefinierte alphanumerische ID, die einen Gerätepool eindeutig identifiziert. Geräte, die zu einem Pool gehören, müssen vom gleichen Typ sein. Bei der Ausführung einer Reihe von Tests werden Geräte im Pool verwendet, um den Workload zu parallelisieren.

`sku`

Ein alphanumerischer Wert, mit dem das getestete Board eindeutig identifiziert wird. Die SKU wird verwendet, um qualifizierte Boards nachzuverfolgen.

> **Note**
>
> Wenn du dein Mainboard im Gerätekatalog für AWS Partner anbieten möchtest, muss die hier angegebene SKU mit der SKU übereinstimmen, die du bei der Angebotserstellung verwendest.

`features`

Ein Array, das die vom Gerät unterstützten Funktionen enthält. AWS IoT Device Tester verwendet diese Informationen, um die durchzuführenden Qualifikationstests auszuwählen.

Unterstützte Werte sind:

`Wifi`

Gibt an, ob Ihr Board über WiFi-Funktionen verfügt.

`Cellular`

Zeigt an, ob Ihr Motherboard über Mobilfunkfunktionen verfügt.

`PKCS11`

Gibt den Kryptografiealgorithmus mit öffentlichen Schlüsseln an, den das Board unterstützt. PKCS11 ist für die Qualifikation erforderlich. Unterstützte Werte sind `ECC`RSA, und`Both`. `Both`gibt an, dass das Board `ECC` sowohl als auch unterstütztRSA.

`KeyProvisioning`

Gibt an, wie ein vertrauenswürdiges X.509-Clientzertifikat auf das Board geschrieben werden kann.

Gültige Werte sind `Import`, `Onboard`, `Both` und`No`. `Onboard`,`Both`, oder die Bereitstellung von `No` Schlüsseln ist für die Qualifizierung erforderlich. `Import`allein ist keine gültige Qualifizierungsoption.

- Verwenden Sie diese Option `Import` nur, wenn Ihr Board den Import von privaten Schlüsseln zulässt. `Import`Die Auswahl ist keine gültige Konfiguration für die Qualifizierung und sollte nur zu Testzwecken verwendet werden, insbesondere bei PKCS11 Testfällen. `Onboard`, `Both` oder `No` ist für die Qualifizierung erforderlich.
- Verwenden Sie diese Option, `Onboard` wenn Ihr Board integrierte private Schlüssel unterstützt (z. B. wenn Ihr Gerät über ein sicheres Element verfügt oder wenn Sie es

vorziehen, Ihr eigenes Geräteschlüsselpaar und Ihr eigenes Zertifikat zu generieren). Stellen Sie sicher, dass Sie in jedem der Geräteabschnitte ein `secureElementConfig`-Element hinzufügen und fügen Sie den absoluten Pfad zur Datei des öffentlichen Schlüssels in das Feld `publicKeyAsciiHexFilePath` ein.

- Verwenden Sie diese `Both` Option, wenn Ihr Board sowohl den Import von privaten Schlüsseln als auch die integrierte Schlüsselgenerierung für die Schlüsselbereitstellung unterstützt.
- Verwenden Sie diese `No` Option, wenn Ihr Board die Schlüsselbereitstellung nicht unterstützt. `No`ist nur dann eine gültige Option, wenn Ihr Gerät ebenfalls vorab bereitgestellt wurde.

`OTA`

Zeigt an, ob Ihr Board die over-the-air Aktualisierungsfunktion (OTA) unterstützt. Das Attribut `OtaDataPlaneProtocol` gibt an, welches OTA-Protokoll auf Datenebene das Gerät unterstützt. Für die Qualifizierung ist ein OTA mit entweder dem HTTP- oder dem MQTT-Datenebenenprotokoll erforderlich. Um die Ausführung von OTA-Tests während des Tests zu überspringen, setzen Sie die OTA-Funktion auf `No` und das `OtaDataPlaneProtocol` Attribut auf. `None` Dies wird kein Qualifikationslauf sein.

`BLE`

Gibt an, ob Ihr Board Bluetooth Low Energy (BLE) unterstützt.

`devices.id`

Eine benutzerdefinierte eindeutige Kennung für das zu testende Gerät.

`devices.connectivity.serialPort`

Der serielle Port des Host-Computers, der zur Herstellung einer Verbindung mit den getesteten Geräten verwendet wird.

`devices.secureElementConfig.PublicKeyAsciiHexFilePath`

Erforderlich, wenn Ihr Board NICHT `pre-provisioned` oder nicht zur Verfügung gestellt `PublicDeviceCertificateArn` wird. Da `Onboard` es sich um eine erforderliche Art der Schlüsselbereitstellung handelt, ist dieses Feld derzeit für die FullTransportInterface TLS-Testgruppe erforderlich. Ob Ihr Gerät es ist`pre-provisioned`, `PublicKeyAsciiHexFilePath` ist optional und muss nicht enthalten sein.

Der folgende Block ist ein absoluter Pfad zu der Datei, die den öffentlichen Hex-Byte-Schlüssel enthält, der aus dem `Onboard` privaten Schlüssel extrahiert wurde.

```
3059 3013 0607 2a86 48ce 3d02 0106 082a
8648 ce3d 0301 0703 4200 04cd 6569 ceb8
1bb9 1e72 339f e8cf 60ef 0f9f b473 33ac
6f19 1813 6999 3fa0 c293 5fae 08f1 1ad0
41b7 345c e746 1046 228e 5a5f d787 d571
dcb2 4e8d 75b3 2586 e2cc 0c
```

Wenn Ihr öffentlicher Schlüssel im.der-Format vorliegt, können Sie den öffentlichen Schlüssel direkt hexcodieren, um die Hex-Datei zu generieren.

Um die Hex-Datei aus einem öffentlichen .der-Schlüssel zu generieren, geben Sie den folgenden Befehl ein: xxd

```
xxd -p pubkey.der > outFile
```

Wenn Ihr öffentlicher Schlüssel im.pem-Format vorliegt, können Sie die Base64-codierten Kopf- und Fußzeilen extrahieren und in das Binärformat dekodieren. Anschließend kodieren Sie die binäre Zeichenfolge hexadezimal, um die Hex-Datei zu generieren.

Gehen Sie wie folgt vor, um eine Hex-Datei für einen öffentlichen PEM-Schlüssel zu generieren:

1. Führen Sie den folgenden base64 Befehl aus, um die Base64-Kopf- und Fußzeile aus dem öffentlichen Schlüssel zu entfernen. Der dekodierte Schlüssel mit dem Namen wird `base64key` dann in die Datei ausgegeben: `pubkey.der`

   ```
   base64 –decode base64key > pubkey.der
   ```

2. Führen Sie den folgenden xxd Befehl aus, um in das Hex-Format `pubkey.der` zu konvertieren. Der resultierende Schlüssel wird gespeichert als *`outFile`*

   ```
   xxd -p pubkey.der > outFile
   ```

`devices.secureElementConfig.PublicDeviceCertificateArn`

Der ARN des Zertifikats von Ihrem sicheren Element, in das hochgeladen wird AWS IoT Core. Informationen zum Hochladen Ihres Zertifikats auf AWS IoT Core finden Sie unter X.509-Client-Zertifikate im AWS IoT Developer Guide.

`devices.secureElementConfig.SecureElementSerialNumber`

(Optional) Die Seriennummer des sicheren Elements. Die Seriennummer wird optional verwendet, um Gerätezertifikate für die JITR-Schlüsselbereitstellung zu erstellen.

`devices.secureElementConfig.preProvisioned`

(Optional) Legen Sie diese Option auf „Ja“ fest, wenn das Gerät über ein vorab bereitgestelltes sicheres Element mit gesperrten Anmeldeinformationen verfügt, das keine Objekte importieren, erstellen oder zerstören kann. Wenn dieses Attribut auf Ja gesetzt ist, müssen Sie die entsprechenden pkcs11-Labels angeben.

`devices.secureElementConfig.pkcs11JITPCodeVerifyRootCertSupport`

(Optional) Auf Ja setzen, wenn die PKCS11 Core-Implementierung des Geräts Speicher für JITP unterstützt. Dadurch wird der `codeverify` JITP-Test beim Testen von Core-PKCS 11 aktiviert, und es müssen die PKCS 11-Labels für den Codeverifizierungsschlüssel, das JITP-Zertifikat und das Stammzertifikat bereitgestellt werden.

`identifiers`

(Optional) Ein Array beliebiger Namen-Wert-Paare. Sie können diese Werte in den im nächsten Abschnitt beschriebenen Build- und Flash-Befehlen verwenden.

Konfiguration von Build-, Flash- und Testeinstellungen

IDT for FreeRTOS erstellt Tests automatisch und überträgt sie auf Ihr Board. Um dies zu aktivieren, müssen Sie IDT so konfigurieren, dass die Build- und Flash-Befehle für Ihre Hardware ausgeführt werden. Die Einstellungen für den Build- und den Flash-Befehl werden in der `userdata.json`-Vorlagendatei im Ordner `config` konfiguriert.

Konfigurieren von Einstellungen für das Testen von Geräten

Build-, Flash- und Testeinstellungen werden in der `configs/userdata.json`-Datei vorgenommen. Das folgende JSON-Beispiel zeigt, wie Sie IDT für FreeRTOS konfigurieren können, um mehrere Geräte zu testen:

```
{
    "sourcePath": "</path/to/freertos>",
    "retainModifiedSourceDirectories": true | false,
    "freeRTOSVersion": "<freertos-version>",
    "freeRTOSTestParamConfigPath": "{{testData.sourcePath}}/path/from/source/path/to/
test_param_config.h",
```

```
    "freeRTOSTestExecutionConfigPath": "{{testData.sourcePath}}/path/from/source/path/
to/test_execution_config.h",
    "buildTool": {
        "name": "your-build-tool-name",
        "version": "your-build-tool-version",
        "command": [
            "<build command> -any-additional-flags {{testData.sourcePath}}"
        ]
    },
    "flashTool": {
        "name": "your-flash-tool-name",
        "version": "your-flash-tool-version",
        "command": [
            "<flash command> -any-additional-flags {{testData.sourcePath}} -any-
additional-flags"
        ]
    },
    "testStartDelayms": 0,
    "echoServerConfiguration": {
      "keyGenerationMethod": "EC | RSA",
      "serverPort": 9000
    },
    "otaConfiguration": {
        "otaE2EFirmwarePath": "{{testData.sourcePath}}/relative-path-to/ota-image-
generated-in-build-process",
        "otaPALCertificatePath": "/path/to/ota/pal/certificate/on/device",
        "deviceFirmwarePath" : "/path/to/firmware/image/name/on/device",
        "codeSigningConfiguration": {
            "signingMethod": "AWS | Custom",
            "signerHashingAlgorithm": "SHA1 | SHA256",
            "signerSigningAlgorithm": "RSA | ECDSA",
            "signerCertificate": "arn:partition:service:region:account-
id:resource:qualifier | /absolute-path-to/signer-certificate-file",
            "untrustedSignerCertificate": "arn:partition:service:region:account-
id:resourcetype:resource:qualifier",
            "signerCertificateFileName": "signerCertificate-file-name",
            "compileSignerCertificate": true | false,
            // ***********Use signerPlatform if you choose AWS for
 signingMethod**************
            "signerPlatform": "AmazonFreeRTOS-Default | AmazonFreeRTOS-TI-CC3220SF"

            ]
         }
    },
```

```
    **********
    This section is used for PKCS #11 labels of private key, public key, device
 certificate, code verification key, JITP certificate, and root certificate.
    When configuring PKCS11, you set up labels and you must provide the labels of the
 device certificate, public key,
    and private key for the key generation type (EC or RSA) it was created with. If
 your device supports PKCS11 storage of JITP certificate,
    code verification key, and root certificate, set
 'pkcs11JITPCodeVerifyRootCertSupport' to 'Yes' in device.json and provide the
 corresponding labels.
    **********
    "pkcs11LabelConfiguration":{
        "pkcs11LabelDevicePrivateKeyForTLS": "<device-private-key-label>",
        "pkcs11LabelDevicePublicKeyForTLS": "<device-public-key-label>",
        "pkcs11LabelDeviceCertificateForTLS": "<device-certificate-label>",
        "pkcs11LabelPreProvisionedECDevicePrivateKeyForTLS": "<preprovisioned-ec-
device-private-key-label>",
        "pkcs11LabelPreProvisionedECDevicePublicKeyForTLS": "<preprovisioned-ec-device-
public-key-label>",
        "pkcs11LabelPreProvisionedECDeviceCertificateForTLS": "<preprovisioned-ec-
device-certificate-label>",
        "pkcs11LabelPreProvisionedRSADevicePrivateKeyForTLS": "<preprovisioned-rsa-
device-private-key-label>",
        "pkcs11LabelPreProvisionedRSADevicePublicKeyForTLS": "<preprovisioned-rsa-
device-public-key-label>",
        "pkcs11LabelPreProvisionedRSADeviceCertificateForTLS": "<preprovisioned-rsa-
device-certificate-label>",
        "pkcs11LabelCodeVerifyKey": "<code-verification-key-label>",
        "pkcs11LabelJITPCertificate": "<JITP-certificate-label>",
        "pkcs11LabelRootCertificate": "<root-certificate-label>"
     }
  }
```

Im Folgenden werden die in der `userdata.json`-Datei verwendeten Attribute aufgelistet:

`sourcePath`

Der Pfad zum Stammverzeichnis des portierten FreeRTOS-Quellcodes.

`retainModifiedSourceDirectories`

(Optional) Überprüft, ob die modifizierten Quellverzeichnisse beibehalten werden sollen, die beim Erstellen und Flashen zu Debugging-Zwecken verwendet wurden. Wenn auf gesetzt`true`, erhalten die geänderten Quellverzeichnisse den Namen RetainedSrc und befinden sich bei

jedem Testgruppenlauf in den Ergebnisprotokollordnern. Wenn nicht enthalten, ist das Feld standardmäßig auf eingestellt. `false`

`freeRTOSTestParamConfigPath`

Der Pfad zur `test_param_config.h` Datei für die FreeRTOS-Libraries-Integration-Tests Integration. Diese Datei muss die `{{testData.sourcePath}}` Platzhaltervariable verwenden, um sie relativ zum Quellcode-Stammverzeichnis zu machen. AWS IoT Device Tester verwendet die Parameter in dieser Datei, um die Tests zu konfigurieren.

`freeRTOSTestExecutionConfigPath`

Der Pfad zur `test_execution_config.h` Datei für die FreeRTOS-Libraries-Integration-Tests Integration. Diese Datei muss die `{{testData.sourcePath}}` Platzhaltervariable verwenden, um sie relativ zum Repository-Stamm zu machen. AWS IoT Device Tester verwendet diese Datei, um zu kontrollieren, welche Tests ausgeführt werden müssen.

`freeRTOSVersion`

Die Version von FreeRTOS, einschließlich der Patch-Version, die in Ihrer Implementierung verwendet wurde. Informationen zu den FreeRTOS-Versionen, AWS IoT Device Tester die mit FreeRTOS kompatibel sind, finden Sie unter Unterstützte Versionen von for FreeRTOS. AWS IoT Device Tester

`buildTool`

Der Befehl zum Erstellen Ihres Quellcodes. Alle Verweise auf den Quellcodepfad im Build-Befehl müssen durch die AWS IoT Device Tester Variable ersetzt werden`{{testData.sourcePath}}`. Verwenden Sie den `{{config.idtRootPath}}` Platzhalter, um relativ zum AWS IoT Device Tester Stammpfad auf ein Build-Skript zu verweisen.

`flashTool`

Der Befehl, um ein Bild auf Ihr Gerät zu flashen. Alle Verweise auf den Quellcodepfad im Flash-Befehl müssen durch die AWS IoT Device Tester Variable `{{testData.sourcePath}}` ersetzt werden. Verwenden Sie den `{{config.idtRootPath}}` Platzhalter, um auf ein Flash-Skript relativ zum AWS IoT Device Tester Stammpfad zu verweisen.

> **Note**
>
> Die neue Struktur der Integrationstests mit FRQ 2.0 erfordert keine Pfadvariablen wie `{{enableTests}}` und. `{{buildImageName}}` Die OTA-End-to-End-Tests werden mit den im FreeRTOS-Libraries-Integration-Tests GitHub Repository bereitgestellten

Konfigurationsvorlagen ausgeführt. Wenn die Dateien im GitHub Repository in Ihrem übergeordneten Quellprojekt vorhanden sind, wird der Quellcode zwischen den Tests nicht geändert. Wenn ein anderes Build-Image für OTA End to End benötigt wird, müssen Sie dieses Image im Build-Skript erstellen und es in der unter angegebenen `userdata.json` Datei angeben`otaConfiguration`.

`testStartDelayms`

Gibt an, wie viele Millisekunden der FreeRTOS-Testläufer wartet, bevor er mit der Ausführung von Tests beginnt. Dies kann nützlich sein, wenn das zu testende Gerät aufgrund von Netzwerk- oder anderen Latenzproblemen wichtige Testinformationen ausgibt, bevor IDT die Möglichkeit hat, eine Verbindung herzustellen und mit der Protokollierung zu beginnen. Dieser Wert gilt nur für FreeRTOS-Testgruppen und nicht für andere Testgruppen, die den FreeRTOS-Testrunner nicht verwenden, wie z. B. die OTA-Tests. Wenn Sie einen Fehler im Zusammenhang mit den erwarteten 10, aber 5 erhalten haben, sollte dieses Feld auf 5000 gesetzt werden.

`echoServerConfiguration`

Die Konfiguration zur Einrichtung des Echoservers für den TLS-Test. Dies ist ein Pflichtfeld.

`keyGenerationMethod`

Der Echoserver ist mit dieser Option konfiguriert. Die Optionen sind EC oder RSA.

`serverPort`

Die Portnummer, auf der der Echo-Server läuft.

`otaConfiguration`

Die Konfiguration für OTA PAL- und OTA E2E-Tests. Dies ist ein Pflichtfeld.

`otaE2EFirmwarePath`

Pfad zum OTA-Bin-Image, das IDT für die OTA-End-to-End-Tests verwendet.

`otaPALCertificatePath`

Der Pfad zum Zertifikat für den OTA-PAL-Test auf dem Gerät. Dies wird verwendet, um die Signatur zu überprüfen. Zum Beispiel ecdsa-sha256-signer.crt.pem.

`deviceFirmwarePath`

Der Pfad zum fest codierten Namen für das Firmware-Image, das gestartet werden soll. Wenn Ihr Gerät das Dateisystem NICHT für den Firmware-Start verwendet, geben Sie dieses Feld

als an `'NA'`. Wenn Ihr Gerät das Dateisystem für den Firmware-Start verwendet, geben Sie den Pfad oder den Namen des Firmware-Startabbilds an.

`codeSigningConfiguration`

`signingMethod`

Die Code-Signaturmethode. Mögliche Werte sind AWS oder Benutzerdefiniert.

> **Note**
>
> Verwenden Sie für die Regionen Peking und Ningxia die Option Benutzerdefiniert. AWS Codesignatur wird in dieser Region nicht unterstützt.

`signerHashingAlgorithm`

Der auf dem Gerät unterstützte Hashing-Algorithmus. Die möglichen Wert sind `SHA1` oder `SHA256`.

`signerSigningAlgorithm`

Der auf dem Gerät unterstützte Signaturalgorithmus. Die möglichen Wert sind `RSA` oder `ECDSA`.

`signerCertificate`

Das für OTA verwendete vertrauenswürdige Zertifikat. Verwenden Sie für die AWS Codesignaturmethode den Amazon-Ressourcennamen (ARN) für das vertrauenswürdige Zertifikat, das in den AWS Certificate Manager hochgeladen wurde. Verwenden Sie für die benutzerdefinierte Codesignaturmethode den absoluten Pfad zur Zertifikatsdatei des Unterzeichners. Informationen zum Erstellen eines vertrauenswürdigen Zertifikats finden Sie unter Erstellen eines Codesignaturzertifikats.

`untrustedSignerCertificate`

Der ARN oder Dateipfad für ein zweites Zertifikat, das in einigen OTA-Tests als nicht vertrauenswürdiges Zertifikat verwendet wird. Informationen zum Erstellen eines Zertifikats finden Sie unter Erstellen eines Codesignaturzertifikats.

`signerCertificateFileName`

Der Dateiname des Codesignaturzertifikats auf dem Gerät. Dieser Wert muss mit dem Dateinamen übereinstimmen, den Sie bei der Ausführung des `aws acm import-certificate` Befehls angegeben haben.

`compileSignerCertificate`

Boolescher Wert, der den Status des Signaturverifizierungszertifikats bestimmt. Gültige Werte sind `true` und `false`.

Setzen Sie diesen Wert auf true, wenn das Zertifikat zur Überprüfung der Signatur des Codesigners nicht bereitgestellt oder geflasht wurde. Es muss in das Projekt kompiliert werden. AWS IoT Device Tester ruft das vertrauenswürdige Zertifikat ab und kompiliert es in. `aws_codesigner_certificate.h`

`signerPlatform`

Der Signier- und Hash-Algorithmus, den AWS Code Signer bei der Erstellung des OTA-Aktualisierungsjobs verwendet. Derzeit lauten die möglichen Werte für dieses Feld `AmazonFreeRTOS-TI-CC3220SF` und `AmazonFreeRTOS-Default`.

- Wählen Sie bei `SHA1` und `RSA` `AmazonFreeRTOS-TI-CC3220SF` aus.
- Wählen Sie bei `SHA256` und `ECDSA` `AmazonFreeRTOS-Default` aus.
- Wenn Sie `SHA256` | `RSA` oder `SHA1` | `ECDSA` für Ihre Konfiguration benötigen, kontaktieren Sie uns, um weitere Unterstützung zu erhalten.
- Konfigurieren Sie `signCommand`, wenn Sie `Custom` für `signingMethod` ausgewählt haben.

`signCommand`

Die beiden Platzhalter „`{{inputImageFilePath}}`“ und „`{{outputSignatureFilePath}}`“ sind im Befehl erforderlich. `{{inputImageFilePath}}` ist der Dateipfad des von IDT erstellten Images, das signiert werden soll. `{{outputSignatureFilePath}}` ist der Dateipfad der Signatur, der vom Skript generiert wird.

`pkcs11LabelConfiguration`

PKCS11 Für die Labelkonfiguration ist mindestens ein Satz von Bezeichnungen aus Gerätezertifikatslabel, Etikett für öffentlichen Schlüssel und Bezeichnung für private Schlüssel erforderlich, um die PKCS11 Testgruppen ausführen zu können. Die erforderlichen PKCS11 Labels basieren auf Ihrer Gerätekonfiguration in der `device.json` Datei. Wenn „pre-provisioned“ auf „Ja“ gesetzt ist`device.json`, muss es sich bei den erforderlichen Bezeichnungen um eine der folgenden Kategorien handeln, je nachdem, was für die PKCS11 Funktion ausgewählt wurde.

- `PreProvisionedEC`
- `PreProvisionedRSA`

Wenn pre-provisioned auf Nein in gesetzt ist`device.json`, lauten die erforderlichen Labels:

- `pkcs11LabelDevicePrivateKeyForTLS`
- `pkcs11LabelDevicePublicKeyForTLS`
- `pkcs11LabelDeviceCertificateForTLS`

Die folgenden drei Bezeichnungen sind nur erforderlich, wenn Sie `pkcs11JITPCodeVerifyRootCertSupport` in Ihrer **`device.json`**Datei Ja für auswählen.

- `pkcs11LabelCodeVerifyKey`
- `pkcs11LabelRootCertificate`
- `pkcs11LabelJITPCertificate`

Die Werte für diese Felder sollten den Werten entsprechen, die im FreeRTOS Porting Guide definiert sind.

`pkcs11LabelDevicePrivateKeyForTLS`

(Optional) Dieses Label wird für das PKCS #11 -Label des privaten Schlüssels verwendet. Bei Geräten mit integrierter Schlüsselbereitstellung und Importunterstützung wird dieses Label zu Testzwecken verwendet. Diese Bezeichnung kann sich von der Bezeichnung unterscheiden, die für den vorab bereitgestellten Fall definiert wurde. Wenn Sie die Schlüsselbereitstellung auf Nein und die Vorbereitstellung auf Ja eingestellt haben, ist dies undefiniert`device.json`.

`pkcs11LabelDevicePublicKeyForTLS`

(Optional) Dieses Label wird für das PKCS #11 -Label des öffentlichen Schlüssels verwendet. Bei Geräten mit integrierter Schlüsselbereitstellung und Importunterstützung wird dieses Label zu Testzwecken verwendet. Diese Bezeichnung kann sich von der Bezeichnung unterscheiden, die für den vorab bereitgestellten Fall definiert wurde. Wenn Sie die Schlüsselbereitstellung auf Nein und die Vorbereitstellung auf Ja eingestellt haben, ist dies undefiniert`device.json`.

`pkcs11LabelDeviceCertificateForTLS`

(Optional) Diese Bezeichnung wird für das PKCS #11 -Label des Gerätezertifikats verwendet. Bei Geräten mit integrierter Schlüsselbereitstellung und Importunterstützung wird dieses Etikett zu Testzwecken verwendet. Diese Bezeichnung kann sich von der Bezeichnung unterscheiden, die für den vorab bereitgestellten Fall definiert wurde. Wenn Sie die Schlüsselbereitstellung auf Nein und die Vorbereitstellung auf Ja eingestellt haben, ist dies undefiniert`device.json`.

pkcs11LabelPreProvisionedECDevicePrivateKeyForTLS

(Optional) Dieses Label wird für das PKCS #11 -Label des privaten Schlüssels verwendet. Für Geräte mit sicheren Elementen oder Hardwarebeschränkungen wird dieses Gerät mit einer anderen Bezeichnung versehen, um die AWS IoT Anmeldeinformationen zu schützen. Wenn Ihr Gerät die Vorabbereitstellung mit einem EC-Schlüssel unterstützt, geben Sie dieses Etikett an. Wenn PreProvisioned auf Ja gesetzt ist`device.json`, muss dieses Label oder beides angegeben werden. `pkcs11LabelPreProvisionedRSADevicePrivateKeyForTLS` Dieses Label kann sich von dem unterscheiden, das für Onboard- und Importfälle definiert wurde.

pkcs11LabelPreProvisionedECDevicePublicKeyForTLS

(Optional) Dieses Label wird für das PKCS #11 -Label des öffentlichen Schlüssels verwendet. Für Geräte mit sicheren Elementen oder Hardwarebeschränkungen wird dieses Gerät mit einer anderen Bezeichnung versehen, um die AWS IoT Anmeldeinformationen zu schützen. Wenn Ihr Gerät die Vorabbereitstellung mit einem EC-Schlüssel unterstützt, geben Sie dieses Etikett an. Wenn PreProvisioned auf Ja gesetzt ist`device.json`, muss dieses Label oder beides angegeben werden. `pkcs11LabelPreProvisionedRSADevicePublicKeyForTLS` Dieses Label kann sich von dem unterscheiden, das für Onboard- und Importfälle definiert wurde.

pkcs11LabelPreProvisionedECDeviceCertificateForTLS

(Optional) Dieses Label wird für das PKCS #11 -Label des Gerätezertifikats verwendet. Für Geräte mit sicheren Elementen oder Hardwarebeschränkungen wird dieses Gerät mit einer anderen Bezeichnung versehen, um die AWS IoT Anmeldeinformationen zu schützen. Wenn Ihr Gerät die Vorabbereitstellung mit einem EC-Schlüssel unterstützt, geben Sie dieses Etikett an. Wenn PreProvisioned auf Ja gesetzt ist`device.json`, muss dieses Label oder beides angegeben werden. `pkcs11LabelPreProvisionedRSADeviceCertificateForTLS` Dieses Label kann sich von dem unterscheiden, das für Onboard- und Importfälle definiert wurde.

pkcs11LabelPreProvisionedRSADevicePrivateKeyForTLS

(Optional) Dieses Label wird für das PKCS #11 -Label des privaten Schlüssels verwendet. Für Geräte mit sicheren Elementen oder Hardwarebeschränkungen wird dieses Gerät mit einer anderen Bezeichnung versehen, um die AWS IoT Anmeldeinformationen zu schützen. Wenn Ihr Gerät die Vorabbereitstellung mit einem RSA-Schlüssel unterstützt, geben Sie diese Bezeichnung an. Wenn PreProvisioned auf Ja gesetzt ist`device.json`, muss dieses Label oder beide angegeben werden. `pkcs11LabelPreProvisionedECDevicePrivateKeyForTLS`

`pkcs11LabelPreProvisionedRSADevicePublicKeyForTLS`

(Optional) Dieses Label wird für das PKCS #11 -Label des öffentlichen Schlüssels verwendet. Für Geräte mit sicheren Elementen oder Hardwarebeschränkungen wird dieses Gerät mit einer anderen Bezeichnung versehen, um die AWS IoT Anmeldeinformationen zu schützen. Wenn Ihr Gerät die Vorabbereitstellung mit einem RSA-Schlüssel unterstützt, geben Sie diese Bezeichnung an. Wenn PreProvisioned auf Ja gesetzt ist`device.json`, muss dieses Label oder beide angegeben werden. `pkcs11LabelPreProvisionedECDevicePublicKeyForTLS`

`pkcs11LabelPreProvisionedRSADeviceCertificateForTLS`

(Optional) Diese Bezeichnung wird für das PKCS #11 -Label des Gerätezertifikats verwendet. Für Geräte mit sicheren Elementen oder Hardwarebeschränkungen wird dieses Gerät mit einer anderen Bezeichnung versehen, um die AWS IoT Anmeldeinformationen zu schützen. Wenn Ihr Gerät die Vorabbereitstellung mit einem RSA-Schlüssel unterstützt, geben Sie diese Bezeichnung an. Wenn PreProvisioned auf Ja gesetzt ist`device.json`, muss dieses Label oder beide angegeben werden. `pkcs11LabelPreProvisionedECDeviceCertificateForTLS`

`pkcs11LabelCodeVerifyKey`

(Optional) Diese Bezeichnung wird für das PKCS #11 -Label des Codeverifizierungsschlüssels verwendet. Wenn Ihr Gerät über PKCS #11 -Speicherunterstützung für das JITP-Zertifikat, den Codeverifizierungsschlüssel und das Stammzertifikat verfügt, geben Sie dieses Etikett an. Wenn `pkcs11JITPCodeVerifyRootCertSupport` in auf Ja gesetzt `device.json` ist, muss dieses Etikett angegeben werden.

`pkcs11LabelJITPCertificate`

(Optional) Diese Bezeichnung wird für das PKCS #11 -Label des JITP-Zertifikats verwendet. Wenn Ihr Gerät über PKCS #11 -Speicherunterstützung für das JITP-Zertifikat, den Codeverifizierungsschlüssel und das Stammzertifikat verfügt, geben Sie dieses Etikett an. Wenn `pkcs11JITPCodeVerifyRootCertSupport` in auf Ja gesetzt `device.json` ist, muss dieses Etikett angegeben werden.

IDT für FreeRTOS-Variablen

Die Befehle zum Erstellen Ihres Codes und zum Flashen des Geräts erfordern möglicherweise Konnektivität oder andere Informationen zu Ihren Geräten, um erfolgreich ausgeführt zu werden. AWS IoT Device Tester ermöglicht es Ihnen, Geräteinformationen in Flash zu referenzieren

und Befehle zu erstellen mit JsonPath. Mithilfe einfacher JsonPath Ausdrücke können Sie die erforderlichen Informationen abrufen, die in Ihrer `device.json` Datei angegeben sind.

Pfadvariablen

IDT for FreeRTOS definiert die folgenden Pfadvariablen, die in Befehlszeilen und Konfigurationsdateien verwendet werden können:

`{{testData.sourcePath}}`

Wird auf den Quellcodepfad erweitert. Wenn Sie diese Variable verwenden, muss sie sowohl in den Flash- als auch in den Build-Befehlen verwendet werden.

`{{device.connectivity.serialPort}}`

Wird auf die serielle Schnittstelle erweitert.

`{{device.identifiers[?(@.name == 'serialNo')].value[0]}}`

Wird zur Seriennummer Ihres Geräts erweitert.

`{{config.idtRootPath}}`

Erweitert auf den Stammpfad AWS IoT Device Tester .

Benutzeroberfläche für IDT für FreeRTOS Qualification Suite 2.0 (FRQ 2.0)

AWS IoT Device Tester for FreeRTOS (IDT for FreeRTOS) umfasst eine webbasierte Benutzeroberfläche (UI), über die Sie mit der IDT-Befehlszeilenanwendung und den zugehörigen Konfigurationsdateien interagieren können. Sie verwenden die Benutzeroberfläche von IDT for FreeRTOS, um eine neue Konfiguration für Ihr Gerät zu erstellen oder eine bestehende zu ändern. Sie können die Benutzeroberfläche auch verwenden, um die IDT-Anwendung aufzurufen und die FreeRTOS-Tests auf Ihrem Gerät auszuführen.

Hinweise zur Verwendung der Befehlszeile zum Ausführen von Qualifikationstests finden Sie unter. Erster Test Ihres Mikrocontroller-Boards

In diesem Abschnitt werden die Voraussetzungen für die Benutzeroberfläche von IDT for FreeRTOS und die Ausführung von Qualifikationstests über die Benutzeroberfläche beschrieben.

Themen

- Richten Sie die IDT-Voraussetzungen ein

- AWS Konfigurieren Sie die Anmeldeinformationen für die Verwendung der IDT-Benutzeroberfläche
- Öffnen Sie die Benutzeroberfläche von IDT for FreeRTOS
- Erstellen Sie eine neue Konfiguration
- Ändern Sie eine bestehende Konfiguration
- Führen Sie Qualifizierungstests durch

Richten Sie die IDT-Voraussetzungen ein

Um Tests über die Benutzeroberfläche von AWS IoT Device Tester (IDT) for FreeRTOS durchzuführen, müssen Sie die Voraussetzungen auf der Richten Sie die Voraussetzungen für die LTS-Qualifikation ein Seite für IDT FreeRTOS Qualification (FRQ) 2.x erfüllen.

AWS Konfigurieren Sie die Anmeldeinformationen für die Verwendung der IDT-Benutzeroberfläche

Sie müssen Ihre IAM-Benutzeranmeldedaten für den AWS Benutzer konfigurieren, in Erstelle ein Konto AWS dem Sie sie erstellt haben. Sie können Ihre Anmeldeinformationen auf zwei Arten angeben:

- In einer Anmeldeinformationsdatei
- Als Umgebungsvariablen

Konfigurieren Sie AWS Anmeldeinformationen mit einer Anmeldeinformationsdatei

IDT verwendet die gleiche Anmeldeinformationsdatei wie das AWS CLI. Weitere Informationen finden Sie unter Konfigurations- und Anmeldeinformationsdateien.

Der Speicherort der Anmeldeinformationsdatei hängt vom verwendeten Betriebssystem ab:

- macOS und Linux — `~/.aws/credentials`
- Windows – `C:\Users\UserName\.aws\credentials`

Fügen Sie der `credentials` Datei Ihre AWS Anmeldeinformationen im folgenden Format hinzu:

```
[default]
aws_access_key_id = your_access_key_id
aws_secret_access_key = your_secret_access_key
```

> **Note**
>
> Wenn Sie das Profil nicht verwenden, müssen Sie den `default` AWS Profilnamen in der Benutzeroberfläche von IDT for FreeRTOS angeben. Weitere Informationen zu Profilen finden Sie unter Konfiguration und Einstellungen für Anmeldeinformationsdateien.

Konfigurieren Sie AWS Anmeldeinformationen mit Umgebungsvariablen

Umgebungsvariablen sind Variablen, die vom Betriebssystem gepflegt und von Systembefehlen verwendet werden. Sie werden nicht gespeichert, wenn Sie die SSH-Sitzung schließen.
Die Benutzeroberfläche von IDT for FreeRTOS verwendet die `AWS_SECRET_ACCESS_KEY` Umgebungsvariablen `AWS_ACCESS_KEY_ID` und, um Ihre Anmeldeinformationen zu speichern. AWS

Um diese Variablen auf Linux, macOS oder Unix festzulegen, verwenden Sie export:

```
export AWS_ACCESS_KEY_ID=your_access_key_id
export AWS_SECRET_ACCESS_KEY=your_secret_access_key
```

In Windows können Sie die Variablen mit set festlegen:

```
set AWS_ACCESS_KEY_ID=your_access_key_id
set AWS_SECRET_ACCESS_KEY=your_secret_access_key
```

Öffnen Sie die Benutzeroberfläche von IDT for FreeRTOS

In diesem Thema wird erklärt, wie Sie die Benutzeroberfläche von IDT for FreeRTOS öffnen, um die FreeRTOS Qualification Suite zu verwenden.

Um die Benutzeroberfläche von IDT für FreeRTOS zu öffnen

1. Laden Sie eine unterstützte Version von IDT for FreeRTOS herunter. Extrahieren Sie dann das heruntergeladene Archiv in ein Verzeichnis, für das Sie Lese- und Schreibberechtigungen haben.
2. Navigieren Sie zum Installationsverzeichnis von IDT for FreeRTOS:

   ```
   cd devicetester-extract-location/bin
   ```

3. Führen Sie den folgenden Befehl aus, um die Benutzeroberfläche von IDT for FreeRTOS zu öffnen:

Linux

```
.devicetester_ui_linux_x86-64
```

Windows

```
./devicetester_ui_win_x64-64
```

macOS

```
./devicetester_ui_mac_x86-64
```

> **Note**
>
> Gehen Sie in macOS zu Systemeinstellungen -> Sicherheit und Datenschutz, damit Ihr System die Benutzeroberfläche ausführen kann. Wenn Sie die Tests ausführen, müssen Sie dies möglicherweise noch dreimal durchführen. das

Die Benutzeroberfläche von IDT for FreeRTOS wird in Ihrem Standardbrowser geöffnet. Die neuesten drei Hauptversionen der folgenden Browser unterstützen die Benutzeroberfläche:

- Google Chrome
- Mozilla Firefox
- Microsoft Edge
- Apple Safari für macOS

> **Note**
>
> Für eine bessere Benutzererfahrung empfehlen wir Google Chrome oder Mozilla Firefox, um auf die Benutzeroberfläche von IDT for FreeRTOS zuzugreifen. Microsoft Internet Explorer wird von der Benutzeroberfläche nicht unterstützt.

> **Important**
>
> Sie müssen Ihre AWS Anmeldeinformationen konfigurieren, bevor Sie die Benutzeroberfläche öffnen. Wenn Sie Ihre Anmeldeinformationen nicht konfiguriert haben, schließen Sie das Browserfenster von IDT for FreeRTOS UI, folgen Sie den Schritten unter und öffnen Sie dann die IDT for FreeRTOS UI erneut. AWS Konfigurieren Sie die Anmeldeinformationen für die Verwendung der IDT-Benutzeroberfläche

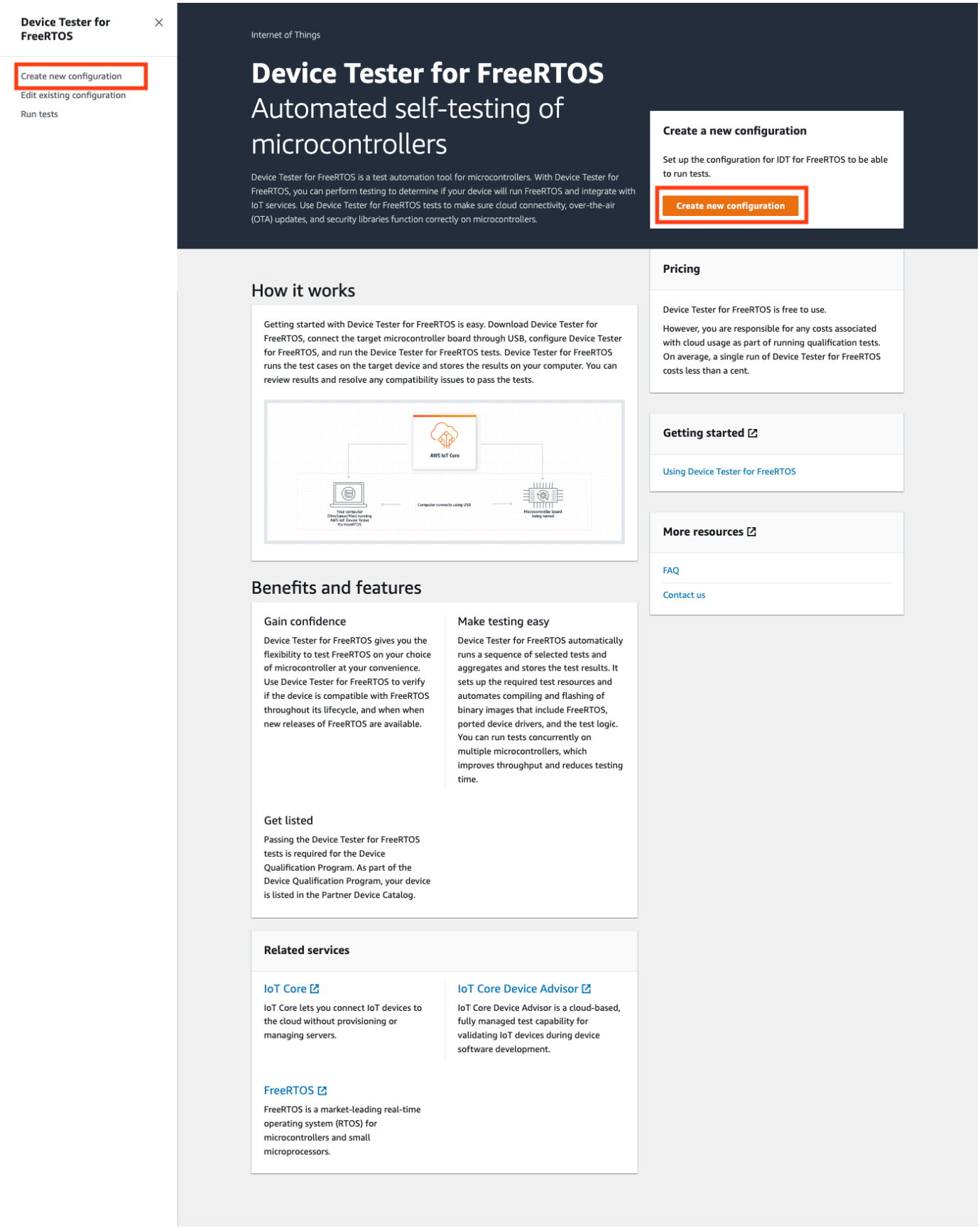
Erstellen Sie eine neue Konfiguration

Wenn Sie zum ersten Mal Benutzer sind, müssen Sie eine neue Konfiguration erstellen, um die JSON-Konfigurationsdateien einzurichten, die IDT for FreeRTOS zum Ausführen von Tests benötigt. Sie können dann Tests ausführen oder die erstellte Konfiguration ändern.

Beispiele für die `userdata.json` Dateien `config.jsondevice.json`, und finden Sie unterErster Test Ihres Mikrocontroller-Boards.

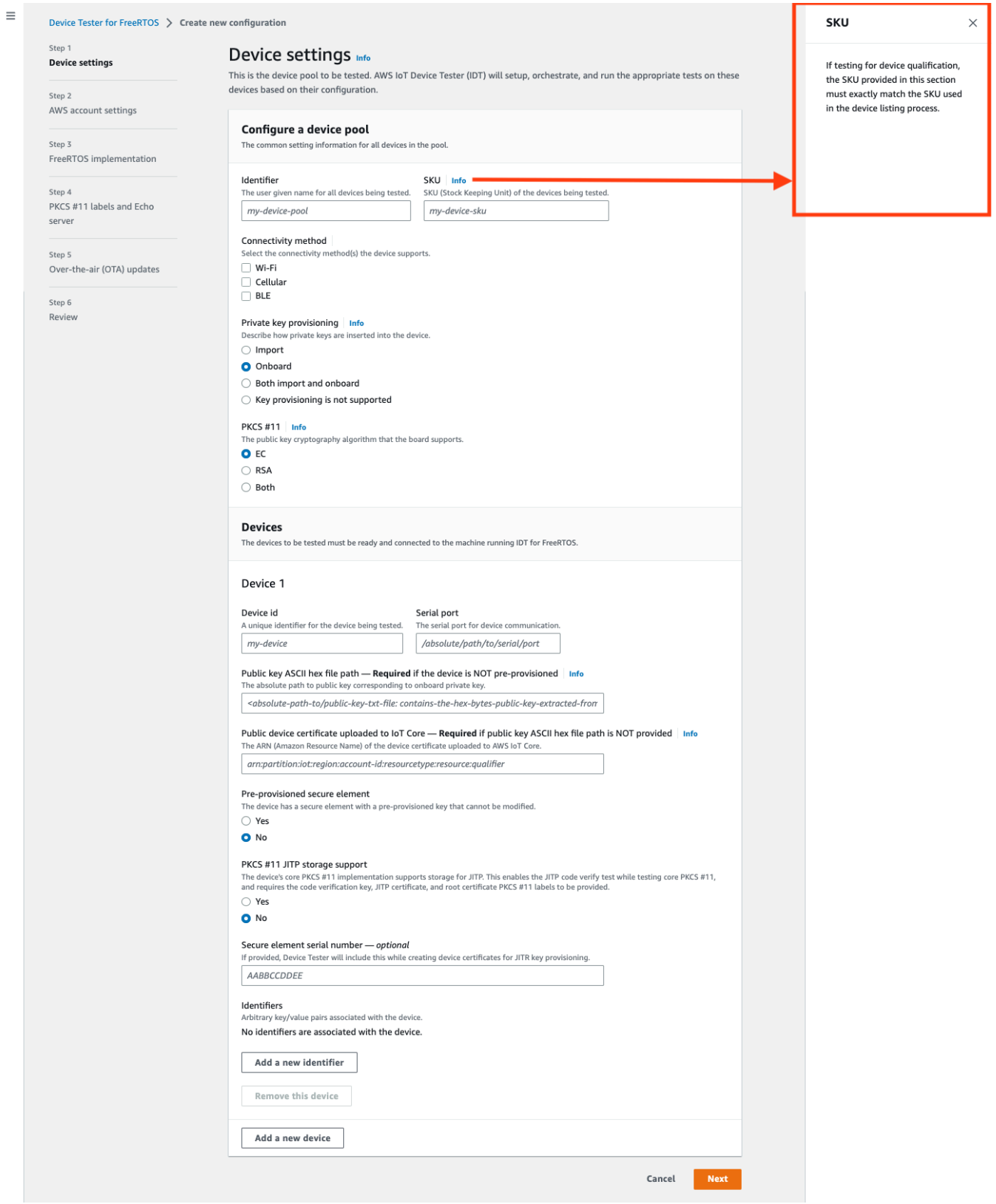
So erstellen Sie eine neue Konfiguration:

1. Öffnen Sie in der Benutzeroberfläche von IDT for FreeRTOS das Navigationsmenü und wählen Sie Neue Konfiguration erstellen aus.

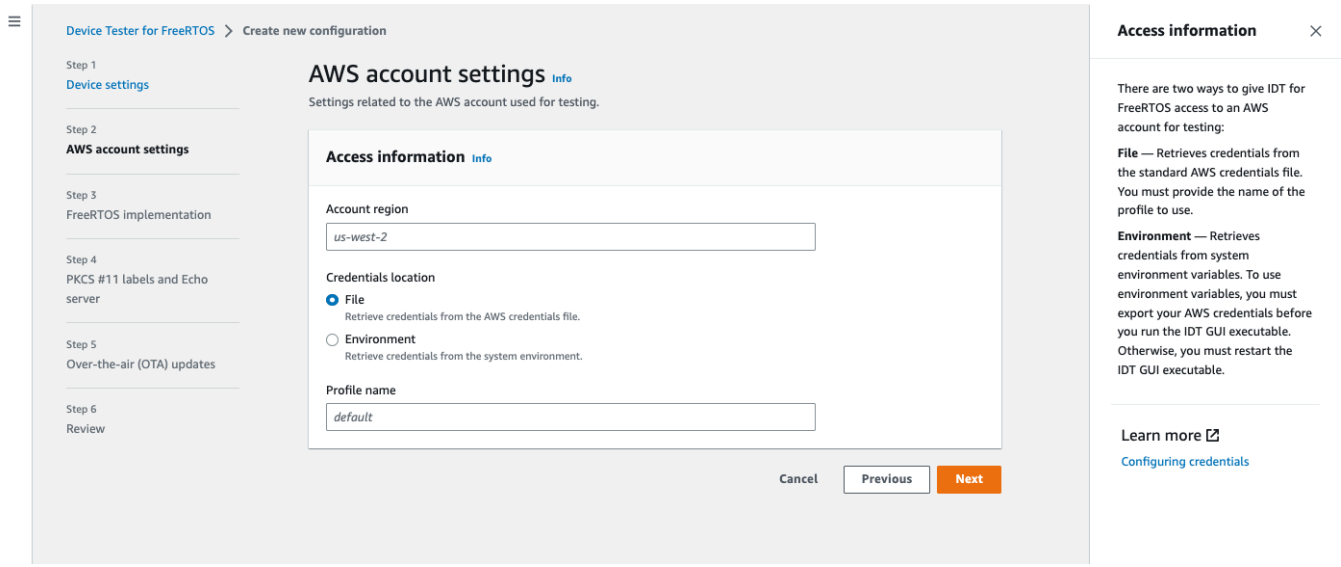

Device Tester for FreeRTOS
Create new configuration
Edit existing configuration
Run tests
Internet of Things
Device Tester for FreeRTOS
Automated self-testing of microcontrollers
Device Tester for FreeRTOS is a test automation tool for microcontrollers. With Device Tester for FreeRTOS, you can perform testing to determine if your device will run FreeRTOS and integrate with IoT services. Use Device Tester for FreeRTOS tests to make sure cloud connectivity, over-the-air (OTA) updates, and security libraries function correctly on microcontrollers.
Create a new configuration
Set up the configuration for IDT for FreeRTOS to be able to run tests.
Create new configuration
How it works
Getting started with Device Tester for FreeRTOS is easy. Download Device Tester for FreeRTOS, connect the target microcontroller board through USB, configure Device Tester for FreeRTOS, and run the Device Tester for FreeRTOS tests. Device Tester for FreeRTOS runs the test cases on the target device and stores the results on your computer. You can review results and resolve any compatibility issues to pass the tests.
AWS IoT Core
Computer connects using USB
Benefits and features
Gain confidence
Device Tester for FreeRTOS gives you the flexibility to test FreeRTOS on your choice of microcontroller at your convenience. Use Device Tester for FreeRTOS to verify if the device is compatible with FreeRTOS throughout its lifecycle, and when when new releases of FreeRTOS are available.
Make testing easy
Device Tester for FreeRTOS automatically runs a sequence of selected tests and aggregates and stores the test results. It sets up the required test resources and automates compiling and flashing of binary images that include FreeRTOS, ported device drivers, and the test logic. You can run tests concurrently on multiple microcontrollers, which improves throughput and reduces testing time.
Get listed
Passing the Device Tester for FreeRTOS tests is required for the Device Qualification Program. As part of the Device Qualification Program, your device is listed in the Partner Device Catalog.
Related services
IoT Core
IoT Core lets you connect IoT devices to the cloud without provisioning or managing servers.
IoT Core Device Advisor
IoT Core Device Advisor is a cloud-based, fully managed test capability for validating IoT devices during device software development.
FreeRTOS
FreeRTOS is a market-leading real-time operating system (RTOS) for microcontrollers and small microprocessors.
Pricing
Device Tester for FreeRTOS is free to use.
However, you are responsible for any costs associated with cloud usage as part of running qualification tests. On average, a single run of Device Tester for FreeRTOS costs less than a cent.
Getting started
Using Device Tester for FreeRTOS
More resources
FAQ
Contact us

2. Folgen Sie dem Konfigurationsassistenten, um die IDT-Konfigurationseinstellungen einzugeben, die für die Durchführung von Qualifikationstests verwendet werden. Der Assistent konfiguriert die folgenden Einstellungen in JSON-Konfigurationsdateien, die sich im *devicetester-extract-location*`/config` Verzeichnis befinden.

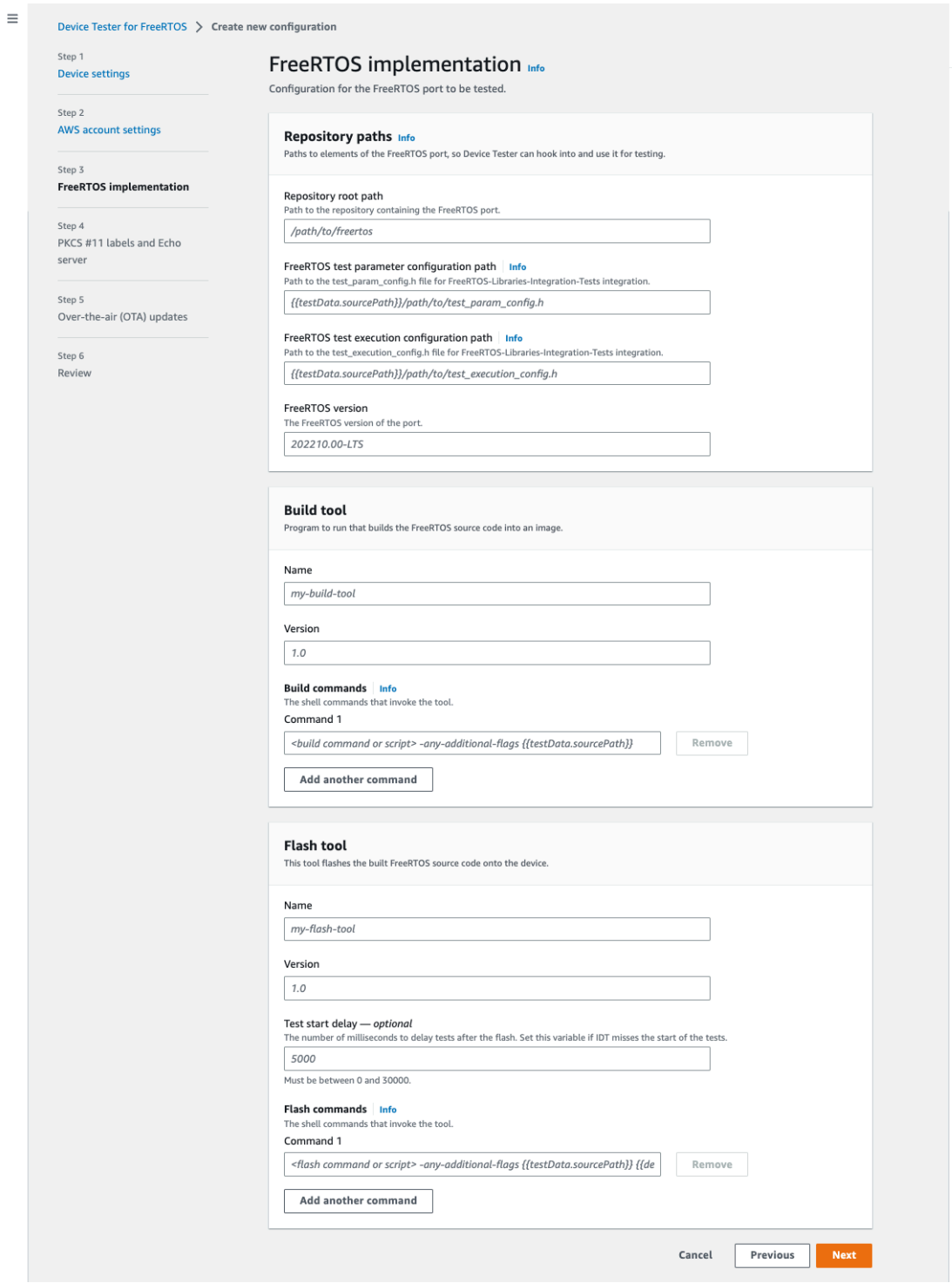
   - Geräteeinstellungen — Die Gerätepool-Einstellungen für die zu testenden Geräte. Diese Einstellungen werden in den `sku` Feldern `id` und konfiguriert, und die Geräte blockieren für den Gerätepool in der `config.json` Datei.


Device Tester for FreeRTOS > Create new configuration
Step 1
Device settings
Step 2
AWS account settings
Step 3
FreeRTOS implementation
Step 4
PKCS #11 labels and Echo server
Step 5
Over-the-air (OTA) updates
Step 6
Review
Device settings Info
This is the device pool to be tested. AWS IoT Device Tester (IDT) will setup, orchestrate, and run the appropriate tests on these devices based on their configuration.
Configure a device pool
The common setting information for all devices in the pool.
Identifier
The user given name for all devices being tested.
my-device-pool
SKU Info
SKU (Stock Keeping Unit) of the devices being tested.
my-device-sku
Connectivity method
Select the connectivity method(s) the device supports.
Wi-Fi
Cellular
BLE
Private key provisioning Info
Describe how private keys are inserted into the device.
Import
Onboard
Both import and onboard
Key provisioning is not supported
PKCS #11 Info
The public key cryptography algorithm that the board supports.
EC
RSA
Both
Devices
The devices to be tested must be ready and connected to the machine running IDT for FreeRTOS.
Device 1
Device id
A unique identifier for the device being tested.
my-device
Serial port
The serial port for device communication.
/absolute/path/to/serial/port
Public key ASCII hex file path — Required if the device is NOT pre-provisioned Info
The absolute path to public key corresponding to onboard private key.
<absolute-path-to/public-key-txt-file: contains-the-hex-bytes-public-key-extracted-from
Public device certificate uploaded to IoT Core — Required if public key ASCII hex file path is NOT provided Info
The ARN (Amazon Resource Name) of the device certificate uploaded to AWS IoT Core.
arn:partition:iot:region:account-id:resourcetype:resource:qualifier
Pre-provisioned secure element
The device has a secure element with a pre-provisioned key that cannot be modified.
Yes
No
PKCS #11 JITP storage support
The device's core PKCS #11 implementation supports storage for JITP. This enables the JITP code verify test while testing core PKCS #11, and requires the code verification key, JITP certificate, and root certificate PKCS #11 labels to be provided.
Yes
No
Secure element serial number — optional
If provided, Device Tester will include this while creating device certificates for JITR key provisioning.
AABBCCDDEE
Identifiers
Arbitrary key/value pairs associated with the device.
No identifiers are associated with the device.
Add a new identifier
Remove this device
Add a new device
Cancel
Next
SKU
If testing for device qualification, the SKU provided in this section must exactly match the SKU used in the device listing process.

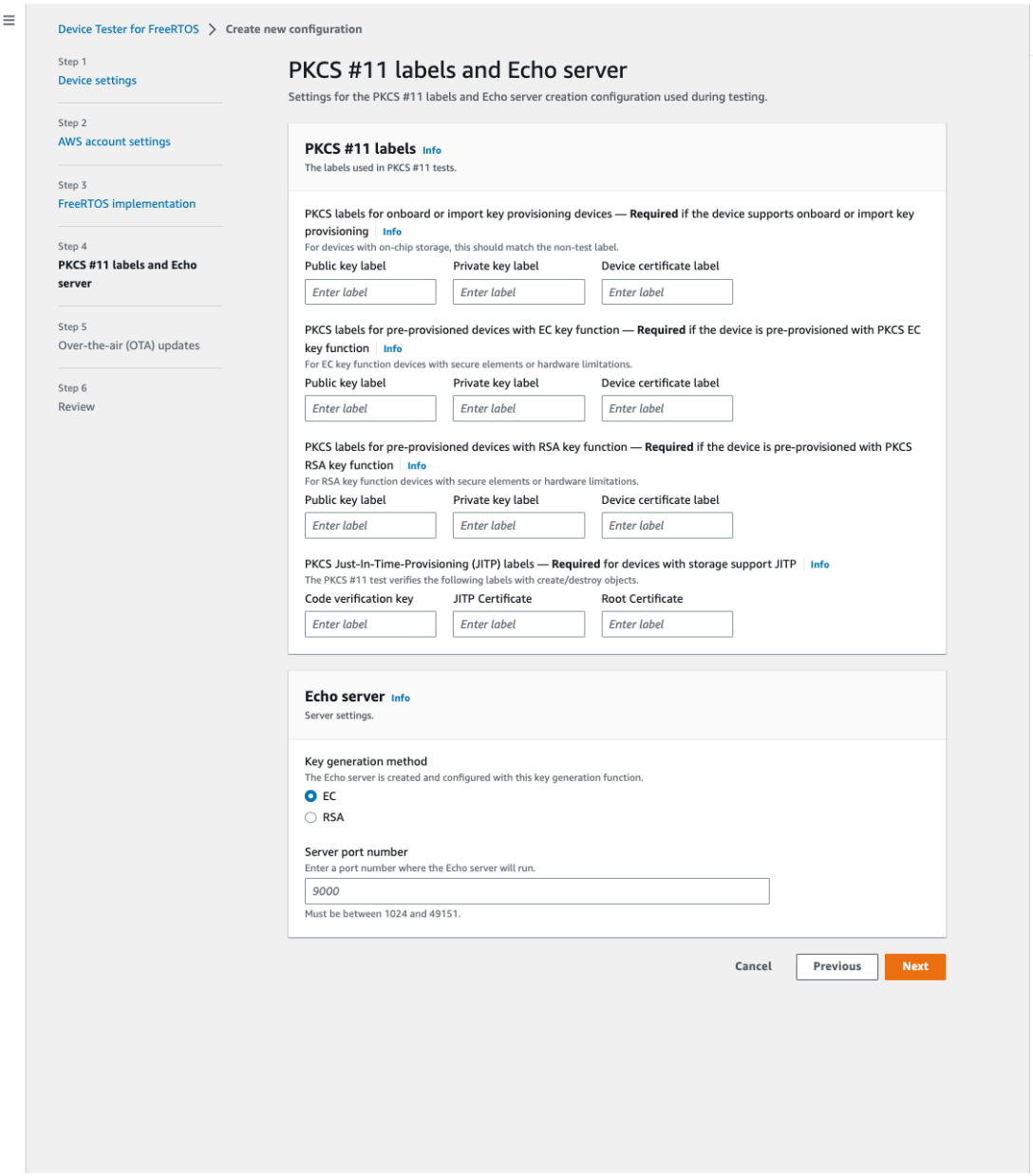
- AWS Kontoeinstellungen — Die AWS-Konto Informationen, die IDT for FreeRTOS verwendet, um AWS Ressourcen während Testläufen zu erstellen. Diese Einstellungen sind in der Datei konfiguriert. `config.json`



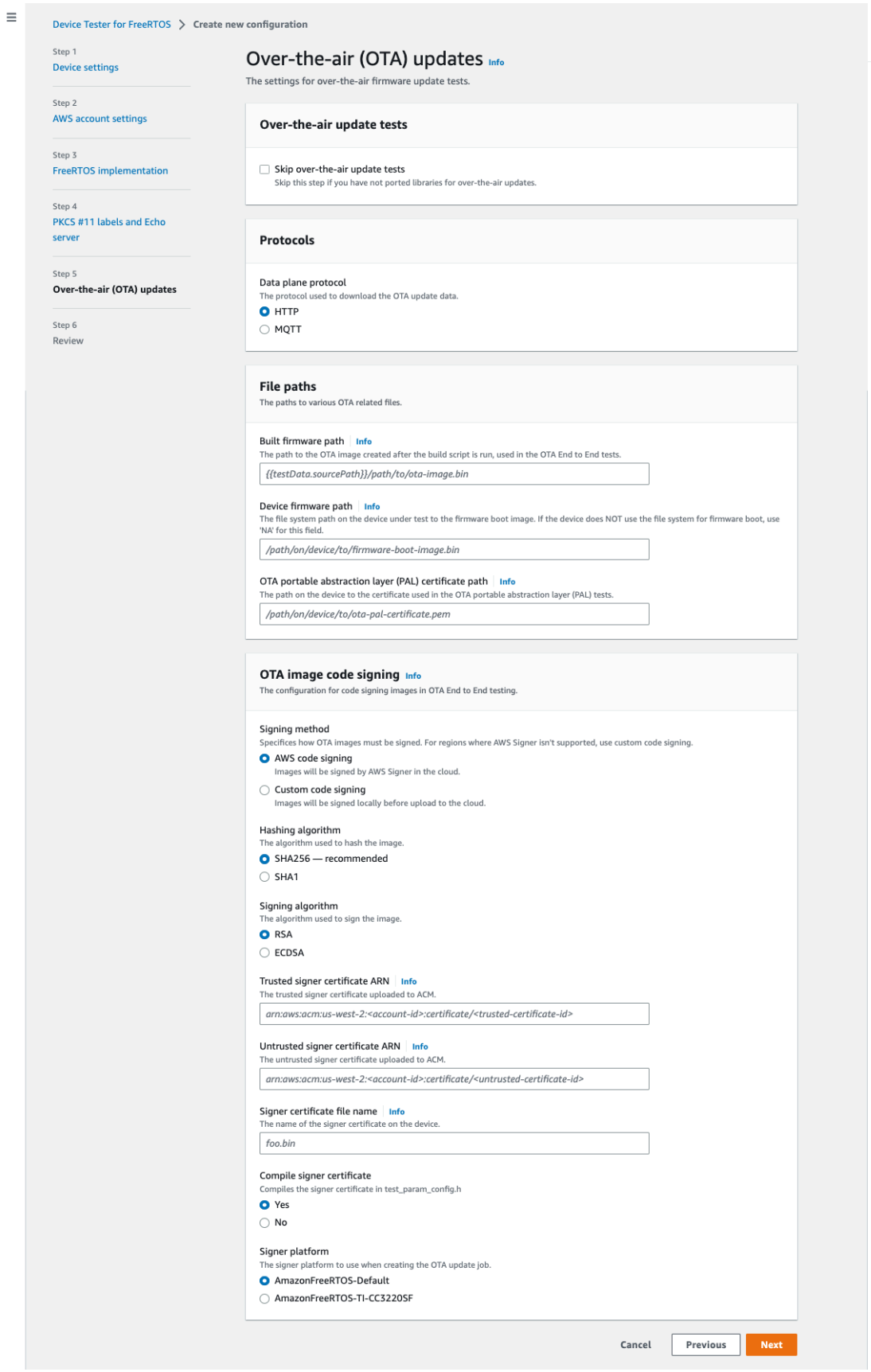
- FreeRTOS-Implementierung — Der absolute Pfad zum FreeRTOS-Repository und zum portierten Code sowie die FreeRTOS-Version, auf der Sie IDT FRQ ausführen möchten. Die Pfade zu den Ausführungs- und Parameter-Config-Header-Dateien aus dem Repository. `FreeRTOS-Libraries-Integration-Tests` GitHub Die Build- und Flash-Befehle für Ihre Hardware, mit denen IDT automatisch Tests auf Ihrem Board erstellen und flashen kann. Diese Einstellungen sind in der `userdata.json` Datei konfiguriert.



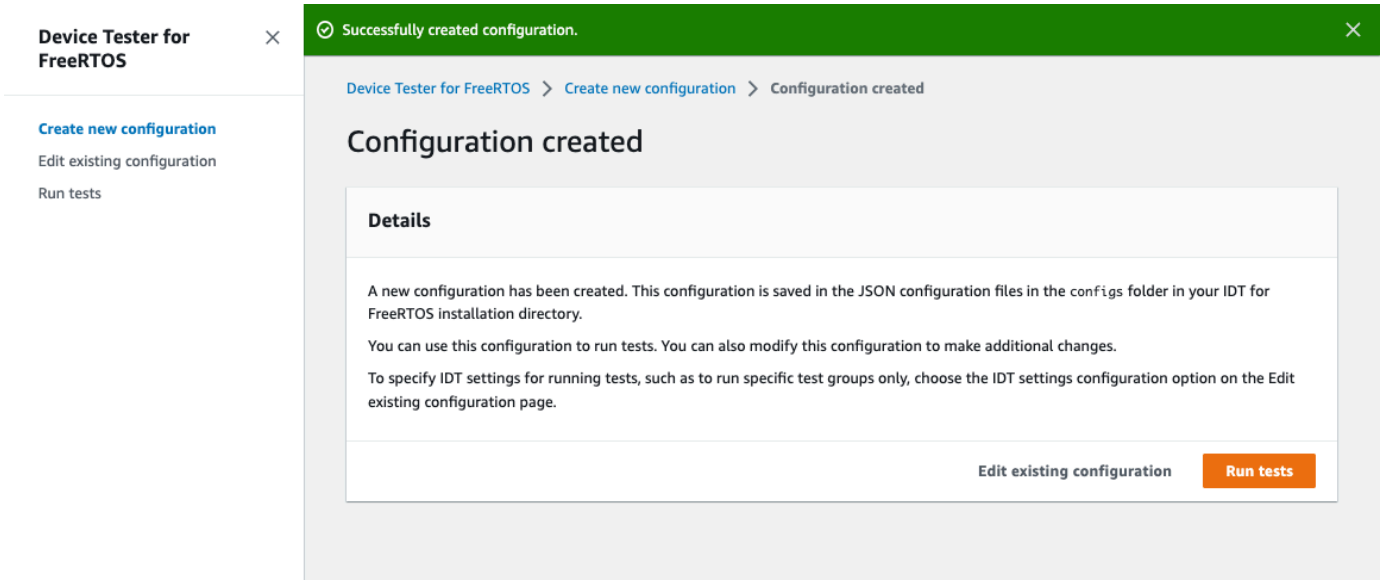
- PKCS #11 -Labels und Echo Server — Die PKCS #11 -Labels, die den auf Ihrer Hardware bereitgestellten Schlüsseln entsprechen, basierend auf der Schlüsselfunktionalität und der Methode zur Schlüsselbereitstellung. Die Konfigurationseinstellungen des Echoservers für die Transport Interface-Tests. Diese Einstellungen werden in den `device.json` Dateien `userdata.json` und konfiguriert.



- Over-the-air (OTA) -Updates — Die Einstellungen, die die OTA-Funktionstests steuern. Diese Einstellungen werden im `features` Block der `userdata.json` Dateien `device.json` und konfiguriert.


Device Tester for FreeRTOS > Create new configuration
Step 1
Device settings
Step 2
AWS account settings
Step 3
FreeRTOS implementation
Step 4
PKCS #11 labels and Echo server
Step 5
Over-the-air (OTA) updates
Step 6
Review
Over-the-air (OTA) updates Info
The settings for over-the-air firmware update tests.
Over-the-air update tests
Skip over-the-air update tests
Skip this step if you have not ported libraries for over-the-air updates.
Protocols
Data plane protocol
The protocol used to download the OTA update data.
HTTP
MQTT
File paths
The paths to various OTA related files.
Built firmware path Info
The path to the OTA image created after the build script is run, used in the OTA End to End tests.
{{testData.sourcePath}}/path/to/ota-image.bin
Device firmware path Info
The file system path on the device under test to the firmware boot image. If the device does NOT use the file system for firmware boot, use 'NA' for this field.
/path/on/device/to/firmware-boot-image.bin
OTA portable abstraction layer (PAL) certificate path Info
The path on the device to the certificate used in the OTA portable abstraction layer (PAL) tests.
/path/on/device/to/ota-pal-certificate.pem
OTA image code signing Info
The configuration for code signing images in OTA End to End testing.
Signing method
Specifices how OTA images must be signed. For regions where AWS Signer isn't supported, use custom code signing.
AWS code signing
Images will be signed by AWS Signer in the cloud.
Custom code signing
Images will be signed locally before upload to the cloud.
Hashing algorithm
The algorithm used to hash the image.
SHA256 — recommended
SHA1
Signing algorithm
The algorithm used to sign the image.
RSA
ECDSA
Trusted signer certificate ARN Info
The trusted signer certificate uploaded to ACM.
arn:aws:acm:us-west-2:<account-id>:certificate/<trusted-certificate-id>
Untrusted signer certificate ARN Info
The untrusted signer certificate uploaded to ACM.
arn:aws:acm:us-west-2:<account-id>:certificate/<untrusted-certificate-id>
Signer certificate file name Info
The name of the signer certificate on the device.
foo.bin
Compile signer certificate
Compiles the signer certificate in test_param_config.h
Yes
No
Signer platform
The signer platform to use when creating the OTA update job.
AmazonFreeRTOS-Default
AmazonFreeRTOS-TI-CC3220SF
Cancel
Previous
Next
Over-the-air (OTA) updates
IDT for FreeRTOS runs tests to verify OTA update behavior, including end-to-end (E2E) and portable abstraction layer (PAL) tests.
These tests are required to qualify a device.
Learn more
FreeRTOS OTA Update tests

3. Überprüfen Sie auf der Überprüfungsseite Ihre Konfigurationsinformationen.



Wenn Sie mit der Überprüfung Ihrer Konfiguration fertig sind, wählen Sie Tests ausführen, um Ihre Qualifizierungstests durchzuführen.

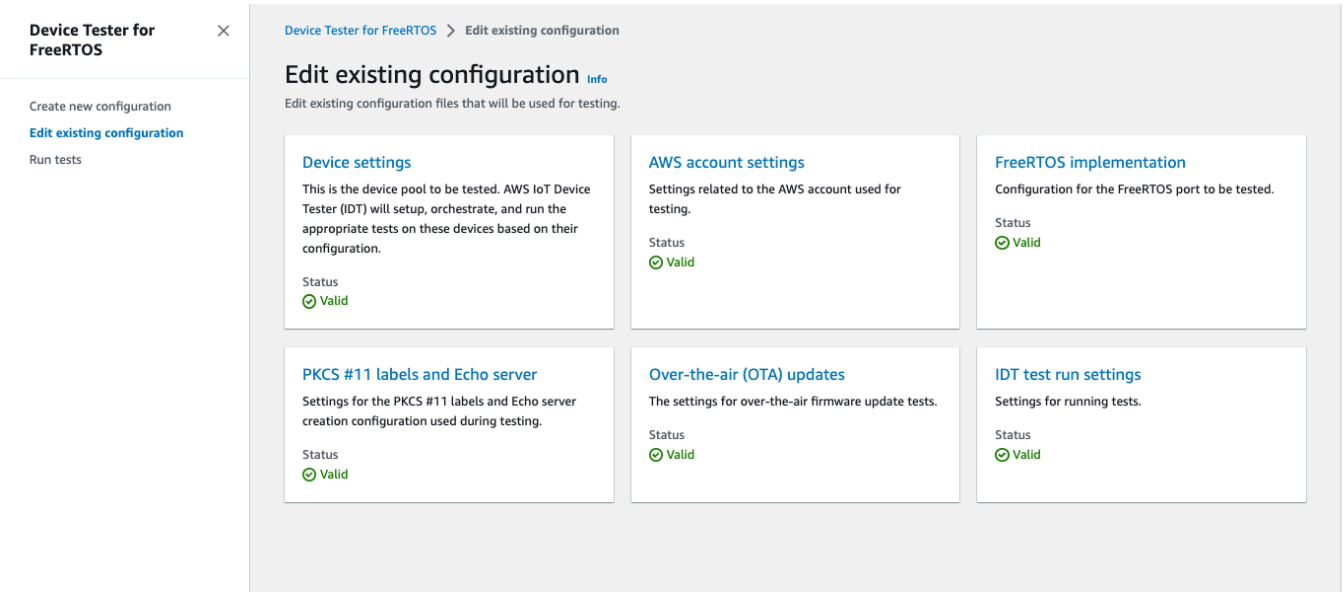
Ändern Sie eine bestehende Konfiguration

Wenn Sie bereits Konfigurationsdateien für IDT for FreeRTOS eingerichtet haben, können Sie die Benutzeroberfläche von IDT for FreeRTOS verwenden, um Ihre bestehende Konfiguration zu ändern. Die vorhandenen Konfigurationsdateien müssen sich im Verzeichnis befinden. *devicetester-extract-location*`/config`

Um eine Konfiguration zu ändern

1. Öffnen Sie in der Benutzeroberfläche von IDT for FreeRTOS das Navigationsmenü und wählen Sie Bestehende Konfiguration bearbeiten aus.

   Das Konfigurations-Dashboard zeigt Informationen zu Ihren vorhandenen Konfigurationseinstellungen an. Wenn eine Konfiguration falsch oder nicht verfügbar ist, lautet der Status für diese Konfiguration`Error validating configuration`.

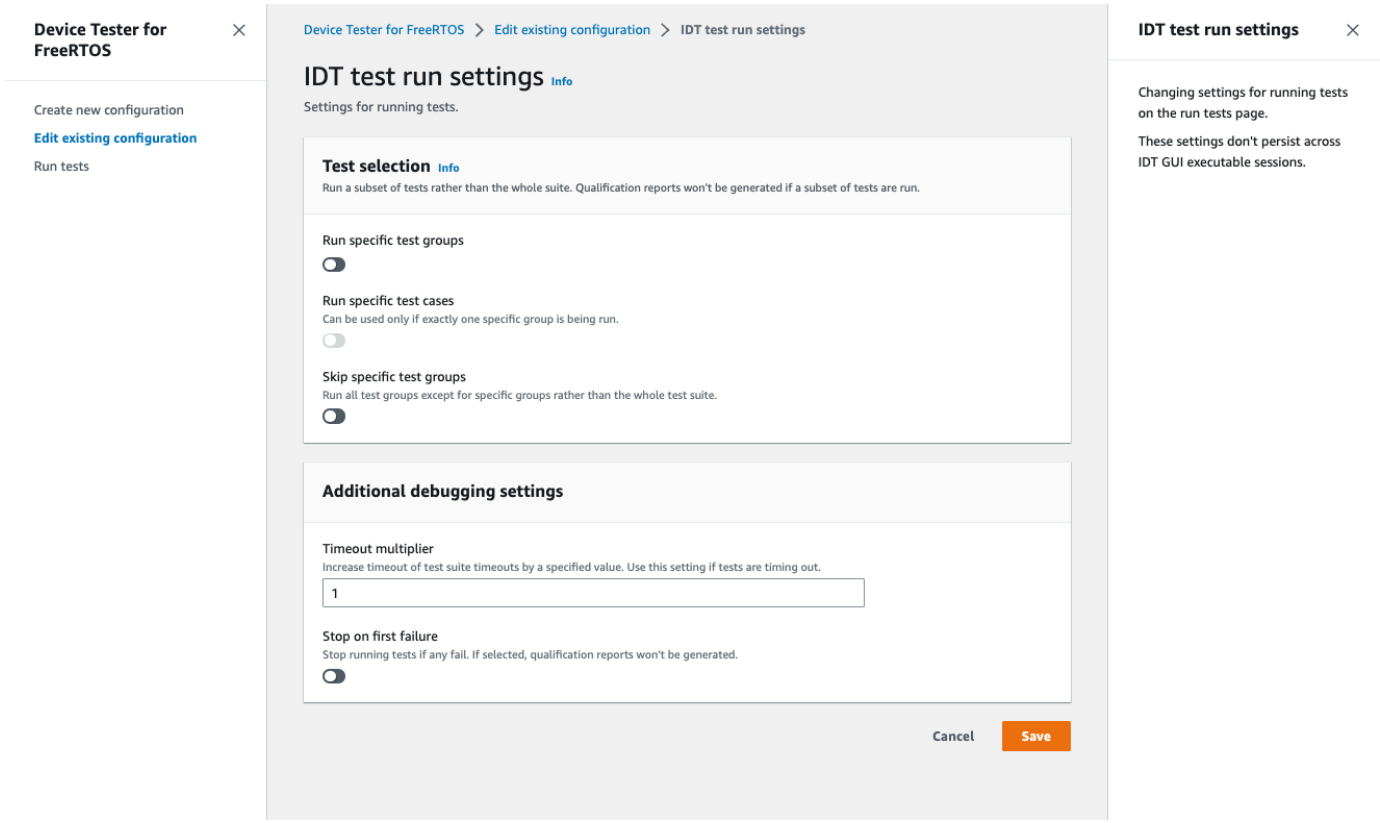


2. Gehen Sie wie folgt vor, um eine bestehende Konfigurationseinstellung zu ändern:

   a. Wählen Sie den Namen einer Konfigurationseinstellung, um die zugehörige Einstellungsseite zu öffnen.

   b. Ändern Sie die Einstellungen und wählen Sie dann Speichern, um die entsprechende Konfigurationsdatei neu zu generieren.

3. Um die IDT for FreeRTOS-Testlaufeinstellungen zu ändern, wählen Sie in der Bearbeitungsansicht IDT-Testlaufeinstellungen aus:



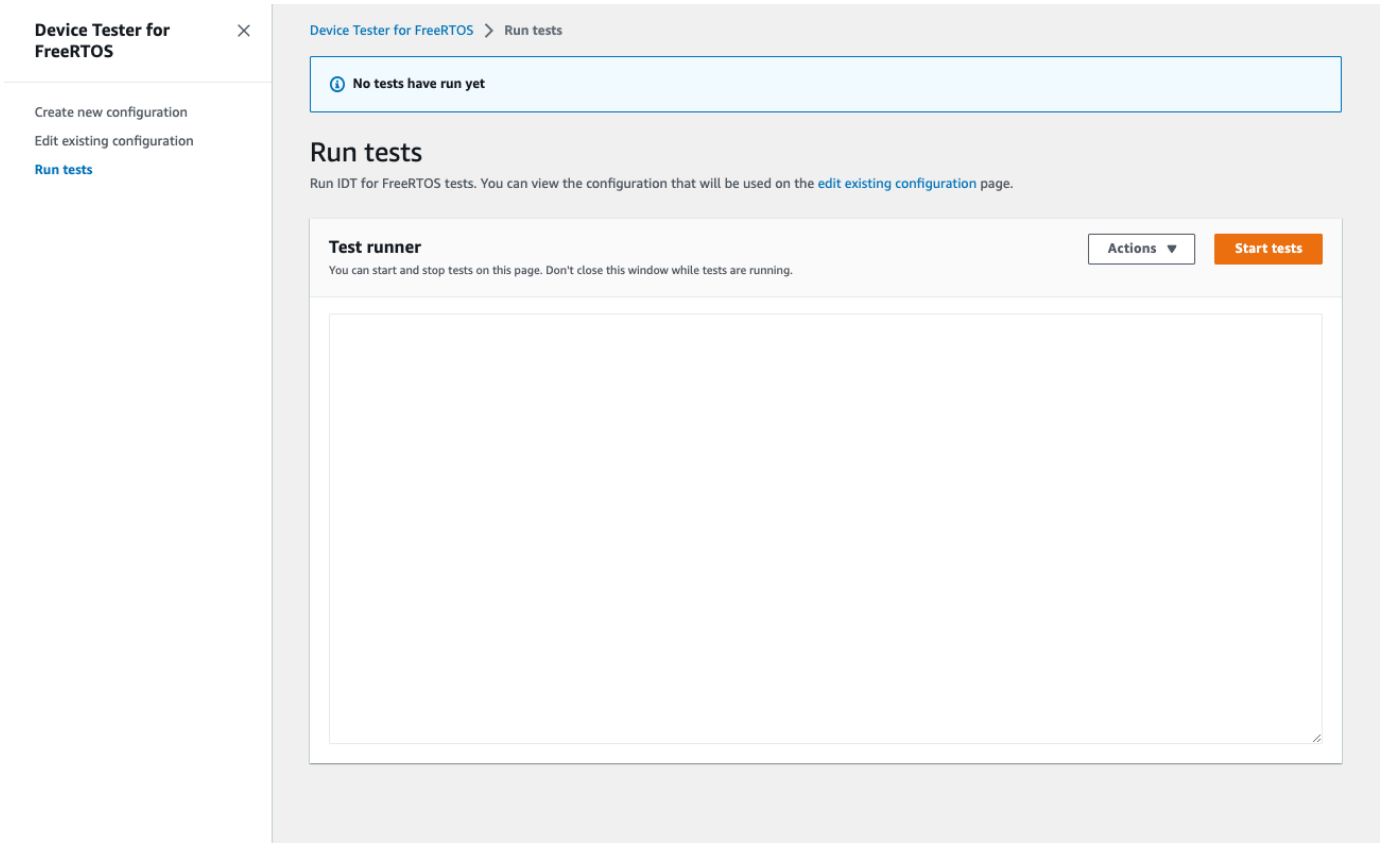
Nachdem Sie die Änderung Ihrer Konfiguration abgeschlossen haben, stellen Sie sicher, dass alle Ihre Konfigurationseinstellungen die Validierung bestehen. Wenn der Status für jede Konfigurationseinstellung lautet`Valid`, können Sie Ihre Qualifizierungstests mit dieser Konfiguration ausführen.

Führen Sie Qualifizierungstests durch

Nachdem Sie eine Konfiguration für die Benutzeroberfläche von IDT for FreeRTOS erstellt haben, können Sie Ihre Qualifikationstests ausführen.

Um Qualifizierungstests durchzuführen

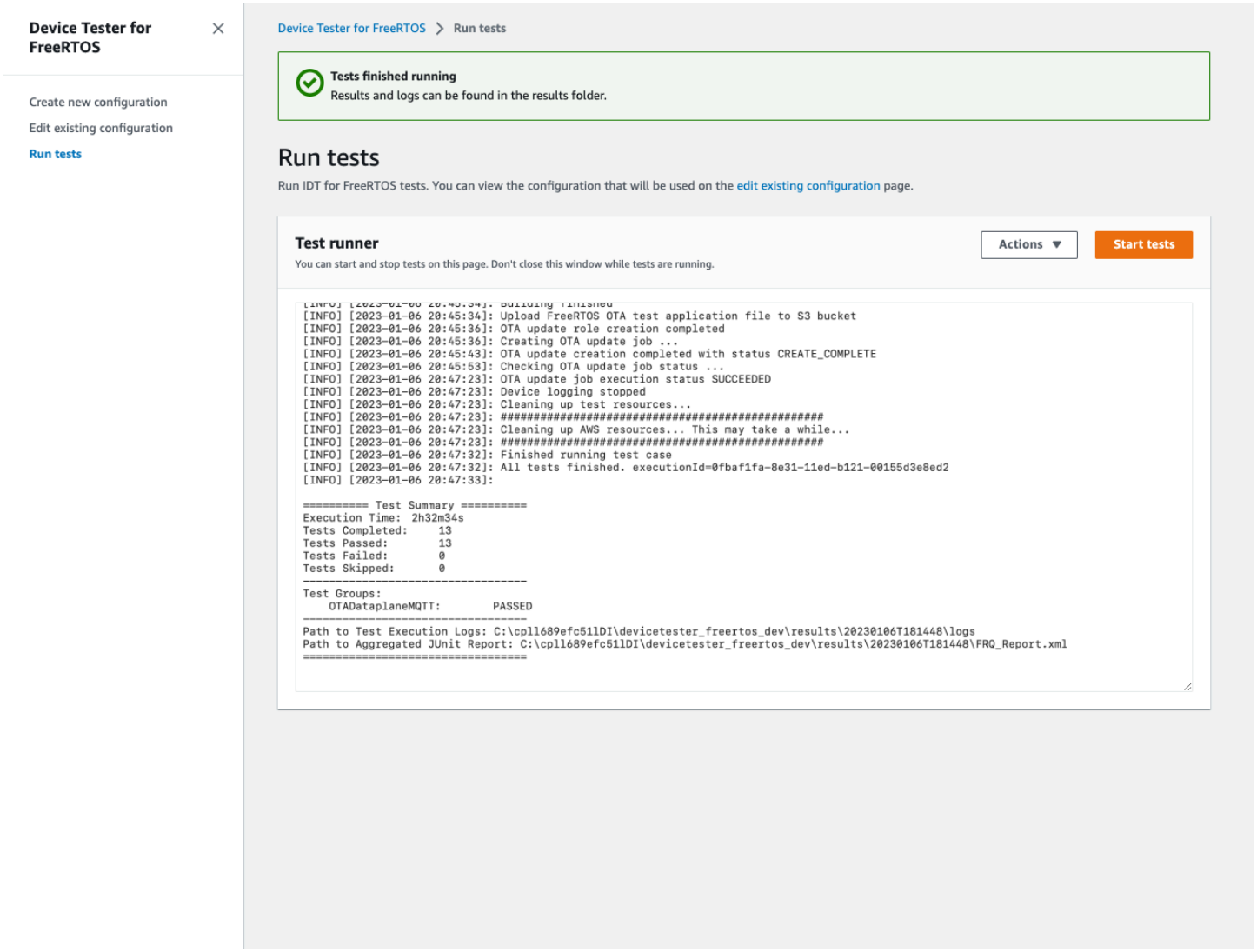
1. Wählen Sie im Navigationsmenü die Option Tests ausführen aus.
2. Wählen Sie Tests starten, um den Testlauf zu starten. Standardmäßig werden alle zutreffenden Tests für Ihre Gerätekonfiguration ausgeführt. IDT for FreeRTOS generiert einen Qualifikationsbericht, wenn alle Tests abgeschlossen sind.



IDT for FreeRTOS führt die Qualifikationstests durch. Anschließend werden die Zusammenfassung des Testlaufs und alle Fehler in der Test Runner-Konsole angezeigt. Nach Abschluss des Testlaufs können Sie die Testergebnisse und Protokolle an den folgenden Orten einsehen:

- Die Testergebnisse befinden sich im *devicetester-extract-location*`/results/`*execution-id* Verzeichnis.
- Testprotokolle befinden sich im *devicetester-extract-location*`/results/`*execution-id*`/logs` Verzeichnis.

Weitere Informationen zu Testergebnissen und Protokollen finden Sie unter Sehen Sie sich das IDT kostenlos an RTOSresults undSehen Sie sich das IDT kostenlos an RTOSlogs.



Führen Sie die FreeRTOS Qualification 2.0 Suite aus

Verwenden Sie die ausführbare Datei AWS IoT Device Tester for FreeRTOS, um mit IDT for FreeRTOS zu interagieren. In den folgenden Befehlszeilenbeispielen wird veranschaulicht, wie Sie die Qualifikationsprüfungen für einen Gerätepool (Satz identischer Geräte) durchführen.

IDT v4.5.2 and later

```
devicetester_[linux | mac | win] run-suite  \
    --suite-id suite-id  \
    --group-id group-id  \
    --pool-id your-device-pool \
    --test-id test-id  \
    --userdata userdata.json
```

Führt eine Reihe von Tests in einem Pool von Geräten aus. Die Datei `userdata.json` muss sich im Verzeichnis *devicetester_extract_location*`/devicetester_freertos_`*[win|mac|linux]*`/configs/` befinden.

> **Note**
>
> Wenn Sie IDT for FreeRTOS unter Windows ausführen, verwenden Sie Schrägstriche (/), um den Pfad zur Datei anzugeben. `userdata.json`

Verwenden Sie den folgenden Befehl zum Ausführen einer bestimmten Testgruppe:

```
devicetester_[linux | mac | win] run-suite  \
    --suite-id FRQ_1.99.0  \
    --group-id group-id  \
    --pool-id pool-id  \
    --userdata userdata.json
```

Die Parameter `suite-id` und `pool-id` sind optional, wenn Sie eine einzige Testsuite in einem einzigen Gerätepool ausführen (d. h. in Ihrer `device.json`-Datei ist nur ein Gerätepool definiert).

Verwenden Sie den folgenden Befehl zum Ausführen eines bestimmten Testfalls in einer Testgruppe:

```
devicetester_[linux | mac | win_x86-64] run-suite  \
    --group-id group-id  \
    --test-id test-id
```

Mit dem Befehl `list-test-cases` können Sie die Testfälle in einer Testgruppe auflisten.

IDT für FreeRTOS-Befehlszeilenoptionen

group-id

(Optional) Die auszuführenden Testgruppen als kommagetrennte Liste. Bei fehlender Angabe führt IDT alle Testgruppen in der Testsuite aus.

pool-id

(Optional) Der zu testende Gerätepool. Dies ist erforderlich, wenn Sie mehrere Gerätepools in `device.json` definieren. Wenn Sie nur einen Gerätepool haben, können Sie diese Option weglassen.

suite-id

(Optional) Die auszuführende Test-Suite-Version. Falls nicht angegeben, verwendet IDT die neueste Version im Verzeichnis der Tests auf Ihrem System.

test-id

(Optional) Die auszuführenden Tests als kommagetrennte Liste. Wenn angegeben, muss `group-id` eine einzelne Gruppe angeben.

Example

```
devicetester_[linux | mac | win_x86-64] run-suite --group-id FreeRTOSVersion --
test-id FreeRTOSVersion
```

h

Verwenden Sie die Hilfe-Option, um mehr über `run-suite`-Optionen zu erfahren.

Example

Beispiel

```
devicetester_[linux | mac | win_x86-64] run-suite -h
```

IDT für FreeRTOS-Befehle

Der Befehl IDT for FreeRTOS unterstützt die folgenden Operationen:

IDT v4.5.2 and later

`help`

Listet Informationen über den angegebenen Befehl auf.

`list-groups`

Listet die Gruppen in der jeweiligen Suite auf.

`list-suites`

Listet die verfügbaren Suites auf.

`list-supported-products`

Listet die unterstützten Produkte und Testsuiteversionen auf.

`list-supported-versions`

Listet die FreeRTOS- und Testsuite-Versionen auf, die von der aktuellen IDT-Version unterstützt werden.

`list-test-cases`

Listet die Testfälle in einer angegebenen Gruppe auf.

`run-suite`

Führt eine Reihe von Tests in einem Pool von Geräten aus.

Verwenden Sie die Option `--suite-id`, um eine Test-Suite-Version anzugeben, oder lassen Sie sie weg, um die neueste Version auf Ihrem System zu verwenden.

Verwenden Sie die `--test-id` um einen einzelnen Testfall auszuführen.

Example

```
devicetester_[linux | mac | win_x86-64] run-suite --group-id FreeRTOSVersion --
test-id FreeRTOSVersion
```

> **Note**
>
> Ab IDT v3.0.0 sucht IDT online nach neueren Testsuiten. Weitere Informationen finden Sie unter Test-Suite-Versionen.

Sehen Sie sich das IDT kostenlos an RTOSresults

Während der Ausführung schreibt IDT Fehler in die Konsole, Protokolldateien und Testberichte. Nachdem IDT die Qualifikations-Testsuite abgeschlossen hat, schreibt es eine Zusammenfassung der Testläufe in die Konsole und erstellt zwei Testberichte. Diese Berichte befinden sich in *`devicetester-extract-location`*`/results/`*`execution-id`*`/`. Beide Berichte erfassen die Ergebnisse von der Ausführung der Qualifikations-Testsuite.

Dies `awsiotdevicetester_report.xml` ist der Qualifizierungstestbericht, den Sie einreichen AWS , um Ihr Gerät im AWS Partner-Gerätekatalog aufzulisten. Die Bericht enthält die folgenden Elemente:

- Die Version IDT für FreeRTOS.
- Die getestete FreeRTOS-Version.
- Die Funktionen von FreeRTOS, die vom Gerät unterstützt werden, basieren auf den bestandenen Tests.
- SKU und Gerätename, die in der `device.json`-Datei angegeben wurden.
- Die Funktionen des Geräts, das in der `device.json`-Datei angegeben wurde.
- Die aggregierte Zusammenfassung der Ergebnisse der Testfälle.
- Eine Aufschlüsselung der Testfallergebnisse nach Bibliotheken, die basierend auf den Geräteeigenschaften getestet wurden.

Das `FRQ_Report.xml` ist ein Bericht im JUnit Standard-XML-Format. Sie können ihn in CI/CD Plattformen wie Jenkins, Bamboo usw. integrieren. Die Bericht enthält die folgenden Elemente:

- Eine aggregierte Zusammenfassung der Ergebnisse der Testfälle.
- Eine Aufschlüsselung der Testfallergebnisse nach Bibliotheken, die basierend auf den Geräteeigenschaften getestet wurden.

Interpretieren Sie die IDT for FreeRTOS-Ergebnisse

Der Berichtsabschnitt in `awsiotdevicetester_report.xml` oder `FRQ_Report.xml` listet die Ergebnisse der durchgeführten Tests auf.

Im ersten XML-Tag `<testsuites>` ist die Gesamtzusammenfassung der Testausführung enthalten. Beispiel:

```
<testsuites name="FRQ results" time="5633" tests="184" failures="0"
errors="0" disabled="0">
```

Im **`<testsuites>`** Tag verwendete Attribute

`name`

Name der Testsuite

time

Zeit (in Sekunden), die zur Ausführung der Qualifikations-Suite erforderlich war

tests

Anzahl der ausgeführten Testfälle

failures

Anzahl der ausgeführten Testfälle, die den Test nicht bestanden haben

errors

Die Anzahl der Testfälle, die IDT für FreeRTOS nicht ausführen konnte.

disabled

Dieses Attribut wird nicht verwendet und kann ignoriert werden.

Wenn es keine Testfallausfälle oder Fehler gibt, erfüllt Ihr Gerät die technischen Voraussetzungen für die Ausführung von FreeRTOS und kann mit Diensten zusammenarbeiten. AWS IoT Wenn Sie Ihr Gerät im AWS Partner-Gerätekatalog auflisten möchten, können Sie diesen Bericht als Qualifikationsnachweis verwenden.

Falls bei Testfällen Fehler auftreten, können Sie den fehlgeschlagenen Testfall identifizieren, indem Sie die XML-Tags von `<testsuites>` überprüfen. Die XML-Tags von `<testsuite>` im `<testsuites>`-Tag zeigen die Ergebniszusammenfassung des Testfalls für eine Testgruppe.

```
<testsuite name="FreeRTOSVersion" package="" tests="1" failures="0"
time="2" disabled="0" errors="0" skipped="0">
```

Das Format ähnelt dem `<testsuites>`-Tag, weist aber ein zusätzliches Attribut mit dem Namen `skipped` auf, das nicht verwendet wird und ignoriert werden kann. Innerhalb der einzelnen XML-Tags von `<testsuite>` befinden sich `<testcase>`-Tags für alle Testfälle, die für eine Testgruppe ausgeführt wurden. Beispiel:

```
<testcase classname="FRQ FreeRTOSVersion" name="FreeRTOSVersion"
attempts="1"></testcase>
```

Im **<awsproduct>** Tag verwendete Attribute

name

Der Name des getesteten Produkts.

`version`

Die Version des getesteten Produkts.

`features`

Die validierten Funktionen Als `required` gekennzeichnete Funktionen sind für die Einreichung Ihres Boards für die Qualifizierung erforderlich. Der folgende Ausschnitt zeigt, wie dies in der `awsiotdevicetester_report.xml` Datei aussieht.

```
<feature name="core-freertos" value="not-supported" type="required"></feature>
```

Als `optional` gekennzeichnete Funktionen sind für die Qualifizierung nicht erforderlich. Die folgenden Codeausschnitte zeigen optionale Funktionen.

```
<feature name="ota-dataplane-mqtt" value="not-supported" type="optional"></feature>
  <feature name="ota-dataplane-http" value="not-supported" type="optional"></
feature>
```

Wenn es keine Testfehler oder Fehler für die erforderlichen Funktionen gibt, erfüllt Ihr Gerät die technischen Voraussetzungen für die Ausführung von FreeRTOS und kann mit Diensten zusammenarbeiten. AWS IoT Wenn Sie Ihr Gerät im AWS Partner-Gerätekatalog auflisten möchten, können Sie diesen Bericht als Qualifikationsnachweis verwenden.

Falls bei Tests Fehler auftreten, können Sie den fehlgeschlagenen Test identifizieren, indem Sie die XML-Tags von `<testsuites>` überprüfen. Die XML-Tags von `<testsuite>` im `<testsuites>`-Tag zeigen die Ergebniszusammenfassung eines Tests für eine Testgruppe. Beispiel:

```
<testsuite name="FreeRTOSVersion" package="" tests="1" failures="1" time="2"
 disabled="0" errors="0" skipped="0">
```

Das Format ähnelt dem `<testsuites>` Tag, hat jedoch ein `skipped` Attribut, das nicht verwendet wird und ignoriert werden kann. Innerhalb der einzelnen `<testsuite>`-XML-Tags befinden sich `<testcase>`-Tags für alle ausgeführten Tests einer Testgruppe. Beispiel:

```
<testcase classname="FreeRTOSVersion" name="FreeRTOSVersion"></testcase>
```

Im **`<testcase>`** Tag verwendete Attribute

`name`

Name des Testfalls

`attempts`

Gibt an, wie oft IDT für FreeRTOS den Testfall ausgeführt hat.

Wenn ein Testfall fehlschlägt oder ein Fehler auftritt, werden `<failure>`- oder `<error>`-Tags hinzugefügt, um das `<testcase>`-Tag mit Informationen für die Fehlerbehebung zu versehen. Beispiel:

```
<testcase classname="FRQ FreeRTOSVersion" name="FreeRTOSVersion">
      <failure type="Failure">Reason for the test case failure</failure>
      <error>Reason for the test case execution error</error>
  </testcase>
```

Weitere Informationen finden Sie unter Beheben von Fehlern in der .

Sehen Sie sich das IDT kostenlos an RTOSlogs

Logs, die IDT for FreeRTOS bei der Testausführung generiert, finden Sie unter. *devicetester-extract-location*/results/*execution-id*/logs Es werden zwei Protokollgruppen generiert:

- `test_manager.log`

  Enthält von IDT für FreeRTOS generierte Protokolle (z. B. Protokolle zur Konfiguration und Berichtserstellung).
- *test_group_id*/*test_case_id*/*test_case_id*.log

  Die Protokolldatei für einen Testfall, einschließlich der Ausgabe des zu testenden Geräts. Die Protokolldatei wird nach der Testgruppe und dem ausgeführten Testfall benannt.

IDT mit FreeRTOS Qualification Suite 1.0 (FRQ 1.0)

> **Important**
>
> Stand Oktober 2022 generiert AWS IoT FreeRTOS Qualification (FRQ) 1.0 keine signierten Qualifikationsberichte. AWS IoT Device Tester Sie können neue AWS IoT FreeRTOS-Geräte nicht für die Aufnahme in den AWS Partnergerätekatalog über das AWS

Gerätequalifizierungsprogramm qualifizieren, wenn Sie IDT FRQ 1.0-Versionen verwenden. Sie können FreeRTOS-Geräte zwar nicht mit IDT FRQ 1.0 qualifizieren, aber Sie können Ihre FreeRTOS-Geräte weiterhin mit FRQ 1.0 testen. Wir empfehlen Ihnen, IDT FRQ 2.0 zu verwenden, um FreeRTOS-Geräte zu qualifizieren und im Partnergerätekatalog aufzulisten.AWS

Sie können IDT für die FreeRTOS-Qualifizierung verwenden, um zu überprüfen, ob das FreeRTOS-Betriebssystem lokal auf Ihrem Gerät funktioniert und mit diesem kommunizieren kann. AWS IoT Insbesondere wird überprüft, ob die Schnittstellen der Portierungsschicht für die FreeRTOS-Bibliotheken korrekt implementiert sind. Es führt end-to-end auch Tests mit durch. AWS IoT Core So wird beispielsweise überprüft, ob Ihr Board MQTT-Nachrichten senden und empfangen und korrekt verarbeiten kann. Die von IDT für FreeRTOS ausgeführten Tests sind im FreeRTOS-Repository definiert. GitHub

Die Tests laufen als Embedded-Anwendungen, die auf Ihr Board geflasht werden. Die Binär-Images der Anwendung enthalten FreeRTOS, die portierten FreeRTOS-Schnittstellen des Halbleiterherstellers, und Platinengerätetreiber. Der Zweck der Tests besteht darin, zu überprüfen, ob die portierten FreeRTOS-Schnittstellen neben den Gerätetreibern korrekt funktionieren.

IDT for FreeRTOS generiert Testberichte, die Sie einreichen können, AWS IoT um Ihre Hardware zum AWS Partnergerätekatalog hinzuzufügen. Weitere Informationen finden Sie unter AWS Device Qualification Program.

IDT for FreeRTOS läuft auf einem Host-Computer (Windows, macOS oder Linux), der an das zu testende Board angeschlossen ist. IDT führt Testfälle aus und fasst Ergebnisse zusammen. Er stellt ebenfalls eine Befehlszeilenschnittstelle zur Verwaltung der Testausführung bereit.

IDT for FreeRTOS testet nicht nur Geräte, sondern erstellt auch Ressourcen (z. B. AWS IoT Dinge, FreeRTOS-Gruppen, Lambda-Funktionen usw.), um den Qualifizierungsprozess zu erleichtern. Um diese Ressourcen zu erstellen, verwendet IDT for FreeRTOS die in der konfigurierten AWS Anmeldeinformationen, `config.json` um API-Aufrufe in Ihrem Namen durchzuführen. Diese Ressourcen werden zu verschiedenen Zeiten während eines Tests bereitgestellt.

Wenn Sie IDT for FreeRTOS auf Ihrem Host-Computer ausführen, werden die folgenden Schritte ausgeführt:

1. Laden und überprüfen Sie die Konfiguration Ihres Geräts und Ihrer Anmeldeinformationen.
2. Führen Sie ausgewählte Tests mit den erforderlichen lokalen und Cloud-Ressourcen durch.

3. Bereinigen Sie lokale und Cloud-Ressourcen.
4. Erstellen Sie Testberichte, die anzeigen, ob Ihr Board die für die Qualifikation erforderlichen Tests bestanden hat.

Themen

- Richten Sie die 1.0-Qualifikationsvoraussetzungen ein
- Erster Test Ihres Mikrocontroller-Boards
- Verwenden Sie die IDT-Benutzeroberfläche, um die FreeRTOS Qualification Suite auszuführen
- Führen Sie Bluetooth Low Energy-Tests durch
- Führen Sie die FreeRTOS Qualification Suite aus
- Sehen Sie sich die Ergebnisse von IDT for FreeRTOS an
- Interpretieren Sie die IDT for FreeRTOS-Ergebnisse
- Die IDT for FreeRTOS-Protokolle anzeigen

Richten Sie die 1.0-Qualifikationsvoraussetzungen ein

In diesem Abschnitt werden die Voraussetzungen für das Testen von Mikrocontrollern mit AWS IoT Device Tester beschrieben.

FreeRTOS herunterladen

Sie können eine Version von FreeRTOS GitHubmit dem folgenden Befehl herunterladen:

```
git clone --branch <FREERTOS_RELEASE_VERSION> --recurse-submodules https://github.com/
aws/amazon-freertos.git
cd amazon-freertos
git submodule update --checkout --init --recursive
```

wo <FREERTOS_RELEASE_VERSION>ist eine Version von FreeRTOS (zum Beispiel 202007.00), die einer IDT-Version entspricht, die unter aufgeführt ist. Unterstützte Versionen von AWS IoT Device Tester Dadurch wird sichergestellt, dass Sie über den vollständigen Quellcode einschließlich der Submodule verfügen und die richtige Version von IDT für Ihre Version von FreeRTOS verwenden und umgekehrt.

Windows hat eine Pfadlängenbegrenzung von 260 Zeichen. Die Pfadstruktur von FreeRTOS ist vielschichtig. Wenn Sie also Windows verwenden, sollten Sie Ihre Dateipfade unter dem Limit von

260 Zeichen halten. Klonen Sie FreeRTOS beispielsweise auf `C:\FreeRTOS` statt. `C:\Users\username\programs\projects\myproj\FreeRTOS\`

FreeRTOS-Qualifikation mit LTS-Bibliotheken

- Damit Ihr Mikrocontroller im AWS Partner Device Catalog als auf Long-Term Support (LTS) basierende Versionen von FreeRTOS unterstützt werden kann, müssen Sie eine Manifestdatei bereitstellen. Weitere Informationen finden Sie in der FreeRTOS Qualification Checklist im FreeRTOS Qualification Guide.
- Um zu überprüfen, ob Ihr Mikrocontroller LTS-basierte Versionen von FreeRTOS unterstützt, und um ihn für die Einreichung im AWS Partner Device Catalog zu qualifizieren, müssen Sie AWS IoT Device Tester (IDT) mit der FreeRTOS Qualification (FRQ) Test Suite Version v1.4.x verwenden.
- Die Support für LTS-basierte Versionen von FreeRTOS ist auf die Version 202012.xx von FreeRTOS beschränkt.

Laden Sie IDT für FreeRTOS herunter

Jede Version von FreeRTOS hat eine entsprechende Version von IDT für FreeRTOS zur Durchführung von Qualifizierungstests. Laden Sie die entsprechende Version von IDT für FreeRTOS von herunter. Unterstützte Versionen von AWS IoT Device Tester

Extrahieren Sie IDT for FreeRTOS an einen Speicherort im Dateisystem, für den Sie Lese- und Schreibberechtigungen haben. Da Microsoft Windows eine Zeichenbeschränkung für die Pfadlänge hat, extrahieren Sie IDT für FreeRTOS in ein Stammverzeichnis wie oder. `C:\` `D:\`

> **Note**
>
> Es wird nicht empfohlen, dass mehrere Benutzer IDT aus einem freigegebenen Speicherort ausführen, z. B. einem NFS-Verzeichnis oder einem freigegebenen Windows-Netzwerkordner. Dies kann zu Abstürzen oder Datenbeschädigung führen. Es wird empfohlen, das IDT-Paket in einem lokalen Laufwerk zu extrahieren.

Erstellen und konfigurieren Sie ein Konto AWS

Melden Sie sich an für eine AWS-Konto

Wenn Sie noch keine haben AWS-Konto, führen Sie die folgenden Schritte aus, um eine zu erstellen.

Um sich für eine anzumelden AWS-Konto

1. Öffnen Sie https://portal.aws.amazon.com/billing/die Anmeldung.
2. Folgen Sie den Online-Anweisungen.

   Während der Anmeldung erhalten Sie einen Telefonanruf oder eine Textnachricht und müssen einen Verifizierungscode über die Telefontasten eingeben.

   Wenn Sie sich für eine anmelden AWS-Konto, Root-Benutzer des AWS-Kontoswird eine erstellt. Der Root-Benutzer hat Zugriff auf alle AWS-Services und Ressourcen des Kontos. Als bewährte Sicherheitsmethode weisen Sie einem Benutzer Administratorzugriff zu und verwenden Sie nur den Root-Benutzer, um Aufgaben auszuführen, die Root-Benutzerzugriff erfordern.

AWS sendet Ihnen nach Abschluss des Anmeldevorgangs eine Bestätigungs-E-Mail. Sie können Ihre aktuellen Kontoaktivitäten jederzeit einsehen und Ihr Konto verwalten, indem Sie zu https://aws.amazon.com/gehen und Mein Konto auswählen.

Erstellen eines Benutzers mit Administratorzugriff

Nachdem Sie sich für einen angemeldet haben AWS-Konto, sichern Sie Ihren Root-Benutzer des AWS-Kontos AWS IAM Identity Center, aktivieren und erstellen Sie einen Administratorbenutzer, sodass Sie den Root-Benutzer nicht für alltägliche Aufgaben verwenden.

Sichern Sie Ihre Root-Benutzer des AWS-Kontos

1. Melden Sie sich AWS-Managementkonsoleals Kontoinhaber an, indem Sie Root-Benutzer auswählen und Ihre AWS-Konto E-Mail-Adresse eingeben. Geben Sie auf der nächsten Seite Ihr Passwort ein.

   Hilfe bei der Anmeldung mit dem Root-Benutzer finden Sie unter Anmelden als Root-Benutzer im AWS-Anmeldung -Benutzerhandbuch zu.
2. Aktivieren Sie die Multi-Faktor-Authentifizierung (MFA) für den Root-Benutzer.

   Anweisungen finden Sie unter Aktivieren eines virtuellen MFA-Geräts für Ihren AWS-Konto Root-Benutzer (Konsole) im IAM-Benutzerhandbuch.

Erstellen eines Benutzers mit Administratorzugriff

1. Aktivieren Sie das IAM Identity Center.

Anweisungen finden Sie unter Aktivieren AWS IAM Identity Center im AWS IAM Identity Center - Benutzerhandbuch.

2. Gewähren Sie einem Administratorbenutzer im IAM Identity Center Benutzerzugriff.

   Ein Tutorial zur Verwendung von IAM-Identity-Center-Verzeichnis als Identitätsquelle finden Sie IAM-Identity-Center-Verzeichnis im Benutzerhandbuch unter Benutzerzugriff mit der Standardeinstellung konfigurieren.AWS IAM Identity Center

Anmelden als Administratorbenutzer

- Um sich mit Ihrem IAM-Identity-Center-Benutzer anzumelden, verwenden Sie die Anmelde-URL, die an Ihre E-Mail-Adresse gesendet wurde, als Sie den IAM-Identity-Center-Benutzer erstellt haben.

  Hilfe bei der Anmeldung mit einem IAM Identity Center-Benutzer finden Sie im AWS-Anmeldung Benutzerhandbuch unter Anmeldung beim AWS Access-Portal.

Weiteren Benutzern Zugriff zuweisen

1. Erstellen Sie im IAM-Identity-Center einen Berechtigungssatz, der den bewährten Vorgehensweisen für die Anwendung von geringsten Berechtigungen folgt.

   Anweisungen hierzu finden Sie unter Berechtigungssatz erstellen im AWS IAM Identity Center - Benutzerhandbuch.

2. Weisen Sie Benutzer einer Gruppe zu und weisen Sie der Gruppe dann Single Sign-On-Zugriff zu.

   Eine genaue Anleitung finden Sie unter Gruppen hinzufügen im AWS IAM Identity Center - Benutzerhandbuch.

AWS IoT Device Tester verwaltete Richtlinie

Die `AWSIoTDeviceTesterForFreeRTOSFullAccess` verwaltete Richtlinie enthält die folgenden AWS IoT Device Tester Berechtigungen für die Versionsprüfung, auto Aktualisierungsfunktionen und die Erfassung von Metriken.

- `iot-device-tester:SupportedVersion`

Erteilt die AWS IoT Device Tester Berechtigung zum Abrufen der Liste der unterstützten Produkte, Testsuiten und IDT-Versionen.

- `iot-device-tester:LatestIdt`

  AWS IoT Device Tester Erteilt die Erlaubnis, die neueste IDT-Version abzurufen, die zum Herunterladen verfügbar ist.

- `iot-device-tester:CheckVersion`

  AWS IoT Device Tester Erteilt die Erlaubnis, die Versionskompatibilität für IDT, Testsuiten und Produkte zu überprüfen.

- `iot-device-tester:DownloadTestSuite`

  AWS IoT Device Tester Erteilt die Erlaubnis zum Herunterladen von Testsuite-Updates.

- `iot-device-tester:SendMetrics`

  AWS Erteilt die Erlaubnis, Metriken zur AWS IoT Device Tester internen Nutzung zu sammeln.

(Optional) Installieren Sie AWS Command Line Interface

Möglicherweise ziehen Sie es vor, den zu verwenden AWS CLI , um einige Operationen auszuführen. Wenn Sie das nicht AWS CLI installiert haben, folgen Sie den Anweisungen unter Installieren von AWS CLI.

Konfigurieren Sie das AWS CLI für die AWS Region, die Sie verwenden möchten, indem Sie es aws configure von einer Befehlszeile aus ausführen. Informationen zu den AWS Regionen, die IDT für FreeRTOS unterstützen, finden Sie unter AWS Regionen und Endpunkte. Weitere Informationen finden Sie unter Schnellkonfiguration mitaws configure. aws configure

Erster Test Ihres Mikrocontroller-Boards

Sie können IDT for FreeRTOS zum Testen verwenden, während Sie die FreeRTOS-Schnittstellen portieren. Nachdem Sie die FreeRTOS-Schnittstellen für die Gerätetreiber Ihres Boards portiert haben, führen Sie AWS IoT Device Tester damit die Qualifizierungstests auf Ihrem Mikrocontroller-Board durch.

Hinzufügen von Portierungsebenen für Bibliotheken

Um FreeRTOS für Ihr Gerät zu portieren, folgen Sie den Anweisungen im FreeRTOS Porting Guide.

AWS Konfigurieren Sie Ihre Zugangsdaten

Sie müssen Ihre AWS Anmeldeinformationen für AWS IoT Device Tester die Kommunikation mit der AWS Cloud konfigurieren. Weitere Informationen finden Sie unter AWS Zugangsdaten und Region für die Entwicklung einrichten. In der *devicetester_extract_location*/devicetester_afreertos_*[win|mac|linux]*/configs/config.json Konfigurationsdatei müssen gültige AWS Anmeldeinformationen angegeben werden.

Themen

- Erstellen Sie einen Gerätepool in IDT für FreeRTOS
- Konfiguration von Build-, Flash- und Testeinstellungen

Erstellen Sie einen Gerätepool in IDT für FreeRTOS

Zu testende Geräte werden in Gerätepools organisiert. Jeder Gerätepool besteht aus einem oder mehreren identischen Geräten. Sie können IDT for FreeRTOS so konfigurieren, dass ein einzelnes Gerät in einem Pool oder mehrere Geräte in einem Pool getestet werden. Um den Qualifizierungsprozess zu beschleunigen, kann IDT for FreeRTOS Geräte mit denselben Spezifikationen parallel testen. Er verwendet eine Round Robin-Methode, um auf jedem Gerät in einem Gerätepool eine andere Testgruppe auszuführen.

Sie können ein oder mehrere Geräte zu einem Gerätepool hinzufügen, indem Sie den Abschnitt `devices` der Vorlage `device.json` im Ordner `configs` bearbeiten.

> **Note**
>
> Alle Geräte im selben Pool müssen dieselbe technische Spezifikation und SKU aufweisen.

Um parallel Builds des Quellcodes für verschiedene Testgruppen zu ermöglichen, kopiert IDT for FreeRTOS den Quellcode in einen Ergebnisordner innerhalb des extrahierten IDT for FreeRTOS-Ordners. Der Quellcodepfad in Ihrem Build- oder Flash-Befehl muss entweder mit der Variablen oder referenziert werden. `testdata.sourcePath` `sdkPath` IDT for FreeRTOS ersetzt diese Variable durch einen temporären Pfad des kopierten Quellcodes. Weitere Informationen finden Sie unter IDT für FreeRTOS-Variablen.

Im Folgenden sehen Sie, wie eine `device.json`-Datei zur Erstellung eines Gerätepools mit mehreren Geräten verwendet wird:

```
[
    {
        "id": "pool-id",
        "sku": "sku",
        "features": [
            {
                "name": "WIFI",
                "value": "Yes | No"
            },
            {
                "name": "Cellular",
                "value": "Yes | No"
            },
            {
                "name": "OTA",
                "value": "Yes | No",
                "configs": [
                    {
                        "name": "OTADataPlaneProtocol",
                        "value": "HTTP | MQTT"
                    }
                ]
            },
            {
                "name": "BLE",
                "value": "Yes | No"
            },
            {
                "name": "TCP/IP",
                "value": "On-chip | Offloaded | No"
            },
            {
                "name": "TLS",
                "value": "Yes | No"
            },
            {
                "name": "PKCS11",
                "value": "RSA | ECC | Both | No"
            },
            {
                "name": "KeyProvisioning",
                "value": "Import | Onboard | No"
            }
```

```
        ],

        "devices": [
          {
            "id": "device-id",
            "connectivity": {
              "protocol": "uart",
              "serialPort": "/dev/tty*"
            },
            **********Remove the section below if the device does not support onboard
 key generation**************
            "secureElementConfig" : {
              "publicKeyAsciiHexFilePath": "absolute-path-to/public-key-txt-file:
 contains-the-hex-bytes-public-key-extracted-from-onboard-private-key",
              "secureElementSerialNumber": "secure-element-serialNo-value",
              "preProvisioned"           : "Yes | No"
            },

 ********************************************************************************
            "identifiers": [
              {
                "name": "serialNo",
                "value": "serialNo-value"
              }
            ]
          }
        ]
    }
]
```

Die folgenden Attribute werden in der Datei `device.json` verwendet:

`id`

Eine benutzerdefinierte alphanumerische ID, die einen Gerätepool eindeutig identifiziert. Geräte, die zu einem Pool gehören, müssen vom gleichen Typ sein. Bei der Ausführung einer Reihe von Tests werden Geräte im Pool verwendet, um den Workload zu parallelisieren.

`sku`

Ein alphanumerischer Wert, mit dem das getestete Board eindeutig identifiziert wird. Die SKU wird verwendet, um qualifizierte Boards nachzuverfolgen.

> **Note**
>
> Wenn Sie Ihr Motherboard im Gerätekatalog für AWS Partner anbieten möchten, muss die hier angegebene SKU mit der SKU übereinstimmen, die Sie bei der Angebotserstellung verwenden.

`features`

Ein Array, das die vom Gerät unterstützten Funktionen enthält. AWS IoT Device Tester verwendet diese Informationen, um die durchzuführenden Qualifikationstests auszuwählen.

Unterstützte Werte sind:

`TCP/IP`

Zeigt an, ob Ihre Platine einen TCP/IP Stack unterstützt und ob dieser On-Chip (MCU) unterstützt oder auf ein anderes Modul ausgelagert wird. TCP/IP ist für die Qualifikation erforderlich.

`WIFI`

Gibt an, ob Ihr Board über WiFi-Funktionen verfügt. Muss auf gesetzt werden, `No` wenn auf gesetzt `Cellular` ist`Yes`.

`Cellular`

Zeigt an, ob Ihr Board über Mobilfunkfunktionen verfügt. Muss auf eingestellt sein, `No` wenn auf eingestellt `WIFI` ist`Yes`. Wenn diese Funktion auf eingestellt ist`Yes`, wird der FullSecureSockets Test mithilfe von AWS t2.micro EC2-Instances ausgeführt, was zu zusätzlichen Kosten für Ihr Konto führen kann. Weitere Informationen dazu finden Sie unter Amazon EC2 – Preise.

`TLS`

Gibt an, ob Ihr Board TLS unterstützt. TLS ist für die Qualifizierung erforderlich.

`PKCS11`

Gibt den Kryptografiealgorithmus mit öffentlichen Schlüsseln an, den das Board unterstützt. PKCS11 ist für die Qualifikation erforderlich. Unterstützte Werte sind `ECC`, `RSA`, `Both` und `No`. `Both` zeigt an, dass das Board sowohl den `ECC`- als auch den `RSA`-Algorithmus unterstützt.

`KeyProvisioning`

Gibt an, wie ein vertrauenswürdiges X.509-Clientzertifikat auf das Board geschrieben werden kann. Gültige Werte sind `Import`, `Onboard` und `No`. Schlüsselbereitstellung ist für die Qualifizierung erforderlich.

- Verwenden Sie `Import`, wenn Ihr Board den Import von privaten Schlüsseln erlaubt. IDT erstellt einen privaten Schlüssel und baut diesen in den FreeRTOS-Quellcode ein.
- Verwenden Sie `Onboard`, wenn Ihr Board die interne Erstellung von privaten Schlüsseln unterstützt (z. B. wenn Ihr Gerät über ein sicheres Element verfügt oder wenn Sie es vorziehen, ein eigenes Geräte-Schlüsselpaar und ein eigenes Zertifikat zu generieren). Stellen Sie sicher, dass Sie in jedem der Geräteabschnitte ein `secureElementConfig`-Element hinzufügen und fügen Sie den absoluten Pfad zur Datei des öffentlichen Schlüssels in das Feld `publicKeyAsciiHexFilePath` ein.
- Wenn Ihr Board die Schlüsselbereitstellung nicht unterstützt, verwenden Sie die Option `No`.

`OTA`

Zeigt an, ob Ihr Board die Aktualisierungsfunktion over-the-air (OTA) unterstützt. Das Attribut `OtaDataPlaneProtocol` gibt an, welches OTA-Protokoll auf Datenebene das Gerät unterstützt. Das Attribut wird ignoriert, wenn die OTA-Funktion vom Gerät nicht unterstützt wird. Wenn ausgewählt `"Both"` ist, wird die Ausführungszeit des OTA-Tests verlängert, da sowohl MQTT-, HTTP- als auch gemischte Tests ausgeführt werden.

> **Note**
>
> Beginnend mit IDT v4.1.0 werden nur `HTTP` und `MQTT` als unterstützte Werte `OtaDataPlaneProtocol` akzeptiert.

`BLE`

Gibt an, ob Ihr Board Bluetooth Low Energy (BLE) unterstützt.

`devices.id`

Eine benutzerdefinierte eindeutige Kennung für das zu testende Gerät.

`devices.connectivity.protocol`

Das Kommunikationsprotokoll, das für die Kommunikation mit diesem Gerät verwendet wird. Unterstützter Wert: `uart`.

`devices.connectivity.serialPort`

Der serielle Port des Host-Computers, der zur Herstellung einer Verbindung mit den getesteten Geräten verwendet wird.

`devices.secureElementConfig.PublicKeyAsciiHexFilePath`

Der absolute Pfad zu der Datei, die den öffentlichen Hex-Byte-Schlüssel enthält, der aus dem integrierten privaten Schlüssel extrahiert wurde.

Beispielformat:

```
3059 3013 0607 2a86 48ce 3d02 0106 082a
8648 ce3d 0301 0703 4200 04cd 6569 ceb8
1bb9 1e72 339f e8cf 60ef 0f9f b473 33ac
6f19 1813 6999 3fa0 c293 5fae 08f1 1ad0
41b7 345c e746 1046 228e 5a5f d787 d571
dcb2 4e8d 75b3 2586 e2cc 0c
```

Wenn Ihr öffentlicher Schlüssel im.der-Format vorliegt, können Sie den öffentlichen Schlüssel direkt hexcodieren, um die Hex-Datei zu generieren.

Beispielbefehl für den öffentlichen Schlüssel .der zur Generierung einer Hex-Datei:

```
xxd -p pubkey.der > outFile
```

Wenn Ihr öffentlicher Schlüssel im.pem-Format vorliegt, können Sie den Base64-kodierten Teil extrahieren, ihn in das Binärformat dekodieren und ihn dann hexadezimalkodieren, um die Hex-Datei zu generieren.

Verwenden Sie beispielsweise diese Befehle, um eine Hex-Datei für einen öffentlichen PEM-Schlüssel zu generieren:

1. Nehmen Sie den Base64-kodierten Teil des Schlüssels heraus (entfernen Sie die Kopf- und Fußzeile) und speichern Sie ihn in einer Datei. Geben Sie ihm beispielsweise einen Namen und führen Sie diesen Befehl aus`base64key`, um ihn in das Format.der zu konvertieren:

   ```
   base64 –decode base64key > pubkey.der
   ```

2. Führen Sie den `xxd` Befehl aus, um ihn in das Hex-Format zu konvertieren.

```
xxd -p pubkey.der > outFile
```

`devices.secureElementConfig.SecureElementSerialNumber`

(Optional) Die Seriennummer des sicheren Elements. Geben Sie dieses Feld an, wenn die Seriennummer zusammen mit dem öffentlichen Schlüssel des Geräts ausgedruckt wird, wenn Sie das demo/test FreeRTOS-Projekt ausführen.

`devices.secureElementConfig.preProvisioned`

(Optional) Wählen Sie „Ja“, wenn das Gerät über ein vorab bereitgestelltes sicheres Element mit gesperrten Anmeldeinformationen verfügt, das keine Objekte importieren, erstellen oder zerstören kann. Diese Konfiguration wird nur wirksam, wenn sie `features` zusammen mit „ECC“ auf „Onboard“ `PKCS11` gesetzt ist. `KeyProvisioning`

`identifiers`

(Optional) Ein Array beliebiger Namen-Wert-Paare. Sie können diese Werte in den im nächsten Abschnitt beschriebenen Build- und Flash-Befehlen verwenden.

Konfiguration von Build-, Flash- und Testeinstellungen

Damit IDT for FreeRTOS automatisch Tests auf Ihrem Board erstellen und flashen kann, müssen Sie IDT so konfigurieren, dass die Build- und Flash-Befehle für Ihre Hardware ausgeführt werden. Die Einstellungen für den Build- und den Flash-Befehl werden in der `userdata.json`-Vorlagendatei im Ordner `config` konfiguriert.

Konfigurieren von Einstellungen für das Testen von Geräten

Build-, Flash- und Testeinstellungen werden in der `configs/userdata.json`-Datei vorgenommen. Wir unterstützen die Echo Server-Konfiguration, indem wir sowohl die Client- als auch die Serverzertifikate und Schlüssel in den ladencustomPath. Weitere Informationen finden Sie unter Einen Echo-Server einrichten im FreeRTOS Porting Guide. Das folgende JSON-Beispiel zeigt, wie Sie IDT für FreeRTOS konfigurieren können, um mehrere Geräte zu testen:

```
{
    "sourcePath": "/absolute-path-to/freertos",
    "vendorPath": "{{testData.sourcePath}}/vendors/vendor-name/boards/board-name",
    // ***********The sdkConfiguration block below is needed if you are not using the
 default, unmodified FreeRTOS repo.
```

```
   // In other words, if you are using the default, unmodified FreeRTOS repo then
remove this block***************
   "sdkConfiguration": {
       "name": "sdk-name",
       "version": "sdk-version",
       "path": "/absolute-path-to/sdk"
   },
   "buildTool": {
       "name": "your-build-tool-name",
       "version": "your-build-tool-version",
       "command": [
           "{{config.idtRootPath}}/relative-path-to/build-parallel.sh
{{testData.sourcePath}} {{enableTests}}"
       ]
   },
   "flashTool": {
       "name": "your-flash-tool-name",
       "version": "your-flash-tool-version",
       "command": [
           "/{{config.idtRootPath}}/relative-path-to/flash-parallel.sh
{{testData.sourcePath}} {{device.connectivity.serialPort}} {{buildImageName}}"
       ],
       "buildImageInfo" : {
           "testsImageName": "tests-image-name",
           "demosImageName": "demos-image-name"
       }
   },
   "testStartDelayms": 0,
   "clientWifiConfig": {
       "wifiSSID": "ssid",
       "wifiPassword": "password",
       "wifiSecurityType": "eWiFiSecurityOpen | eWiFiSecurityWEP | eWiFiSecurityWPA |
eWiFiSecurityWPA2 | eWiFiSecurityWPA3"
   },
   "testWifiConfig": {
       "wifiSSID": "ssid",
       "wifiPassword": "password",
       "wifiSecurityType": "eWiFiSecurityOpen | eWiFiSecurityWEP | eWiFiSecurityWPA |
eWiFiSecurityWPA2 | eWiFiSecurityWPA3"
   },
   //**********
   //This section is used to start echo server based on server certificate generation
method,
```

```
    //When certificateGenerationMethod is set as Automatic specify the eccCurveFormat
 to generate certifcate and key based on curve format,
    //When certificateGenerationMethod is set as Custom specify the certificatePath and
 PrivateKeyPath to be used to start echo server
    //**********
    "echoServerCertificateConfiguration": {
      "certificateGenerationMethod": "Automatic | Custom",
      "customPath": {
          "clientCertificatePath":"/path/to/clientCertificate",
          "clientPrivateKeyPath": "/path/to/clientPrivateKey",
          "serverCertificatePath":"/path/to/serverCertificate",
          "serverPrivateKeyPath": "/path/to/serverPrivateKey"
      },
    "eccCurveFormat": "P224 | P256 | P384 | P521"
    },
    "echoServerConfiguration": {
        "securePortForSecureSocket": 33333, // Secure tcp port used by SecureSocket
 test. Default value is 33333. Ensure that the port configured isn't blocked by the
 firewall or your corporate network
       "insecurePortForSecureSocket": 33334, // Insecure tcp port used by SecureSocket
 test. Default value is 33334. Ensure that the port configured isn't blocked by the
 firewall or your corporate network
       "insecurePortForWiFi": 33335 // Insecure tcp port used by Wi-Fi test. Default
 value is 33335. Ensure that the port configured isn't blocked by the firewall or your
 corporate network
    },
    "otaConfiguration": {
        "otaFirmwareFilePath": "{{testData.sourcePath}}/relative-path-to/ota-image-
generated-in-build-process",
       "deviceFirmwareFileName": "ota-image-name-on-device",
       "otaDemoConfigFilePath": "{{testData.sourcePath}}/relative-path-to/ota-demo-
config-header-file",
       "codeSigningConfiguration": {
           "signingMethod": "AWS | Custom",
           "signerHashingAlgorithm": "SHA1 | SHA256",
           "signerSigningAlgorithm": "RSA | ECDSA",
           "signerCertificate": "arn:partition:service:region:account-
id:resource:qualifier | /absolute-path-to/signer-certificate-file",
           "signerCertificateFileName": "signerCertificate-file-name",
           "compileSignerCertificate": boolean,
           // ***********Use signerPlatform if you choose aws for
 signingMethod***************
           "signerPlatform": "AmazonFreeRTOS-Default | AmazonFreeRTOS-TI-CC3220SF",
```

```
            "untrustedSignerCertificate": "arn:partition:service:region:account-
id:resourcetype:resource:qualifier",
            // ***********Use signCommand if you choose custom for
 signingMethod***************
            "signCommand": [
                "/absolute-path-to/sign.sh {{inputImageFilePath}}
 {{outputSignatureFilePath}}"
            ]
        }
    },
    // ***********Remove the section below if you're not configuring
 CMake***************
    "cmakeConfiguration": {
        "boardName": "board-name",
        "vendorName": "vendor-name",
        "compilerName": "compiler-name",
        "frToolchainPath": "/path/to/freertos/toolchain",
        "cmakeToolchainPath": "/path/to/cmake/toolchain"
    },
    "freertosFileConfiguration": {
        "required": [
            {
                "configName": "pkcs11Config",
                "filePath": "{{testData.sourcePath}}/vendors/vendor-name/boards/board-
name/aws_tests/config_files/core_pkcs11_config.h"
            },
            {
                "configName": "pkcs11TestConfig",
                "filePath": "{{testData.sourcePath}}/vendors/vendor-name/boards/board-
name/aws_tests/config_files/iot_test_pkcs11_config.h"
            }
        ],
        "optional": [
            {
                "configName": "otaAgentTestsConfig",
                "filePath": "{{testData.sourcePath}}/vendors/vendor-name/boards/board-
name/aws_tests/config_files/ota_config.h"
            },
            {
                "configName": "otaAgentDemosConfig",
                "filePath": "{{testData.sourcePath}}/vendors/vendor-name/boards/board-
name/aws_demos/config_files/ota_config.h"
            },
            {
```

```
                "configName": "otaDemosConfig",
                "filePath": "{{testData.sourcePath}}/vendors/vendor-name/boards/board-
name/aws_demos/config_files/ota_demo_config.h"
            }
        ]
    }
}
```

Im Folgenden werden die in der `userdata.json`-Datei verwendeten Attribute aufgelistet:

`sourcePath`

Der Pfad zum Stammverzeichnis des portierten FreeRTOS-Quellcodes. Für parallel Tests mit einem SDK `sourcePath` kann das über den `{{userData.sdkConfiguration.path}}` Platzhalter eingestellt werden. Beispiel:

```
{ "sourcePath":"{{userData.sdkConfiguration.path}}/freertos" }
```

`vendorPath`

Der Pfad zum herstellerspezifischen FreeRTOS-Code. Für serielle Tests kann der `vendorPath` als absoluter Pfad festgelegt werden. Beispiel:

```
{ "vendorPath":"C:/path-to-freertos/vendors/espressif/boards/esp32" }
```

Für parallele Tests kann der `vendorPath` mit dem Platzhalter `{{testData.sourcePath}}` eingestellt werden. Beispiel:

```
{ "vendorPath":"{{testData.sourcePath}}/vendors/espressif/boards/esp32" }
```

Die `vendorPath` Variable ist nur notwendig, wenn sie ohne SDK läuft, andernfalls kann sie entfernt werden.

> **Note**
>
> Wenn Tests ohne SDK parallel ausgeführt werden, muss der `{{testData.sourcePath}}` Platzhalter in den `flashTool` Feldern`vendorPath`,`buildTool`, verwendet werden. Wenn der Test mit einem einzigen Gerät ausgeführt wird, müssen absolute Pfade in den Feldern `vendorPath`,

`buildTool`, `flashTool` verwendet werden. Bei der Ausführung mit einem SDK muss der `{{sdkPath}}` Platzhalter in den Befehlen `sourcePathbuildTool`, und `flashTool` verwendet werden.

`sdkConfiguration`

Wenn Sie FreeRTOS mit Änderungen an der Datei- und Ordnerstruktur qualifizieren, die über das für die Portierung erforderliche Maß hinausgehen, müssen Sie Ihre SDK-Informationen in diesem Block konfigurieren. Wenn Sie sich nicht mit einem portierten FreeRTOS innerhalb eines SDK qualifizieren, sollten Sie diesen Block komplett weglassen.

`sdkConfiguration.name`

Der Name des SDK, das Sie mit FreeRTOS verwenden. Wenn Sie kein SDK verwenden, sollte der gesamte `sdkConfiguration` Block weggelassen werden.

`sdkConfiguration.version`

Die Version des SDK, das Sie mit FreeRTOS verwenden. Wenn Sie kein SDK verwenden, sollte der gesamte `sdkConfiguration` Block weggelassen werden.

`sdkConfiguration.path`

Der absolute Pfad zu Ihrem SDK-Verzeichnis, das Ihren FreeRTOS-Code enthält. Wenn Sie kein SDK verwenden, sollte der gesamte `sdkConfiguration` Block weggelassen werden.

`buildTool`

Der vollständige Pfad zu Ihrem Build-Skript (.bat oder .sh), das die Befehle zur Erstellung Ihres Quellcodes enthält. Alle Verweise auf den Quellcodepfad im Build-Befehl müssen durch die AWS IoT Device Tester Variable ersetzt werden `{{testdata.sourcePath}}` und Verweise auf den SDK-Pfad sollten durch ersetzt werden`{{sdkPath}}`. Verwenden Sie den `{{config.idtRootPath}}` Platzhalter, um auf den absoluten oder relativen IDT-Pfad zu verweisen.

`testStartDelayms`

Gibt an, wie viele Millisekunden der FreeRTOS-Testläufer wartet, bevor er mit der Ausführung von Tests beginnt. Dies kann nützlich sein, wenn das zu testende Gerät aufgrund von Netzwerk- oder anderen Latenzen mit der Ausgabe wichtiger Testinformationen beginnt, bevor IDT die Möglichkeit hat, eine Verbindung herzustellen und mit der Protokollierung zu beginnen. Der zulässige

Höchstwert ist 30000 ms (30 Sekunden). Dieser Wert gilt nur für FreeRTOS-Testgruppen und nicht für andere Testgruppen, die den FreeRTOS-Testrunner nicht verwenden, wie z. B. die OTA-Tests.

`flashTool`

Vollständiger Pfad zu Ihrem Flash-Skript (.sh oder.bat), der die Flash-Befehle für Ihr Gerät enthält. Alle Verweise auf den Quellcodepfad im Befehl flash müssen durch die Variable IDT for FreeRTOS ersetzt werden `{{testdata.sourcePath}}` und alle Verweise auf Ihren SDK-Pfad müssen durch die Variable IDT for FreeRTOS ersetzt werden. `{{sdkPath}}` Verwenden Sie den `{{config.idtRootPath}}` Platzhalter, um auf den absoluten oder relativen IDT-Pfad zu verweisen.

`buildImageInfo`

`testsImageName`

Der Name der Datei, die vom Build-Befehl beim Erstellen von Tests aus dem Ordner erzeugt wurde. *`freertos-source`*`/tests`

`demosImageName`

Der Name der Datei, die vom Build-Befehl beim Erstellen von Tests aus dem *`freertos-source`*`/demos` Ordner erzeugt wurde.

`clientWifiConfig`

Die Client-WLAN-Konfiguration. Die Tests für die WLAN-Bibliothek erfordern ein MCU-Board, um eine Verbindung mit zwei Zugriffspunkten herzustellen. (Die beiden Zugangspunkte können identisch sein.) Dieses Attribut konfiguriert die WLAN-Einstellungen für den ersten Zugriffspunkt. In einigen WLAN-Testfällen wird erwartet, dass der Zugriffspunkt über ein gewisses Maß an Sicherheit verfügt und nicht offen ist. Bitte stellen Sie sicher, dass sich beide Access Points im selben Subnetz befinden wie der Host-Computer, auf dem IDT ausgeführt wird.

`wifi_ssid`

Die WLAN-SSID.

`wifi_password`

Das WLAN-Passwort.

`wifiSecurityType`

Die Art der verwendeten WLAN-Sicherheit. Einer der Werte:

- `eWiFiSecurityOpen`
- `eWiFiSecurityWEP`
- `eWiFiSecurityWPA`
- `eWiFiSecurityWPA2`
- `eWiFiSecurityWPA3`

> **Note**
>
> Wenn Ihr Board kein WLAN unterstützt, müssen Sie Ihrer `device.json`-Datei dennoch den Abschnitt `clientWifiConfig` hinzufügen. Werte für diese Attribute können Sie jedoch weglassen.

`testWifiConfig`

Die WLAN-Testkonfiguration. Die Tests für die WLAN-Bibliothek erfordern ein MCU-Board, um eine Verbindung mit zwei Zugriffspunkten herzustellen. (Die beiden Zugangspunkte können identisch sein.) Dieses Attribut konfiguriert die WLAN-Einstellungen für den zweiten Zugriffspunkt. In einigen WLAN-Testfällen wird erwartet, dass der Zugriffspunkt über ein gewisses Maß an Sicherheit verfügt und nicht offen ist. Bitte stellen Sie sicher, dass sich beide Access Points im selben Subnetz befinden wie der Host-Computer, auf dem IDT ausgeführt wird.

`wifiSSID`

Die WLAN-SSID.

`wifiPassword`

Das WLAN-Passwort.

`wifiSecurityType`

Die Art der verwendeten WLAN-Sicherheit. Einer der Werte:

- `eWiFiSecurityOpen`
- `eWiFiSecurityWEP`
- `eWiFiSecurityWPA`
- `eWiFiSecurityWPA2`
- `eWiFiSecurityWPA3`

> **Note**
>
> Wenn Ihr Board kein WLAN unterstützt, müssen Sie Ihrer `device.json`-Datei dennoch den Abschnitt `testWifiConfig` hinzufügen. Werte für diese Attribute können Sie jedoch weglassen.

`echoServerCertificateConfiguration`

Der konfigurierbare Platzhalter für die Generierung von Echo-Serverzertifikaten für Secure-Socket-Tests. Dies ist ein Pflichtfeld.

`certificateGenerationMethod`

Gibt an, ob das Serverzertifikat automatisch generiert oder manuell bereitgestellt wird.

`customPath`

Es `certificateGenerationMethod` ist „Benutzerdefiniert“ `certificatePath` und `privateKeyPath` sie sind erforderlich.

`certificatePath`

Gibt den Dateipfad für das Serverzertifikat an.

`privateKeyPath`

Gibt den Dateipfad für den privaten Schlüssel an.

`eccCurveFormat`

Gibt das von der Karte unterstützte Kurvenformat an. `PKCS11`Erforderlich, wenn in `device.json` auf „ecc“ gesetzt ist. Gültige Werte sind „P224", „P256", „P384" oder „P521".

`echoServerConfiguration`

Die konfigurierbaren Echo-Server-Ports für und sichere Socket-Tests. WiFi Dies ist ein optionales Feld.

`securePortForSecureSocket`

Der Port, der zum Einrichten eines Echo-Servers mit TLS für den Secure Sockets-Test verwendet wird. Der Standardwert ist 33333. Stellen Sie sicher, dass der konfigurierte Port nicht von einer Firewall oder Ihrem Unternehmensnetzwerk blockiert wird.

`insecurePortForSecureSocket`

Der Port, der zum Einrichten eines Echo-Servers ohne TLS für den Secure Sockets-Test verwendet wird. Der Standardwert für den Test ist 33334. Stellen Sie sicher, dass der konfigurierte Port nicht von einer Firewall oder Ihrem Unternehmensnetzwerk blockiert wird.

`insecurePortForWiFi`

Der Port, der verwendet wird, um den Echo-Server ohne TLS für WiFi Tests einzurichten. Der Standardwert für den Test ist 33335. Stellen Sie sicher, dass der konfigurierte Port nicht von einer Firewall oder Ihrem Unternehmensnetzwerk blockiert wird.

`otaConfiguration`

Die OTA-Konfiguration. [Optional]

`otaFirmwareFilePath`

Der vollständige Pfad zum OTA-Image, das nach dem Build erstellt wird. Beispiel, `{{testData.sourcePath}}/`*`relative-path/to/ota/image/from/source/root`*.

`deviceFirmwareFileName`

Der vollständige Dateipfad auf dem MCU-Gerät, auf dem sich die OTA-Firmware befindet. Einige Geräte verwenden dieses Feld nicht, aber Sie müssen trotzdem einen Wert angeben.

`otaDemoConfigFilePath`

Der vollständige Pfad zu `aws_demo_config.h`, zu finden in *`afr-source`*`/vendors/vendor/boards/board/aws_demos/config_files/`. Diese Dateien sind in der Portierungscode-Vorlage enthalten, die FreeRTOS bereitstellt.

`codeSigningConfiguration`

Die Code-Signaturkonfiguration.

`signingMethod`

Die Code-Signaturmethode. Die möglichen Wert sind `AWS` oder `Custom`.

> **Note**
>
> Verwenden Sie für die Regionen Peking und Ningxia. `Custom` `AWS`Codesignatur wird in diesen Regionen nicht unterstützt.

`signerHashingAlgorithm`

Der auf dem Gerät unterstützte Hashing-Algorithmus. Die möglichen Wert sind `SHA1` oder `SHA256`.

`signerSigningAlgorithm`

Der auf dem Gerät unterstützte Signaturalgorithmus. Die möglichen Wert sind `RSA` oder `ECDSA`.

`signerCertificate`

Das für OTA verwendete vertrauenswürdige Zertifikat.

Verwenden Sie als AWS Codesignaturmethode den Amazon-Ressourcennamen (ARN) für das vertrauenswürdige Zertifikat, das in den hochgeladen wurde AWS Certificate Manager.

Verwenden Sie für benutzerdefinierte Codesignaturmethode den absoluten Pfad zur Signiererzertifikatdatei.

Weitere Hinweise zum Erstellen eines vertrauenswürdigen Zertifikats finden Sie unter Erstellen eines Zertifikats für die Codesignierung.

`signerCertificateFileName`

Der Dateiname des Codesignaturzertifikats auf dem Gerät. Dieser Wert muss mit dem Dateinamen übereinstimmen, den Sie bei der Ausführung des `aws acm import-certificate` Befehls angegeben haben.

Weitere Informationen finden Sie unter Erstellen eines Zertifikats für die Codesignierung.

`compileSignerCertificate`

`true`Wird auf gesetzt, wenn das Zertifikat zur Überprüfung der Signatur des Codesigners nicht bereitgestellt oder geflasht wurde und daher in das Projekt kompiliert werden muss. AWS IoT Device Tester ruft das vertrauenswürdige Zertifikat ab und kompiliert es in. `aws_codesigner_certifiate.h`

`untrustedSignerCertificate`

Der ARN oder Dateipfad für ein zweites Zertifikat, das in einigen OTA-Tests als nicht vertrauenswürdiges Zertifikat verwendet wird. Weitere Informationen zum Erstellen eines Zertifikats finden Sie unter Erstellen eines Codesignaturzertifikats.

`signerPlatform`

Der Signier- und Hash-Algorithmus, den AWS Code Signer bei der Erstellung des OTA-Aktualisierungsjobs verwendet. Derzeit lauten die möglichen Werte für dieses Feld `AmazonFreeRTOS-TI-CC3220SF` und `AmazonFreeRTOS-Default`.

- Wählen Sie bei `SHA1` und `RSA` `AmazonFreeRTOS-TI-CC3220SF` aus.
- Wählen Sie bei `SHA256` und `ECDSA` `AmazonFreeRTOS-Default` aus.

Wenn Sie `SHA256 | RSA` oder `SHA1 | ECDSA` für Ihre Konfiguration benötigen, kontaktieren Sie uns, um weitere Unterstützung zu erhalten.

Konfigurieren Sie `signCommand`, wenn Sie `Custom` für `signingMethod` ausgewählt haben.

`signCommand`

Der Befehl, der zum Ausführen benutzerdefinierter Codesignaturen verwendet wird. Sie finden die Vorlage im Verzeichnis „`/configs/script_templates`“.

Die beiden Platzhalter „`{{inputImageFilePath}}`“ und „`{{outputSignatureFilePath}}`“ sind im Befehl erforderlich. `{{inputImageFilePath}}` ist der Dateipfad des von IDT erstellten Images, das signiert werden soll. `{{outputSignatureFilePath}}` ist der Dateipfad der Signatur, der vom Skript generiert wird.

`cmakeConfiguration`

CMake Konfiguration [optional]

> **Note**
>
> Um CMake Testfälle auszuführen, müssen Sie den Boardnamen, den Herstellernamen und entweder das `frToolchainPath` Oder angeben`compilerName`. Sie können das auch angeben`cmakeToolchainPath`, wenn Sie einen benutzerdefinierten Pfad zur CMake Toolchain haben.

`boardName`

Der Name des Boards, das getestet wird. Der Boardname sollte mit dem Ordnernamen unter *`path/to/afr/source/code`*`/vendors/`*`vendor`*`/boards/`*`board`* übereinstimmen.

vendorName

Der Herstellername für die zu testende Karte. Der Anbieter sollte mit dem Ordnernamen unter *path/to/afr/source/code*/vendors/*vendor* übereinstimmen.

compilerName

Der Name des Compilers.

frToolchainPath

Der vollqualifizierte Pfad zur Compiler-Toolchain.

cmakeToolchainPath

Der vollständig qualifizierte Pfad zur Toolchain. CMake Dies ist ein optionales Feld.

freertosFileConfiguration

Die Konfiguration der FreeRTOS-Dateien, die IDT vor der Ausführung von Tests ändert.

required

In diesem Abschnitt werden die erforderlichen Tests angegeben, deren Konfigurationsdateien Sie verschoben haben PKCS11, z. B. TLS usw.

configName

Der Name des Tests, der konfiguriert wird.

filePath

Der absolute Pfad zu den Konfigurationsdateien innerhalb des *freertos* Repos. Verwenden Sie die `{{testData.sourcePath}}` Variable, um den Pfad zu definieren.

optional

In diesem Abschnitt werden optionale Tests angegeben, deren Konfigurationsdateien Sie verschoben haben WiFi, z. B. OTA usw.

configName

Der Name des Tests, der konfiguriert wird.

filePath

Der absolute Pfad zu den Konfigurationsdateien innerhalb des *freertos* Repos. Verwenden Sie die `{{testData.sourcePath}}` Variable, um den Pfad zu definieren.

> **Note**
>
> Um CMake Testfälle auszuführen, müssen Sie den Boardnamen, den Herstellernamen und entweder das `afrToolchainPath` Oder angeben`compilerName`. Sie können auch angeben`cmakeToolchainPath`, ob Sie einen benutzerdefinierten Pfad zur CMake Toolchain haben.

IDT für FreeRTOS-Variablen

Die Befehle zum Erstellen Ihres Codes und zum Flashen des Geräts erfordern möglicherweise Konnektivität oder andere Informationen zu Ihren Geräten, um erfolgreich ausgeführt zu werden. AWS IoT Device Tester ermöglicht es Ihnen, Geräteinformationen in Flash zu referenzieren und Befehle zu erstellen mit JsonPath. Mithilfe einfacher JsonPath Ausdrücke können Sie die erforderlichen Informationen abrufen, die in Ihrer `device.json` Datei angegeben sind.

Pfadvariablen

IDT for FreeRTOS definiert die folgenden Pfadvariablen, die in Befehlszeilen und Konfigurationsdateien verwendet werden können:

`{{testData.sourcePath}}`

Wird auf den Quellcodepfad erweitert. Wenn Sie diese Variable verwenden, muss sie sowohl in den Flash- als auch in den Build-Befehlen verwendet werden.

`{{sdkPath}}`

Wird auf den Wert in Ihrem erweitert, `userData.sdkConfiguration.path` wenn es in den Befehlen Build und Flash verwendet wird.

`{{device.connectivity.serialPort}}`

Wird auf die serielle Schnittstelle erweitert.

`{{device.identifiers[?(@.name == 'serialNo')].value[0]}}`

Wird zur Seriennummer Ihres Geräts erweitert.

`{{enableTests}}`

Ganzzahliger Wert, der angibt, ob der Build für Tests (Wert 1) oder Demos (Wert 0) bestimmt ist.

{{buildImageName}}

Der Dateiname des mit dem Build-Befehl erstellten Image.

{{otaCodeSignerPemFile}}

PEM-Datei für den OTA-Codesigner.

{{config.idtRootPath}}

Erweitert auf den AWS IoT Device Tester Stammpfad. Diese Variable ersetzt den absoluten Pfad für IDT, wenn sie von den Befehlen build und flash verwendet wird.

Verwenden Sie die IDT-Benutzeroberfläche, um die FreeRTOS Qualification Suite auszuführen

Ab IDT v4.3.0 enthält AWS IoT Device Tester for FreeRTOS (IDT-FreeRTOS) eine webbasierte Benutzeroberfläche, über die Sie mit der ausführbaren IDT-Befehlszeile und den zugehörigen Konfigurationsdateien interagieren können. Sie können die IDT-FreeRTOS-Benutzeroberfläche verwenden, um eine neue Konfiguration zum Ausführen von IDT-Tests zu erstellen oder eine bestehende Konfiguration zu ändern. Sie können die Benutzeroberfläche auch verwenden, um die IDT-Programmdatei aufzurufen und Tests auszuführen.

Die IDT-FreeRTOS-Benutzeroberfläche bietet die folgenden Funktionen:

- Vereinfachen Sie die Einrichtung von Konfigurationsdateien für IDT-FreeRTOS-Tests.
- Vereinfachen Sie die Verwendung von IDT-FreeRTOS zur Durchführung von Qualifikationstests.

Hinweise zur Verwendung der Befehlszeile zum Ausführen von Qualifikationstests finden Sie unter. Erster Test Ihres Mikrocontroller-Boards

Dieser Abschnitt beschreibt die Voraussetzungen für die Verwendung der IDT-FreeRTOS-Benutzeroberfläche und zeigt Ihnen, wie Sie mit der Ausführung von Qualifizierungstests in der Benutzeroberfläche beginnen.

Themen

- Richten Sie die Voraussetzungen für den Betrieb der FreeRTOS Qualification Suite ein
- Erste Schritte mit der IDT-FreeRTOS-Benutzeroberfläche

Richten Sie die Voraussetzungen für den Betrieb der FreeRTOS Qualification Suite ein

In diesem Abschnitt werden die Voraussetzungen für das Testen von Mikrocontrollern mit AWS IoT Device Tester beschrieben.

Themen

- Verwenden Sie einen unterstützten Webbrowser
- FreeRTOS herunterladen
- Laden Sie IDT für FreeRTOS herunter
- Erstellen und konfigurieren Sie ein Konto AWS
- AWS IoT Device Tester verwaltete Richtlinie

Verwenden Sie einen unterstützten Webbrowser

Die IDT-FreeRTOS-Benutzeroberfläche unterstützt die folgenden Webbrowser.

Browser	Version
Google Chrome	Die letzten drei Hauptversionen
Mozilla Firefox	Die letzten drei Hauptversionen
Microsoft Edge	Die letzten drei Hauptversionen
Apple Safari für macOS	Die letzten drei Hauptversionen

Wir empfehlen Ihnen, Google Chrome oder Mozilla Firefox für eine bessere Benutzererfahrung zu verwenden.

> **Note**
>
> Die IDT-FreeRTOS-Benutzeroberfläche unterstützt Microsoft Internet Explorer nicht.

FreeRTOS herunterladen

Sie können eine Version von FreeRTOS GitHubmit dem folgenden Befehl herunterladen:

```
git clone --branch <FREERTOS_RELEASE_VERSION> --recurse-submodules https://github.com/
aws/amazon-freertos.git
cd amazon-freertos
git submodule update --checkout --init --recursive
```

wo <FREERTOS_RELEASE_VERSION>ist eine Version von FreeRTOS (zum Beispiel 202007.00), die einer IDT-Version entspricht, die unter aufgeführt ist. Unterstützte Versionen von AWS IoT Device Tester Dadurch wird sichergestellt, dass Sie über den vollständigen Quellcode einschließlich der Submodule verfügen und die richtige Version von IDT für Ihre Version von FreeRTOS verwenden und umgekehrt.

Windows hat eine Pfadlängenbegrenzung von 260 Zeichen. Die Pfadstruktur von FreeRTOS ist vielschichtig. Wenn Sie also Windows verwenden, sollten Sie Ihre Dateipfade unter dem Limit von 260 Zeichen halten. Klonen Sie FreeRTOS beispielsweise auf `C:\FreeRTOS` statt. `C:\Users\username\programs\projects\myproj\FreeRTOS\`

Überlegungen zur LTS-Qualifizierung (Qualifizierung für FreeRTOS, das LTS-Bibliotheken verwendet)

- Damit Ihr Mikrocontroller im AWS Partner Device Catalog als auf Long-Term Support (LTS) basierende Versionen von FreeRTOS unterstützt werden kann, müssen Sie eine Manifestdatei bereitstellen. Weitere Informationen finden Sie in der FreeRTOS Qualification Checklist im FreeRTOS Qualification Guide.
- Um zu überprüfen, ob Ihr Mikrocontroller LTS-basierte Versionen von FreeRTOS unterstützt, und um ihn für die Einreichung im AWS Partner Device Catalog zu qualifizieren, müssen Sie AWS IoT Device Tester (IDT) mit der FreeRTOS Qualification (FRQ) Test Suite Version v1.4.x verwenden.
- Die Support für LTS-basierte Versionen von FreeRTOS ist auf die Version 202012.xx von FreeRTOS beschränkt.

Laden Sie IDT für FreeRTOS herunter

Jede Version von FreeRTOS hat eine entsprechende Version von IDT für FreeRTOS zur Durchführung von Qualifizierungstests. Laden Sie die entsprechende Version von IDT für FreeRTOS von herunter. Unterstützte Versionen von AWS IoT Device Tester

Extrahieren Sie IDT for FreeRTOS an einen Speicherort im Dateisystem, für den Sie Lese- und Schreibberechtigungen haben. Da Microsoft Windows eine Zeichenbeschränkung für die Pfadlänge hat, extrahieren Sie IDT für FreeRTOS in ein Stammverzeichnis wie oder. `C:\` `D:\`

> **Note**
>
> Es wird empfohlen, das IDT-Paket auf ein lokales Laufwerk zu extrahieren. Wenn mehrere Benutzer IDT von einem gemeinsam genutzten Speicherort ausführen können, z. B. von einem NFS-Verzeichnis oder einem gemeinsam genutzten Windows-Netzwerkordner, kann dies dazu führen, dass das System nicht reagiert oder Daten beschädigt werden.

Erstellen und konfigurieren Sie ein Konto AWS

Melden Sie sich an für ein AWS-Konto

Wenn Sie noch keine haben AWS-Konto, führen Sie die folgenden Schritte aus, um eine zu erstellen.

Um sich für eine anzumelden AWS-Konto

1. Öffnen Sie https://portal.aws.amazon.com/billing/die Anmeldung.
2. Folgen Sie den Online-Anweisungen.

   Während der Anmeldung erhalten Sie einen Telefonanruf oder eine Textnachricht und müssen einen Verifizierungscode über die Telefontasten eingeben.

   Wenn Sie sich für eine anmelden AWS-Konto, Root-Benutzer des AWS-Kontoswird eine erstellt. Der Root-Benutzer hat Zugriff auf alle AWS-Services und Ressourcen des Kontos. Als bewährte Sicherheitsmethode weisen Sie einem Benutzer Administratorzugriff zu und verwenden Sie nur den Root-Benutzer, um Aufgaben auszuführen, die Root-Benutzerzugriff erfordern.

AWS sendet Ihnen nach Abschluss des Anmeldevorgangs eine Bestätigungs-E-Mail. Du kannst jederzeit deine aktuellen Kontoaktivitäten einsehen und dein Konto verwalten, indem du zu https://aws.amazon.com/gehst und Mein Konto auswählst.

Erstellen eines Benutzers mit Administratorzugriff

Nachdem Sie sich für einen angemeldet haben AWS-Konto, sichern Sie Ihren Root-Benutzer des AWS-Kontos AWS IAM Identity Center, aktivieren und erstellen Sie einen Administratorbenutzer, sodass Sie den Root-Benutzer nicht für alltägliche Aufgaben verwenden.

Sichern Sie Ihre Root-Benutzer des AWS-Kontos

1. Melden Sie sich AWS-Managementkonsoleals Kontoinhaber an, indem Sie Root-Benutzer auswählen und Ihre AWS-Konto E-Mail-Adresse eingeben. Geben Sie auf der nächsten Seite Ihr Passwort ein.

   Hilfe bei der Anmeldung mit dem Root-Benutzer finden Sie unter Anmelden als Root-Benutzer im AWS-Anmeldung -Benutzerhandbuch zu.
2. Aktivieren Sie die Multi-Faktor-Authentifizierung (MFA) für den Root-Benutzer.

   Anweisungen finden Sie unter Aktivieren eines virtuellen MFA-Geräts für Ihren AWS-Konto Root-Benutzer (Konsole) im IAM-Benutzerhandbuch.

Erstellen eines Benutzers mit Administratorzugriff

1. Aktivieren Sie das IAM Identity Center.

   Anweisungen finden Sie unter Aktivieren AWS IAM Identity Center im AWS IAM Identity Center -Benutzerhandbuch.
2. Gewähren Sie einem Administratorbenutzer im IAM Identity Center Benutzerzugriff.

   Ein Tutorial zur Verwendung von IAM-Identity-Center-Verzeichnis als Identitätsquelle finden Sie IAM-Identity-Center-Verzeichnis im Benutzerhandbuch unter Benutzerzugriff mit der Standardeinstellung konfigurieren.AWS IAM Identity Center

Anmelden als Administratorbenutzer

- Um sich mit Ihrem IAM-Identity-Center-Benutzer anzumelden, verwenden Sie die Anmelde-URL, die an Ihre E-Mail-Adresse gesendet wurde, als Sie den IAM-Identity-Center-Benutzer erstellt haben.

  Hilfe bei der Anmeldung mit einem IAM Identity Center-Benutzer finden Sie im AWS-Anmeldung Benutzerhandbuch unter Anmeldung beim AWS Access-Portal.

Weiteren Benutzern Zugriff zuweisen

1. Erstellen Sie im IAM-Identity-Center einen Berechtigungssatz, der den bewährten Vorgehensweisen für die Anwendung von geringsten Berechtigungen folgt.

Anweisungen hierzu finden Sie unter Berechtigungssatz erstellen im AWS IAM Identity Center - Benutzerhandbuch.

2. Weisen Sie Benutzer einer Gruppe zu und weisen Sie der Gruppe dann Single Sign-On-Zugriff zu.

   Eine genaue Anleitung finden Sie unter Gruppen hinzufügen im AWS IAM Identity Center - Benutzerhandbuch.

AWS IoT Device Tester verwaltete Richtlinie

Damit der Gerätetester ausgeführt und Messwerte gesammelt werden kann, enthält die `AWSIoTDeviceTesterForFreeRTOSFullAccess` verwaltete Richtlinie die folgenden Berechtigungen:

- `iot-device-tester:SupportedVersion`

  Erteilt die Erlaubnis, die Liste der von IDT unterstützten FreeRTOS-Versionen und Testsuite-Versionen abzurufen, sodass sie auf der verfügbar sind. AWS CLI
- `iot-device-tester:LatestIdt`

  Erteilt die Erlaubnis, die neueste AWS IoT Device Tester Version herunterzuladen, die zum Herunterladen verfügbar ist.
- `iot-device-tester:CheckVersion`

  Gewährt die Berechtigung, zu überprüfen, ob Kombinationen aus Produkt, Testsuite und AWS IoT Device Tester -Versionen kompatibel sind.
- `iot-device-tester:DownloadTestSuite`

  Erteilt die Erlaubnis AWS IoT Device Tester zum Herunterladen von Testsuiten.
- `iot-device-tester:SendMetrics`

  Erteilt die Erlaubnis zur Veröffentlichung AWS IoT Device Tester von Nutzungsmetrikdaten.

Erste Schritte mit der IDT-FreeRTOS-Benutzeroberfläche

Dieser Abschnitt zeigt Ihnen, wie Sie die IDT-FreeRTOS-Benutzeroberfläche verwenden, um Ihre Konfiguration zu erstellen oder zu ändern, und zeigt Ihnen dann, wie Sie Tests ausführen.

Themen

- Anmeldeinformationen konfigurieren AWS
- Öffnen Sie die IDT-FreeRTOS-Benutzeroberfläche
- Erstellen Sie eine neue Konfiguration
- Ändern Sie eine bestehende Konfiguration
- Führen Sie Qualifizierungstests durch

Anmeldeinformationen konfigurieren AWS

Sie müssen die Anmeldeinformationen für den AWS Benutzer konfigurieren, den Sie in erstellt habenErstellen und konfigurieren Sie ein Konto AWS. Sie können Ihre Anmeldeinformationen auf zwei Arten angeben:

- In einer Anmeldeinformationsdatei
- Als Umgebungsvariablen

Konfigurieren Sie AWS Anmeldeinformationen mit einer Anmeldeinformationsdatei

IDT verwendet die gleiche Anmeldeinformationsdatei wie das AWS CLI. Weitere Informationen finden Sie unter Konfigurations- und Anmeldeinformationsdateien.

Der Speicherort der Anmeldeinformationsdatei variiert je nach verwendetem Betriebssystem:

- macOS Linux: `~/.aws/credentials`
- Windows: `C:\Users\`*`UserName`*`\.aws\credentials`

Fügen Sie der `credentials` Datei Ihre AWS Anmeldeinformationen im folgenden Format hinzu:

```
[default]
aws_access_key_id = <your_access_key_id>
aws_secret_access_key = <your_secret_access_key>
```

> **Note**
>
> Wenn Sie das `default` AWS Profil nicht verwenden, geben Sie unbedingt den Profilnamen in der IDT-FreeRTOS-Benutzeroberfläche an. Weitere Informationen zu Profilen finden Sie unter Konfiguration und Einstellungen für Anmeldeinformationsdateien.

Konfigurieren Sie AWS Anmeldeinformationen mit Umgebungsvariablen

Umgebungsvariablen sind Variablen, die vom Betriebssystem gepflegt und von Systembefehlen verwendet werden. Sie werden nicht gespeichert, wenn Sie die SSH-Sitzung schließen. Die IDT-FreeRTOS-Benutzeroberfläche verwendet die `AWS_SECRET_ACCESS_KEY` Umgebungsvariablen `AWS_ACCESS_KEY_ID` und, um Ihre Anmeldeinformationen zu speichern. AWS

Um diese Variablen auf Linux, macOS oder Unix festzulegen, verwenden Sie export:

```
export AWS_ACCESS_KEY_ID=<your_access_key_id>
export AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

In Windows können Sie die Variablen mit set festlegen:

```
set AWS_ACCESS_KEY_ID=<your_access_key_id>
set AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

Öffnen Sie die IDT-FreeRTOS-Benutzeroberfläche

Um die IDT-FreeRTOS-Benutzeroberfläche zu öffnen

1. Laden Sie eine unterstützte IDT-FreeRTOS-Version herunter und extrahieren Sie das heruntergeladene Archiv an einen Speicherort auf Ihrem Dateisystem, für den Sie Lese- und Schreibberechtigungen haben.
2. Führen Sie den folgenden Befehl aus, um zum IDT-FreeRTOS-Installationsverzeichnis zu navigieren:

   ```
   cd devicetester-extract-location/bin
   ```

3. Führen Sie den folgenden Befehl aus, um die IDT-FreeRTOS-Benutzeroberfläche zu öffnen:

Linux

```
.devicetestergui_linux_x86-64.exe
```

Windows

```
./devicetestergui_win_x64-64
```

macOS

```
./devicetestergui_mac_x86-64
```

> **Note**
>
> Gehen Sie auf einem Mac zu Systemeinstellungen -> Sicherheit und Datenschutz, damit Ihr System die Benutzeroberfläche ausführen kann. Wenn Sie die Tests ausführen, müssen Sie dies möglicherweise noch dreimal tun.

Die IDT-FreeRTOS-Benutzeroberfläche wird in Ihrem Standardbrowser geöffnet. Informationen zu unterstützten Browsern finden Sie unter. Verwenden Sie einen unterstützten Webbrowser

Erstellen Sie eine neue Konfiguration

Wenn Sie zum ersten Mal Benutzer sind, müssen Sie eine neue Konfiguration erstellen, um die JSON-Konfigurationsdateien einzurichten, die IDT-FreeRTOS zum Ausführen von Tests benötigt. Anschließend können Sie Tests ausführen oder die erstellte Konfiguration ändern.

Beispiele für die `userdata.json` Dateien `config.jsondevice.json`, und finden Sie unterErster Test Ihres Mikrocontroller-Boards. Ein Beispiel für die `resource.json` Datei, die nur für die Ausführung von Bluetooth Low Energy (BLE) -Tests verwendet wird, finden Sie unterFühren Sie Bluetooth Low Energy-Tests durch.

Um eine neue Konfiguration zu erstellen

1. Öffnen Sie in der IDT-FreeRTOS-Benutzeroberfläche das Navigationsmenü und wählen Sie dann Neue Konfiguration erstellen.

> **Important**
>
> Sie müssen Ihre AWS Anmeldeinformationen konfigurieren, bevor Sie die Benutzeroberfläche öffnen. Wenn Sie Ihre Anmeldeinformationen nicht konfiguriert haben, schließen Sie das IDT-FreeRTOS-UI-Browserfenster, folgen Sie den Schritten unterAnmeldeinformationen konfigurieren AWS, und öffnen Sie dann die IDT-FreeRTOS-Benutzeroberfläche erneut.

2. Folgen Sie dem Konfigurationsassistenten, um die IDT-Konfigurationseinstellungen einzugeben, die für die Durchführung von Qualifikationstests verwendet werden. Der Assistent konfiguriert die folgenden Einstellungen in JSON-Konfigurationsdateien, die sich im *`devicetester-extract-location`*`/config` Verzeichnis befinden.

   - AWS settings — Die AWS-Konto Informationen, die IDT-FreeRTOS verwendet, um AWS Ressourcen während Testläufen zu erstellen. Diese Einstellungen sind in der Datei konfiguriert. `config.json`
   - FreeRTOS-Repository — Der absolute Pfad zum FreeRTOS-Repository und zum portierten Code sowie die Art der Qualifikation, die Sie durchführen möchten. Diese Einstellungen sind in der Datei konfiguriert. `userdata.json`

     Sie müssen FreeRTOS für Ihr Gerät portieren, bevor Sie Qualifikationstests durchführen können. Weitere Informationen finden Sie im FreeRTOS Porting Guide
   - Build and Flash — Die Build- und Flash-Befehle für Ihre Hardware, mit denen IDT automatisch Tests auf Ihrem Board erstellen und flashen kann. Diese Einstellungen sind in der `userdata.json` Datei konfiguriert.
   - Geräte — Die Gerätepool-Einstellungen für die zu testenden Geräte. Diese Einstellungen werden in den `sku` Feldern `id` und und im `devices` Block für den Gerätepool in der `device.json` Datei konfiguriert.
   - Netzwerk — Die Einstellungen zum Testen der Netzwerkkommunikationsunterstützung für Ihre Geräte. Diese Einstellungen werden im `features` Block der `device.json` Datei und in den `testWifiConfig` Blöcken `clientWifiConfig` und in der `userdata.json` Datei konfiguriert.
   - Echo-Server — Die Echo-Server-Konfigurationseinstellungen für Secure-Socket-Tests. Diese Einstellungen sind in der `userdata.json` Datei konfiguriert.

Weitere Hinweise zur Echo-Serverkonfiguration finden Sie unterhttps://docs.aws.amazon.com/freertos/latest/portingguide/afr-echo-server.html.

- CMake— (Optional) Die Einstellungen für die Ausführung von CMake Build-Funktionstests. Diese Konfiguration ist nur erforderlich, wenn Sie sie CMake als Build-System verwenden. Diese Einstellungen sind in der `userdata.json` Datei konfiguriert.
- BLE — Die Einstellungen für die Durchführung von Bluetooth Low Energy-Funktionstests. Diese Einstellungen werden im `features` Block der `device.json` Datei und in der `resource.json` Datei konfiguriert.
- OTA — Die Einstellungen für die Ausführung von OTA-Funktionstests. Diese Einstellungen werden im `features` Block der `device.json` Datei und in der `userdata.json` Datei konfiguriert.

3. Überprüfen Sie auf der Seite „Überprüfen“ Ihre Konfigurationsinformationen.

Wenn Sie mit der Überprüfung Ihrer Konfiguration fertig sind, wählen Sie Tests ausführen aus, um Ihre Qualifizierungstests durchzuführen.

Ändern Sie eine bestehende Konfiguration

Wenn Sie bereits Konfigurationsdateien für IDT eingerichtet haben, können Sie die IDT-FreeRTOS-Benutzeroberfläche verwenden, um Ihre bestehende Konfiguration zu ändern. Stellen Sie sicher, dass Ihre vorhandenen Konfigurationsdateien im Verzeichnis verfügbar sind. *`devicetester-extract-location`*`/config`

Um eine neue Konfiguration zu ändern

1. Öffnen Sie in der IDT-FreeRTOS-Benutzeroberfläche das Navigationsmenü und wählen Sie dann Bestehende Konfiguration bearbeiten aus.

   Das Konfigurations-Dashboard zeigt Informationen zu Ihren vorhandenen Konfigurationseinstellungen an. Wenn eine Konfiguration falsch oder nicht verfügbar ist, lautet der Status für diese Konfiguration`Error validating configuration`.

2. Gehen Sie wie folgt vor, um eine bestehende Konfigurationseinstellung zu ändern:

   a. Wählen Sie den Namen einer Konfigurationseinstellung, um die zugehörige Einstellungsseite zu öffnen.

b. Ändern Sie die Einstellungen und wählen Sie dann Speichern, um die entsprechende Konfigurationsdatei neu zu generieren.

Nachdem Sie die Änderung Ihrer Konfiguration abgeschlossen haben, stellen Sie sicher, dass alle Ihre Konfigurationseinstellungen die Validierung bestanden haben. Wenn der Status für jede Konfigurationseinstellung lautet`Valid`, können Sie Ihre Qualifizierungstests mit dieser Konfiguration ausführen.

Führen Sie Qualifizierungstests durch

Nachdem Sie eine Konfiguration für IDT-FreerTOS erstellt haben, können Sie Ihre Qualifizierungstests ausführen.

Um Qualifizierungstests durchzuführen

1. Überprüfen Sie Ihre Konfiguration.
2. Wählen Sie im Navigationsmenü die Option Tests ausführen aus.
3. Um den Testlauf zu starten, wählen Sie Tests starten.

IDT-FreerTOS führt die Qualifikationstests aus und zeigt die Zusammenfassung des Testlaufs sowie alle Fehler in der Test Runner-Konsole an. Nach Abschluss des Testlaufs können Sie die Testergebnisse und Protokolle an den folgenden Orten einsehen:

- Die Testergebnisse befinden sich im *`devicetester-extract-location`*`/results/`*`execution-id`* Verzeichnis.
- Testprotokolle befinden sich im *`devicetester-extract-location`*`/results/`*`execution-id`*`/logs` Verzeichnis.

Weitere Informationen zu Testergebnissen und Protokollen finden Sie unter Sehen Sie sich die Ergebnisse von IDT for FreeRTOS an undDie IDT for FreeRTOS-Protokolle anzeigen.

Führen Sie Bluetooth Low Energy-Tests durch

In diesem Abschnitt wird beschrieben, wie Sie Bluetooth Low Energy-Tests mit AWS IoT Device Tester for FreeRTOS einrichten und ausführen.

Bluetooth-Tests sind für wichtige Kernqualifikationen nicht erforderlich. Wenn Sie Ihr Gerät nicht mit FreeRTOS Bluetooth-Unterstützung testen möchten, können Sie diese Einrichtung überspringen. Achten Sie darauf, dass die BLE-Funktion in device.json auf eingestellt ist. No

Voraussetzungen

- Folgen Sie den Anweisungen in Erster Test Ihres Mikrocontroller-Boards.
- Ein Raspberry Pi 4B oder 3B+. (Erforderlich zum Ausführen der Raspberry Pi BLE-Begleitanwendung)
- Ein MicroSD-Karten- und SD-Karten-Adapter für die Raspberry Pi-Software.

Einrichtung von Raspberry Pi

Um die BLE-Funktionen des zu testenden Geräts (DUT) zu testen, benötigen Sie einen Raspberry Pi Model 4B oder 3B+.

So konfigurieren Sie Ihren Raspberry Pi zum Ausführen von BLE-Tests

1. Laden Sie eines der benutzerdefinierten Yocto-Images herunter, das die für die Durchführung der Tests erforderliche Software enthält.

   - Bild für Raspberry Pi 4B
   - Bild für Raspberry Pi 3B+

   > **Note**
   >
   > Das Yocto-Image sollte nur zum Testen mit FreeRTOS und nicht AWS IoT Device Tester für andere Zwecke verwendet werden.

2. Stellen Sie das Yocto-Image auf der SD-Karte für Raspberry Pi bereit.

   - Verwenden Sie ein Tool zum Schreiben von SD-Karten wie Etcher, um die heruntergeladene Datei auf die SD-Karte zu flashen. *image-name*`.rpi-sd.img` Da das Betriebssystem-Image sehr groß ist, kann dieser Schritt einige Zeit in Anspruch nehmen. Werfen Sie die SD-Karte dann aus und setzen Sie die microSD-Karte in Ihren Raspberry Pi ein.

3. Konfigurieren Sie Ihren Raspberry Pi.

a. Für den ersten Systemstart empfehlen wir, den Raspberry Pi mit einem Monitor, einer Tastatur und einer Maus zu verbinden.

b. Schließen Sie Ihren Raspberry Pi an eine Micro-USB-Energiequelle an.

c. Melden Sie sich mit den Standard-Anmeldeinformationen an. Geben Sie für die Benutzer-ID **`root`** ein. Geben Sie für Passwort **`idtafr`** ein.

d. Verbinden Sie den Raspberry Pi über eine Ethernet- oder WLAN-Verbindung mit Ihrem Netzwerk.

   i. Um Ihren Raspberry Pi über WLAN zu verbinden, öffnen Sie `/etc/wpa_supplicant.conf` auf dem Raspberry Pi und fügen Sie Ihre WLAN-Anmeldeinformationen zur `Network`-Konfiguration hinzu.

```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=0
update_config=1

network={
        scan_ssid=1
        ssid="your-wifi-ssid"
        psk="your-wifi-password"
        }
```

   ii. Führen Sie `ifup wlan0` aus, um die WLAN-Verbindung zu starten. Es kann einige Minuten dauern, eine Verbindung mit Ihrem WLAN-Netzwerk herzustellen.

e. Führen Sie für eine Ethernet-Verbindung `ifconfig eth0` aus. Führen Sie für eine WLAN-Verbindung `ifconfig wlan0` aus. Notieren Sie sich die IP-Adresse, die als `inet addr` in der Befehlsausgabe angezeigt wird. Sie benötigen die IP-Adresse zu einem späteren Zeitpunkt in diesem Verfahren.

f. (Optional) Die Tests führen Befehle auf dem Raspberry Pi unter Verwendung der Standard-Anmeldeinformationen für das Yocto Image über SSH aus. Als zusätzliche Sicherheitsmaßnahme empfehlen wir, die Authentifizierung über öffentliche Schlüssel für SSH einzurichten und passwortbasiertes SSH zu deaktivieren.

   i. Erstellen Sie mithilfe des OpenSSL-Befehls `ssh-keygen` einen SSH-Schlüssel. Wenn Sie bereits ein SSK-Schlüsselpaar auf Ihrem Host-Computer haben, empfiehlt es sich, ein neues zu erstellen, damit AWS IoT Device Tester sich FreeRTOS auf Ihrem Raspberry Pi anmelden kann.

> **Note**
>
> Windows wird nicht mit einem installierten SSH-Client geliefert. Weitere Informationen zum Installieren eines SSH-Clients unter Windows finden Sie unter Download SSH Software.

ii. Der Befehl `ssh-keygen` fordert Sie auf, einen Namen und Pfad zum Speichern des Schlüsselpaars einzugeben. Standardmäßig werden die Schlüsselpaardateien `id_rsa` (privater Schlüssel) und `id_rsa.pub` (öffentlicher Schlüssel) genannt. Unter macOS und Linux ist der Standard-Speicherort dieser Dateien `~/.ssh/`. Unter Windows ist der Standard-Speicherort `C:\Users\`*`user-name`*.

iii. Wenn Sie zur Eingabe einer Schlüsselphrase aufgefordert werden, drücken Sie einfach die Eingabetaste, um fortzufahren.

iv. Um Ihren SSH-Schlüssel zu Ihrem Raspberry Pi hinzuzufügen, damit AWS IoT Device Tester sich FreeRTOS auf dem Gerät anmelden kann, verwenden Sie den `ssh-copy-id` Befehl von Ihrem Host-Computer. Dieser Befehl fügt Ihren öffentlichen Schlüssel zur Datei `~/.ssh/authorized_keys` auf Ihrem Raspberry Pi hinzu.

```
ssh-copy-id root@raspberry-pi-ip-address
```

v. Wenn Sie zur Eingabe eines Passworts aufgefordert werden, geben Sie **`idtafr`** ein. Dies ist das Standard-Passwort für das Yocto Image.

> **Note**
>
> Der Befehl `ssh-copy-id` setzt voraus, dass der öffentliche Schlüssel die Bezeichnung `id_rsa.pub` trägt. Unter macOS und Linux ist der Standard-Speicherort `~/.ssh/`. Unter Windows ist der Standard-Speicherort `C:\Users\`*`user-name`*`\.ssh`. Wenn Sie dem öffentlichen Schlüssel einen anderen Namen gegeben oder ihn an einem anderen Ort gespeichert haben, müssen Sie den vollqualifizierten Pfad zu Ihrem öffentlichen SSH-Schlüssel mit der Option `-i` zu `ssh-copy-id` (z. B. `ssh-copy-id -i ~/my/path/myKey.pub`) angeben. Weitere Informationen zum Erstellen von SSH-Schlüsseln und Kopieren von öffentlichen Schlüsseln finden Sie unter SSH-COPY-ID.

vi. Um zu testen, ob die Authentifizierung über öffentliche Schlüssel funktioniert, führen Sie `ssh -i` *`/my/path/myKey`* `root@`*`raspberry-pi-device-ip`* aus.

Wenn Sie nicht zur Eingabe eines Passworts aufgefordert werden, funktioniert Ihre Authentifizierung über öffentliche Schlüssel.

vii. Stellen Sie sicher, dass Sie sich mit einem öffentlichen Schlüssel bei Ihrem Raspberry Pi anmelden können, und deaktivieren Sie anschließend das passwortbasierte SSH.

A. Bearbeiten Sie die Datei `/etc/ssh/sshd_config` auf dem Raspberry Pi.

B. Legen Sie für das Attribut `PasswordAuthentication` no fest.

C. Speichern und schließen Sie die Datei `sshd_config`.

D. Laden Sie den SSH-Server erneut, indem Sie `/etc/init.d/sshd reload` ausführen.

g. Erstellen Sie eine `resource.json`-Datei.

i. Erstellen Sie in dem Verzeichnis, in das Sie AWS IoT Device Tester extrahiert haben, eine Datei mit dem Namen. `resource.json`

ii. Fügen Sie der Datei die folgenden Informationen über Ihren Raspberry Pi hinzu und *`rasp-pi-ip-address`* ersetzen Sie sie durch die IP-Adresse Ihres Raspberry Pi.

```
[
    {
        "id": "ble-test-raspberry-pi",
        "features": [
            {"name":"ble", "version":"4.2"}
        ],
        "devices": [
            {
                "id": "ble-test-raspberry-pi-1",
                "connectivity": {
                    "protocol": "ssh",
                    "ip": "rasp-pi-ip-address"
                }
            }
        ]
    }
]
```

iii. Wenn Sie sich nicht dafür entschieden haben, die Authentifizierung mit öffentlichem Schlüssel für SSH zu verwenden, fügen Sie dem `connectivity` Abschnitt der `resource.json` Datei Folgendes hinzu.

```
"connectivity": {
    "protocol": "ssh",
    "ip": "rasp-pi-ip-address",
    "auth": {
        "method": "password",
        "credentials": {
            "user": "root",
            "password": "idtafr"
        }
    }
}
```

iv. (Optional) Wenn Sie die Authentifizierung über öffentliche Schlüssel für SSH verwenden möchten, fügen Sie den folgenden `connectivity`-Abschnitt der `resource.json`-Datei hinzu.

```
"connectivity": {
    "protocol": "ssh",
    "ip": "rasp-pi-ip-address",
    "auth": {
        "method": "pki",
        "credentials": {
            "user": "root",
            "privKeyPath": "location-of-private-key"
        }
    }
}
```

FreeRTOS-Geräteeinrichtung

Legen Sie für die `BLE`-Funktion in Ihrer `device.json`-Datei `Yes` fest. Wenn Sie mit einer `device.json`-Datei von einem Zeitpunkt, bevor Bluetooth-Tests verfügbar waren, starten, müssen Sie die Funktion für BLE zum `features`-Array hinzufügen:

```
{
    ...
```

```
    "features": [
        {
            "name": "BLE",
            "value": "Yes"
        },
    ...
}
```

Führen Sie die BLE-Tests aus

Nachdem Sie die BLE-Funktion in `device.json` aktiviert haben, werden die BLE-Tests ausgeführt, wenn Sie `devicetester_`*`[linux | mac | win_x86-64]`* `run-suite` ohne Angabe einer Gruppen-ID ausführen.

Wenn Sie die BLE-Tests separat ausführen möchten, können Sie die Gruppen-ID für BLE angeben: `devicetester_`*`[linux | mac | win_x86-64]`* `run-suite --userdata` *`path-to-userdata`*`/userdata.json --group-id FullBLE`.

Um eine möglichst zuverlässige Leistung zu gewährleisten, platzieren Sie Ihren Raspberry Pi in der Nähe des zu prüfenden Geräts (Device under Test, DUT).

Beheben Sie Fehler bei BLE-Tests

Stellen Sie sicher, dass Sie die unter Erster Test Ihres Mikrocontroller-Boards aufgeführten Schritte befolgt haben. Wenn andere Tests als BLE fehlschlagen, liegt das Problem mit hoher Wahrscheinlichkeit nicht an der Bluetooth-Konfiguration.

Führen Sie die FreeRTOS Qualification Suite aus

Sie verwenden die ausführbare Datei AWS IoT Device Tester for FreeRTOS, um mit IDT for FreeRTOS zu interagieren. In den folgenden Befehlszeilenbeispielen wird veranschaulicht, wie Sie die Qualifikationsprüfungen für einen Gerätepool (Satz identischer Geräte) durchführen.

IDT v3.0.0 and later

```
devicetester_[linux | mac | win] run-suite  \
    --suite-id suite-id  \
    --group-id group-id  \
    --pool-id your-device-pool \
    --test-id test-id  \
    --upgrade-test-suite y|n  \
```

```
    --update-idt y|n  \
    --update-managed-policy y|n  \
    --userdata userdata.json
```

Führt eine Reihe von Tests in einem Pool von Geräten aus. Die Datei `userdata.json` muss sich im Verzeichnis *devicetester_extract_location*`/devicetester_afreertos_`*[win|mac|linux]*`/configs/` befinden.

> **Note**
>
> Wenn Sie IDT for FreeRTOS unter Windows ausführen, verwenden Sie Schrägstriche (/), um den Pfad zur Datei anzugeben. `userdata.json`

Verwenden Sie den folgenden Befehl zum Ausführen einer bestimmten Testgruppe:

```
devicetester_[linux | mac | win] run-suite  \
    --suite-id FRQ_1.0.1  \
    --group-id group-id  \
    --pool-id pool-id  \
    --userdata userdata.json
```

Die Parameter `suite-id` und `pool-id` sind optional, wenn Sie eine einzige Testsuite in einem einzigen Gerätepool ausführen (d. h. in Ihrer `device.json`-Datei ist nur ein Gerätepool definiert).

Verwenden Sie den folgenden Befehl zum Ausführen eines bestimmten Testfalls in einer Testgruppe:

```
devicetester_[linux | mac | win_x86-64] run-suite  \
    --group-id group-id  \
    --test-id test-id
```

Mit dem Befehl `list-test-cases` können Sie die Testfälle in einer Testgruppe auflisten.

IDT für FreeRTOS-Befehlszeilenoptionen

group-id

(Optional) Die auszuführenden Testgruppen als kommagetrennte Liste. Bei fehlender Angabe führt IDT alle Testgruppen in der Testsuite aus.

pool-id

(Optional) Der zu testende Gerätepool. Dies ist erforderlich, wenn Sie mehrere Gerätepools in `device.json` definieren. Wenn Sie nur einen Gerätepool haben, können Sie diese Option weglassen.

suite-id

(Optional) Die auszuführende Test-Suite-Version. Falls nicht angegeben, verwendet IDT die neueste Version im Verzeichnis der Tests auf Ihrem System.

> **Note**
>
> Ab IDT v3.0.0 sucht IDT online nach neueren Testsuiten. Weitere Informationen finden Sie unter Test-Suite-Versionen.

test-id

(Optional) Die auszuführenden Tests als kommagetrennte Liste. Wenn angegeben, muss `group-id` eine einzelne Gruppe angeben.

Example

```
devicetester_[linux | mac | win_x86-64] run-suite --group-id mqtt --test-id
 mqtt_test
```

update-idt

(Optional) Wenn dieser Parameter nicht festgelegt ist und eine neuere IDT-Version verfügbar ist, werden Sie aufgefordert, IDT zu aktualisieren. Wenn dieser Parameter auf gesetzt ist`Y`, stoppt IDT die Testausführung, wenn festgestellt wird, dass eine neuere Version verfügbar ist. Wenn dieser Parameter auf gesetzt ist`N`, setzt IDT die Testausführung fort.

update-managed-policy

(Optional) Wenn dieser Parameter nicht verwendet wird und IDT feststellt, dass Ihre verwaltete Richtlinie nicht verwendet wird up-to-date, werden Sie aufgefordert, Ihre verwaltete Richtlinie zu aktualisieren. Wenn dieser Parameter auf gesetzt ist`Y`, stoppt IDT die Testausführung, wenn festgestellt wird, dass Ihre verwaltete Richtlinie dies nicht tut. up-to-date Wenn dieser Parameter auf gesetzt ist`N`, setzt IDT die Testausführung fort.

upgrade-test-suite

(Optional) Wenn diese Version nicht verwendet wird und eine neuere Test-Suite-Version verfügbar ist, werden Sie aufgefordert, sie herunterzuladen. Um die Eingabeaufforderung auszublenden, geben Sie y an, damit immer die neueste Testsuite heruntergeladen wird, oder n, damit die angegebene Testsuite oder die neueste Version auf Ihrem System verwendet wird.

Example

Beispiel

Verwenden Sie den folgenden Befehl, um immer die neueste Testsuite herunterzuladen und zu verwenden.

```
devicetester_[linux | mac | win_x86-64] run-suite --userdata userdata file --
group-id group ID --upgrade-test-suite y
```

Verwenden Sie den folgenden Befehl, um die neueste Testsuite auf Ihrem System zu verwenden.

```
devicetester_[linux | mac | win_x86-64] run-suite --userdata userdata file --
group-id group ID --upgrade-test-suite n
```

h

Verwenden Sie die Hilfe-Option, um mehr über `run-suite`-Optionen zu erfahren.

Example

Beispiel

```
devicetester_[linux | mac | win_x86-64] run-suite -h
```

IDT v1.7.0 and earlier

```
devicetester_[linux | mac | win] run-suite  \
    --suite-id suite-id  \
    --pool-id your-device-pool  \
```

```
    --userdata userdata.json
```

Die Datei `userdata.json` sollte sich im Verzeichnis *devicetester_extract_location*`/devicetester_afreertos_`*[win|mac|linux]*`/configs/` befinden.

> **Note**
>
> Wenn Sie IDT for FreeRTOS unter Windows ausführen, verwenden Sie Schrägstriche (/), um den Pfad zur Datei anzugeben. `userdata.json`

Verwenden Sie den folgenden Befehl zum Ausführen einer bestimmten Testgruppe.

```
devicetester_[linux | mac | win] run-suite  \
    --suite-id FRQ_1 --group-id group-id  \
    --pool-id pool-id  \
    --userdata userdata.json
```

`suite-id` und `pool-id` sind optional, wenn Sie eine einzige Testsuite in einem einzigen Gerätepool ausführen (d. h. Sie haben nur einen Gerätepool in Ihrer `device.json`-Datei definiert).

IDT für FreeRTOS-Befehlszeilenoptionen

group-id

(Optional) Gibt die Testgruppe an.

pool-id

Gibt den Gerätepool an, der getestet werden soll. Wenn Sie nur einen Gerätepool haben, können Sie diese Option weglassen.

suite-id

(Optional) Gibt die Testsuite an, die ausgeführt werden soll.

IDT für FreeRTOS-Befehle

Der Befehl IDT for FreeRTOS unterstützt die folgenden Operationen:

IDT v3.0.0 and later

`help`

Listet Informationen über den angegebenen Befehl auf.

`list-groups`

Listet die Gruppen in der jeweiligen Suite auf.

`list-suites`

Listet die verfügbaren Suites auf.

`list-supported-products`

Listet die unterstützten Produkte und Testsuiteversionen auf.

`list-supported-versions`

Listet die FreeRTOS- und Testsuite-Versionen auf, die von der aktuellen IDT-Version unterstützt werden.

`list-test-cases`

Listet die Testfälle in einer angegebenen Gruppe auf.

`run-suite`

Führt eine Reihe von Tests in einem Pool von Geräten aus.

Verwenden Sie die Option `--suite-id`, um eine Test-Suite-Version anzugeben, oder lassen Sie sie weg, um die neueste Version auf Ihrem System zu verwenden.

Verwenden Sie die `--test-id` um einen einzelnen Testfall auszuführen.

Example

```
devicetester_[linux | mac | win_x86-64] run-suite --group-id mqtt --test-id
 mqtt_test
```

Eine vollständige Liste der Optionen finden Sie unter Führen Sie die FreeRTOS Qualification Suite aus.

> **Note**
>
> Ab IDT v3.0.0 sucht IDT online nach neueren Testsuiten. Weitere Informationen finden Sie unter Test-Suite-Versionen.

IDT v1.7.0 and earlier

`help`

Listet Informationen über den angegebenen Befehl auf.

`list-groups`

Listet die Gruppen in der jeweiligen Suite auf.

`list-suites`

Listet die verfügbaren Suites auf.

`run-suite`

Führt eine Reihe von Tests in einem Pool von Geräten aus.

Tests für erneute Qualifikationen

Sobald neue Versionen von IDT for FreeRTOS Qualifizierungstests veröffentlicht werden oder wenn Sie Ihre platinenspezifischen Pakete oder Gerätetreiber aktualisieren, können Sie IDT for FreeRTOS verwenden, um Ihre Mikrocontroller-Boards zu testen. Stellen Sie für nachfolgende Qualifikationen sicher, dass Sie über die neuesten Versionen von FreeRTOS und IDT for FreeRTOS verfügen, und führen Sie die Qualifizierungstests erneut durch.

Sehen Sie sich die Ergebnisse von IDT for FreeRTOS an

Während der Ausführung schreibt IDT Fehler in die Konsole, Protokolldateien und Testberichte. Nachdem IDT die Qualifikations-Testsuite abgeschlossen hat, schreibt es eine Zusammenfassung der Testläufe in die Konsole und erstellt zwei Testberichte. Diese Berichte befinden sich in *`devicetester-extract-location`*`/results/`*`execution-id`*`/`. Beide Berichte erfassen die Ergebnisse von der Ausführung der Qualifikations-Testsuite.

Dies `awsiotdevicetester_report.xml` ist der Qualifizierungstestbericht, den Sie einreichen AWS , um Ihr Gerät im AWS Partner-Gerätekatalog aufzulisten. Die Bericht enthält die folgenden Elemente:

- Die Version IDT für FreeRTOS.
- Die getestete FreeRTOS-Version.
- Die Funktionen von FreeRTOS, die vom Gerät unterstützt werden, basieren auf den bestandenen Tests.
- SKU und Gerätename, die in der `device.json`-Datei angegeben wurden.
- Die Funktionen des Geräts, das in der `device.json`-Datei angegeben wurde.
- Die aggregierte Zusammenfassung der Ergebnisse der Testfälle.
- Eine Aufschlüsselung der Testfallergebnisse nach Bibliotheken, die auf der Grundlage der Gerätefunktionen getestet wurden (FullWiFiz. B. fullMQTT usw.).
- Ob diese Qualifikation von FreeRTOS für Version 202012.00 gilt, die LTS-Bibliotheken verwendet.

Das `FRQ_Report.xml` ist ein Bericht im Standard-XML-Format. JUnit Sie können ihn in CI/CD Plattformen wie Jenkins, Bamboo usw. integrieren. Die Bericht enthält die folgenden Elemente:

- Eine aggregierte Zusammenfassung der Ergebnisse der Testfälle.
- Eine Aufschlüsselung der Testfallergebnisse nach Bibliotheken, die basierend auf den Geräteeigenschaften getestet wurden.

Interpretieren Sie die IDT for FreeRTOS-Ergebnisse

Der Berichtsabschnitt in `awsiotdevicetester_report.xml` oder `FRQ_Report.xml` listet die Ergebnisse der durchgeführten Tests auf.

Im ersten XML-Tag `<testsuites>` ist die Gesamtzusammenfassung der Testausführung enthalten. Beispiel:

```
<testsuites name="FRQ results" time="5633" tests="184" failures="0"
errors="0" disabled="0">
```

Im **`<testsuites>`** Tag verwendete Attribute

`name`

Name der Testsuite

`time`

Zeit (in Sekunden), die zur Ausführung der Qualifikations-Suite erforderlich war

`tests`

Anzahl der ausgeführten Testfälle

`failures`

Anzahl der ausgeführten Testfälle, die den Test nicht bestanden haben

`errors`

Die Anzahl der Testfälle, die IDT für FreeRTOS nicht ausführen konnte.

`disabled`

Dieses Attribut wird nicht verwendet und kann ignoriert werden.

Wenn es keine Testfallausfälle oder Fehler gibt, erfüllt Ihr Gerät die technischen Voraussetzungen für die Ausführung von FreeRTOS und kann mit Diensten zusammenarbeiten. AWS IoT Wenn Sie Ihr Gerät im AWS Partner-Gerätekatalog auflisten möchten, können Sie diesen Bericht als Qualifikationsnachweis verwenden.

Falls bei Testfällen Fehler auftreten, können Sie den fehlgeschlagenen Testfall identifizieren, indem Sie die XML-Tags von `<testsuites>` überprüfen. Die XML-Tags von `<testsuite>` im `<testsuites>`-Tag zeigen die Ergebniszusammenfassung des Testfalls für eine Testgruppe.

```
<testsuite name="FullMQTT" package="" tests="16" failures="0" time="76"
disabled="0" errors="0" skipped="0">
```

Das Format ähnelt dem `<testsuites>`-Tag, weist aber ein zusätzliches Attribut mit dem Namen `skipped` auf, das nicht verwendet wird und ignoriert werden kann. Innerhalb der einzelnen XML-Tags von `<testsuite>` befinden sich `<testcase>`-Tags für alle Testfälle, die für eine Testgruppe ausgeführt wurden. Beispiel:

```
<testcase classname="mcu.Full_MQTT" name="AFQP_MQTT_Connect_HappyCase"
attempts="1"></testcase>
```

Im **`<awsproduct>`** Tag verwendete Attribute

`name`

Der Name des getesteten Produkts.

`version`

Die Version des getesteten Produkts.

`sdk`

Wenn Sie IDT mit einem SDK ausgeführt haben, enthält dieser Block den Namen und die Version Ihres SDK. Wenn Sie IDT nicht mit einem SDK ausgeführt haben, enthält dieser Block:

```
<sdk>
    <name>N/A</vame>
    <version>N/A</version>
</sdk>
```

`features`

Die validierten Funktionen Als `required` gekennzeichnete Funktionen sind für die Einreichung Ihres Boards für die Qualifizierung erforderlich. Der folgende Ausschnitt zeigt, wie dies in der Datei aussieht. `awsiotdevicetester_report.xml`

```
<feature name="core-freertos" value="not-supported" type="required"></feature>
```

Als `optional` gekennzeichnete Funktionen sind für die Qualifizierung nicht erforderlich. Die folgenden Codeausschnitte zeigen optionale Funktionen.

```
<feature name="ota-dataplane-mqtt" value="not-supported" type="optional"></feature>
<feature name="ota-dataplane-http" value="not-supported" type="optional"></feature>
```

Wenn es keine Testfehler oder Fehler für die erforderlichen Funktionen gibt, erfüllt Ihr Gerät die technischen Voraussetzungen für die Ausführung von FreeRTOS und kann mit Diensten zusammenarbeiten. AWS IoT Wenn Sie Ihr Gerät im AWS Partner-Gerätekatalog auflisten möchten, können Sie diesen Bericht als Qualifikationsnachweis verwenden.

Falls bei Tests Fehler auftreten, können Sie den fehlgeschlagenen Test identifizieren, indem Sie die XML-Tags von `<testsuites>` überprüfen. Die XML-Tags von `<testsuite>` im `<testsuites>`-Tag zeigen die Ergebniszusammenfassung eines Tests für eine Testgruppe. Beispiel:

```
<testsuite name="FreeRTOSVersion" package="" tests="1" failures="1" time="2"
 disabled="0" errors="0" skipped="0">
```

Das Format ähnelt dem `<testsuites>` Tag, hat jedoch ein `skipped` Attribut, das nicht verwendet wird und ignoriert werden kann. Innerhalb der einzelnen `<testsuite>`-XML-Tags befinden sich `<testcase>`-Tags für alle ausgeführten Tests einer Testgruppe. Beispiel:

```
<testcase classname="FreeRTOSVersion" name="FreeRTOSVersion"></testcase>
```

`lts`

True, wenn Sie sich für eine Version von FreeRTOS qualifizieren, die LTS-Bibliotheken verwendet, andernfalls false.

Im Tag verwendete Attribute **`<testcase>`**

`name`

Name des Testfalls

`attempts`

Gibt an, wie oft IDT für FreeRTOS den Testfall ausgeführt hat.

Wenn ein Testfall fehlschlägt oder ein Fehler auftritt, werden `<failure>`- oder `<error>`-Tags hinzugefügt, um das `<testcase>`-Tag mit Informationen für die Fehlerbehebung zu versehen. Beispiel:

```
<testcase classname="mcu.Full_MQTT" name="AFQP_MQTT_Connect_HappyCase">
    <failure type="Failure">Reason for the test case failure</failure>
    <error>Reason for the test case execution error</error>
</testcase>
```

Weitere Informationen finden Sie unter Beheben von Fehlern in der .

Die IDT for FreeRTOS-Protokolle anzeigen

Logs, die IDT for FreeRTOS bei der Testausführung generiert, finden Sie unter. *`devicetester-extract-location`*`/results/`*`execution-id`*`/logs` Es werden zwei Protokollgruppen generiert:

`test_manager.log`

Enthält von IDT für FreeRTOS generierte Protokolle (z. B. Protokolle zur Konfiguration und Berichtserstellung).

***`test_group_id`*`__`*`test_case_id`*`.log`** (z. B. **`FullMQTT__Full_MQTT.log`**)

Die Protokolldatei für einen Testfall, einschließlich der Ausgabe des zu testenden Geräts. Die Protokolldatei wird nach der Testgruppe und dem ausgeführten Testfall benannt.

Entwickeln und betreiben Sie Ihre eigenen IDT-Testsuiten

Ab IDT v4.0.0 kombiniert IDT for FreeRTOS ein standardisiertes Konfigurations-Setup und ein Ergebnisformat mit einer Testsuite-Umgebung, mit der Sie benutzerdefinierte Testsuiten für Ihre Geräte und Gerätesoftware entwickeln können. Sie können benutzerdefinierte Tests für Ihre eigene interne Validierung hinzufügen oder sie Ihren Kunden zur Geräteverifizierung zur Verfügung stellen.

Verwenden Sie IDT, um benutzerdefinierte Testsuiten wie folgt zu entwickeln und auszuführen:

Um benutzerdefinierte Testsuiten zu entwickeln

- Erstellen Sie Testsuiten mit benutzerdefinierter Testlogik für das Gerät, das Sie testen möchten.
- Stellen Sie IDT Ihre benutzerdefinierten Testsuiten für Testläufer zur Verfügung. Fügen Sie Informationen zu bestimmten Einstellungskonfigurationen für Ihre Testsuiten hinzu.

Um benutzerdefinierte Testsuiten auszuführen

- Richten Sie das Gerät ein, das Sie testen möchten.
- Implementieren Sie die Einstellungskonfigurationen entsprechend den Anforderungen der Testsuiten, die Sie verwenden möchten.
- Verwenden Sie IDT, um Ihre benutzerdefinierten Testsuiten auszuführen.
- Sehen Sie sich die Testergebnisse und Ausführungsprotokolle für die von IDT ausgeführten Tests an.

Laden Sie die neueste Version von AWS IoT Device Tester for FreeRTOS herunter

Laden Sie die neueste Version von IDT herunter und extrahieren Sie die Software an einen Speicherort in Ihrem Dateisystem, für den Sie Lese- und Schreibberechtigungen haben.

> **Note**
>
> IDT unterstützt nicht die Ausführung durch mehrere Benutzer von einem gemeinsam genutzten Speicherort aus, z. B. einem NFS-Verzeichnis oder einem freigegebenen Windows-Netzwerkordner. Es wird empfohlen, das IDT-Paket auf ein lokales Laufwerk zu extrahieren und die IDT-Binärdatei auf Ihrer lokalen Workstation auszuführen.
> Windows hat eine Pfadlängenbegrenzung von 260 Zeichen. Wenn Sie Windows verwenden, extrahieren Sie IDT in ein Stammverzeichnis wie `C:\` oder `D:\`, um Ihre Pfade unter der Grenze von 260 Zeichen zu halten.

Arbeitsablauf der Testsuite

Testsuiten bestehen aus drei Arten von Dateien:

- Konfigurationsdateien, die IDT Informationen zur Ausführung der Testsuite liefern.
- Testen Sie ausführbare Dateien, die IDT zum Ausführen von Testfällen verwendet.
- Zusätzliche Dateien, die zum Ausführen von Tests erforderlich sind.

Führen Sie die folgenden grundlegenden Schritte aus, um benutzerdefinierte IDT-Tests zu erstellen:

1. Erstellen Sie Konfigurationsdateien für Ihre Testsuite.
2. Erstellen Sie ausführbare Testfalldateien, die die Testlogik für Ihre Testsuite enthalten.
3. Überprüfen und dokumentieren Sie die Konfigurationsinformationen, die Testläufer benötigen, um die Testsuite auszuführen.
4. Stellen Sie sicher, dass IDT Ihre Testsuite ausführen und die Testergebnisse erwartungsgemäß liefern kann.

Folgen Sie den Anweisungen unter, um schnell eine benutzerdefinierte Beispielsuite zu erstellen und auszuführen. Tutorial: Erstellen Sie die IDT-Testsuite als Beispiel und führen Sie sie aus

Erste Schritte zum Erstellen einer benutzerdefinierten Testsuite in Python finden Sie unterTutorial: Entwickeln Sie eine einfache IDT-Testsuite.

Tutorial: Erstellen Sie die IDT-Testsuite als Beispiel und führen Sie sie aus

Der AWS IoT Device Tester Download enthält den Quellcode für eine Beispiel-Testsuite. Sie können dieses Tutorial abschließen, um die Beispiel-Testsuite zu erstellen und auszuführen, um zu verstehen, wie Sie AWS IoT Device Tester for FreeRTOS verwenden können, um benutzerdefinierte Testsuiten auszuführen. Obwohl dieses Tutorial SSH verwendet, ist es hilfreich zu lernen, wie man es AWS IoT Device Tester mit FreeRTOS-Geräten verwendet.

In diesem Tutorial werden Sie die folgenden Schritte ausführen:

1. Erstellen Sie die Beispiel-Testsuite
2. Verwenden Sie IDT, um die Beispieltestsuite auszuführen

Themen

- Richten Sie die Voraussetzungen für die Beispieltestsuite ein
- Geräteinformationen für IDT konfigurieren
- Erstellen Sie die Beispiel-Testsuite
- Verwenden Sie IDT, um die Beispieltestsuite auszuführen
- Beheben von Fehlern

Richten Sie die Voraussetzungen für die Beispieltestsuite ein

Zum Durcharbeiten dieses Tutorials ist Folgendes erforderlich:

- Anforderungen an den Host-Computer
  - Aktuelle Version von AWS IoT Device Tester
  - Python 3.7 oder höher

    Führen Sie den folgenden Befehl aus, um die auf Ihrem Computer installierte Version von Python zu überprüfen:

    ```
    python3 --version
    ```

    Wenn die Verwendung dieses Befehls unter Windows einen Fehler zurückgibt, verwenden Sie `python --version` stattdessen. Wenn die zurückgegebene Versionsnummer 3.7 oder höher

ist, führen Sie den folgenden Befehl in einem Powershell-Terminal aus, um ihn `python3` als Alias für Ihren `python` Befehl festzulegen.

```
Set-Alias -Name "python3" -Value "python"
```

Wenn keine Versionsinformationen zurückgegeben werden oder wenn die Versionsnummer kleiner als 3.7 ist, folgen Sie den Anweisungen unter Python herunterladen, um Python 3.7+ zu installieren. Weitere Informationen finden Sie in der Python-Dokumentation.

- urllib3

  Führen Sie den folgenden Befehl aus, um zu überprüfen, ob die Installation korrekt `urllib3` ist:

  ```
  python3 -c 'import urllib3'
  ```

  Wenn `urllib3` es nicht installiert ist, führen Sie den folgenden Befehl aus, um es zu installieren:

  ```
  python3 -m pip install urllib3
  ```

- Anforderungen an Speichergeräte
  - Ein Gerät mit einem Linux-Betriebssystem und einer Netzwerkverbindung zu demselben Netzwerk wie Ihr Host-Computer.

    Wir empfehlen Ihnen, einen Raspberry Pi mit Raspberry Pi OS zu verwenden. Stellen Sie sicher, dass Sie SSH auf Ihrem Raspberry Pi eingerichtet haben, um eine Remoteverbindung herzustellen.

Geräteinformationen für IDT konfigurieren

Konfigurieren Sie Ihre Geräteinformationen für IDT, um den Test durchzuführen. Sie müssen die `device.json` Vorlage im *`<device-tester-extract-location>`*`/configs` Ordner mit den folgenden Informationen aktualisieren.

```
[
  {
    "id": "pool",
    "sku": "N/A",
    "devices": [
      {
```

```
        "id": "<device-id>",
        "connectivity": {
          "protocol": "ssh",
          "ip": "<ip-address>",
          "port": "<port>",
          "auth": {
            "method": "pki | password",
            "credentials": {
              "user": "<user-name>",
              "privKeyPath": "/path/to/private/key",
              "password": "<password>"
            }
          }
        }
      }
    ]
  }
]
```

Geben Sie im `devices` Objekt die folgenden Informationen ein:

`id`

Eine benutzerdefinierte eindeutige Kennung für Ihr Gerät.

`connectivity.ip`

Die IP-Adresse Ihres Geräts.

`connectivity.port`

Optional. Die Portnummer, die für SSH-Verbindungen zu Ihrem Gerät verwendet werden soll.

`connectivity.auth`

Authentifizierungsinformationen für die Verbindung.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

`connectivity.auth.method`

Die Authentifizierungsmethode, die für den Zugriff über ein bestimmtes Verbindungsprotokoll auf ein Gerät verwendet wird.

Unterstützte Werte sind:

- `pki`
- `password`

`connectivity.auth.credentials`

Die für die Authentifizierung verwendeten Anmeldeinformationen.

`connectivity.auth.credentials.user`

Der Benutzername, mit dem Sie sich auf Ihrem Gerät angemeldet haben.

`connectivity.auth.credentials.privKeyPath`

Der vollständige Pfad zu dem privaten Schlüssel, mit dem Sie sich auf Ihrem Gerät angemeldet haben.

Dieser Wert gilt nur, wenn `connectivity.auth.method` auf `pki` festgelegt ist.

`devices.connectivity.auth.credentials.password`

Das Passwort, mit dem Sie sich auf Ihrem Gerät angemeldet haben.

Dieser Wert gilt nur, wenn `connectivity.auth.method` auf `password` festgelegt ist.

> **Note**
>
> Geben Sie `privKeyPath` nur an, wenn `method` auf `pki` gesetzt ist.
> Geben Sie `password` nur an, wenn `method` auf `password` gesetzt ist.

Erstellen Sie die Beispiel-Testsuite

Der *`<device-tester-extract-location>`*`/samples/python` Ordner enthält Beispielkonfigurationsdateien, Quellcode und das IDT Client SDK, die Sie mithilfe der bereitgestellten Build-Skripten zu einer Testsuite kombinieren können. Die folgende Verzeichnisstruktur zeigt den Speicherort dieser Beispieldateien:

```
<device-tester-extract-location>
### ...
### tests
### samples
#   ### ...
#   ### python
```

```
#       ### configuration
#       ### src
#       ### build-scripts
#           ### build.sh
#           ### build.ps1
### sdks
    ### ...
    ### python
        ### idt_client
```

Um die Testsuite zu erstellen, führen Sie die folgenden Befehle auf Ihrem Host-Computer aus:

Windows

```
cd <device-tester-extract-location>/samples/python/build-scripts
./build.ps1
```

Linux, macOS, or UNIX

```
cd <device-tester-extract-location>/samples/python/build-scripts
./build.sh
```

Dadurch wird die Beispiel-Testsuite in dem `IDTSampleSuitePython_1.0.0` Ordner innerhalb des *<device-tester-extract-location>*`/tests` Ordners erstellt. Sehen Sie sich die Dateien im `IDTSampleSuitePython_1.0.0` Ordner an, um zu verstehen, wie die Beispieltestsuite strukturiert ist, und um verschiedene Beispiele für ausführbare Testfälle und Testkonfigurationsdateien zu sehen.

> **Note**
>
> Die Beispieltestsuite enthält Python-Quellcode. Nehmen Sie keine vertraulichen Informationen in Ihren Testsuite-Code auf.

Nächster Schritt: Verwenden Sie IDT, um die von Ihnen erstellte Beispiel-Testsuite auszuführen.

Verwenden Sie IDT, um die Beispieltestsuite auszuführen

Führen Sie die folgenden Befehle auf Ihrem Host-Computer aus, um die Beispiel-Testsuite auszuführen:

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id IDTSampleSuitePython
```

IDT führt die Beispieltestsuite aus und streamt die Ergebnisse an die Konsole. Wenn der Test abgeschlossen ist, sehen Sie die folgenden Informationen:

```
========== Test Summary ==========
Execution Time:         5s
Tests Completed:        4
Tests Passed:           4
Tests Failed:           0
Tests Skipped:          0
---------------------------------
Test Groups:
    sample_group:       PASSED
---------------------------------
Path to AWS IoT Device Tester Report: /path/to/devicetester/
results/87e673c6-1226-11eb-9269-8c8590419f30/awsiotdevicetester_report.xml
Path to Test Execution Logs: /path/to/devicetester/
results/87e673c6-1226-11eb-9269-8c8590419f30/logs
Path to Aggregated JUnit Report: /path/to/devicetester/
results/87e673c6-1226-11eb-9269-8c8590419f30/IDTSampleSuitePython_Report.xml
```

Beheben von Fehlern

Verwenden Sie die folgenden Informationen, um Probleme beim Ausfüllen des Tutorials zu lösen.

Der Testfall wird nicht erfolgreich ausgeführt

- Wenn der Test nicht erfolgreich ausgeführt wird, streamt IDT die Fehlerprotokolle an die Konsole, die Ihnen bei der Problembehandlung des Testlaufs helfen kann. Stellen Sie sicher, dass Sie alle Voraussetzungen für dieses Tutorial erfüllen.

Es kann keine Verbindung zu dem zu testenden Gerät hergestellt werden

Überprüfen Sie Folgendes:

- Ihre `device.json` Datei enthält die richtige IP-Adresse, den richtigen Port und die richtigen Authentifizierungsinformationen.
- Sie können von Ihrem Host-Computer aus eine Verbindung zu Ihrem Gerät über SSH herstellen.

Tutorial: Entwickeln Sie eine einfache IDT-Testsuite

Eine Testsuite kombiniert Folgendes:

- Ausführbare Testdatei, die die Testlogik enthält
- Konfigurationsdateien, die die Testsuite beschreiben

Dieses Tutorial zeigt Ihnen, wie Sie IDT for FreeRTOS verwenden, um eine Python-Testsuite zu entwickeln, die einen einzigen Testfall enthält. Obwohl dieses Tutorial SSH verwendet, ist es hilfreich zu lernen, wie man es AWS IoT Device Tester mit FreeRTOS-Geräten verwendet.

In diesem Tutorial werden Sie die folgenden Schritte ausführen:

1. Erstellen Sie ein Testsuite-Verzeichnis
2. Erstellen Sie Konfigurationsdateien
3. Erstellen Sie die ausführbare Testfalldatei
4. Führen Sie die Testsuite aus

Folgen Sie den nachstehenden Schritten, um ein Tutorial zur Entwicklung einer einfachen IDT-Testsuite abzuschließen.

Themen

- Richten Sie die Voraussetzungen für eine einfache IDT-Testsuite ein
- Erstellen Sie ein Testsuite-Verzeichnis
- Erstellen Sie Konfigurationsdateien
- Holen Sie sich das IDT-Client-SDK
- Erstellen Sie die ausführbare Testfalldatei
- Geräteinformationen für IDT konfigurieren
- Führen Sie die Testsuite aus
- Beheben von Fehlern
- Erstellen Sie Konfigurationsdateien für die IDT-Testsuite
- Konfigurieren Sie den IDT-Testorchestrator
- Konfigurieren Sie die IDT-Zustandsmaschine
- Erstellen Sie eine ausführbare IDT-Testfalldatei

- Verwenden Sie den IDT-Kontext
- Konfigurieren Sie Einstellungen für Testläufer
- Debuggen Sie benutzerdefinierte Testsuiten und führen Sie sie aus
- Überprüfen Sie die IDT-Testergebnisse und -Protokolle
- Senden Sie IDT-Nutzungsmetriken

Richten Sie die Voraussetzungen für eine einfache IDT-Testsuite ein

Zum Durcharbeiten dieses Tutorials ist Folgendes erforderlich:

- Anforderungen an den Host-Computer
  - Aktuelle Version von AWS IoT Device Tester
  - Python 3.7 oder höher

    Führen Sie den folgenden Befehl aus, um die auf Ihrem Computer installierte Version von Python zu überprüfen:

    ```
    python3 --version
    ```

    Wenn die Verwendung dieses Befehls unter Windows einen Fehler zurückgibt, verwenden Sie `python --version` stattdessen. Wenn die zurückgegebene Versionsnummer 3.7 oder höher ist, führen Sie den folgenden Befehl in einem Powershell-Terminal aus, um ihn `python3` als Alias für Ihren `python` Befehl festzulegen.

    ```
    Set-Alias -Name "python3" -Value "python"
    ```

    Wenn keine Versionsinformationen zurückgegeben werden oder wenn die Versionsnummer kleiner als 3.7 ist, folgen Sie den Anweisungen unter Python herunterladen, um Python 3.7+ zu installieren. Weitere Informationen finden Sie in der Python-Dokumentation.
  - urllib3

    Führen Sie den folgenden Befehl aus, um zu überprüfen, ob die Installation korrekt `urllib3` ist:

    ```
    python3 -c 'import urllib3'
    ```

    Wenn `urllib3` es nicht installiert ist, führen Sie den folgenden Befehl aus, um es zu installieren:

```
python3 -m pip install urllib3
```

- Anforderungen an Speichergeräte
  - Ein Gerät mit einem Linux-Betriebssystem und einer Netzwerkverbindung zu demselben Netzwerk wie Ihr Host-Computer.

    Wir empfehlen Ihnen, einen Raspberry Pi mit Raspberry Pi OS zu verwenden. Stellen Sie sicher, dass Sie SSH auf Ihrem Raspberry Pi eingerichtet haben, um eine Remoteverbindung herzustellen.

Erstellen Sie ein Testsuite-Verzeichnis

IDT unterteilt Testfälle innerhalb jeder Testsuite logisch in Testgruppen. Jeder Testfall muss sich innerhalb einer Testgruppe befinden. Erstellen Sie für dieses Tutorial einen Ordner mit dem Namen `MyTestSuite_1.0.0` und erstellen Sie in diesem Ordner den folgenden Verzeichnisbaum:

```
MyTestSuite_1.0.0
### suite
    ### myTestGroup
        ### myTestCase
```

Erstellen Sie Konfigurationsdateien

Ihre Testsuite muss die folgenden erforderlichen Konfigurationsdateien enthalten:

Erforderliche Dateien

`suite.json`

Enthält Informationen über die Testsuite. Siehe Konfigurieren Sie suite.json.

`group.json`

Enthält Informationen über eine Testgruppe. Sie müssen für jede Testgruppe in Ihrer Testsuite eine `group.json` Datei erstellen. Siehe Konfigurieren Sie group.json.

`test.json`

Enthält Informationen zu einem Testfall. Sie müssen für jeden Testfall in Ihrer Testsuite eine `test.json` Datei erstellen. Siehe Konfigurieren Sie test.json.

1. Erstellen Sie in dem `MyTestSuite_1.0.0/suite` Ordner eine `suite.json` Datei mit der folgenden Struktur:

```
{
    "id": "MyTestSuite_1.0.0",
    "title": "My Test Suite",
    "details": "This is my test suite.",
    "userDataRequired": false
}
```

2. Erstellen Sie in dem `MyTestSuite_1.0.0/myTestGroup` Ordner eine `group.json` Datei mit der folgenden Struktur:

```
{
    "id": "MyTestGroup",
    "title": "My Test Group",
    "details": "This is my test group.",
    "optional": false
}
```

3. Erstellen Sie in dem `MyTestSuite_1.0.0/myTestGroup/myTestCase` Ordner eine `test.json` Datei mit der folgenden Struktur:

```
{
    "id": "MyTestCase",
    "title": "My Test Case",
    "details": "This is my test case.",
    "execution": {
        "timeout": 300000,
        "linux": {
            "cmd": "python3",
            "args": [
                "myTestCase.py"
            ]
        },
        "mac": {
            "cmd": "python3",
            "args": [
                "myTestCase.py"
            ]
        },
        "win": {
```

```
            "cmd": "python3",
            "args": [
                "myTestCase.py"
            ]
        }
    }
}
```

Der Verzeichnisbaum für Ihren `MyTestSuite_1.0.0` Ordner sollte jetzt wie folgt aussehen:

```
MyTestSuite_1.0.0
### suite
    ### suite.json
    ### myTestGroup
        ### group.json
        ### myTestCase
            ### test.json
```

Holen Sie sich das IDT-Client-SDK

Sie verwenden das IDT-Client-SDK, damit IDT mit dem zu testenden Gerät interagieren und Testergebnisse melden kann. Für dieses Tutorial verwenden Sie die Python-Version des SDK.

Kopieren Sie den *<device-tester-extract-location>*`/sdks/python/` Ordner aus dem `idt_client` Ordner in Ihren `MyTestSuite_1.0.0/suite/myTestGroup/myTestCase` Ordner.

Führen Sie den folgenden Befehl aus, um zu überprüfen, ob das SDK erfolgreich kopiert wurde.

```
cd MyTestSuite_1.0.0/suite/myTestGroup/myTestCase
python3 -c 'import idt_client'
```

Erstellen Sie die ausführbare Testfalldatei

Ausführbare Testfalldateien enthalten die Testlogik, die Sie ausführen möchten. Eine Testsuite kann mehrere ausführbare Testfalldateien enthalten. Für dieses Tutorial erstellen Sie nur eine ausführbare Testfalldatei.

1. Erstellen Sie die Testsuite-Datei.

   Erstellen Sie in dem `MyTestSuite_1.0.0/suite/myTestGroup/myTestCase` Ordner eine `myTestCase.py` Datei mit dem folgenden Inhalt:

```
from idt_client import *

def main():
    # Use the client SDK to communicate with IDT
    client = Client()

if __name__ == "__main__":
    main()
```

2. Verwenden Sie die Funktionen des Client-SDK, um Ihrer `myTestCase.py` Datei die folgende Testlogik hinzuzufügen:

   a. Führen Sie auf dem zu testenden Gerät einen SSH-Befehl aus.

```
from idt_client import *

def main():
    # Use the client SDK to communicate with IDT
    client = Client()

    # Create an execute on device request
    exec_req = ExecuteOnDeviceRequest(ExecuteOnDeviceCommand("echo 'hello
 world'"))

    # Run the command
    exec_resp = client.execute_on_device(exec_req)

    # Print the standard output
    print(exec_resp.stdout)

if __name__ == "__main__":
    main()
```

   b. Senden Sie das Testergebnis an IDT.

```
from idt_client import *

def main():
    # Use the client SDK to communicate with IDT
    client = Client()

    # Create an execute on device request
```

```
    exec_req = ExecuteOnDeviceRequest(ExecuteOnDeviceCommand("echo 'hello
 world'"))

    # Run the command
    exec_resp = client.execute_on_device(exec_req)

    # Print the standard output
    print(exec_resp.stdout)

    # Create a send result request
    sr_req = SendResultRequest(TestResult(passed=True))

    # Send the result
    client.send_result(sr_req)

if __name__ == "__main__":
    main()
```

Geräteinformationen für IDT konfigurieren

Konfigurieren Sie Ihre Geräteinformationen für IDT, um den Test auszuführen. Sie müssen die `device.json` Vorlage im *<device-tester-extract-location>*`/configs` Ordner mit den folgenden Informationen aktualisieren.

```
[
  {
    "id": "pool",
    "sku": "N/A",
    "devices": [
      {
        "id": "<device-id>",
        "connectivity": {
          "protocol": "ssh",
          "ip": "<ip-address>",
          "port": "<port>",
          "auth": {
            "method": "pki | password",
            "credentials": {
              "user": "<user-name>",
              "privKeyPath": "/path/to/private/key",
              "password": "<password>"
            }
```

```
          }
        }
      }
    ]
  }
]
```

Geben Sie im `devices` Objekt die folgenden Informationen ein:

`id`

Eine benutzerdefinierte eindeutige Kennung für Ihr Gerät.

`connectivity.ip`

Die IP-Adresse Ihres Geräts.

`connectivity.port`

Optional. Die Portnummer, die für SSH-Verbindungen zu Ihrem Gerät verwendet werden soll.

`connectivity.auth`

Authentifizierungsinformationen für die Verbindung.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

`connectivity.auth.method`

Die Authentifizierungsmethode, die für den Zugriff über ein bestimmtes Verbindungsprotokoll auf ein Gerät verwendet wird.

Unterstützte Werte sind:

- `pki`
- `password`

`connectivity.auth.credentials`

Die für die Authentifizierung verwendeten Anmeldeinformationen.

`connectivity.auth.credentials.user`

Der Benutzername, mit dem Sie sich auf Ihrem Gerät angemeldet haben.

`connectivity.auth.credentials.privKeyPath`

Der vollständige Pfad zu dem privaten Schlüssel, mit dem Sie sich auf Ihrem Gerät angemeldet haben.

Dieser Wert gilt nur, wenn `connectivity.auth.method` auf `pki` festgelegt ist.

`devices.connectivity.auth.credentials.password`

Das Passwort, mit dem Sie sich auf Ihrem Gerät angemeldet haben.

Dieser Wert gilt nur, wenn `connectivity.auth.method` auf `password` festgelegt ist.

> **Note**
>
> Geben Sie `privKeyPath` nur an, wenn `method` auf `pki` gesetzt ist.
> Geben Sie `password` nur an, wenn `method` auf `password` gesetzt ist.

Führen Sie die Testsuite aus

Nachdem Sie Ihre Testsuite erstellt haben, möchten Sie sicherstellen, dass sie wie erwartet funktioniert. Führen Sie dazu die folgenden Schritte aus, um die Testsuite mit Ihrem vorhandenen Gerätepool auszuführen.

1. Kopieren Sie Ihren `MyTestSuite_1.0.0` Ordner in*`<device-tester-extract-location>`*`/tests`.
2. Führen Sie die folgenden Befehle aus:

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id MyTestSuite
```

IDT führt Ihre Testsuite aus und streamt die Ergebnisse an die Konsole. Wenn der Test abgeschlossen ist, sehen Sie die folgenden Informationen:

```
time="2020-10-19T09:24:47-07:00" level=info msg=Using pool: pool
time="2020-10-19T09:24:47-07:00" level=info msg=Using test suite "MyTestSuite_1.0.0"
 for execution
time="2020-10-19T09:24:47-07:00" level=info msg=b'hello world\n'
 suiteId=MyTestSuite groupId=myTestGroup testCaseId=myTestCase deviceId=my-device
 executionId=9a52f362-1227-11eb-86c9-8c8590419f30
time="2020-10-19T09:24:47-07:00" level=info msg=All tests finished.
 executionId=9a52f362-1227-11eb-86c9-8c8590419f30
time="2020-10-19T09:24:48-07:00" level=info msg=
```

```
========== Test Summary ==========
Execution Time:         1s
Tests Completed:        1
Tests Passed:           1
Tests Failed:           0
Tests Skipped:          0
---------------------------------
Test Groups:
    myTestGroup:        PASSED
---------------------------------
Path to AWS IoT Device Tester Report: /path/to/devicetester/
results/9a52f362-1227-11eb-86c9-8c8590419f30/awsiotdevicetester_report.xml
Path to Test Execution Logs: /path/to/devicetester/
results/9a52f362-1227-11eb-86c9-8c8590419f30/logs
Path to Aggregated JUnit Report: /path/to/devicetester/
results/9a52f362-1227-11eb-86c9-8c8590419f30/MyTestSuite_Report.xml
```

Beheben von Fehlern

Verwenden Sie die folgenden Informationen, um Probleme beim Ausfüllen des Tutorials zu lösen.

Der Testfall wird nicht erfolgreich ausgeführt

Wenn der Test nicht erfolgreich ausgeführt wird, streamt IDT die Fehlerprotokolle an die Konsole, die Ihnen bei der Problembehandlung des Testlaufs helfen kann. Bevor Sie die Fehlerprotokolle überprüfen, überprüfen Sie Folgendes:

- Das IDT-Client-SDK befindet sich im richtigen Ordner, wie unter beschrieben. Holen Sie sich das IDT-Client-SDK
- Sie erfüllen alle Voraussetzungen für dieses Tutorial. Weitere Informationen finden Sie unter Richten Sie die Voraussetzungen für eine einfache IDT-Testsuite ein.

Es kann keine Verbindung zu dem zu testenden Gerät hergestellt werden

Überprüfen Sie Folgendes:

- Ihre `device.json` Datei enthält die richtige IP-Adresse, den richtigen Port und die richtigen Authentifizierungsinformationen.
- Sie können von Ihrem Host-Computer aus eine Verbindung zu Ihrem Gerät über SSH herstellen.

Erstellen Sie Konfigurationsdateien für die IDT-Testsuite

In diesem Abschnitt werden die Formate beschrieben, in denen Sie Konfigurationsdateien erstellen, die Sie beim Schreiben einer benutzerdefinierten Testsuite einbeziehen.

Erforderliche Konfigurationsdateien

`suite.json`

Enthält Informationen über die Testsuite. Siehe Konfigurieren Sie suite.json.

`group.json`

Enthält Informationen über eine Testgruppe. Sie müssen für jede Testgruppe in Ihrer Testsuite eine `group.json` Datei erstellen. Siehe Konfigurieren Sie group.json.

`test.json`

Enthält Informationen zu einem Testfall. Sie müssen für jeden Testfall in Ihrer Testsuite eine `test.json` Datei erstellen. Siehe Konfigurieren Sie test.json.

Optionale Konfigurationsdateien

`test_orchestrator.yaml` oder **`state_machine.json`**

Definiert, wie Tests ausgeführt werden, wenn IDT die Testsuite ausführt. SSe Konfigurieren Sie test_orchestrator.yaml.

> **Note**
>
> Ab IDT v4.5.2 verwenden Sie die `test_orchestrator.yaml` Datei, um den Test-Workflow zu definieren. In früheren Versionen von IDT verwenden Sie die Datei. `state_machine.json` Informationen zur Zustandsmaschine finden Sie unterKonfigurieren Sie die IDT-Zustandsmaschine.

`userdata_schema.json`

Definiert das Schema für die `userdata.json`Datei, die Testläufer in ihre Einstellungskonfiguration aufnehmen können. Die `userdata.json` Datei wird für alle zusätzlichen Konfigurationsinformationen verwendet, die für die Ausführung des Tests

erforderlich sind, aber nicht in der `device.json` Datei enthalten sind. Siehe Konfigurieren Sie userdata_schema.json.

Die Konfigurationsdateien werden *<custom-test-suite-folder>* wie hier gezeigt in Ihrem abgelegt.

```
<custom-test-suite-folder>
### suite
    ### suite.json
    ### test_orchestrator.yaml
    ### userdata_schema.json
    ### <test-group-folder>
        ### group.json
        ### <test-case-folder>
            ### test.json
```

Konfigurieren Sie suite.json

Die `suite.json` Datei legt Umgebungsvariablen fest und bestimmt, ob Benutzerdaten für die Ausführung der Testsuite erforderlich sind. Verwenden Sie die folgende Vorlage, um Ihre *<custom-test-suite-folder>*`/suite/suite.json` Datei zu konfigurieren:

```
{
    "id": "<suite-name>_<suite-version>",
    "title": "<suite-title>",
    "details": "<suite-details>",
    "userDataRequired": true | false,
    "environmentVariables": [
        {
            "key": "<name>",
            "value": "<value>",
        },
        ...
        {
            "key": "<name>",
            "value": "<value>",
        }
    ]
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`id`

Eine eindeutige benutzerdefinierte ID für die Testsuite. Der Wert von `id` muss mit dem Namen des Testsuite-Ordners übereinstimmen, in dem sich die `suite.json` Datei befindet. Der Name der Suite und die Suite-Version müssen außerdem die folgenden Anforderungen erfüllen:

- *<suite-name>*darf keine Unterstriche enthalten.
- *<suite-version>*wird bezeichnet als*x.x.x*, wo x ist eine Zahl.

Die ID wird in IDT-generierten Testberichten angezeigt.

`title`

Ein benutzerdefinierter Name für das Produkt oder die Funktion, das von dieser Testsuite getestet wird. Der Name wird in der IDT-CLI für Testläufer angezeigt.

`details`

Eine kurze Beschreibung des Zwecks der Testsuite.

`userDataRequired`

Definiert, ob Testläufer benutzerdefinierte Informationen in eine `userdata.json` Datei aufnehmen müssen. Wenn Sie diesen Wert auf setzen`true`, müssen Sie die `userdata_schema.json`Datei auch in Ihren Testsuite-Ordner aufnehmen.

`environmentVariables`

Optional. Ein Array von Umgebungsvariablen, die für diese Testsuite festgelegt werden sollen.

`environmentVariables.key`

Der Name der Umgebungsvariable.

`environmentVariables.value`

Der Wert der Umgebungsvariable.

Konfigurieren Sie group.json

Die `group.json` Datei definiert, ob eine Testgruppe erforderlich oder optional ist. Verwenden Sie die folgende Vorlage, um Ihre *<custom-test-suite-folder>*`/suite/`*<test-group>*`/group.json` Datei zu konfigurieren:

```
{
```

```
    "id": "<group-id>",
    "title": "<group-title>",
    "details": "<group-details>",
    "optional": true | false,
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`id`

Eine eindeutige benutzerdefinierte ID für die Testgruppe. Der Wert von `id` muss mit dem Namen des Testgruppenordners übereinstimmen, in dem sich die `group.json` Datei befindet, und darf keine Unterstriche () _ enthalten. Die ID wird in IDT-generierten Testberichten verwendet.

`title`

Ein beschreibender Name für die Testgruppe. Der Name wird in der IDT-CLI für Testläufer angezeigt.

`details`

Eine kurze Beschreibung des Zwecks der Testgruppe.

`optional`

Optional. Stellen Sie diese Option ein`true`, um diese Testgruppe als optionale Gruppe anzuzeigen, nachdem IDT die Ausführung der erforderlichen Tests abgeschlossen hat. Der Standardwert ist `false`.

Konfigurieren Sie test.json

Die `test.json` Datei bestimmt die ausführbaren Testfalldateien und die Umgebungsvariablen, die von einem Testfall verwendet werden. Weitere Hinweise zum Erstellen von ausführbaren Testfalldateien finden Sie unter. Erstellen Sie eine ausführbare IDT-Testfalldatei

Verwenden Sie die folgende Vorlage, um Ihre *<custom-test-suite-folder>*`/suite/`*<test-group>*`/`*<test-case>*`/test.json` Datei zu konfigurieren:

```
{
    "id": "<test-id>",
    "title": "<test-title>",
    "details": "<test-details>",
```

```
    "requireDUT": true | false,
    "requiredResources": [
        {
            "name": "<resource-name>",
            "features": [
                {
                    "name": "<feature-name>",
                    "version": "<feature-version>",
                    "jobSlots": <job-slots>
                }
            ]
        }
    ],
    "execution": {
        "timeout": <timeout>,
        "mac": {
            "cmd": "/path/to/executable",
            "args": [
                "<argument>"
            ],
        },
        "linux": {
            "cmd": "/path/to/executable",
            "args": [
                "<argument>"
            ],
        },
        "win": {
            "cmd": "/path/to/executable",
            "args": [
                "<argument>"
            ]
        }
    },
    "environmentVariables": [
        {
            "key": "<name>",
            "value": "<value>",
        }
    ]
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`id`

Eine eindeutige benutzerdefinierte ID für den Testfall. Der Wert von `id` muss mit dem Namen des Testfallordners übereinstimmen, in dem sich die `test.json` Datei befindet, und darf keine Unterstriche () `_` enthalten. Die ID wird in IDT-generierten Testberichten verwendet.

`title`

Ein beschreibender Name für den Testfall. Der Name wird in der IDT-CLI für Testläufer angezeigt.

`details`

Eine kurze Beschreibung des Zwecks des Testfalls.

`requireDUT`

Optional. `true`Ist auf eingestellt, wenn ein Gerät für die Ausführung dieses Tests erforderlich ist, andernfalls auf`false`. Der Standardwert ist `true`. Testläufer konfigurieren die Geräte, mit denen sie den Test ausführen, in ihrer `device.json` Datei.

`requiredResources`

Optional. Ein Array, das Informationen über Ressourcengeräte bereitstellt, die für die Ausführung dieses Tests benötigt werden.

`requiredResources.name`

Der eindeutige Name, der dem Ressourcengerät gegeben werden soll, wenn dieser Test ausgeführt wird.

`requiredResources.features`

Eine Reihe von benutzerdefinierten Funktionen für Ressourcengeräte.

`requiredResources.features.name`

Der Name der Funktion. Die Gerätefunktion, für die Sie dieses Gerät verwenden möchten. Dieser Name wird mit dem Funktionsnamen abgeglichen, den der Test-Runner in der `resource.json` Datei angegeben hat.

`requiredResources.features.version`

Optional. Die Version der Funktion. Dieser Wert wird mit der vom Test-Runner in der `resource.json` Datei bereitgestellten Feature-Version verglichen. Wenn keine Version bereitgestellt wird, wird die Funktion nicht überprüft. Wenn für die Funktion keine Versionsnummer erforderlich ist, lassen Sie dieses Feld leer.

`requiredResources.features.jobSlots`

Optional. Die Anzahl der gleichzeitigen Tests, die diese Funktion unterstützen kann. Der Standardwert ist 1. Wenn Sie möchten, dass IDT unterschiedliche Geräte für einzelne Funktionen verwendet, empfehlen wir Ihnen, diesen Wert auf 1 festzulegen.

`execution.timeout`

Die Zeit (in Millisekunden), die IDT wartet, bis der Test abgeschlossen ist. Weitere Informationen zum Einstellen dieses Werts finden Sie unter. Erstellen Sie eine ausführbare IDT-Testfalldatei

`execution.os`

Die ausführbaren Testfalldateien, die auf dem Betriebssystem des Host-Computers ausgeführt werden sollen, basieren auf dem Betriebssystem, auf dem IDT ausgeführt wird. Unterstützte Werte sind `linux`, `mac` und `win`.

`execution.os.cmd`

Der Pfad zu der ausführbaren Testfalldatei, die Sie für das angegebene Betriebssystem ausführen möchten. Dieser Speicherort muss sich im Systempfad befinden.

`execution.os.args`

Optional. Die Argumente, die zur Ausführung der ausführbaren Testfalldatei angegeben werden müssen.

`environmentVariables`

Optional. Ein Array von Umgebungsvariablen, die für diesen Testfall festgelegt wurden.

`environmentVariables.key`

Der Name der Umgebungsvariable.

`environmentVariables.value`

Der Wert der Umgebungsvariable.

> **Note**
>
> Wenn Sie dieselbe Umgebungsvariable in der `test.json` Datei und in der `suite.json` Datei angeben, hat der Wert in der `test.json` Datei Vorrang.

Konfigurieren Sie test_orchestrator.yaml

Ein Testorchestrator ist ein Konstrukt, das den Ausführungsablauf der Testsuite steuert. Er bestimmt den Startstatus einer Testsuite, verwaltet Zustandsübergänge auf der Grundlage benutzerdefinierter Regeln und setzt den Übergang durch diese Zustände fort, bis der Endstatus erreicht ist.

Wenn Ihre Testsuite keinen benutzerdefinierten Test-Orchestrator enthält, generiert IDT einen Test-Orchestrator für Sie.

Der Standard-Test-Orchestrator führt die folgenden Funktionen aus:

- Bietet Testläufern die Möglichkeit, anstelle der gesamten Testsuite bestimmte Testgruppen auszuwählen und auszuführen.
- Wenn keine bestimmten Testgruppen ausgewählt sind, wird jede Testgruppe in der Testsuite in zufälliger Reihenfolge ausgeführt.
- Generiert Berichte und druckt eine Konsolenübersicht aus, in der die Testergebnisse für jede Testgruppe und jeden Testfall angezeigt werden.

Weitere Informationen zur Funktionsweise des IDT Test Orchestrator finden Sie unter. Konfigurieren Sie den IDT-Testorchestrator

Konfigurieren Sie userdata_schema.json

Die `userdata_schema.json` Datei bestimmt das Schema, in dem Testläufer Benutzerdaten bereitstellen. Benutzerdaten sind erforderlich, wenn Ihre Testsuite Informationen benötigt, die nicht in der `device.json` Datei enthalten sind. Beispielsweise benötigen Ihre Tests möglicherweise Anmeldeinformationen für Wi-Fi-Netzwerke, bestimmte offene Ports oder Zertifikate, die ein Benutzer bereitstellen muss. Diese Informationen können IDT als Eingabeparameter zur Verfügung gestellt werden`userdata`, dessen Wert eine `userdata.json` Datei ist, die Benutzer in ihrem *`<device-tester-extract-location>`*`/config` Ordner erstellen. Das Format der `userdata.json` Datei basiert auf der `userdata_schema.json` Datei, die Sie in die Testsuite aufnehmen.

Um anzugeben, dass Testläufer eine `userdata.json` Datei bereitstellen müssen:

1. Stellen Sie in der `suite.json` Datei `userDataRequired` auf ein`true`.
2. Erstellen Sie in Ihrem *`<custom-test-suite-folder>`* eine `userdata_schema.json` Datei.
3. Bearbeiten Sie die `userdata_schema.json` Datei, um ein gültiges IETF-Draft v4-JSON-Schema zu erstellen.

Wenn IDT Ihre Testsuite ausführt, liest es automatisch das Schema und verwendet es, um die vom Testläufer bereitgestellte `userdata.json` Datei zu validieren. Falls gültig, ist der Inhalt der `userdata.json` Datei sowohl im IDT-Kontext als auch im Test-Orchestrator-Kontext verfügbar.

Konfigurieren Sie den IDT-Testorchestrator

Ab IDT v4.5.2 enthält IDT eine neue Test Orchestrator-Komponente. Der Testorchestrator ist eine IDT-Komponente, die den Ausführungsablauf der Testsuite steuert und den Testbericht generiert, nachdem IDT alle Tests ausgeführt hat. Der Testorchestrator bestimmt die Testauswahl und die Reihenfolge, in der Tests ausgeführt werden, auf der Grundlage benutzerdefinierter Regeln.

Wenn Ihre Testsuite keinen benutzerdefinierten Test-Orchestrator enthält, generiert IDT einen Test-Orchestrator für Sie.

Der Standard-Test-Orchestrator führt die folgenden Funktionen aus:

- Bietet Testläufern die Möglichkeit, anstelle der gesamten Testsuite bestimmte Testgruppen auszuwählen und auszuführen.
- Wenn keine bestimmten Testgruppen ausgewählt sind, wird jede Testgruppe in der Testsuite in zufälliger Reihenfolge ausgeführt.
- Generiert Berichte und druckt eine Konsolenübersicht aus, in der die Testergebnisse für jede Testgruppe und jeden Testfall angezeigt werden.

Der Testorchestrator ersetzt die IDT-Zustandsmaschine. Wir empfehlen dringend, anstelle der IDT State Machine den Test Orchestrator für die Entwicklung Ihrer Testsuiten zu verwenden. Der Test-Orchestrator bietet die folgenden verbesserten Funktionen:

- Verwendet ein deklaratives Format im Vergleich zum imperativen Format, das die IDT-Zustandsmaschine verwendet. Auf diese Weise können Sie angeben, welche Tests Sie ausführen möchten und wann Sie sie ausführen möchten.
- Verwaltet die Bearbeitung bestimmter Gruppen, die Berichtsgenerierung, die Fehlerbehandlung und die Ergebnisverfolgung, sodass Sie diese Aktionen nicht manuell verwalten müssen.
- Verwendet das YAML-Format, das Kommentare standardmäßig unterstützt.
- Benötigt 80 Prozent weniger Festplattenspeicher als der Test-Orchestrator, um denselben Workflow zu definieren.
- Fügt eine Validierung vor dem Test hinzu, um sicherzustellen, dass Ihre Workflow-Definition keine falschen Test IDs - oder zirkulären Abhängigkeiten enthält.

Testen Sie das Orchestrator-Format

Sie können die folgende Vorlage verwenden, um Ihre eigene *custom-test-suite-folder*/suite/test_orchestrator.yaml Datei zu konfigurieren:

```
Aliases:
  string: context-expression

ConditionalTests:
  - Condition: context-expression
    Tests:
      - test-descriptor

Order:
  - - group-descriptor
    - group-descriptor

Features:
  - Name: feature-name
    Value: support-description
    Condition: context-expression
    Tests:
        - test-descriptor
    OneOfTests:
        - test-descriptor
    IsRequired: boolean
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`Aliases`

Optional. Benutzerdefinierte Zeichenketten, die Kontextausdrücken zugeordnet sind. Aliase ermöglichen es Ihnen, benutzerfreundliche Namen zu generieren, um Kontextausdrücke in Ihrer Test-Orchestrator-Konfiguration zu identifizieren. Dies ist besonders nützlich, wenn Sie komplexe Kontextausdrücke oder Ausdrücke erstellen, die Sie an mehreren Stellen verwenden.

Sie können Kontextausdrücke verwenden, um Kontextabfragen zu speichern, mit denen Sie auf Daten aus anderen IDT-Konfigurationen zugreifen können. Weitere Informationen finden Sie unter Greifen Sie auf Daten im Kontext zu.

Example

Beispiel

```
Aliases:
    FizzChosen: "'{{$pool.features[?(@.name == 'Fizz')].value[0]}}' == 'yes'"
    BuzzChosen: "'{{$pool.features[?(@.name == 'Buzz')].value[0]}}' == 'yes'"
    FizzBuzzChosen: "'{{$aliases.FizzChosen}}' && '{{$aliases.BuzzChosen}}'"
```

`ConditionalTests`

Optional. Eine Liste der Bedingungen und der entsprechenden Testfälle, die ausgeführt werden, wenn jede Bedingung erfüllt ist. Jede Bedingung kann mehrere Testfälle haben. Sie können einen bestimmten Testfall jedoch nur einer Bedingung zuweisen.

Standardmäßig führt IDT jeden Testfall aus, der keiner Bedingung in dieser Liste zugewiesen ist. Wenn Sie diesen Abschnitt nicht angeben, führt IDT alle Testgruppen in der Testsuite aus.

Jedes Element in der `ConditionalTests` Liste enthält die folgenden Parameter:

`Condition`

Ein Kontextausdruck, der einen booleschen Wert ergibt. Wenn der ausgewertete Wert wahr ist, führt IDT die im Parameter angegebenen Testfälle aus. `Tests`

`Tests`

Die Liste der Testdeskriptoren.

Jeder Testdeskriptor verwendet die Testgruppen-ID und einen oder mehrere Testfälle, um die einzelnen Tests IDs zu identifizieren, die von einer bestimmten Testgruppe aus ausgeführt werden sollen. Der Testdeskriptor verwendet das folgende Format:

```
GroupId: group-id
CaseIds: [test-id, test-id] # optional
```

Example

Beispiel

Im folgenden Beispiel werden generische Kontextausdrücke verwendet, die Sie als `Aliases` definieren können.

```
ConditionalTests:
```

```
    - Condition: "{{$aliases.Condition1}}"
      Tests:
          - GroupId: A
          - GroupId: B
    - Condition: "{{$aliases.Condition2}}"
      Tests:
          - GroupId: D
    - Condition: "{{$aliases.Condition1}} || {{$aliases.Condition2}}"
      Tests:
          - GroupId: C
```

Basierend auf den definierten Bedingungen wählt IDT Testgruppen wie folgt aus:

- Wenn `Condition1` dies zutrifft, führt IDT die Tests in den Testgruppen A, B und C durch.
- Wenn `Condition2` dies zutrifft, führt IDT die Tests in den Testgruppen C und D durch.

`Order`

Optional. Die Reihenfolge, in der die Tests ausgeführt werden. Sie geben die Testreihenfolge auf der Ebene der Testgruppen an. Wenn Sie diesen Abschnitt nicht angeben, führt IDT alle zutreffenden Testgruppen in zufälliger Reihenfolge aus. Der Wert von `Order` ist eine Liste von Gruppendeskriptorlisten. Jede Testgruppe, in der Sie nicht aufgeführt sind`Order`, kann parallel zu jeder anderen aufgelisteten Testgruppe ausgeführt werden.

Jede Gruppendeskriptorliste enthält einen oder mehrere Gruppendeskriptoren und gibt die Reihenfolge an, in der die Gruppen ausgeführt werden sollen, die in den einzelnen Deskriptoren angegeben sind. Sie können die folgenden Formate verwenden, um einzelne Gruppendeskriptoren zu definieren:

- *`group-id`*— Die Gruppen-ID einer vorhandenen Testgruppe.
- [*`group-id`*, *`group-id`*]— Liste der Testgruppen, die in beliebiger Reihenfolge relativ zueinander ausgeführt werden können.
- `"*"`— Platzhalter. Dies entspricht der Liste aller Testgruppen, die noch nicht in der aktuellen Gruppendeskriptorliste angegeben sind.

Der Wert für `Order` muss außerdem die folgenden Anforderungen erfüllen:

- Die Testgruppe IDs , die Sie in einem Gruppendeskriptor angeben, muss in Ihrer Testsuite vorhanden sein.
- Jede Gruppendeskriptorliste muss mindestens eine Testgruppe enthalten.

- Jede Gruppendeskriptorliste muss eine eindeutige Gruppe enthalten. IDs Sie können eine Testgruppen-ID nicht innerhalb einzelner Gruppendeskriptoren wiederholen.
- Eine Gruppendeskriptorliste kann höchstens einen Platzhalter-Gruppendeskriptor enthalten. Der Platzhalter-Gruppendeskriptor muss das erste oder das letzte Element in der Liste sein.

Example

Beispiel

Für eine Testsuite, die die Testgruppen A, B, C, D und E enthält, zeigt die folgende Beispielliste verschiedene Möglichkeiten, um anzugeben, dass IDT zuerst die Testgruppe A, dann die Testgruppe B und dann die Testgruppen C, D und E in beliebiger Reihenfolge ausführen soll.

- ```
  Order:
      - - A
        - B
        - [C, D, E]
  ```
- ```
  Order:
      - - A
        - B
        - "*"
  ```
- ```
  Order:
      - - A
        - B

      - - B
        - C

      - - B
        - D

      - - B
        - E
  ```

`Features`

Optional. Die Liste der Produktfunktionen, die IDT der `awsiotdevicetester_report.xml` Datei hinzufügen soll. Wenn Sie diesen Abschnitt nicht angeben, fügt IDT dem Bericht keine Produktfunktionen hinzu.
  ```

Bei einer Produktfunktion handelt es sich um benutzerdefinierte Informationen über bestimmte Kriterien, die ein Gerät möglicherweise erfüllt. Beispielsweise kann die MQTT-Produktfunktion angeben, dass das Gerät MQTT-Nachrichten ordnungsgemäß veröffentlicht. In werden Produktfunktionen als`awsiotdevicetester_report.xml`, oder als benutzerdefinierter benutzerdefinierter Wert festgelegt `supported``not-supported`, je nachdem, ob die angegebenen Tests bestanden wurden.

Jedes Element in der `Features` Liste besteht aus den folgenden Parametern:

`Name`

Der Name der Funktion.

`Value`

Optional. Der benutzerdefinierte Wert, den Sie anstelle von im Bericht verwenden möchten`supported`. Wenn dieser Wert nicht angegeben ist, legt Based IDT den Feature-Wert auf `supported` oder `not-supported` basierend auf Testergebnissen fest. Wenn Sie dasselbe Feature mit unterschiedlichen Bedingungen testen, können Sie für jede Instanz dieses Features in der `Features` Liste einen benutzerdefinierten Wert verwenden, und IDT verkettet die Feature-Werte für unterstützte Bedingungen. Die Ausgabe des obigen Befehls sieht in etwa folgendermaßen aus (JSON format).

`Condition`

Ein Kontextausdruck, der zu einem booleschen Wert ausgewertet wird. Wenn der ausgewertete Wert wahr ist, fügt IDT die Funktion dem Testbericht hinzu, nachdem die Ausführung der Testsuite abgeschlossen ist. Wenn der ausgewertete Wert falsch ist, ist der Test nicht im Bericht enthalten.

`Tests`

Optional. Die Liste der Testdeskriptoren. Alle Tests, die in dieser Liste aufgeführt sind, müssen bestanden werden, damit die Funktion unterstützt wird.

Jeder Testdeskriptor in dieser Liste verwendet die Testgruppen-ID und einen oder mehrere Testfälle, um die einzelnen Tests IDs zu identifizieren, die von einer bestimmten Testgruppe aus ausgeführt werden sollen. Der Testdeskriptor verwendet das folgende Format:

```
GroupId: group-id
CaseIds: [test-id, test-id] # optional
```

Sie müssen `OneOfTests` für jedes Feature in der `Features` Liste entweder `Tests` oder angeben.

`OneOfTests`

Optional. Die Liste der Testdeskriptoren. Mindestens einer der in dieser Liste aufgeführten Tests muss bestanden werden, damit die Funktion unterstützt wird.

Jeder Testdeskriptor in dieser Liste verwendet die Testgruppen-ID und einen oder mehrere Testfälle, um die einzelnen Tests IDs zu identifizieren, die von einer bestimmten Testgruppe aus ausgeführt werden sollen. Der Testdeskriptor verwendet das folgende Format:

```
GroupId: group-id
CaseIds: [test-id, test-id] # optional
```

Sie müssen `OneOfTests` für jedes Feature in der `Features` Liste entweder `Tests` oder angeben.

`IsRequired`

Der boolesche Wert, der definiert, ob die Funktion im Testbericht erforderlich ist. Der Standardwert ist `false`.

Testen Sie den Orchestrator-Kontext

Der Test-Orchestrator-Kontext ist ein schreibgeschütztes JSON-Dokument, das Daten enthält, die dem Test-Orchestrator während der Ausführung zur Verfügung stehen. Der Test-Orchestrator-Kontext ist nur vom Test-Orchestrator aus zugänglich und enthält Informationen, die den Testablauf bestimmen. Sie können beispielsweise Informationen verwenden, die von Testläufern in der `userdata.json` Datei konfiguriert wurden, um festzustellen, ob ein bestimmter Test ausgeführt werden muss.

Der Test-Orchestrator-Kontext verwendet das folgende Format:

```
{
    "pool": {
        <device-json-pool-element>
    },
    "userData": {
        <userdata-json-content>
    },
```

```
    "config": {
        <config-json-content>
    }
}
```

`pool`

Informationen über den Gerätepool, der für den Testlauf ausgewählt wurde. Für einen ausgewählten Gerätepool werden diese Informationen aus dem entsprechenden Gerätepool-Array-Element der obersten Ebene abgerufen, das in der `device.json` Datei definiert ist.

`userData`

Informationen in der `userdata.json` Datei.

`config`

Informationen in der `config.json` Datei.

Sie können den Kontext mithilfe der JSONPath Notation abfragen. Die Syntax für JSONPath Abfragen in Statusdefinitionen lautet`{{query}}`. Wenn Sie auf Daten aus dem Test Orchestrator-Kontext zugreifen, stellen Sie sicher, dass jeder Wert eine Zeichenfolge, eine Zahl oder einen booleschen Wert ergibt.

Weitere Hinweise zur Verwendung der JSONPath Notation für den Zugriff auf Daten aus dem Kontext finden Sie unter. Verwenden Sie den IDT-Kontext

Konfigurieren Sie die IDT-Zustandsmaschine

> **Important**
>
> Ab IDT v4.5.2 ist diese Zustandsmaschine veraltet. Wir empfehlen dringend, den neuen Test-Orchestrator zu verwenden. Weitere Informationen finden Sie unter Konfigurieren Sie den IDT-Testorchestrator.

Eine Zustandsmaschine ist ein Konstrukt, das den Ausführungsablauf der Testsuite steuert. Sie bestimmt den Startstatus einer Testsuite, verwaltet Zustandsübergänge auf der Grundlage benutzerdefinierter Regeln und setzt den Übergang durch diese Zustände fort, bis der Endstatus erreicht ist.

Wenn Ihre Testsuite keine benutzerdefinierte Zustandsmaschine enthält, generiert IDT eine Zustandsmaschine für Sie. Die Standard-Zustandsmaschine erfüllt die folgenden Funktionen:

- Bietet Testläufern die Möglichkeit, anstelle der gesamten Testsuite bestimmte Testgruppen auszuwählen und auszuführen.
- Wenn keine bestimmten Testgruppen ausgewählt sind, wird jede Testgruppe in der Testsuite in zufälliger Reihenfolge ausgeführt.
- Generiert Berichte und druckt eine Konsolenübersicht aus, in der die Testergebnisse für jede Testgruppe und jeden Testfall angezeigt werden.

Die Zustandsmaschine für eine IDT-Testsuite muss die folgenden Kriterien erfüllen:

- Jeder Status entspricht einer Aktion, die IDT ausführen muss, z. B. dem Ausführen einer Testgruppe oder eines Produkts oder einer Berichtsdatei.
- Beim Übergang zu einem Status wird die mit dem Status verknüpfte Aktion ausgeführt.
- Jeder Status definiert die Übergangsregel für den nächsten Status.
- Der Endstatus muss entweder `Succeed` oder sein`Fail`.

Format der Zustandsmaschine

Sie können die folgende Vorlage verwenden, um Ihre eigene *<custom-test-suite-folder>*`/suite/state_machine.json` Datei zu konfigurieren:

```
{
  "Comment": "<description>",
  "StartAt": "<state-name>",
  "States": {
    "<state-name>": {
      "Type": "<state-type>",
      // Additional state configuration
    }

    // Required states
    "Succeed": {
      "Type": "Succeed"
    },
    "Fail": {
      "Type": "Fail"
    }
```

```
  }
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`Comment`

Eine Beschreibung der Zustandsmaschine.

`StartAt`

Der Name des Status, in dem IDT mit der Ausführung der Testsuite beginnt. Der Wert von `StartAt` muss auf einen der im `States` Objekt aufgelisteten Zustände gesetzt werden.

`States`

Ein Objekt, das benutzerdefinierte Statusnamen gültigen IDT-Staaten zuordnet. Jeder Bundesstaat. *`state-name`*Objekt enthält die Definition eines gültigen Zustands, der dem zugeordnet ist*`state-name`*.

Das `States` Objekt muss die `Fail` Zustände `Succeed` und enthalten. Hinweise zu gültigen Bundesstaaten finden Sie unterGültige Staaten und Bundesstaatendefinitionen.

Gültige Staaten und Bundesstaatendefinitionen

In diesem Abschnitt werden die Zustandsdefinitionen aller gültigen Staaten beschrieben, die in der IDT-Zustandsmaschine verwendet werden können. Einige der folgenden Staaten unterstützen Konfigurationen auf Testfallebene. Wir empfehlen jedoch, Regeln für den Statusübergang auf Testgruppenebene statt auf Testfallebene zu konfigurieren, sofern dies nicht unbedingt erforderlich ist.

Definitionen von Bundesstaaten

- RunTask
- Choice
- Parallel
- AddProductFeatures
- Bericht
- LogMessage
- SelectGroup

- Fehler
- Succeed

RunTask

Der `RunTask` Staat führt Testfälle aus einer in der Testsuite definierten Testgruppe aus.

```
{
    "Type": "RunTask",
    "Next": "<state-name>",
    "TestGroup": "<group-id>",
    "TestCases": [
        "<test-id>"
    ],
    "ResultVar": "<result-name>"
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`Next`

Der Name des Status, zu dem nach der Ausführung der Aktionen im aktuellen Status übergegangen werden soll.

`TestGroup`

Optional. Die ID der Testgruppe, die ausgeführt werden soll. Wenn dieser Wert nicht angegeben ist, führt IDT die Testgruppe aus, die der Testläufer auswählt.

`TestCases`

Optional. Ein Array von Testfällen IDs aus der in `TestGroup` angegebenen Gruppe. Basierend auf den Werten von `TestGroup` und `TestCases` bestimmt IDT das Verhalten der Testausführung wie folgt:

- Wenn `TestGroup` sowohl als auch angegeben `TestCases` sind, führt IDT die angegebenen Testfälle aus der Testgruppe aus.
- Wenn `TestCases` angegeben, aber nicht angegeben `TestGroup` ist, führt IDT die angegebenen Testfälle aus.
- Wenn `TestGroup` angegeben, aber nicht angegeben `TestCases` ist, führt IDT alle Testfälle innerhalb der angegebenen Testgruppe aus.

- Wenn weder `TestGroup` oder angegeben `TestCases` ist, führt IDT alle Testfälle aus der Testgruppe aus, die der Testläufer aus der IDT-CLI auswählt. Um die Gruppenauswahl für Testläufer zu aktivieren, müssen Sie `RunTask` sowohl Status als auch `Choice` Status in Ihre `statemachine.json` Datei aufnehmen. Ein Beispiel dafür, wie das funktioniert, finden Sie unter Beispiel für eine Zustandsmaschine: Vom Benutzer ausgewählte Testgruppen ausführen.

  Weitere Informationen zur Aktivierung von IDT-CLI-Befehlen für Testläufer finden Sie unterthe section called “IDT-CLI-Befehle aktivieren”.

`ResultVar`

Der Name der Kontextvariablen, die mit den Ergebnissen des Testlaufs festgelegt werden soll. Geben Sie diesen Wert nicht an, wenn Sie keinen Wert für angegeben habenTestGroup. IDT legt den Wert der Variablen, die Sie definieren, auf `true` oder `ResultVar` auf der `false` Grundlage der folgenden Werte fest:

- Wenn der Variablenname die Form hat*`text`*_*`text`*_`passed`, wird der Wert darauf gesetzt, ob alle Tests in der ersten Testgruppe bestanden oder übersprungen wurden.
- In allen anderen Fällen wird der Wert darauf gesetzt, ob alle Tests in allen Testgruppen bestanden wurden oder ob sie übersprungen wurden.

In der Regel verwenden Sie `RunTask` state, um eine Testgruppen-ID anzugeben, ohne einen einzelnen Testfall anzugeben IDs, sodass IDT alle Testfälle in der angegebenen Testgruppe ausführt. Alle Testfälle, die von diesem Status ausgeführt werden, werden parallel in zufälliger Reihenfolge ausgeführt. Wenn jedoch für alle Testfälle ein Gerät ausgeführt werden muss und nur ein einziges Gerät verfügbar ist, werden die Testfälle stattdessen sequentiell ausgeführt.

Fehlerbehandlung

Wenn eine der angegebenen Testgruppen oder Testfälle nicht gültig ist, IDs gibt dieser Status den `RunTaskError` Ausführungsfehler aus. Wenn der Status auf einen Ausführungsfehler stößt, wird auch die `hasExecutionError` Variable im Zustandsmaschinenkontext auf gesetzt`true`.

Choice

Mit `Choice` diesem Status können Sie basierend auf benutzerdefinierten Bedingungen dynamisch den nächsten Status festlegen, zu dem der Übergang erfolgen soll.

```
{
    "Type": "Choice",
```

```
    "Default": "<state-name>",
    "FallthroughOnError": true | false,
    "Choices": [
        {
            "Expression": "<expression>",
            "Next": "<state-name>"
        }
    ]
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`Default`

Der Standardstatus, in den der Übergang erfolgen soll, wenn keiner der in definierten Ausdrücke ausgewertet werden `Choices` kann. `true`

`FallthroughOnError`

Optional. Gibt das Verhalten an, wenn der Status bei der Auswertung von Ausdrücken auf einen Fehler stößt. Legt fest, `true` ob Sie einen Ausdruck überspringen möchten, wenn die Auswertung zu einem Fehler führt. Wenn keine Ausdrücke übereinstimmen, wechselt die Zustandsmaschine in den `Default` Status. Wenn der `FallthroughOnError` Wert nicht angegeben ist, wird standardmäßig der Wert verwendet. `false`

`Choices`

Eine Reihe von Ausdrücken und Zuständen, um zu bestimmen, in welchen Status nach der Ausführung der Aktionen im aktuellen Status übergegangen werden soll.

`Choices.Expression`

Eine Ausdruckszeichenfolge, die einen booleschen Wert ergibt. Wenn der Ausdruck zu ausgewertet wird`true`, geht die Zustandsmaschine in den Zustand über, der in definiert ist. `Choices.Next` Ausdruckszeichenfolgen rufen Werte aus dem Zustandsmaschinen-Kontext ab und führen dann Operationen an ihnen durch, um einen booleschen Wert zu erhalten. Hinweise zum Zugriff auf den Zustandsmaschinenkontext finden Sie unter. Kontext der Zustandsmaschine

`Choices.Next`

Der Name des Zustands, zu dem der Übergang erfolgen soll, wenn der in definierte Ausdruck zu `Choices.Expression` `true` ausgewertet wird.

Fehlerbehandlung

In den folgenden Fällen kann für den `Choice` Status eine Fehlerbehandlung erforderlich sein:

- Einige Variablen in den Auswahlausdrücken sind im Zustandsmaschinen-Kontext nicht vorhanden.
- Das Ergebnis eines Ausdrucks ist kein boolescher Wert.
- Das Ergebnis einer JSON-Suche ist keine Zeichenfolge, Zahl oder boolescher Wert.

In diesem Status können Sie keinen `Catch` Block verwenden, um Fehler zu behandeln. Wenn Sie die Ausführung der Zustandsmaschine beenden möchten, wenn sie auf einen Fehler stößt, müssen Sie `FallthroughOnError` auf einstellen`false`. Wir empfehlen jedoch, dass Sie die Einstellung `FallthroughOnError` auf `true` festlegen und je nach Anwendungsfall eine der folgenden Aktionen ausführen:

- Wenn davon ausgegangen wird, dass eine Variable, auf die Sie zugreifen, in einigen Fällen nicht existiert, verwenden Sie den Wert von `Default` und zusätzliche `Choices` Blöcke, um den nächsten Status anzugeben.
- Wenn eine Variable, auf die Sie zugreifen, immer existieren sollte, setzen Sie den `Default` Status auf`Fail`.

Parallel

Mit dem `Parallel` Status können Sie neue Zustandsmaschinen definieren und parallel zueinander ausführen.

```
{
    "Type": "Parallel",
    "Next": "<state-name>",
    "Branches": [
        <state-machine-definition>
    ]
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`Next`

Der Name des Status, zu dem nach der Ausführung der Aktionen im aktuellen Status übergegangen werden soll.

`Branches`

Eine Reihe von Zustandsmaschinendefinitionen, die ausgeführt werden sollen. Jede Zustandsmaschinen-Definition muss ihre eigenen `StartAtSucceed`, und `Fail` -Zustände enthalten. Die Zustandsmaschinendefinitionen in diesem Array können nicht auf Zustände verweisen, die außerhalb ihrer eigenen Definition liegen.

> **Note**
>
> Da jeder Zustandsmaschine denselben Zustandsmaschinenkontext verwendet, kann das Setzen von Variablen in einem Zweig und das anschließende Lesen dieser Variablen aus einem anderen Zweig zu unerwartetem Verhalten führen.

Der `Parallel` Status wechselt erst in den nächsten Status, nachdem er alle Branch-State-Machines ausgeführt hat. Jeder Status, für den ein Gerät erforderlich ist, wartet mit der Ausführung, bis das Gerät verfügbar ist. Wenn mehrere Geräte verfügbar sind, führt dieser Status Testfälle aus mehreren Gruppen parallel aus. Wenn nicht genügend Geräte verfügbar sind, werden die Testfälle sequentiell ausgeführt. Da Testfälle in zufälliger Reihenfolge ausgeführt werden, wenn sie parallel ausgeführt werden, können verschiedene Geräte verwendet werden, um Tests derselben Testgruppe auszuführen.

Fehlerbehandlung

Stellen Sie sicher, dass sowohl die Zweigzustandsmaschine als auch die übergeordnete Zustandsmaschine in den `Fail` Status wechseln, in dem Ausführungsfehler behoben werden.

Da Branch-State-Maschinen keine Ausführungsfehler an den übergeordneten Zustandsmaschinen übertragen, können Sie einen `Catch` Block nicht verwenden, um Ausführungsfehler in Branch-State-Machines zu behandeln. Verwenden Sie den `hasExecutionErrors` Wert stattdessen im Kontext des Shared State Machines. Ein Beispiel dafür, wie das funktioniert, finden Sie unterBeispiel State Machine: Zwei Testgruppen parallel ausführen.

AddProductFeatures

Mit `AddProductFeatures` diesem Status können Sie der von IDT generierten `awsiotdevicetester_report.xml` Datei Produktmerkmale hinzufügen.

Bei einer Produktfunktion handelt es sich um benutzerdefinierte Informationen über bestimmte Kriterien, die ein Gerät möglicherweise erfüllt. Beispielsweise kann die `MQTT` Produktfunktion

angeben, dass das Gerät MQTT-Nachrichten ordnungsgemäß veröffentlicht. Im Bericht werden Produktfunktionen als, oder als benutzerdefinierter Wert festgelegt `supportednot-supported`, je nachdem, ob die angegebenen Tests bestanden wurden.

> **Note**
>
> Der `AddProductFeatures` Staat generiert selbst keine Berichte. Dieser Status muss in den `ReportStatus` übergehen, in dem Berichte generiert werden können.

```
{
    "Type": "Parallel",
    "Next": "<state-name>",
    "Features": [
        {
            "Feature": "<feature-name>",
            "Groups": [
                "<group-id>"
            ],
            "OneOfGroups": [
                "<group-id>"
            ],
            "TestCases": [
                "<test-id>"
            ],
            "IsRequired": true | false,
            "ExecutionMethods": [
                "<execution-method>"
            ]
        }
    ]
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`Next`

Der Name des Status, zu dem nach der Ausführung der Aktionen im aktuellen Status übergegangen werden soll.

`Features`

Eine Reihe von Produktfunktionen, die in der `awsiotdevicetester_report.xml` Datei angezeigt werden sollen.

`Feature`

Der Name der Funktion

`FeatureValue`

Optional. Der benutzerdefinierte Wert, der anstelle von im Bericht verwendet werden soll`supported`. Wenn dieser Wert nicht angegeben ist, wird der Feature-Wert basierend auf den Testergebnissen auf `supported` oder gesetzt`not-supported`.

Wenn Sie einen benutzerdefinierten Wert für verwenden`FeatureValue`, können Sie dasselbe Feature mit unterschiedlichen Bedingungen testen, und IDT verkettet die Feature-Werte für die unterstützten Bedingungen. Der folgende Auszug zeigt beispielsweise das `MyFeature` Feature mit zwei separaten Feature-Werten:

```
...
{
    "Feature": "MyFeature",
    "FeatureValue": "first-feature-supported",
    "Groups": ["first-feature-group"]
},
{
    "Feature": "MyFeature",
    "FeatureValue": "second-feature-supported",
    "Groups": ["second-feature-group"]
},
...
```

Wenn beide Testgruppen erfolgreich sind, wird der Feature-Wert auf `first-feature-supported, second-feature-supported` gesetzt.

`Groups`

Optional. Ein Array von Testgruppen IDs. Alle Tests innerhalb jeder angegebenen Testgruppe müssen bestanden werden, damit die Funktion unterstützt wird.

`OneOfGroups`

Optional. Ein Array von Testgruppen IDs. Alle Tests innerhalb mindestens einer der angegebenen Testgruppen müssen bestanden werden, damit die Funktion unterstützt wird.

`TestCases`

Optional. Eine Reihe von Testfällen IDs. Wenn Sie diesen Wert angeben, gilt Folgendes:

- Alle angegebenen Testfälle müssen bestanden werden, damit die Funktion unterstützt wird.
- `Groups`darf nur eine Testgruppen-ID enthalten.
- `OneOfGroups`darf nicht angegeben werden.

`IsRequired`

Optional. Stellen Sie auf ein`false`, um diese Funktion im Bericht als optionale Funktion zu kennzeichnen. Der Standardwert ist `true`.

`ExecutionMethods`

Optional. Eine Reihe von Ausführungsmethoden, die dem in der `device.json` Datei angegebenen `protocol` Wert entsprechen. Wenn dieser Wert angegeben ist, müssen Testläufer einen `protocol` Wert angeben, der einem der Werte in diesem Array entspricht, um das Feature in den Bericht aufzunehmen. Wenn dieser Wert nicht angegeben wird, wird das Feature immer in den Bericht aufgenommen.

Um den `AddProductFeatures` Status verwenden zu können, müssen Sie den Wert von `ResultVar` in the `RunTask` state auf einen der folgenden Werte festlegen:

- Wenn Sie einen einzelnen Testfall angegeben haben IDs, setzen Sie ihn `ResultVar` auf*`group-id_test-id`*_passed.
- Wenn Sie keinen individuellen Testfall angegeben haben IDs, legen Sie `ResultVar` den Wert auf fest*`group-id`*_passed.

Der `AddProductFeatures` Staat sucht auf folgende Weise nach Testergebnissen:

- Wenn Sie keinen Testfall angegeben haben IDs, wird das Ergebnis für jede Testgruppe anhand des Werts der *`group-id`*_passed Variablen im State-Machine-Kontext bestimmt.
- Wenn Sie einen Testfall angegeben haben IDs, wird das Ergebnis für jeden der Tests anhand des Werts der *`group-id_test-id`*_passed Variablen im Zustandsmaschinen-Kontext bestimmt.

Fehlerbehandlung

Wenn eine in diesem Status angegebene Gruppen-ID keine gültige Gruppen-ID ist, führt dieser Status zu einem `AddProductFeaturesError` Ausführungsfehler. Wenn im Status ein Ausführungsfehler auftritt, wird auch die `hasExecutionErrors` Variable im Zustandsmaschinenkontext auf gesetzt`true`.

Bericht

Der `Report` Status generiert die *`suite-name`*`_Report.xml` `awsiotdevicetester_report.xml` UND-Dateien. In diesem Status wird der Bericht auch an die Konsole gestreamt.

```
{
    "Type": "Report",
    "Next": "<state-name>"
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`Next`

Der Name des Status, zu dem nach der Ausführung der Aktionen im aktuellen Status übergegangen werden soll.

Sie sollten immer gegen Ende der Testausführung in den `Report` Status wechseln, damit Testläufer die Testergebnisse einsehen können. In der Regel ist der nächste Status nach diesem Status`Succeed`.

Fehlerbehandlung

Wenn in diesem Status Probleme beim Generieren der Berichte auftreten, wird der `ReportError` Ausführungsfehler ausgegeben.

LogMessage

Der `LogMessage` Status generiert die `test_manager.log` Datei und streamt die Protokollnachricht an die Konsole.

```
{
    "Type": "LogMessage",
```

```
    "Next": "<state-name>"
    "Level": "info | warn | error"
    "Message": "<message>"
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`Next`

Der Name des Status, zu dem nach der Ausführung der Aktionen im aktuellen Status übergegangen werden soll.

`Level`

Die Fehlerstufe, auf der die Protokollnachricht erstellt werden soll. Wenn Sie eine ungültige Stufe angeben, generiert dieser Status eine Fehlermeldung und verwirft sie.

`Message`

Die zu protokollierende Nachricht.

SelectGroup

Der `SelectGroup` Status aktualisiert den Kontext der Zustandsmaschine, um anzugeben, welche Gruppen ausgewählt wurden. Die in diesem Status festgelegten Werte werden von allen nachfolgenden `Choice` Staaten verwendet.

```
{
    "Type": "SelectGroup",
    "Next": "<state-name>"
    "TestGroups": [
        <group-id>"
    ]
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`Next`

Der Name des Status, zu dem nach der Ausführung der Aktionen im aktuellen Status übergegangen werden soll.

`TestGroups`

Eine Reihe von Testgruppen, die als ausgewählt markiert werden. Für jede Testgruppen-ID in diesem Array wird die *`group-id`*`_selected` Variable `true` im Kontext auf gesetzt. Stellen Sie sicher, dass Sie eine gültige Testgruppe angeben IDs , da IDT nicht überprüft, ob die angegebenen Gruppen existieren.

Fehler

Der `Fail` Status weist darauf hin, dass die Zustandsmaschine nicht korrekt ausgeführt wurde. Dies ist ein Endzustand für die Zustandsmaschine, und jede Zustandsmaschine muss diesen Zustand enthalten.

```
{
    "Type": "Fail"
}
```

Succeed

Der `Succeed` Status gibt an, dass die Zustandsmaschine korrekt ausgeführt wurde. Dies ist ein Endzustand für die Zustandsmaschine, und jede Zustandsmaschine muss diesen Zustand enthalten.

```
{
    "Type": "Succeed"
}
```

Kontext der Zustandsmaschine

Der Zustandsmaschinenkontext ist ein schreibgeschütztes JSON-Dokument, das Daten enthält, die der Zustandsmaschine während der Ausführung zur Verfügung stehen. Der Zustandsmaschinen-Kontext ist nur von der Zustandsmaschine aus zugänglich und enthält Informationen, die den Testablauf bestimmen. Sie können beispielsweise Informationen verwenden, die von Testläufern in der `userdata.json` Datei konfiguriert wurden, um zu ermitteln, ob ein bestimmter Test ausgeführt werden muss.

Der State-Machine-Kontext verwendet das folgende Format:

```
{
    "pool": {
        <device-json-pool-element>
```

```
    },
    "userData": {
        <userdata-json-content>
    },
    "config": {
        <config-json-content>
    },
    "suiteFailed": true | false,
    "specificTestGroups": [
        "<group-id>"
    ],
    "specificTestCases": [
        "<test-id>"
    ],
    "hasExecutionErrors": true
}
```

`pool`

Informationen über den Gerätepool, der für den Testlauf ausgewählt wurde. Für einen ausgewählten Gerätepool werden diese Informationen aus dem entsprechenden Gerätepool-Array-Element der obersten Ebene abgerufen, das in der `device.json` Datei definiert ist.

`userData`

Informationen in der `userdata.json` Datei.

`config`

Informationen an die `config.json` Datei anheften.

`suiteFailed`

Der Wert wird auf den `false` Zeitpunkt gesetzt, zu dem die Zustandsmaschine gestartet wird. Wenn eine Testgruppe in einem `RunTask` Status ausfällt, wird dieser Wert `true` für die verbleibende Dauer der State-Machine-Ausführung auf gesetzt.

`specificTestGroups`

Wenn der Testläufer anstelle der gesamten Testsuite bestimmte Testgruppen zur Ausführung auswählt, wird dieser Schlüssel erstellt und enthält die Liste der spezifischen Testgruppen IDs.

`specificTestCases`

Wenn der Testläufer anstelle der gesamten Testsuite bestimmte Testfälle zur Ausführung auswählt, wird dieser Schlüssel erstellt und enthält die Liste der spezifischen Testfälle IDs.

`hasExecutionErrors`

Wird nicht beendet, wenn die Zustandsmaschine gestartet wird. Wenn in einem Status ein Ausführungsfehler auftritt, wird diese Variable erstellt und `true` für die verbleibende Dauer der State-Machine-Ausführung auf „gesetzt“.

Sie können den Kontext mithilfe der JSONPath Notation abfragen. Die Syntax für JSONPath Abfragen in Statusdefinitionen lautet`{{$.query}}`. In einigen Bundesstaaten können Sie JSONPath Abfragen als Platzhalterzeichenfolgen verwenden. IDT ersetzt die Platzhalterzeichenfolgen durch den Wert der ausgewerteten JSONPath Abfrage aus dem Kontext. Sie können Platzhalter für die folgenden Werte verwenden:

- Der `TestCases` Wert in `RunTask` Bundesstaaten.
- Der `Expression` `Choice` Wertstatus.

Wenn Sie auf Daten aus dem State Machine-Kontext zugreifen, stellen Sie sicher, dass die folgenden Bedingungen erfüllt sind:

- Ihre JSON-Pfade müssen beginnen mit `$.`
- Jeder Wert muss eine Zeichenfolge, eine Zahl oder einen booleschen Wert ergeben.

Weitere Hinweise zur Verwendung der JSONPath Notation für den Zugriff auf Daten aus dem Kontext finden Sie unter. Verwenden Sie den IDT-Kontext

Ausführungsfehler

Ausführungsfehler sind Fehler in der Zustandsmaschinen-Definition, auf die der Zustandsmaschine bei der Ausführung eines Zustands stößt. IDT protokolliert Informationen zu jedem Fehler in der `test_manager.log` Datei und streamt die Protokollnachricht an die Konsole.

Sie können die folgenden Methoden verwenden, um Ausführungsfehler zu behandeln:

- Fügen Sie der Statusdefinition einen `CatchBlock` hinzu.
- Überprüfen Sie den `hasExecutionErrors`Wert des Werts im Kontext der Zustandsmaschine.

Fangen

Um es zu verwenden`Catch`, fügen Sie Ihrer Bundesstaatendefinition Folgendes hinzu:

```
"Catch": [
    {
        "ErrorEquals": [
            "<error-type>"
        ]
        "Next": "<state-name>"
    }
]
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`Catch.ErrorEquals`

Eine Reihe von Fehlertypen, die abgefangen werden sollen. Wenn ein Ausführungsfehler mit einem der angegebenen Werte übereinstimmt, wechselt die Zustandsmaschine in den unter angegebenen Status`Catch.Next`. In den einzelnen Statusdefinitionen finden Sie Informationen zur Art des Fehlers, den sie erzeugt.

`Catch.Next`

Der nächste Status, in den übergegangen werden soll, wenn im aktuellen Status ein Ausführungsfehler auftritt, der einem der in angegebenen Werte entspricht`Catch.ErrorEquals`.

Catch-Blöcke werden sequentiell behandelt, bis einer der Blöcke übereinstimmt. Wenn der Wert „Keine Fehler“ mit den in den Catch-Blöcken aufgelisteten Fehlern übereinstimmt, wird die Ausführung der Zustandsmaschinen fortgesetzt. Da Ausführungsfehler auf falsche Statusdefinitionen zurückzuführen sind, empfehlen wir, dass Sie in den Status Fail wechseln, wenn in einem Status ein Ausführungsfehler auftritt.

hasExecutionError

Wenn in einigen Staaten Ausführungsfehler auftreten, geben sie nicht nur den Fehler aus, sondern setzen den `hasExecutionError` Wert auch `true` im Kontext der Zustandsmaschine auf. Sie können diesen Wert verwenden, um zu erkennen, wann ein Fehler auftritt, und dann einen `Choice` Status verwenden, um die Zustandsmaschine in den `Fail` Status zu versetzen.

Diese Methode hat die folgenden Eigenschaften.

- Die Zustandsmaschine beginnt mit keinem Wert, der zugewiesen wurde`hasExecutionError`, und dieser Wert ist erst verfügbar, wenn ein bestimmter Status ihn festlegt. Das bedeutet, dass Sie den Status `false` für die `Choice` Staaten, die `Fallthrough0nError` auf diesen Wert zugreifen,

explizit auf setzen müssen, um zu verhindern, dass der Zustandsmaschine angehalten wird, wenn keine Ausführungsfehler auftreten.

- Sobald der Wert auf gesetzt ist`true`, `hasExecutionError` wird er niemals auf False gesetzt oder aus dem Kontext entfernt. Das bedeutet, dass dieser Wert nur nützlich ist, wenn er zum ersten Mal auf gesetzt wird`true`, und für alle nachfolgenden Zustände bietet er keinen aussagekräftigen Wert.
- Der `hasExecutionError` Wert wird von allen Zweigzustandsmaschinen im `Parallel` Bundesstaat gemeinsam genutzt, was je nach Reihenfolge, in der auf ihn zugegriffen wird, zu unerwarteten Ergebnissen führen kann.

Aufgrund dieser Eigenschaften empfehlen wir nicht, diese Methode zu verwenden, wenn Sie stattdessen einen Catch-Block verwenden können.

Beispiel für Zustandsmaschinen

Dieser Abschnitt enthält einige Beispielkonfigurationen von Zustandsmaschinen.

Beispiele

- Beispiel für eine Zustandsmaschine: Führen Sie eine einzelne Testgruppe aus
- Beispiel für eine Zustandsmaschine: Führen Sie vom Benutzer ausgewählte Testgruppen aus
- Beispiel für eine Zustandsmaschine: Führen Sie eine einzelne Testgruppe mit Produktfunktionen aus
- Beispiel State Machine: Zwei Testgruppen parallel ausführen

Beispiel für eine Zustandsmaschine: Führen Sie eine einzelne Testgruppe aus

Dieser Zustandsmaschine:

- Führt die Testgruppe mit der ID aus`GroupA`, die in der Suite in einer `group.json` Datei vorhanden sein muss.
- Prüft auf Ausführungsfehler und wechselt zu, `Fail` ob welche gefunden wurden.
- Generiert einen Bericht und wechselt zu, `Succeed` ob keine Fehler vorliegen, und `Fail` andernfalls.

```
{
    "Comment": "Runs a single group and then generates a report.",
    "StartAt": "RunGroupA",
```

```
    "States": {
        "RunGroupA": {
            "Type": "RunTask",
            "Next": "Report",
            "TestGroup": "GroupA",
            "Catch": [
                {
                    "ErrorEquals": [
                        "RunTaskError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "Report": {
            "Type": "Report",
            "Next": "Succeed",
            "Catch": [
                {
                    "ErrorEquals": [
                        "ReportError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "Succeed": {
            "Type": "Succeed"
        },
        "Fail": {
            "Type": "Fail"
        }
    }
}
```

Beispiel für eine Zustandsmaschine: Führen Sie vom Benutzer ausgewählte Testgruppen aus

Dieser Zustandsmaschine:

- Prüft, ob der Testläufer bestimmte Testgruppen ausgewählt hat. Die Zustandsmaschine sucht nicht nach bestimmten Testfällen, da Testläufer keine Testfälle auswählen können, ohne auch eine Testgruppe auszuwählen.
- Wenn Testgruppen ausgewählt sind:

  - Führt die Testfälle innerhalb der ausgewählten Testgruppen aus. Zu diesem Zweck spezifiziert die Zustandsmaschine nicht explizit Testgruppen oder Testfälle im `RunTask` Status.
  - Generiert einen Bericht, nachdem alle Tests ausgeführt und beendet wurden.
- Wenn keine Testgruppen ausgewählt sind:
  - Führt Tests in einer Testgruppe aus`GroupA`.
  - Generiert Berichte und beendet das Programm.

```
{
    "Comment": "Runs specific groups if the test runner chose to do that, otherwise
 runs GroupA.",
    "StartAt": "SpecificGroupsCheck",
    "States": {
        "SpecificGroupsCheck": {
            "Type": "Choice",
            "Default": "RunGroupA",
            "FallthroughOnError": true,
            "Choices": [
                {
                    "Expression": "{{$.specificTestGroups[0]}} != ''",
                    "Next": "RunSpecificGroups"
                }
            ]
        },
        "RunSpecificGroups": {
            "Type": "RunTask",
            "Next": "Report",
            "Catch": [
                {
                    "ErrorEquals": [
                        "RunTaskError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "RunGroupA": {
            "Type": "RunTask",
            "Next": "Report",
            "TestGroup": "GroupA",
            "Catch": [
```

```
                {
                    "ErrorEquals": [
                        "RunTaskError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "Report": {
            "Type": "Report",
            "Next": "Succeed",
            "Catch": [
                {
                    "ErrorEquals": [
                        "ReportError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "Succeed": {
            "Type": "Succeed"
        },
        "Fail": {
            "Type": "Fail"
        }
    }
}
```

Beispiel für eine Zustandsmaschine: Führen Sie eine einzelne Testgruppe mit Produktfunktionen aus

Dieser Zustandsmaschine:

- Führt die Testgruppe aus`GroupA`.
- Prüft auf Ausführungsfehler und wechselt, `Fail` ob welche gefunden wurden.
- Fügt der `awsiotdevicetester_report.xml` Datei das `FeatureThatDependsOnGroupA` Feature hinzu:
  - Wenn der `GroupA` Wert erfolgreich ist, wird das Feature auf gesetzt`supported`.
  - Die Funktion ist im Bericht nicht als optional gekennzeichnet.
- Generiert einen Bericht und wechselt zu, `Succeed` wenn keine Fehler vorliegen, und `Fail` andernfalls

```
{
    "Comment": "Runs GroupA and adds product features based on GroupA",
    "StartAt": "RunGroupA",
    "States": {
        "RunGroupA": {
            "Type": "RunTask",
            "Next": "AddProductFeatures",
            "TestGroup": "GroupA",
            "ResultVar": "GroupA_passed",
            "Catch": [
                {
                    "ErrorEquals": [
                        "RunTaskError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "AddProductFeatures": {
            "Type": "AddProductFeatures",
            "Next": "Report",
            "Features": [
                {
                    "Feature": "FeatureThatDependsOnGroupA",
                    "Groups": [
                        "GroupA"
                    ],
                    "IsRequired": true
                }
            ]
        },
        "Report": {
            "Type": "Report",
            "Next": "Succeed",
            "Catch": [
                {
                    "ErrorEquals": [
                        "ReportError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
```

```
        "Succeed": {
            "Type": "Succeed"
        },
        "Fail": {
            "Type": "Fail"
        }
    }
}
```

Beispiel State Machine: Zwei Testgruppen parallel ausführen

Diese Zustandsmaschine:

- Führt die Gruppen `GroupA` und die `GroupB` Testgruppen parallel aus. Die `ResultVar` Variablen, die im Kontext der `RunTask` Bundesstaaten gespeichert sind, stehen dem `AddProductFeatures` Staat zur Verfügung.
- Prüft auf Ausführungsfehler und wechselt, `Fail` falls welche gefunden wurden. Diese Zustandsmaschine verwendet keinen `Catch` Block, da diese Methode keine Ausführungsfehler in Zweigzustandsmaschinen erkennt.
- Fügt der `awsiotdevicetester_report.xml` Datei Funktionen hinzu, die auf den Gruppen basieren, die erfolgreich sind
  - Bei `GroupA` erfolgreicher Prüfung wird das Feature auf gesetzt`supported`.
  - Die Funktion ist im Bericht nicht als optional gekennzeichnet.
- Generiert einen Bericht und wechselt zu, `Succeed` wenn keine Fehler vorliegen, und `Fail` andernfalls

Wenn zwei Geräte im Gerätepool konfiguriert sind, `GroupB` können beide `GroupA` Geräte gleichzeitig ausgeführt werden. Wenn jedoch einer `GroupA` oder `GroupB` mehrere Tests enthalten sind, können beide Geräte diesen Tests zugewiesen werden. Wenn nur ein Gerät konfiguriert ist, werden die Testgruppen nacheinander ausgeführt.

```
{
    "Comment": "Runs GroupA and GroupB in parallel",
    "StartAt": "RunGroupAAndB",
    "States": {
        "RunGroupAAndB": {
            "Type": "Parallel",
            "Next": "CheckForErrors",
            "Branches": [
```

```
                {
                    "Comment": "Run GroupA state machine",
                    "StartAt": "RunGroupA",
                    "States": {
                        "RunGroupA": {
                            "Type": "RunTask",
                            "Next": "Succeed",
                            "TestGroup": "GroupA",
                            "ResultVar": "GroupA_passed",
                            "Catch": [
                                {
                                    "ErrorEquals": [
                                        "RunTaskError"
                                    ],
                                    "Next": "Fail"
                                }
                            ]
                        },
                        "Succeed": {
                            "Type": "Succeed"
                        },
                        "Fail": {
                            "Type": "Fail"
                        }
                    }
                },
                {
                    "Comment": "Run GroupB state machine",
                    "StartAt": "RunGroupB",
                    "States": {
                        "RunGroupA": {
                            "Type": "RunTask",
                            "Next": "Succeed",
                            "TestGroup": "GroupB",
                            "ResultVar": "GroupB_passed",
                            "Catch": [
                                {
                                    "ErrorEquals": [
                                        "RunTaskError"
                                    ],
                                    "Next": "Fail"
                                }
                            ]
                        },
```

```
                    "Succeed": {
                        "Type": "Succeed"
                    },
                    "Fail": {
                        "Type": "Fail"
                    }
                }
            }
        ]
    },
    "CheckForErrors": {
        "Type": "Choice",
        "Default": "AddProductFeatures",
        "FallthroughOnError": true,
        "Choices": [
            {
                "Expression": "{{$.hasExecutionErrors}} == true",
                "Next": "Fail"
            }
        ]
    },
    "AddProductFeatures": {
        "Type": "AddProductFeatures",
        "Next": "Report",
        "Features": [
            {
                "Feature": "FeatureThatDependsOnGroupA",
                "Groups": [
                    "GroupA"
                ],
                "IsRequired": true
            },
            {
                "Feature": "FeatureThatDependsOnGroupB",
                "Groups": [
                    "GroupB"
                ],
                "IsRequired": true
            }
        ]
    },
    "Report": {
        "Type": "Report",
        "Next": "Succeed",
```

```
            "Catch": [
                {
                    "ErrorEquals": [
                        "ReportError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "Succeed": {
            "Type": "Succeed"
        },
        "Fail": {
            "Type": "Fail"
        }
    }
}
```

Erstellen Sie eine ausführbare IDT-Testfalldatei

Sie können eine ausführbare Testfalldatei auf folgende Weise erstellen und in einem Testsuite-Ordner ablegen:

- Für Testsuiten, die Argumente oder Umgebungsvariablen aus den `test.json` Dateien verwenden, um zu bestimmen, welche Tests ausgeführt werden sollen, können Sie einen einzelnen ausführbaren Testfall für die gesamte Testsuite oder eine ausführbare Testdatei für jede Testgruppe in der Testsuite erstellen.
- Für eine Testsuite, in der Sie bestimmte Tests auf der Grundlage bestimmter Befehle ausführen möchten, erstellen Sie für jeden Testfall in der Testsuite eine ausführbare Testfalldatei.

Als Testautor können Sie bestimmen, welcher Ansatz für Ihren Anwendungsfall geeignet ist, und die ausführbare Testfalldatei entsprechend strukturieren. Stellen Sie sicher, dass Sie in jeder `test.json` Datei den richtigen Pfad für die ausführbare Testfalldatei angeben und dass die angegebene ausführbare Datei korrekt ausgeführt wird.

Wenn alle Geräte für die Ausführung eines Testfalls bereit sind, liest IDT die folgenden Dateien:

- Der `test.json` für den ausgewählten Testfall festgelegte Prozess bestimmt, welche Prozesse gestartet und welche Umgebungsvariablen gesetzt werden sollen.
- Die `suite.json` für die Testsuite bestimmt die zu setzenden Umgebungsvariablen.

IDT startet den erforderlichen ausführbaren Testprozess auf der Grundlage der in der `test.json` Datei angegebenen Befehle und Argumente und übergibt die erforderlichen Umgebungsvariablen an den Prozess.

Verwenden Sie das IDT Client SDK

Mit dem IDT-Client SDKs können Sie das Schreiben von Testlogik in Ihre Testdatei mithilfe von API-Befehlen vereinfachen, die Sie verwenden können, um mit IDT und Ihren zu testenden Geräten zu interagieren. IDT bietet derzeit Folgendes: SDKs

- IDT-Client-SDK für Python
- IDT Client SDK for Go
- IDT Client SDK for Java

Diese SDKs befinden sich im *<device-tester-extract-location>*`/sdks` Ordner. Wenn Sie eine neue ausführbare Testfalldatei erstellen, müssen Sie das SDK, das Sie verwenden möchten, in den Ordner kopieren, der Ihre ausführbare Testfalldatei enthält, und in Ihrem Code auf das SDK verweisen. Dieser Abschnitt enthält eine kurze Beschreibung der verfügbaren API-Befehle, die Sie in Ihren ausführbaren Testfalldateien verwenden können.

In diesem Abschnitt

- Interaktion mit Geräten
- IDT-Interaktion
- Interaktion mit dem Host

Interaktion mit Geräten

Mit den folgenden Befehlen können Sie mit dem zu testenden Gerät kommunizieren, ohne zusätzliche Funktionen für die Geräteinteraktion und das Konnektivitätsmanagement implementieren zu müssen.

`ExecuteOnDevice`

Ermöglicht es Testsuiten, Shell-Befehle auf einem Gerät auszuführen, das SSH- oder Docker-Shell-Verbindungen unterstützt.

`CopyToDevice`

Ermöglicht Testsuiten, eine lokale Datei vom Host-Computer, auf dem IDT ausgeführt wird, an einen bestimmten Speicherort auf einem Gerät zu kopieren, das SSH- oder Docker-Shell-Verbindungen unterstützt.

`ReadFromDevice`

Ermöglicht es Testsuiten, von der seriellen Schnittstelle von Geräten zu lesen, die UART-Verbindungen unterstützen.

> **Note**
>
> Da IDT keine direkten Verbindungen zu Geräten verwaltet, die mithilfe von Gerätezugriffsinformationen aus dem Kontext hergestellt werden, empfehlen wir, diese API-Befehle für Geräteinteraktionen in Ihren ausführbaren Testfalldateien zu verwenden. Wenn diese Befehle jedoch nicht Ihren Testfallanforderungen entsprechen, können Sie Gerätezugriffsinformationen aus dem IDT-Kontext abrufen und sie verwenden, um über die Testsuite eine direkte Verbindung zum Gerät herzustellen.
> Um eine direkte Verbindung herzustellen, rufen Sie die Informationen in den `device.connectivity` `resource.devices.connectivity` Feldern für Ihr zu testendes Gerät bzw. für Ressourcengeräte ab. Weitere Informationen zur Verwendung des IDT-Kontextes finden Sie unterVerwenden Sie den IDT-Kontext.

IDT-Interaktion

Mit den folgenden Befehlen können Ihre Testsuiten mit IDT kommunizieren.

`PollForNotifications`

Ermöglicht Testsuiten, nach Benachrichtigungen von IDT zu suchen.

`GetContextValue` und **`GetContextString`**

Ermöglicht Testsuiten das Abrufen von Werten aus dem IDT-Kontext. Weitere Informationen finden Sie unter Verwenden Sie den IDT-Kontext.

`SendResult`

Ermöglicht es Testsuiten, Testfallergebnisse an IDT zu melden. Dieser Befehl muss am Ende jedes Testfalls in einer Testsuite aufgerufen werden.

Interaktion mit dem Host

Mit dem folgenden Befehl können Ihre Testsuiten mit dem Host-Computer kommunizieren.

`PollForNotifications`

Ermöglicht Testsuiten, nach Benachrichtigungen von IDT zu suchen.

`GetContextValue` und **`GetContextString`**

Ermöglicht Testsuiten das Abrufen von Werten aus dem IDT-Kontext. Weitere Informationen finden Sie unter Verwenden Sie den IDT-Kontext.

`ExecuteOnHost`

Ermöglicht Testsuiten die Ausführung von Befehlen auf dem lokalen Computer und ermöglicht IDT die Verwaltung des Lebenszyklus der ausführbaren Testfälle.

IDT-CLI-Befehle aktivieren

Der `run-suite` Befehl IDT CLI bietet mehrere Optionen, mit denen Test Runner die Testausführung anpassen kann. Damit Testläufer diese Optionen zum Ausführen Ihrer benutzerdefinierten Testsuite verwenden können, implementieren Sie Unterstützung für die IDT-CLI. Wenn Sie keine Unterstützung implementieren, können Testläufer weiterhin Tests ausführen, aber einige CLI-Optionen funktionieren nicht richtig. Um ein optimales Kundenerlebnis zu bieten, empfehlen wir, die Unterstützung für die folgenden Argumente für den `run-suite` Befehl in der IDT-CLI zu implementieren:

`timeout-multiplier`

Gibt einen Wert größer als 1,0 an, der auf alle Timeouts beim Ausführen von Tests angewendet wird.

Testläufer können dieses Argument verwenden, um das Timeout für die Testfälle zu erhöhen, die sie ausführen möchten. Wenn ein Testläufer dieses Argument in seinem `run-suite` Befehl angibt, verwendet IDT es, um den Wert der Umgebungsvariablen IDT_TEST_TIMEOUT zu berechnen, und legt das Feld im IDT-Kontext fest. `config.timeoutMultiplier` Um dieses Argument zu unterstützen, müssen Sie wie folgt vorgehen:

- Anstatt den Timeout-Wert direkt aus der `test.json` Datei zu verwenden, lesen Sie die Umgebungsvariable IDT_TEST_TIMEOUT, um den korrekt berechneten Timeout-Wert zu erhalten.

- Rufen Sie den `config.timeoutMultiplier` Wert aus dem IDT-Kontext ab und wenden Sie ihn auf Timeouts mit langer Laufzeit an.

Weitere Hinweise zum vorzeitigen Beenden aufgrund von Timeout-Ereignissen finden Sie unter. Geben Sie das Exit-Verhalten an

`stop-on-first-failure`

Gibt an, dass IDT die Ausführung aller Tests beenden soll, wenn ein Fehler auftritt.

Wenn ein Testläufer dieses Argument in seinem `run-suite` Befehl angibt, beendet IDT die Ausführung von Tests, sobald ein Fehler auftritt. Wenn Testfälle jedoch parallel ausgeführt werden, kann dies zu unerwarteten Ergebnissen führen. Um Support zu implementieren, stellen Sie sicher, dass Ihre Testlogik alle laufenden Testfälle anweist, anzuhalten, temporäre Ressourcen zu bereinigen und ein Testergebnis an IDT zu melden, wenn IDT auf dieses Ereignis trifft. Weitere Informationen zum vorzeitigen Beenden von Fehlern finden Sie unter. Geben Sie das Exit-Verhalten an

`group-id` und **`test-id`**

Gibt an, dass IDT nur die ausgewählten Testgruppen oder Testfälle ausführen soll.

Testläufer können diese Argumente mit ihrem `run-suite` Befehl verwenden, um das folgende Verhalten bei der Testausführung anzugeben:

- Führt alle Tests innerhalb der angegebenen Testgruppen aus.
- Führt eine Auswahl von Tests innerhalb einer angegebenen Testgruppe aus.

Um diese Argumente zu unterstützen, muss die Zustandsmaschine für Ihre Testsuite einen bestimmten Satz von `RunTask Choice` Zuständen in Ihrer Zustandsmaschine enthalten. Wenn Sie keine benutzerdefinierte Zustandsmaschine verwenden, enthält die standardmäßige IDT-Zustandsmaschine die erforderlichen Zustände für Sie, und Sie müssen keine zusätzlichen Maßnahmen ergreifen. Wenn Sie jedoch eine benutzerdefinierte Zustandsmaschine verwenden, verwenden Sie sie Beispiel für eine Zustandsmaschine: Führen Sie vom Benutzer ausgewählte Testgruppen aus als Beispiel, um die erforderlichen Zustände zu Ihrer Zustandsmaschine hinzuzufügen.

Weitere Informationen zu IDT-CLI-Befehlen finden Sie unterDebuggen Sie benutzerdefinierte Testsuiten und führen Sie sie aus.

Schreiben Sie Ereignisprotokolle

Während der Test läuft, senden Sie Daten an `stdout` `stderr` die Konsole und schreiben dort Ereignisprotokolle und Fehlermeldungen. Hinweise zum Format von Konsolenmeldungen finden Sie unterNachrichtenformat der Konsole.

Wenn das IDT die Ausführung der Testsuite beendet hat, sind diese Informationen auch in der `test_manager.log` Datei im *<devicetester-extract-location>*`/results/`*<execution-id>*`/logs` Ordner verfügbar.

Sie können jeden Testfall so konfigurieren, dass die Protokolle des Testlaufs, einschließlich der Protokolle des zu testenden Geräts, in die *<group-id>*`_`*<test-id>* Datei im *<device-tester-extract-location>*`/results/`*execution-id*`/logs` Ordner geschrieben werden. Rufen Sie dazu den Pfad zur Protokolldatei aus dem IDT-Kontext mit der `testData.logFilePath` Abfrage ab, erstellen Sie eine Datei unter diesem Pfad und schreiben Sie den gewünschten Inhalt hinein. IDT aktualisiert den Pfad automatisch auf der Grundlage des laufenden Testfalls. Wenn Sie sich dafür entscheiden, die Protokolldatei für einen Testfall nicht zu erstellen, wird keine Datei für diesen Testfall generiert.

Sie können Ihre ausführbare Textdatei auch so einrichten, dass sie bei Bedarf zusätzliche Protokolldateien im *<device-tester-extract-location>*`/logs` Ordner erstellt. Wir empfehlen Ihnen, eindeutige Präfixe für Protokolldateinamen anzugeben, damit Ihre Dateien nicht überschrieben werden.

Ergebnisse an IDT melden

IDT schreibt Testergebnisse in die `awsiotdevicetester_report.xml` und die *suite-name*`_report.xml` Dateien. Diese Berichtsdateien befinden sich in*<device-tester-extract-location>*`/results/`*<execution-id>*`/`. Beide Berichte erfassen die Ergebnisse der Ausführung der Testsuite. Weitere Informationen zu den Schemas, die IDT für diese Berichte verwendet, finden Sie unter Überprüfen Sie die IDT-Testergebnisse und -Protokolle

Um den Inhalt der *suite-name*`_report.xml` Datei aufzufüllen, müssen Sie den `SendResult` Befehl verwenden, um die Testergebnisse an IDT zu melden, bevor die Testausführung abgeschlossen ist. Wenn IDT die Ergebnisse eines Tests nicht finden kann, gibt es einen Fehler für den Testfall aus. Der folgende Python-Auszug zeigt die Befehle zum Senden eines Testergebnisses an IDT:

```
request-variable = SendResultRequest(TestResult(result))
client.send_result(request-variable)
```

Wenn Sie Ergebnisse nicht über die API melden, sucht IDT im Ordner mit den Testartefakten nach Testergebnissen. Der Pfad zu diesem Ordner wird in der Datei im `testData.testArtifactsPath` IDT-Kontext gespeichert. In diesem Ordner verwendet IDT die erste alphabetisch sortierte XML-Datei, die es findet, als Testergebnis.

Wenn Ihre Testlogik JUnit XML-Ergebnisse liefert, können Sie die Testergebnisse in eine XML-Datei im Ordner artefacts schreiben, um die Ergebnisse direkt an IDT weiterzugeben, anstatt die Ergebnisse zu analysieren und sie dann über die API an IDT zu senden.

Wenn Sie diese Methode verwenden, stellen Sie sicher, dass Ihre Testlogik die Testergebnisse korrekt zusammenfasst, und formatieren Sie Ihre Ergebnisdatei im gleichen Format wie die Datei. *`suite-name`*`_report.xml` IDT führt keine Überprüfung der von Ihnen bereitgestellten Daten durch, mit den folgenden Ausnahmen:

- IDT ignoriert alle Eigenschaften des Tags. `testsuites` Stattdessen werden die Tag-Eigenschaften anhand anderer gemeldeter Testgruppenergebnisse berechnet.
- Darin `testsuites` muss mindestens ein `testsuite` Tag vorhanden sein.

Da IDT für alle Testfälle denselben Ordner mit Artefakten verwendet und Ergebnisdateien nicht zwischen Testläufen löscht, kann diese Methode auch zu fehlerhaften Berichten führen, wenn IDT die falsche Datei liest. Es wird empfohlen, für alle Testfälle denselben Namen für die generierte XML-Ergebnisdatei zu verwenden, um die Ergebnisse für jeden Testfall zu überschreiben und sicherzustellen, dass IDT die richtigen Ergebnisse zur Verfügung hat. Sie können in Ihrer Testsuite zwar einen gemischten Ansatz für die Berichterstattung verwenden, d. h. für einige Testfälle eine XML-Ergebnisdatei verwenden und für andere die Ergebnisse über die API einreichen, wir empfehlen diesen Ansatz jedoch nicht.

Geben Sie das Exit-Verhalten an

Konfigurieren Sie Ihre ausführbare Textdatei so, dass sie immer mit dem Exit-Code 0 beendet wird, auch wenn ein Testfall einen Fehler oder ein Fehlerergebnis meldet. Verwenden Sie Exit-Codes ungleich Null nur, um anzuzeigen, dass ein Testfall nicht ausgeführt wurde oder dass die ausführbare Testfalldatei keine Ergebnisse an IDT übermitteln konnte. Wenn IDT einen Exit-Code ungleich Null empfängt, bedeutet dies, dass der Testfall auf einen Fehler gestoßen ist, der seine Ausführung verhindert hat.

IDT kann in den folgenden Fällen anfordern oder erwarten, dass die Ausführung eines Testfalls beendet wird, bevor er abgeschlossen ist. Verwenden Sie diese Informationen, um Ihre ausführbare Testfalldatei so zu konfigurieren, dass jedes dieser Ereignisse anhand des Testfalls erkannt wird:

Timeout (Zeitüberschreitung)

Tritt auf, wenn ein Testfall länger als der in der `test.json` Datei angegebene Timeout-Wert ausgeführt wird. Wenn der Testläufer das `timeout-multiplier` Argument verwendet hat, um einen Timeout-Multiplikator anzugeben, berechnet IDT den Timeout-Wert mit dem Multiplikator.

Verwenden Sie die Umgebungsvariable IDT_TEST_TIMEOUT, um dieses Ereignis zu erkennen. Wenn ein Testläufer einen Test startet, setzt IDT den Wert der Umgebungsvariablen IDT_TEST_TIMEOUT auf den berechneten Timeout-Wert (in Sekunden) und übergibt die Variable an die ausführbare Testfalldatei. Sie können den Variablenwert lesen, um einen geeigneten Timer festzulegen.

Unterbrechen

Tritt auf, wenn der Test-Runner IDT unterbricht. Zum Beispiel durch Drücken von. Ctrl+C

Da Terminals Signale an alle untergeordneten Prozesse weiterleiten, können Sie in Ihren Testfällen einfach einen Signal-Handler konfigurieren, um Interrupt-Signale zu erkennen.

Alternativ können Sie die API regelmäßig abfragen, um den Wert des `CancellationRequested` booleschen Werts in der API-Antwort zu überprüfen. `PollForNotifications`
Wenn IDT ein Interruptsignal empfängt, setzt es den Wert des booleschen Werts auf. `CancellationRequested` `true`

Stoppt beim ersten Fehler

Tritt auf, wenn ein Testfall, der parallel zum aktuellen Testfall ausgeführt wird, fehlschlägt und der Testläufer das `stop-on-first-failure` Argument verwendet hat, um anzugeben, dass IDT beendet werden soll, wenn ein Fehler auftritt.

Um dieses Ereignis zu erkennen, können Sie die API regelmäßig abfragen, um den Wert des `CancellationRequested` booleschen Werts in der `PollForNotifications` API-Antwort zu überprüfen. Wenn IDT auf einen Fehler stößt und so konfiguriert ist, dass es beim ersten Fehler stoppt, wird der Wert des booleschen Werts `CancellationRequested` auf gesetzt. `true`

Wenn eines dieser Ereignisse eintritt, wartet IDT 5 Minuten, bis alle derzeit laufenden Testfälle abgeschlossen sind. Wenn nicht alle laufenden Testfälle innerhalb von 5 Minuten beendet werden, erzwingt IDT, jeden ihrer Prozesse zu beenden. Wenn IDT vor dem Ende der Prozesse keine Testergebnisse erhalten hat, werden die Testfälle als Timeout markiert. Als bewährte Methode sollten Sie sicherstellen, dass Ihre Testfälle die folgenden Aktionen ausführen, wenn sie auf eines der Ereignisse stoßen:

1. Beenden Sie die Ausführung der normalen Testlogik.
2. Bereinigen Sie alle temporären Ressourcen, z. B. Testartefakte, auf dem zu testenden Gerät.
3. Melden Sie IDT ein Testergebnis, z. B. einen Testfehler oder einen Fehler.
4. Beenden.

Verwenden Sie den IDT-Kontext

Wenn IDT eine Testsuite ausführt, kann die Testsuite auf einen Datensatz zugreifen, anhand dessen bestimmt werden kann, wie die einzelnen Tests ausgeführt werden. Diese Daten werden als IDT-Kontext bezeichnet. Beispielsweise wird die Benutzerdatenkonfiguration, die von Testläufern in einer `userdata.json` Datei bereitgestellt wird, Testsuiten im IDT-Kontext zur Verfügung gestellt.

Der IDT-Kontext kann als schreibgeschütztes JSON-Dokument betrachtet werden. Testsuiten können mithilfe von Standard-JSON-Datentypen wie Objekten, Arrays, Zahlen usw. Daten aus dem Kontext abrufen und Daten in den Kontext schreiben.

Kontextschema

Der IDT-Kontext verwendet das folgende Format:

```
{
    "config": {
        <config-json-content>
        "timeoutMultiplier": timeout-multiplier,
        "idtRootPath": <path/to/IDT/root>
    },
    "device": {
        <device-json-device-element>
    },
    "devicePool": {
        <device-json-pool-element>
    },
    "resource": {
        "devices": [
            {
                <resource-json-device-element>
                "name": "<resource-name>"
            }
        ]
    },
    "testData": {
```

```
        "awsCredentials": {
            "awsAccessKeyId": "<access-key-id>",
            "awsSecretAccessKey": "<secret-access-key>",
            "awsSessionToken": "<session-token>"
        },
        "logFilePath": "/path/to/log/file"
    },
    "userData": {
        <userdata-json-content>
    }
}
```

`config`

Informationen aus der `config.json`Datei. Das `config` Feld enthält außerdem die folgenden zusätzlichen Felder:

`config.timeoutMultiplier`

Der Multiplikator für jeden Timeout-Wert, der von der Testsuite verwendet wird. Dieser Wert wird vom Test-Runner über die IDT-CLI angegeben. Der Standardwert ist 1.

`config.idRootPath`

Dieser Wert ist ein Platzhalter für den absoluten Pfadwert von IDT bei der Konfiguration der Datei. `userdata.json` Dies wird von den Befehlen Build und Flash verwendet.

`device`

Informationen über das für den Testlauf ausgewählte Gerät. Diese Information entspricht dem `devices` Array-Element in der `device.json`Datei für das ausgewählte Gerät.

`devicePool`

Informationen über den Gerätepool, der für den Testlauf ausgewählt wurde. Diese Informationen entsprechen dem Element des Gerätepool-Arrays der obersten Ebene, das in der `device.json` Datei für den ausgewählten Gerätepool definiert ist.

`resource`

Informationen zu Ressourcengeräten aus der `resource.json` Datei.

`resource.devices`

Diese Informationen entsprechen dem in der `resource.json` Datei definierten `devices` Array. Jedes `devices` Element enthält das folgende zusätzliche Feld:

`resource.device.name`

Der Name des Ressourcengeräts. Dieser Wert ist auf den `requiredResource.name` Wert in der `test.json` Datei festgelegt.

`testData.awsCredentials`

Die AWS Anmeldeinformationen, die vom Test für die Verbindung mit der AWS Cloud verwendet wurden. Diese Informationen werden aus der `config.json` Datei abgerufen.

`testData.logFilePath`

Der Pfad zur Protokolldatei, in die der Testfall Protokollnachrichten schreibt. Die Testsuite erstellt diese Datei, falls sie nicht existiert.

`userData`

Informationen, die vom Testrunner in der `userdata.json`Datei bereitgestellt wurden.

Greifen Sie auf Daten im Kontext zu

Sie können den Kontext anhand der JSONPath Notation aus Ihren Konfigurationsdateien und aus Ihrer ausführbaren Textdatei mit dem `GetContextValue` und abfragen `GetContextString` APIs. Die Syntax für JSONPath Zeichenketten für den Zugriff auf den IDT-Kontext variiert wie folgt:

- In `suite.json` und `test.json` verwenden `{{`*`query`*`}}` Sie. Das heißt, verwenden Sie nicht das Stammelement`$.`, um Ihren Ausdruck zu starten.
- In`statemachine.json`, du benutzt`{{$.`*`query`*`}}`.
- In API-Befehlen verwenden Sie je nach Befehl *`query`* oder`{{$.`*`query`*`}}`. Weitere Informationen finden Sie in der Inline-Dokumentation im SDKs.

In der folgenden Tabelle werden die Operatoren in einem typischen JSONPath Foobar-Ausdruck beschrieben:

Operator	Description
$	Das Stammelement. Da es sich bei dem Kontextwert der obersten Ebene für IDT um ein Objekt handelt, verwenden Sie ihn in der Regel, um Ihre Abfragen `$.` zu starten.

Operator	Description
`.childName`	Greift auf das untergeordnete Element mit dem Namen eines Objekts `childName` zu. Wenn es auf ein Array angewendet wird, ergibt dies ein neues Array, bei dem dieser Operator auf jedes Element angewendet wird. Beim Elementnamen wird zwischen Groß- und Kleinschreibung unterschieden. Die Abfrage für den Zugriff auf den `awsRegion` Wert im `config` Objekt lautet beispielsweise `$.config.awsRegion` .
`[start:end]`	Filtert Elemente aus einem Array und ruft Elemente ab, die mit dem `start` Index beginnen und bis zum `end` Index aufsteigen, jeweils inklusive.
`[index1, index2, ... , indexN]`	Filtert Elemente aus einem Array und ruft nur Elemente aus den angegebenen Indizes ab.
`[?(expr)]`	Filtert Elemente aus einem Array mithilfe des `expr` Ausdrucks. Dieser Ausdruck muss einen booleschen Wert ergeben.

Verwenden Sie die folgende Syntax, um Filterausdrücke zu erstellen:

```
<jsonpath> | <value> operator <jsonpath> | <value>
```

In dieser Syntax gilt:

- `jsonpath`ist ein JSONPath , das die Standard-JSON-Syntax verwendet.
- `value`ist ein beliebiger benutzerdefinierter Wert, der die Standard-JSON-Syntax verwendet.
- `operator`ist einer der folgenden Operatoren:
  - <(Weniger als)
  - <=(Weniger als oder gleich)

- ==(Gleich)

  Wenn der Wert JSONPath oder in Ihrem Ausdruck ein Array, ein boolescher Wert oder ein Objektwert ist, dann ist dies der einzige unterstützte binäre Operator, den Sie verwenden können.
- >=(Größer als oder gleich)
- >(Größer als)
- =~(Übereinstimmung mit regulären Ausdrücken). Um diesen Operator in einem Filterausdruck zu verwenden, muss der Wert JSONPath oder auf der linken Seite des Ausdrucks eine Zeichenfolge ergeben, und auf der rechten Seite muss es sich um einen Musterwert handeln, der der RE2Syntax folgt.

Sie können JSONPath Abfragen in der Form {{*query*}} als Platzhalterzeichenfolgen innerhalb der `environmentVariables` Felder `args` und in `test.json` Dateien und innerhalb der `environmentVariables` Felder in `suite.json` Dateien verwenden. IDT führt eine Kontextsuche durch und füllt die Felder mit dem ausgewerteten Wert der Abfrage auf. In der `suite.json` Datei können Sie beispielsweise Platzhalterzeichenfolgen verwenden, um Umgebungsvariablenwerte anzugeben, die sich mit jedem Testfall ändern, und IDT füllt die Umgebungsvariablen mit dem richtigen Wert für jeden Testfall. Wenn Sie jedoch Platzhalterzeichenfolgen in `test.json` und `suite.json` -Dateien verwenden, gelten für Ihre Abfragen die folgenden Überlegungen:

- Sie müssen jedes Vorkommen des `devicePool` Schlüssels in Ihrer Abfrage in Kleinbuchstaben angeben. Das heißt, verwenden Sie `devicepool` stattdessen.
- Für Arrays können Sie nur Zeichenkettenarrays verwenden. Darüber hinaus verwenden Arrays ein nicht standardmäßiges Format. `item1, item2,...,itemN` Wenn das Array nur ein Element enthält, wird es als serialisiert`item`, sodass es nicht von einem Zeichenkettenfeld zu unterscheiden ist.
- Sie können keine Platzhalter verwenden, um Objekte aus dem Kontext abzurufen.

Aus diesen Gründen empfehlen wir, wann immer möglich, die API für den Zugriff auf den Kontext in Ihrer Testlogik anstelle von Platzhalterzeichenfolgen in `test.json` und `suite.json` -Dateien zu verwenden. In einigen Fällen kann es jedoch praktischer sein, JSONPath Platzhalter zu verwenden, um einzelne Zeichenketten abzurufen, die als Umgebungsvariablen festgelegt werden sollen.

Konfigurieren Sie Einstellungen für Testläufer

Um benutzerdefinierte Testsuiten auszuführen, müssen Testläufer ihre Einstellungen auf der Grundlage der Testsuite konfigurieren, die sie ausführen möchten. Die Einstellungen werden auf der Grundlage von Vorlagen für Konfigurationsdateien angegeben, die sich im *<device-tester-extract-location>*`/configs/` Ordner befinden. Falls erforderlich, müssen Testläufer auch AWS Anmeldeinformationen einrichten, die IDT für die Verbindung mit der AWS Cloud verwendet.

Als Testautor müssen Sie diese Dateien konfigurieren, um Ihre Testsuite zu debuggen. Sie müssen den Testläufern Anweisungen geben, damit sie die folgenden Einstellungen nach Bedarf für die Ausführung Ihrer Testsuiten konfigurieren können.

Konfigurieren von device.json

Die `device.json` Datei enthält Informationen über die Geräte, auf denen Tests ausgeführt werden (z. B. IP-Adresse, Anmeldeinformationen, Betriebssystem und CPU-Architektur).

Testläufer können diese Informationen mithilfe der folgenden `device.json` Vorlagendatei bereitstellen, die sich im *<device-tester-extract-location>*`/configs/` Ordner befindet.

```
[
    {
        "id": "<pool-id>",
        "sku": "<pool-sku>",
        "features": [
            {
                "name": "<feature-name>",
                "value": "<feature-value>",
                "configs": [
                    {
                        "name": "<config-name>",
                        "value": "<config-value>"
                    }
                ],
            }
        ],
        "devices": [
            {
                "id": "<device-id>",
                "pairedResource": "<device-id>", //used for no-op protocol
                "connectivity": {
                    "protocol": "ssh | uart | docker | no-op",
```

```
                    // ssh
                    "ip": "<ip-address>",
                    "port": <port-number>,
                    "publicKeyPath": "<public-key-path>",
                    "auth": {
                        "method": "pki | password",
                        "credentials": {
                            "user": "<user-name>",
                            // pki
                            "privKeyPath": "/path/to/private/key",

                            // password
                            "password": "<password>",
                        }
                    },

                    // uart
                    "serialPort": "<serial-port>",

                    // docker
                    "containerId": "<container-id>",
                    "containerUser": "<container-user-name>",
                }
            }
        ]
    }
]
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`id`

Eine benutzerdefinierte alphanumerische ID, die eine Sammlung von Geräten, den sogenannten Gerätepool, eindeutig identifiziert. Geräte, die zu einem Pool gehören, müssen über identische Hardware verfügen. Bei der Ausführung einer Reihe von Tests werden Geräte im Pool verwendet, um die Workload zu parallelisieren. Mehrere Geräte werden verwendet, um verschiedene Tests auszuführen.

`sku`

Ein alphanumerischer Wert, durch den das zu testende Gerät eindeutig identifiziert wird. Die SKU wird verwendet, um qualifizierte Geräte nachzuverfolgen.

> **Note**
>
> Wenn du dein Board im Gerätekatalog für AWS Partner anbieten möchtest, muss die hier angegebene SKU mit der SKU übereinstimmen, die du bei der Angebotserstellung verwendest.

`features`

Optional. Ein Array, das die Funktionen enthält, die das Gerät unterstützt. Gerätefunktionen sind benutzerdefinierte Werte, die Sie in Ihrer Testsuite konfigurieren. Sie müssen Ihren Testläufern Informationen über die Namen und Werte der Funktionen zur Verfügung stellen, die in die `device.json` Datei aufgenommen werden sollen. Wenn Sie beispielsweise ein Gerät testen möchten, das als MQTT-Server für andere Geräte fungiert, können Sie Ihre Testlogik so konfigurieren, dass bestimmte unterstützte Stufen für ein Feature mit dem Namen `MQTT_QoS` validiert werden. Testläufer geben diesen Funktionsnamen an und setzen den Funktionswert auf die von ihrem Gerät unterstützten QoS-Stufen. Sie können die bereitgestellten Informationen mit der Abfrage aus dem IDT-Kontext oder mit der `devicePool.features` Abfrage aus dem State-Machine-Kontext abrufen. `pool.features`

`features.name`

Der Name des Features.

`features.value`

Die unterstützten Feature-Werte.

`features.configs`

Konfigurationseinstellungen für die Funktion, falls erforderlich.

`features.config.name`

Der Name der Konfigurationseinstellung.

`features.config.value`

Die unterstützten Einstellungswerte.

`devices`

Eine Reihe von Geräten im Pool, die getestet werden sollen. Es ist mindestens ein Gerät erforderlich.

`devices.id`

Eine benutzerdefinierte eindeutige Kennung für das zu testende Gerät.

`devices.pairedResource`

Eine benutzerdefinierte eindeutige Kennung für ein Ressourcengerät. Dieser Wert ist erforderlich, wenn Sie Geräte testen, die das `no-op` Konnektivitätsprotokoll verwenden.

`connectivity.protocol`

Das Kommunikationsprotokoll, das für die Kommunikation mit diesem Gerät verwendet wird. Jedes Gerät in einem Pool muss dasselbe Protokoll verwenden.

Derzeit sind die einzigen unterstützten Werte `ssh` und `uart` für physische Geräte, `docker` für Docker-Container und `no-op` für Geräte, die keine direkte Verbindung mit dem IDT-Hostcomputer haben, aber ein Ressourcengerät als physische Middleware für die Kommunikation mit dem Host-Computer benötigen.

Für No-Op-Geräte konfigurieren Sie die Ressourcen-Geräte-ID in. `devices.pairedResource` Sie müssen diese ID auch in der `resource.json` Datei angeben. Bei dem gekoppelten Gerät muss es sich um ein Gerät handeln, das physisch mit dem zu testenden Gerät gekoppelt ist. Nachdem IDT das gekoppelte Ressourcengerät identifiziert und eine Verbindung zu diesem hergestellt hat, stellt IDT gemäß den in der Datei beschriebenen Funktionen keine Verbindung zu anderen Ressourcengeräten her. `test.json`

`connectivity.ip`

Die IP-Adresse des zu testenden Geräts.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

`connectivity.port`

Optional. Die Portnummer, die für SSH-Verbindungen verwendet werden soll.

Der Standardwert ist 22.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

`connectivity.publicKeyPath`

Optional. Der vollständige Pfad zum öffentlichen Schlüssel, der zur Authentifizierung von Verbindungen mit dem zu testenden Gerät verwendet wird. Wenn Sie das

angebenpublicKeyPath, validiert IDT den öffentlichen Schlüssel des Geräts, wenn es eine SSH-Verbindung zu dem zu testenden Gerät herstellt. Wenn dieser Wert nicht angegeben ist, stellt IDT eine SSH-Verbindung her, validiert aber nicht den öffentlichen Schlüssel des Geräts.

Wir empfehlen dringend, dass Sie den Pfad zum öffentlichen Schlüssel angeben und eine sichere Methode verwenden, um diesen öffentlichen Schlüssel abzurufen. Für standardmäßige SSH-Clients auf Befehlszeilenbasis wird der öffentliche Schlüssel in der Datei bereitgestellt. known_hosts Wenn Sie eine separate öffentliche Schlüsseldatei angeben, muss diese Datei dasselbe Format wie die known_hosts Datei verwenden, d. h.. `ip-address key-type public-key`

connectivity.auth

Authentifizierungsinformationen für die Verbindung.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

connectivity.auth.method

Die Authentifizierungsmethode, die für den Zugriff über ein bestimmtes Verbindungsprotokoll auf ein Gerät verwendet wird.

Unterstützte Werte sind:

- `pki`
- `password`

connectivity.auth.credentials

Die für die Authentifizierung verwendeten Anmeldeinformationen.

connectivity.auth.credentials.password

Das Passwort für die Anmeldung am Gerät wird überprüft.

Dieser Wert gilt nur, wenn `connectivity.auth.method` auf `password` festgelegt ist.

connectivity.auth.credentials.privKeyPath

Der vollständige Pfad zum privaten Schlüssel, der für die Anmeldung bei dem zu testenden Gerät verwendet wird.

Dieser Wert gilt nur, wenn `connectivity.auth.method` auf `pki` festgelegt ist.

`connectivity.auth.credentials.user`

Der Benutzername für die Anmeldung bei dem zu testenden Gerät.

`connectivity.serialPort`

Optional. Die serielle Schnittstelle, an die das Gerät angeschlossen ist.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `uart` festgelegt ist.

`connectivity.containerId`

Die Container-ID oder der Name des getesteten Docker-Containers.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `docker` festgelegt ist.

`connectivity.containerUser`

Optional. Der Name des Benutzers gegenüber dem Benutzer innerhalb des Containers. Der Standardwert ist der im Dockerfile angegebene Benutzer.

Der Standardwert ist 22.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `docker` festgelegt ist.

> **Note**
>
> Um zu überprüfen, ob Testläufer die falsche Geräteverbindung für einen Test konfigurieren, können Sie Daten `pool.Devices[0].Connectivity.Protocol` aus dem Zustandsmaschinenkontext abrufen und mit dem erwarteten Wert in einem `Choice` Status vergleichen. Wenn ein falsches Protokoll verwendet wird, drucken Sie eine Nachricht mit dem `LogMessage` Status und wechseln Sie zum `Fail` Status. Alternativ können Sie den Fehlerbehandlungscode verwenden, um einen Testfehler für falsche Gerätetypen zu melden.

(Optional) Konfigurieren Sie userdata.json

Die `userdata.json` Datei enthält alle zusätzlichen Informationen, die für eine Testsuite erforderlich sind, aber nicht in der Datei angegeben sind. `device.json` Das Format dieser Datei hängt von der `userdata_scheme.json`Datei ab, die in der Testsuite definiert ist. Wenn Sie ein Testautor sind,

stellen Sie sicher, dass Sie diese Informationen Benutzern zur Verfügung stellen, die die von Ihnen geschriebenen Testsuiten ausführen.

(Optional) Konfigurieren Sie resource.json

Die `resource.json` Datei enthält Informationen über alle Geräte, die als Ressourcengeräte verwendet werden. Ressourcengeräte sind Geräte, die zum Testen bestimmter Funktionen eines zu testenden Geräts erforderlich sind. Um beispielsweise die Bluetooth-Fähigkeit eines Geräts zu testen, können Sie ein Ressourcengerät verwenden, um zu testen, ob Ihr Gerät erfolgreich eine Verbindung zu dem Gerät herstellen kann. Ressourcengeräte sind optional, und Sie können so viele Ressourcengeräte benötigen, wie Sie benötigen. Als Testautor verwenden Sie die Datei test.json, um die Funktionen der Ressourcengeräte zu definieren, die für einen Test erforderlich sind. Testläufer verwenden dann die `resource.json` Datei, um einen Pool von Ressourcengeräten bereitzustellen, die über die erforderlichen Funktionen verfügen. Stellen Sie sicher, dass Sie diese Informationen Benutzern zur Verfügung stellen, die die von Ihnen geschriebenen Testsuiten ausführen werden.

Testläufer können diese Informationen mithilfe der folgenden `resource.json` Vorlagendatei bereitstellen, die sich im *<device-tester-extract-location>*`/configs/` Ordner befindet.

```
[
    {
        "id": "<pool-id>",
        "features": [
            {
                "name": "<feature-name>",
                "version": "<feature-value>",
                "jobSlots": <job-slots>
            }
        ],
        "devices": [
            {
                "id": "<device-id>",
                "connectivity": {
                    "protocol": "ssh | uart | docker",
                    // ssh
                    "ip": "<ip-address>",
                    "port": <port-number>,
                    "publicKeyPath": "<public-key-path>",
                    "auth": {
                        "method": "pki | password",
                        "credentials": {
                            "user": "<user-name>",
```

```
                        // pki
                        "privKeyPath": "/path/to/private/key",

                        // password
                        "password": "<password>",
                    }
                },

                // uart
                "serialPort": "<serial-port>",

                // docker
                "containerId": "<container-id>",
                "containerUser": "<container-user-name>",
            }
        }
    ]
}
]
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`id`

Eine benutzerdefinierte alphanumerische ID, die eine Sammlung von Geräten, den sogenannten Gerätepool, eindeutig identifiziert. Geräte, die zu einem Pool gehören, müssen über identische Hardware verfügen. Bei der Ausführung einer Reihe von Tests werden Geräte im Pool verwendet, um die Workload zu parallelisieren. Mehrere Geräte werden verwendet, um verschiedene Tests auszuführen.

`features`

Optional. Ein Array, das die Funktionen enthält, die das Gerät unterstützt. Die in diesem Feld erforderlichen Informationen sind in den test.json-Dateien in der Testsuite definiert und bestimmen, welche Tests ausgeführt werden und wie diese Tests ausgeführt werden. Wenn die Testsuite keine Funktionen benötigt, ist dieses Feld nicht erforderlich.

`features.name`

Der Name der Funktion.

`features.version`

Die Feature-Version.

`features.jobSlots`

Einstellung, die angibt, wie viele Tests das Gerät gleichzeitig verwenden können. Der Standardwert ist 1.

`devices`

Eine Reihe von Geräten im Pool, die getestet werden sollen. Es ist mindestens ein Gerät erforderlich.

`devices.id`

Eine benutzerdefinierte eindeutige Kennung für das zu testende Gerät.

`connectivity.protocol`

Das Kommunikationsprotokoll, das für die Kommunikation mit diesem Gerät verwendet wird. Jedes Gerät in einem Pool muss dasselbe Protokoll verwenden.

Derzeit werden nur Werte `uart` für physische Geräte `ssh` und `docker` für Docker-Container unterstützt.

`connectivity.ip`

Die IP-Adresse des zu testenden Geräts.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

`connectivity.port`

Optional. Die Portnummer, die für SSH-Verbindungen verwendet werden soll.

Der Standardwert ist 22.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

`connectivity.publicKeyPath`

Optional. Der vollständige Pfad zum öffentlichen Schlüssel, der zur Authentifizierung von Verbindungen mit dem zu testenden Gerät verwendet wird. Wenn Sie das angebenpublicKeyPath, validiert IDT den öffentlichen Schlüssel des Geräts, wenn es eine SSH-Verbindung zu dem zu testenden Gerät herstellt. Wenn dieser Wert nicht angegeben ist, stellt IDT eine SSH-Verbindung her, validiert aber nicht den öffentlichen Schlüssel des Geräts.

Wir empfehlen dringend, dass Sie den Pfad zum öffentlichen Schlüssel angeben und eine sichere Methode verwenden, um diesen öffentlichen Schlüssel abzurufen. Für standardmäßige

SSH-Clients auf Befehlszeilenbasis wird der öffentliche Schlüssel in der Datei bereitgestellt. `known_hosts` Wenn Sie eine separate öffentliche Schlüsseldatei angeben, muss diese Datei dasselbe Format wie die `known_hosts` Datei verwenden, d. h.. `ip-address key-type public-key`

`connectivity.auth`

Authentifizierungsinformationen für die Verbindung.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

`connectivity.auth.method`

Die Authentifizierungsmethode, die für den Zugriff über ein bestimmtes Verbindungsprotokoll auf ein Gerät verwendet wird.

Unterstützte Werte sind:

- `pki`
- `password`

`connectivity.auth.credentials`

Die für die Authentifizierung verwendeten Anmeldeinformationen.

`connectivity.auth.credentials.password`

Das Passwort für die Anmeldung am Gerät wird überprüft.

Dieser Wert gilt nur, wenn `connectivity.auth.method` auf `password` festgelegt ist.

`connectivity.auth.credentials.privKeyPath`

Der vollständige Pfad zum privaten Schlüssel, der für die Anmeldung bei dem zu testenden Gerät verwendet wird.

Dieser Wert gilt nur, wenn `connectivity.auth.method` auf `pki` festgelegt ist.

`connectivity.auth.credentials.user`

Der Benutzername für die Anmeldung bei dem zu testenden Gerät.

`connectivity.serialPort`

Optional. Die serielle Schnittstelle, an die das Gerät angeschlossen ist.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `uart` festgelegt ist.

`connectivity.containerId`

Die Container-ID oder der Name des getesteten Docker-Containers.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `docker` festgelegt ist.

`connectivity.containerUser`

Optional. Der Name des Benutzers gegenüber dem Benutzer innerhalb des Containers. Der Standardwert ist der im Dockerfile angegebene Benutzer.

Der Standardwert ist 22.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `docker` festgelegt ist.

(Optional) Konfigurieren Sie config.json

Die `config.json` Datei enthält Konfigurationsinformationen für IDT. In der Regel müssen Testläufer diese Datei nicht ändern, es sei denn, sie müssen ihre AWS Benutzeranmeldeinformationen für IDT und optional eine AWS Region angeben. Wenn AWS Anmeldeinformationen mit den erforderlichen Berechtigungen bereitgestellt werden, werden Nutzungsmetriken AWS IoT Device Tester erfasst und an diese gesendet. AWS Dies ist eine Opt-in-Funktion, die zur Verbesserung der IDT-Funktionalität verwendet wird. Weitere Informationen finden Sie unter Senden Sie IDT-Nutzungsmetriken.

Testläufer können ihre AWS Anmeldeinformationen auf eine der folgenden Arten konfigurieren:

- Anmeldeinformationsdatei

  IDT verwendet die gleiche Anmeldeinformationsdatei wie das AWS CLI. Weitere Informationen finden Sie unter Konfigurations- und Anmeldeinformationsdateien.

  Der Speicherort der Datei mit den Anmeldeinformationen variiert je nach verwendetem Betriebssystem:

  - macOS Linux: `~/.aws/credentials`
  - Windows: `C:\Users\`*`UserName`*`\.aws\credentials`
- Umgebungsvariablen

  Umgebungsvariablen sind Variablen, die vom Betriebssystem gepflegt und von Systembefehlen verwendet werden. Variablen, die während einer SSH-Sitzung definiert wurden, sind nach dem Schließen dieser Sitzung nicht verfügbar. IDT kann die `AWS_SECRET_ACCESS_KEY`

Umgebungsvariablen `AWS_ACCESS_KEY_ID` und zum Speichern von Anmeldeinformationen verwenden AWS

Um diese Variablen auf Linux, macOS oder Unix festzulegen, verwenden Sie export:

```
export AWS_ACCESS_KEY_ID=<your_access_key_id>
export AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

In Windows können Sie die Variablen mit set festlegen:

```
set AWS_ACCESS_KEY_ID=<your_access_key_id>
set AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

Um AWS Anmeldeinformationen für IDT zu konfigurieren, bearbeiten Testläufer den `auth` Abschnitt in der `config.json` Datei, die sich im *<device-tester-extract-location>*`/configs/` Ordner befindet.

```
{
    "log": {
        "location": "logs"
    },
    "configFiles": {
        "root": "configs",
        "device": "configs/device.json"
    },
    "testPath": "tests",
    "reportPath": "results",
    "awsRegion": "<region>",
    "auth": {
        "method": "file | environment",
        "credentials": {
            "profile": "<profile-name>"
        }
    }
}
]
```

Nachfolgend sind alle Pflichtfelder beschrieben:

> **Note**
>
> Alle Pfade in dieser Datei sind relativ zu definiert. *<device-tester-extract-location>*

log.location

Der Pfad zum Protokollordner in der*<device-tester-extract-location>*.

configFiles.root

Der Pfad zu dem Ordner, der die Konfigurationsdateien enthält.

configFiles.device

Der Pfad zur `device.json` Datei.

testPath

Der Pfad zu dem Ordner, der Testsuiten enthält.

reportPath

Der Pfad zu dem Ordner, der Testergebnisse enthält, nachdem IDT eine Testsuite ausgeführt hat.

awsRegion

Optional. Die AWS Region, die die Testsuiten verwenden werden. Wenn nicht festgelegt, verwenden Testsuiten die in jeder Testsuite angegebene Standardregion.

auth.method

Die Methode, die IDT zum Abrufen von AWS Anmeldeinformationen verwendet. Unterstützte Werte sind `file` das Abrufen von Anmeldeinformationen aus einer Anmeldeinformationsdatei und `environment` das Abrufen von Anmeldeinformationen mithilfe von Umgebungsvariablen.

auth.credentials.profile

Das zu verwendende Anmeldeinformationsprofil aus der Anmeldeinformationsdatei. Diese Eigenschaft gilt nur, wenn `auth.method` auf `file` festgelegt ist.

Debuggen Sie benutzerdefinierte Testsuiten und führen Sie sie aus

Nachdem die erforderliche Konfiguration festgelegt wurde, kann IDT Ihre Testsuite ausführen. Die Laufzeit der vollständigen Testsuite hängt von der Hardware und der Zusammensetzung

der Testsuite ab. Als Referenz: Es dauert ungefähr 30 Minuten, bis die vollständige FreeRTOS-Qualifizierungstestsuite auf einem Raspberry Pi 3B abgeschlossen ist.

Während Sie Ihre Testsuite schreiben, können Sie IDT verwenden, um die Testsuite im Debug-Modus auszuführen, um Ihren Code vor der Ausführung zu überprüfen oder ihn Testläufern zur Verfügung zu stellen.

Führen Sie IDT im Debug-Modus aus

Da Testsuiten auf IDT angewiesen sind, um mit Geräten zu interagieren, den Kontext bereitzustellen und Ergebnisse zu erhalten, können Sie Ihre Testsuiten nicht einfach in einer IDE debuggen, ohne dass IDT interagiert. Zu diesem Zweck stellt die IDT-CLI den `debug-test-suite` Befehl bereit, mit dem Sie IDT im Debug-Modus ausführen können. Führen Sie den folgenden Befehl aus, um die verfügbaren Optionen für anzuzeigen: `debug-test-suite`

```
devicetester_[linux | mac | win_x86-64] debug-test-suite -h
```

Wenn Sie IDT im Debug-Modus ausführen, startet IDT weder die Testsuite noch den Test-Orchestrator. Stattdessen interagiert IDT mit Ihrer IDE, um auf Anfragen von der in der IDE ausgeführten Testsuite zu antworten und die Protokolle auf der Konsole auszudrucken. IDT hat kein Timeout und wartet mit dem Beenden, bis es manuell unterbrochen wird. Im Debug-Modus führt IDT auch den Test-Orchestrator nicht aus und generiert keine Berichtsdateien. Um Ihre Testsuite zu debuggen, müssen Sie Ihre IDE verwenden, um einige Informationen bereitzustellen, die IDT normalerweise aus den Konfigurationsdateien bezieht. Stellen Sie sicher, dass Sie die folgenden Informationen angeben:

- Umgebungsvariablen und Argumente für jeden Test. IDT liest diese Informationen nicht von `test.json` oder`suite.json`.
- Argumente zur Auswahl von Ressourcengeräten. IDT liest diese Informationen nicht aus`test.json`.

Gehen Sie wie folgt vor, um Ihre Testsuiten zu debuggen:

1. Erstellen Sie die Einstellungskonfigurationsdateien, die für die Ausführung der Testsuite erforderlich sind. Wenn Ihre Testsuite beispielsweise die`device.json`, und erfordert`resource.json`, stellen Sie sicher`user data.json`, dass Sie sie alle nach Bedarf konfigurieren.

2. Führen Sie den folgenden Befehl aus, um IDT in den Debug-Modus zu versetzen und alle Geräte auszuwählen, die für die Ausführung des Tests erforderlich sind.

   ```
   devicetester_[linux | mac | win_x86-64] debug-test-suite [options]
   ```

   Nachdem Sie diesen Befehl ausgeführt haben, wartet IDT auf Anfragen von der Testsuite und beantwortet sie dann. IDT generiert auch die Umgebungsvariablen, die für den Fallprozess für das IDT Client SDK erforderlich sind.

3. Verwenden Sie in Ihrer IDE die `debug` Konfiguration `run` oder, um Folgendes zu tun:

   a. Legen Sie die Werte der von IDT generierten Umgebungsvariablen fest.

   b. Legen Sie den Wert aller Umgebungsvariablen oder Argumente fest, die Sie in Ihrer `test.json` AND-Datei angegeben haben. `suite.json`

   c. Legen Sie nach Bedarf Haltepunkte fest.

4. Führen Sie die Testsuite in Ihrer IDE aus.

   Sie können die Testsuite so oft wie nötig debuggen und erneut ausführen. IDT tritt im Debug-Modus nicht auf.

5. Nachdem Sie das Debuggen abgeschlossen haben, unterbrechen Sie IDT, um den Debug-Modus zu verlassen.

IDT-CLI-Befehle zum Ausführen von Tests

Im folgenden Abschnitt werden die IDT-CLI-Befehle beschrieben:

IDT v4.0.0

`help`

Listet Informationen über den angegebenen Befehl auf.

`list-groups`

Listet die Gruppen in der jeweiligen Testsuite auf.

`list-suites`

Listet die verfügbaren Testsuites auf.

`list-supported-products`

Listet die unterstützten Produkte für Ihre Version von IDT auf, in diesem Fall FreeRTOS-Versionen und FreeRTOS Qualification Test Suite-Versionen, die für die aktuelle IDT-Version verfügbar sind.

`list-test-cases`

Listet die Testfälle in einer bestimmten Testgruppe auf. Die folgende Option wird unterstützt:

- `group-id`. Die Testgruppe, nach der gesucht werden soll. Diese Option ist erforderlich und muss eine einzelne Gruppe angeben.

`run-suite`

Führt eine Reihe von Tests in einem Pool von Geräten aus. Im Folgenden sind einige häufig verwendete Optionen aufgeführt:

- `suite-id`. Die auszuführende Version der Testsuite. Wenn nicht angegeben, verwendet IDT die neueste Version im `tests`-Ordner.
- `group-id`. Die auszuführenden Testgruppen als kommagetrennte Liste. Bei fehlender Angabe führt IDT alle Testgruppen in der Testsuite aus.
- `test-id`. Die auszuführenden Testfälle als kommagetrennte Liste. Wenn angegeben, muss `group-id` eine einzelne Gruppe angeben.
- `pool-id`. Der zu testende Gerätepool. Testläufer müssen einen Pool angeben, wenn in Ihrer `device.json` Datei mehrere Gerätepools definiert sind.
- `timeout-multiplier`. Konfiguriert IDT so, dass das in der `test.json` Datei angegebene Timeout für die Testausführung für einen Test mit einem benutzerdefinierten Multiplikator geändert wird.
- `stop-on-first-failure`. Konfiguriert IDT so, dass die Ausführung beim ersten Fehler gestoppt wird. Diese Option sollte mit `group-id` verwendet werden, um die angegebenen Testgruppen zu debuggen.
- `userdata`. Legt die Datei fest, die Benutzerdateninformationen enthält, die zum Ausführen der Testsuite erforderlich sind. Dies ist nur erforderlich, wenn `userdataRequired` es in der `suite.json` Datei für die Testsuite auf true gesetzt ist.

Weitere Informationen zu `run-suite`-Optionen erhalten Sie mit der `help`-Option:

```
devicetester_[linux | mac | win_x86-64] run-suite -h
```

`debug-test-suite`

Führen Sie die Testsuite im Debug-Modus aus. Weitere Informationen finden Sie unter Führen Sie IDT im Debug-Modus aus.

Überprüfen Sie die IDT-Testergebnisse und -Protokolle

In diesem Abschnitt wird das Format beschrieben, in dem IDT Konsolenprotokolle und Testberichte generiert.

Nachrichtenformat der Konsole

AWS IoT Device Tester verwendet ein Standardformat für das Drucken von Nachrichten an die Konsole, wenn eine Testsuite gestartet wird. Der folgende Auszug zeigt ein Beispiel für eine von IDT generierte Konsolennachricht.

```
[INFO] [2000-01-02 03:04:05]: Using suite: MyTestSuite_1.0.0
 executionId=9a52f362-1227-11eb-86c9-8c8590419f30
```

Die meisten Konsolennachrichten bestehen aus den folgenden Feldern:

`time`

Ein vollständiger ISO 8601-Zeitstempel für das protokollierte Ereignis.

`level`

Die Nachrichtenebene für das protokollierte Ereignis. In der Regel ist die Ebene der protokollierten Nachricht eine von `info`, `warn`, oder`error`. IDT gibt eine `fatal` `panic` OR-Nachricht aus, wenn es auf ein erwartetes Ereignis trifft, das dazu führt, dass es vorzeitig beendet wird.

`msg`

Die protokollierte Nachricht.

`executionId`

Eine eindeutige ID-Zeichenfolge für den aktuellen IDT-Prozess. Diese ID wird verwendet, um zwischen einzelnen IDT-Läufen zu unterscheiden.

Von einer Testsuite generierte Konsolennachrichten enthalten zusätzliche Informationen über das zu testende Gerät und die Testsuite, Testgruppe und Testfälle, die IDT ausführt. Der folgende Auszug zeigt ein Beispiel für eine Konsolennachricht, die aus einer Testsuite generiert wurde.

```
[INFO] [2000-01-02 03:04:05]: Hello world! suiteId=MyTestSuitegroupId=myTestGroup
 testCaseId=myTestCase deviceId=my-
deviceexecutionId=9a52f362-1227-11eb-86c9-8c8590419f30
```

Der für die Testsuite spezifische Teil der Konsolennachricht enthält die folgenden Felder:

`suiteId`

Der Name der Testsuite, die gerade läuft.

`groupId`

Die ID der Testgruppe, die gerade läuft.

`testCaseId`

Die ID des aktuell laufenden Testfalls.

`deviceId`

Eine ID des zu testenden Geräts, das der aktuelle Testfall verwendet.

Die Testzusammenfassung enthält Informationen über die Testsuite, die Testergebnisse für jede Gruppe, die ausgeführt wurde, und die Speicherorte der generierten Protokolle und Berichtsdateien. Das folgende Beispiel zeigt eine Meldung mit einer Testzusammenfassung.

```
========== Test Summary ==========
Execution Time:     5m00s
Tests Completed:    4
Tests Passed:       3
Tests Failed:       1
Tests Skipped:      0
---------------------------------
Test Groups:
    GroupA:         PASSED
    GroupB:         FAILED
---------------------------------
Failed Tests:
```

```
    Group Name: GroupB
        Test Name: TestB1
            Reason: Something bad happened
--------------------------------
Path to AWS IoT Device Tester Report: /path/to/awsiotdevicetester_report.xml
Path to Test Execution Logs: /path/to/logs
Path to Aggregated JUnit Report: /path/to/MyTestSuite_Report.xml
```

AWS IoT Device Tester Berichtsschema

`awsiotdevicetester_report.xml`list ein unterschriebener Bericht, der die folgenden Informationen enthält:

- Die IDT-Version.
- Die Version der Testsuite.
- Die Berichtssignatur und der Schlüssel, die zum Signieren des Berichts verwendet wurden.
- Die Geräte-SKU und der in der `device.json` Datei angegebene Gerätepoolname.
- Die Produktversion und die getesteten Gerätefunktionen.
- Die aggregierte Zusammenfassung der Testergebnisse. Diese Informationen entsprechen denen, die in der *`suite-name`*`_report.xml` Datei enthalten sind.

```
<apnreport>
    <awsiotdevicetesterversion>idt-version</awsiotdevicetesterversion>
    <testsuiteversion>test-suite-version</testsuiteversion>
    <signature>signature</signature>
    <keyname>keyname</keyname>
    <session>
        <testsession>execution-id</testsession>
        <starttime>start-time</starttime>
        <endtime>end-time</endtime>
    </session>
    <awsproduct>
        <name>product-name</name>
        <version>product-version</version>
        <features>
            <feature name="<feature-name>" value="supported | not-supported | <feature-
value>" type="optional | required"/>
        </features>
    </awsproduct>
```

```
    <device>
        <sku>device-sku</sku>
        <name>device-name</name>
        <features>
            <feature name="<feature-name>" value="<feature-value>"/>
        </features>
        <executionMethod>ssh | uart | docker</executionMethod>
    </device>
    <devenvironment>
        <os name="<os-name>"/>
    </devenvironment>
    <report>
        <suite-name-report-contents>
    </report>
</apnreport>
```

Die Datei `awsiotdevicetester_report.xml` enthält ein `<awsproduct>`-Tag mit Informationen zum getesteten Produkt und den Produktfunktionen, die nach einer Reihe von Tests validiert wurden.

Im **`<awsproduct>`** Tag verwendete Attribute

`name`

Der Name des getesteten Produkts.

`version`

Die Version des getesteten Produkts.

`features`

Die validierten Funktionen Als markierte Funktionen `required` sind erforderlich, damit die Testsuite das Gerät validieren kann. Der folgende Ausschnitt zeigt, wie diese Informationen in der Datei `awsiotdevicetester_report.xml` angezeigt werden.

```
<feature name="ssh" value="supported" type="required"></feature>
```

Als markierte Funktionen `optional` sind für die Validierung nicht erforderlich. Die folgenden Codeausschnitte zeigen optionale Funktionen.

```
<feature name="hsi" value="supported" type="optional"></feature>
<feature name="mqtt" value="not-supported" type="optional"></feature>
```

Berichtsschema der Testsuite

Der *suite-name*_Result.xml Bericht ist im JUnit XML-Format. Sie können ihn in Continuous Integration and Deployment-Plattformen wie Jenkins, Bamboo usw. integrieren. Der Bericht enthält eine Zusammenfassung der Testergebnisse.

```
<testsuites name="<suite-name> results" time="<run-duration>" tests="<number-of-test>"
 failures="<number-of-tests>" skipped="<number-of-tests>" errors="<number-of-tests>"
 disabled="0">
    <testsuite name="<test-group-id>" package="" tests="<number-of-tests>"
 failures="<number-of-tests>" skipped="<number-of-tests>" errors="<number-of-tests>"
 disabled="0">
        <!--success-->
        <testcase classname="<classname>" name="<name>" time="<run-duration>"/>
        <!--failure-->
        <testcase classname="<classname>" name="<name>" time="<run-duration>">
            <failure type="<failure-type>">
                reason
            </failure>
        </testcase>
        <!--skipped-->
        <testcase classname="<classname>" name="<name>" time="<run-duration>">
            <skipped>
                reason
            </skipped>
        </testcase>
        <!--error-->
        <testcase classname="<classname>" name="<name>" time="<run-duration>">
            <error>
                reason
            </error>
        </testcase>
    </testsuite>
</testsuites>
```

Im Berichtsabschnitt sowohl im awsiotdevicetester_report.xml als auch im Abschnitt *suite-name*_report.xml werden die durchgeführten Tests und die Ergebnisse aufgeführt.

Im ersten XML-Tag <testsuites> ist die Zusammenfassung der Testausführung enthalten. Beispiel:

```
<testsuites name="MyTestSuite results" time="2299" tests="28" failures="0" errors="0"
 disabled="0">
```

Im **<testsuites>** Tag verwendete Attribute

name

Name der Testsuite

time

Die Zeit in Sekunden, die zum Ausführen der Testsuite benötigt wurde.

tests

Die Anzahl der ausgeführten Tests.

failures

Die Anzahl der ausgeführten Tests, die den Test nicht bestanden haben

errors

Die Anzahl der Tests, die IDT nicht ausführen konnte.

disabled

Dieses Attribut wird nicht verwendet und kann ignoriert werden.

Falls bei Tests Fehler auftreten, können Sie den fehlgeschlagenen Test identifizieren, indem Sie die XML-Tags von `<testsuites>` überprüfen. Die XML-Tags von `<testsuite>` im `<testsuites>`-Tag zeigen die Ergebniszusammenfassung eines Tests für eine Testgruppe. Beispiel:

```
<testsuite name="combination" package="" tests="1" failures="0" time="161" disabled="0"
 errors="0" skipped="0">
```

Das Format ähnelt dem `<testsuites>`-Tag, weist aber das Attribut `skipped` auf, das nicht verwendet wird und ignoriert werden kann. Innerhalb der einzelnen `<testsuite>`-XML-Tags befinden sich `<testcase>`-Tags für alle ausgeführten Tests einer Testgruppe. Beispiel:

```
<testcase classname="Security Test" name="IP Change Tests" attempts="1"></testcase>
```

Im **<testcase>** Tag verwendete Attribute

`name`

Der Name des Tests

`attempts`

Gibt an, wie oft IDT den Testfall ausgeführt hat.

Wenn ein Testfall fehlschlägt oder ein Fehler auftritt, werden `<failure>`- oder `<error>`-Tags hinzugefügt, um das `<testcase>`-Tag mit Informationen für die Fehlerbehebung zu versehen. Beispiel:

```
<testcase classname="mcu.Full_MQTT" name="MQTT_TestCase" attempts="1">
 <failure type="Failure">Reason for the test failure</failure>
 <error>Reason for the test execution error</error>
</testcase>
```

Senden Sie IDT-Nutzungsmetriken

Wenn Sie AWS Anmeldeinformationen mit den erforderlichen Berechtigungen angeben, AWS IoT Device Tester sammelt und sendet Nutzungsdaten an. AWS Dies ist eine Opt-in-Funktion, die zur Verbesserung der IDT-Funktionalität verwendet wird. IDT sammelt Informationen wie die folgenden:

- Die AWS Konto-ID, die zum Ausführen von IDT verwendet wurde
- Die IDT-CLI-Befehle, die zum Ausführen von Tests verwendet werden
- Die Testsuite, die ausgeführt wird
- Die Testsuiten im *`<device-tester-extract-location>`* Ordner
- Die Anzahl der im Gerätepool konfigurierten Geräte
- Testfallnamen und Laufzeiten
- Informationen zu den Testergebnissen, z. B. ob Tests bestanden oder fehlgeschlagen sind, ob Fehler aufgetreten sind oder ob sie übersprungen wurden
- Getestete Produktmerkmale
- Verhalten beim Beenden von IDT, z. B. unerwartete oder vorzeitige Austritte

Alle Informationen, die IDT sendet, werden auch in einer `metrics.log` Datei im Ordner protokolliert. *`<device-tester-extract-location>`*`/results/`*`<execution-id>`*`/` In der Protokolldatei

können Sie die Informationen einsehen, die während eines Testlaufs gesammelt wurden. Diese Datei wird nur generiert, wenn Sie Nutzungsmetriken erfassen möchten.

Um die Erfassung von Messwerten zu deaktivieren, müssen Sie keine zusätzlichen Maßnahmen ergreifen. Speichern Sie Ihre AWS Anmeldeinformationen einfach nicht, und wenn Sie AWS Anmeldeinformationen gespeichert haben, konfigurieren Sie die `config.json` Datei nicht für den Zugriff darauf.

Melden Sie sich an für ein AWS-Konto

Wenn Sie noch keine haben AWS-Konto, führen Sie die folgenden Schritte aus, um eine zu erstellen.

Um sich für eine anzumelden AWS-Konto

1. Öffnen Sie https://portal.aws.amazon.com/billing/die Anmeldung.
2. Folgen Sie den Online-Anweisungen.

   Während der Anmeldung erhalten Sie einen Telefonanruf oder eine Textnachricht und müssen einen Verifizierungscode über die Telefontasten eingeben.

   Wenn Sie sich für eine anmelden AWS-Konto, Root-Benutzer des AWS-Kontoswird eine erstellt. Der Root-Benutzer hat Zugriff auf alle AWS-Services und Ressourcen des Kontos. Als bewährte Sicherheitsmethode weisen Sie einem Benutzer Administratorzugriff zu und verwenden Sie nur den Root-Benutzer, um Aufgaben auszuführen, die Root-Benutzerzugriff erfordern.

AWS sendet Ihnen nach Abschluss des Anmeldevorgangs eine Bestätigungs-E-Mail. Du kannst jederzeit deine aktuellen Kontoaktivitäten einsehen und dein Konto verwalten, indem du zu https://aws.amazon.com/gehst und Mein Konto auswählst.

Erstellen eines Benutzers mit Administratorzugriff

Nachdem Sie sich für einen angemeldet haben AWS-Konto, sichern Sie Ihren Root-Benutzer des AWS-Kontos AWS IAM Identity Center, aktivieren und erstellen Sie einen Administratorbenutzer, sodass Sie den Root-Benutzer nicht für alltägliche Aufgaben verwenden.

Sichern Sie Ihre Root-Benutzer des AWS-Kontos

1. Melden Sie sich AWS-Managementkonsoleals Kontoinhaber an, indem Sie Root-Benutzer auswählen und Ihre AWS-Konto E-Mail-Adresse eingeben. Geben Sie auf der nächsten Seite Ihr Passwort ein.

Hilfe bei der Anmeldung mit dem Root-Benutzer finden Sie unter Anmelden als Root-Benutzer im AWS-Anmeldung -Benutzerhandbuch zu.

2. Aktivieren Sie die Multi-Faktor-Authentifizierung (MFA) für den Root-Benutzer.

   Anweisungen finden Sie unter Aktivieren eines virtuellen MFA-Geräts für Ihren AWS-Konto Root-Benutzer (Konsole) im IAM-Benutzerhandbuch.

Erstellen eines Benutzers mit Administratorzugriff

1. Aktivieren Sie das IAM Identity Center.

   Anweisungen finden Sie unter Aktivieren AWS IAM Identity Center im AWS IAM Identity Center -Benutzerhandbuch.

2. Gewähren Sie einem Administratorbenutzer im IAM Identity Center Benutzerzugriff.

   Ein Tutorial zur Verwendung von IAM-Identity-Center-Verzeichnis als Identitätsquelle finden Sie IAM-Identity-Center-Verzeichnis im Benutzerhandbuch unter Benutzerzugriff mit der Standardeinstellung konfigurieren.AWS IAM Identity Center

Anmelden als Administratorbenutzer

- Um sich mit Ihrem IAM-Identity-Center-Benutzer anzumelden, verwenden Sie die Anmelde-URL, die an Ihre E-Mail-Adresse gesendet wurde, als Sie den IAM-Identity-Center-Benutzer erstellt haben.

  Hilfe bei der Anmeldung mit einem IAM Identity Center-Benutzer finden Sie im AWS-Anmeldung Benutzerhandbuch unter Anmeldung beim AWS Access-Portal.

Weiteren Benutzern Zugriff zuweisen

1. Erstellen Sie im IAM-Identity-Center einen Berechtigungssatz, der den bewährten Vorgehensweisen für die Anwendung von geringsten Berechtigungen folgt.

   Anweisungen hierzu finden Sie unter Berechtigungssatz erstellen im AWS IAM Identity Center -Benutzerhandbuch.

2. Weisen Sie Benutzer einer Gruppe zu und weisen Sie der Gruppe dann Single Sign-On-Zugriff zu.

Eine genaue Anleitung finden Sie unter Gruppen hinzufügen im AWS IAM Identity Center - Benutzerhandbuch.

Um Zugriff zu gewähren, fügen Sie Ihren Benutzern, Gruppen oder Rollen Berechtigungen hinzu:

- Benutzer und Gruppen in AWS IAM Identity Center:

  Erstellen Sie einen Berechtigungssatz. Befolgen Sie die Anweisungen unter Erstellen eines Berechtigungssatzes im AWS IAM Identity Center -Benutzerhandbuch.
- Benutzer, die in IAM über einen Identitätsanbieter verwaltet werden:

  Erstellen Sie eine Rolle für den Identitätsverbund. Befolgen Sie die Anleitung unter Eine Rolle für einen externen Identitätsanbieter (Verbund) erstellen im IAM-Benutzerhandbuch.
- IAM-Benutzer:
  - Erstellen Sie eine Rolle, die Ihr Benutzer annehmen kann. Befolgen Sie die Anleitung unter Eine Rolle für einen IAM-Benutzer erstellen im IAM-Benutzerhandbuch.
  - (Nicht empfohlen) Weisen Sie einem Benutzer eine Richtlinie direkt zu oder fügen Sie einen Benutzer zu einer Benutzergruppe hinzu. Befolgen Sie die Anweisungen unter Hinzufügen von Berechtigungen zu einem Benutzer (Konsole) im IAM-Benutzerhandbuch.

Geben Sie AWS Anmeldeinformationen für IDT ein

Gehen Sie wie folgt vor, damit IDT auf Ihre AWS Anmeldeinformationen zugreifen und Messwerte an AWS senden kann:

1. Speichern Sie die AWS Anmeldeinformationen für Ihren IAM-Benutzer als Umgebungsvariablen oder in einer Anmeldeinformationsdatei:

   a. Führen Sie den folgenden Befehl aus, um Umgebungsvariablen zu verwenden:

   ```
   AWS_ACCESS_KEY_ID=access-key
   AWS_SECRET_ACCESS_KEY=secret-access-key
   ```

   b. Um die Datei mit den Anmeldeinformationen zu verwenden, fügen Sie der Datei die folgenden Informationen hinzu `.aws/credentials file:`

   ```
   [profile-name]
   ```

```
aws_access_key_id=access-key
aws_secret_access_key=secret-access-key
```

2. Konfigurieren Sie den `auth` Abschnitt der `config.json` Datei. Weitere Informationen finden Sie unter (Optional) Konfigurieren Sie config.json.

Pflegen Sie die Versionen der Testsuiten

IDT for FreeRTOS organisiert Testressourcen in Testsuiten und Testgruppen:

- Eine Testsuite besteht aus einer Reihe von Testgruppen, die verwendet werden, um zu überprüfen, ob ein Gerät mit bestimmten Versionen von FreeRTOS funktioniert.
- Eine Testgruppe ist der Satz einzelner Tests, die sich auf ein bestimmtes Funktion beziehen, z. B. BLE und MQTT Messaging.

Ab IDT v3.0.0 sind Testsuites beginnend mit 1.0.0 im Format `major.minor.patch` versioniert. Wenn Sie IDT herunterladen, enthält das Paket die neueste Test-Suite-Version.

Wenn Sie IDT in der Befehlszeilenschnittstelle starten, prüft IDT, ob eine neuere Test-Suite-Version verfügbar ist. Wenn ja, werden Sie aufgefordert, auf die neue Version zu aktualisieren. Sie können wählen, ob Sie aktualisieren oder mit Ihren aktuellen Tests fortfahren möchten.

> **Note**
>
> IDT unterstützt zur Qualifizierung die drei neuesten Test-Suite-Versionen. Weitere Informationen finden Sie unter Machen Sie sich mit den Support-Richtlinien vertraut für AWS IoT Device Tester.

Mit dem Befehl `upgrade-test-suite` können Sie Testsuiten herunterladen. Oder Sie können den optionalen Parameter verwenden, `-upgrade-test-suite` *`flag`* wenn Sie IDT starten. Dabei *`flag`* kann 'y' angegeben werden, um immer die neueste Version herunterzuladen, oder 'n', um die vorhandene Version zu verwenden.

Sie können den `list-supported-versions` Befehl auch ausführen, um die FreeRTOS- und Testsuite-Versionen aufzulisten, die von der aktuellen Version von IDT unterstützt werden.

Neue Tests können neue IDT-Konfigurationseinstellungen einführen. Wenn die Einstellungen optional sind, benachrichtigt IDT Sie und setzt die Ausführung der Tests fort. Wenn die Einstellungen

erforderlich sind, benachrichtigt IDT Sie und beendet die Ausführung. Nachdem Sie die Einstellungen konfiguriert haben, können Sie die Ausführung der Tests fortsetzen.

Beheben von Fehlern in der

Jede Ausführung einer Testsuite verfügt über eine eindeutige Ausführungs-ID, die verwendet wird, um im Verzeichnis `results` einen Ordner mit dem Namen `results/`*`execution-id`* zu erstellen. Die einzelnen Testgruppenprotokolle befinden sich im Verzeichnis `results/`*`execution-id`*`/logs`. Verwenden Sie die IDT for FreeRTOS-Konsolenausgabe, um die Ausführungs-ID, Testfall-ID und Testgruppen-ID des fehlgeschlagenen Testfalls zu ermitteln, und öffnen Sie dann die Protokolldatei für diesen Testfall mit dem Namen. `results/`*`execution-id`*`/logs/`*`test_group_id`*`__`*`test_case_id`*`.log` Die Informationen in dieser Datei beinhalten Folgendes:

- Vollständige Build- und Flash-Befehlsausgabe.
- Testausführungsausgabe.
- Ausführlichere IDT für die FreeRTOS-Konsolenausgabe.

Wir empfehlen folgenden Workflow für die Fehlersuche:

1. Wenn der Fehler "*`user/role`*ist nicht berechtigt, auf diese Ressource zuzugreifen“ angezeigt wird, stellen Sie sicher, dass Sie die Berechtigungen wie unter konfiguriert haben. Erstellen und konfigurieren Sie ein Konto AWS
2. Lesen Sie die Konsolenausgabe, um Informationen zu finden, wie z. B. Ausführungs-UUID und aktuell ausgeführte Aufgaben.
3. Suchen Sie in der Datei `FRQ_Report.xml` nach Fehleranweisungen für jeden Test. Dieses Verzeichnis enthält Ausführungsprotokolle für jede Testgruppe.
4. Schauen Sie in den Protokolldateien unter nach`/results/`*`execution-id`*`/logs`.
5. Untersuchen Sie einen der folgenden Problembereiche:
   - Gerätekonfiguration, wie z. B. JSON-Konfigurationsdateien im Ordner `/configs/`.
   - Geräteschnittstelle Überprüfen Sie die Protokolle, um festzustellen, welche Schnittstelle fehlschlägt.
   - Geräte-Tools Stellen Sie sicher, dass die Toolchains zum Erstellen und Flashen des Gerätes korrekt installiert und konfiguriert sind.

- Stellen Sie für FRQ 1.x.x sicher, dass eine saubere, geklonte Version des FreeRTOS-Quellcodes verfügbar ist. FreeRTOS-Versionen sind entsprechend der FreeRTOS-Version gekennzeichnet. Verwenden Sie die folgenden Befehle, um eine bestimmte Version des Codes zu klonen:

```
git clone --branch version-number https://github.com/aws/amazon-freertos.git
cd amazon-freertos
git submodule update --checkout --init --recursive
```

Beheben Sie Fehler bei der Gerätekonfiguration

Wenn Sie IDT für FreeRTOS verwenden, müssen Sie die richtigen Konfigurationsdateien einrichten, bevor Sie die Binärdatei ausführen. Wenn Sie Parsing- und Konfigurationsfehler erhalten, sollten Sie als Erstes eine für Ihre Umgebung geeignete Konfigurationsvorlage suchen und anwenden. Diese Vorlagen befinden sich im Verzeichnis *IDT_ROOT*`/configs`.

Wenn weiterhin Probleme auftreten, beachten Sie den folgenden Debugging-Vorgang.

Wo suche ich?

Lesen Sie zunächst die Konsolenausgabe, um Informationen zu finden, z. B. die Ausführungs-UUID, die in dieser Dokumentation als `execution-id` bezeichnet wird.

Suchen Sie als Nächstes in der Datei `FRQ_Report.xml` im Verzeichnis `/results/`*execution-id*. Diese Datei enthält alle ausgeführten Testfälle und Fehlerausschnitte zu jedem Fehler. Suchen Sie zum Abrufen aller Ausführungsprotokolle für jeden Testfall jeweils nach der Datei `/results/`*execution-id*`/logs/`*test_group_id*`__`*test_case_id*`.log`.

IDT-Fehlercodes

In der folgenden Tabelle werden die von IDT für FreeRTOS generierten Fehlercodes erklärt:

Fehlercode	Name des Fehlercodes	Mögliche Ursache	Fehlerbehebung
201	InvalidInputError	Felder in `device.json` , `config.js`	Stellen Sie sicher, dass die Pflichtfelder

Fehlercode	Name des Fehlercodes	Mögliche Ursache	Fehlerbehebung
		on oder `userdata.json` fehlen entweder oder haben ein falsches Format.	in den aufgelisteten Dateien nicht fehlen und dass sie das erforderliche Format aufweisen. Weitere Informationen finden Sie unter Erster Test Ihres Mikrocontroller-Boards.

Fehlercode	Name des Fehlercodes	Mögliche Ursache	Fehlerbehebung
202	ValidationError	Felder in `device.json` , `config.json` oder `userdata.json` enthalten ungültige Werte.	Überprüfen Sie die Fehlermeldung auf der rechten Seite des Fehlercodes im Bericht: • Ungültige AWS Region — Geben Sie eine gültige AWS Region in Ihrer Datei an. `config.json` Weitere Informationen zu AWS Regionen finden Sie unter Regionen und Endpunkte. • Ungültige AWS Anmeldeinformationen — Legen Sie gültige AWS Anmeldeinformationen auf Ihrem Testcomputer fest (über Umgebungsvariablen oder die Anmeldeinformationsdatei). Überprüfen Sie, ob das Authentifizierungsfeld korrekt konfiguriert ist. Weitere Informationen finden Sie

Fehlercode	Name des Fehlercodes	Mögliche Ursache	Fehlerbehebung
			unter Erstellen und konfigurieren Sie ein Konto AWS.
203	CopySourceCodeError	Der FreeRTOS-Quellcode konnte nicht in das angegebene Verzeichnis kopiert werden.	Überprüfen Sie Folgendes: • Prüfen Sie, ob ein gültiger `sourcePath` in der Datei `userdata.json` angegeben ist. • Löschen Sie den `build` Ordner im FreeRTOS-Quellcodeverzeichnis, falls er existiert. Weitere Informationen finden Sie unter Konfiguration von Build-, Flash- und Testeinstellungen. • Windows hat eine Zeichenbeschränkung für Dateipfadnamen. Ein langer Dateipfadname verursacht einen Fehler.

Fehlercode	Name des Fehlercodes	Mögliche Ursache	Fehlerbehebung
204	BuildSourceError	Der FreeRTOS-Quellcode konnte nicht kompiliert werden.	Überprüfen Sie Folgendes: • Überprüfen Sie, ob die Informationen unter `buildTool` in der Datei `userdata.json` korrekt sind. • Wenn Sie `cmake` als Build-Tool verwenden, stellen Sie sicher, dass `{{enableTests}}` im `buildTool` -Befehl angegeben ist. Weitere Informationen finden Sie unter Konfiguration von Build-, Flash- und Testeinstellungen. • Wenn Sie IDT for FreeRTOS in einen Dateipfad auf Ihrem System extrahiert haben, der beispielsweise Leerzeichen enthält, benötigen Sie möglicherweise zusätzliche Anführung

Fehlercode	Name des Fehlercodes	Mögliche Ursache	Fehlerbehebung
			szeichen in Ihren Build-Befehlen`C:\Users\My Name\Desktop\` , um sicherzustellen, dass die Pfade ordnungsgemäß analysiert werden. Das Gleiche wird möglicherweise für Ihre Flash-Befehle benötigt.
205	FlashOrRunTestError	IDT FreeRTOS kann FreeRTOS nicht auf Ihrem DUT flashen oder ausführen.	Überprüfen Sie, ob die Informationen unter `flashTool` in der Datei `userdata.json` korrekt sind. Weitere Informationen finden Sie unter Konfiguration von Build-, Flash- und Testeinstellungen.

Fehlercode	Name des Fehlercodes	Mögliche Ursache	Fehlerbehebung
206	StartEchoServerError	IDT FreeRTOS kann den Echo-Server für die WiFi oder Secure Socket-Tests nicht starten.	Stellen Sie sicher, dass die unter `echoServerConfiguration` in Ihrer `userdata.json` -Datei konfigurierten Ports nicht von der Firewall oder den Netzwerkeinstellungen verwendet oder blockiert werden.

Fehler beim Parsen der Konfigurationsdatei debuggen

Es kann vorkommen, dass ein Tippfehler in einer JSON-Konfiguration zu Parsing-Fehlern führt. In den meisten Fällen ist die Ursache des Problems eine fehlende Klammer oder ein fehlendes Komma oder Anführungszeichen in Ihrer JSON-Datei. IDT for FreeRTOS führt eine JSON-Validierung durch und druckt Debugging-Informationen. Gedruckt werden die Zeile, in der der Fehler aufgetreten ist, sowie Zeilennummer und Spaltennummer des Syntaxfehlers. Diese Informationen sollten ausreichen, um Ihnen bei der Behebung des Fehlers zu helfen. Wenn Sie jedoch immer noch Probleme haben, den Fehler zu finden, können Sie die Validierung manuell in Ihrer IDE, einem Texteditor wie Atom oder Sublime oder über ein Online-Tool wie durchführen. JSONLint

Debug-Testergebnisse, Analysieren von Fehlern

Beim Ausführen einer Testgruppe wie FullTransportInterfaceTLS FreeRTOS-Libraries-Integration-Tests, Full PKCS11 _Core, Full _Onboard_ECC, Full PKCS11 _Onboard_RSA, Full _ PKCS11 _ECC, Full PKCS11 _ PreProvisioned _RSA oder IDT for PreProvisioned FreeRTOS analysiert die Testergebnisse des Testgeräts mit der seriellen OTACoreVerbindung. PKCS11 Manchmal können zusätzliche serielle Ausgänge am Gerät die Auswertung der Testergebnisse beeinträchtigen.

Im oben genannten Fall werden seltsame Gründe für das Scheitern eines Testfalls ausgegeben, z. B. Zeichenfolgen, die von nicht verwandten Geräteausgängen stammen. Die IDT for FreeRTOS-

Testfallprotokolldatei (die alle seriellen Ausgaben enthält, die IDT for FreeRTOS während des Tests erhalten hat) kann Folgendes enthalten:

```
<unrelated device output>
TEST(Full_PKCS11_Capabilities, PKCS11_Capabilities)<unrelated device output>
<unrelated device output>
 PASS
```

Im obigen Beispiel verhindert die unabhängige Geräteausgabe, dass IDT for FreeRTOS das Testergebnis PASS erkennt.

Überprüfen Sie Folgendes, um ein optimales Testen sicherzustellen.

- Stellen Sie sicher, dass die auf dem Gerät verwendeten Protokollierungsmakros threadsicher sind. Weitere Informationen finden Sie unter Implementieren der Bibliotheksprotokollierungsmakros.
- Stellen Sie sicher, dass die serielle Verbindung während der Tests nur minimal ausgibt. Andere Geräteausgänge können ein Problem darstellen, selbst wenn Ihre Protokollierungsmakros ordnungsgemäß threadsicher sind, da die Testergebnisse während des Tests in separaten Aufrufen ausgegeben werden.

Ein IDT for FreeRTOS-Testfallprotokoll würde idealerweise eine unterbrechungsfreie Ausgabe der Testergebnisse wie folgt anzeigen:

```
---------STARTING TESTS---------
TEST(Full_OTA_PAL, otaPal_CloseFile_ValidSignature) PASS
TEST(Full_OTA_PAL, otaPal_CloseFile_InvalidSignatureBlockWritten) PASS
-----------------------
2 Tests 0 Failures 0 Ignored
```

Fehler bei der Integritätsprüfung debuggen

Wenn Sie die FRQ 1.x.x-Version von FreeRTOS verwenden, gelten die folgenden Integritätsprüfungen.

Wenn Sie die kostenlose RTOSIntegrity Testgruppe ausführen und Fehler auftreten, stellen Sie zunächst sicher, dass Sie keine der Verzeichnisdateien geändert haben. *`freertos`* Falls dies

nicht der Fall ist und weiterhin Probleme auftreten, stellen Sie sicher, dass Sie den richtigen Zweig verwenden. Wenn Sie den `list-supported-products` Befehl IDT ausführen, können Sie herausfinden, welchen markierten Zweig des *`freertos`* Repositorys Sie verwenden sollten.

Wenn Sie den richtigen markierten Zweig des `freertos` Repositorys geklont haben und immer noch Probleme haben, stellen Sie sicher, dass Sie den Befehl auch ausgeführt haben. `submodule update` Der Klon-Workflow für das `freertos` Repo sieht wie folgt aus.

```
git clone --branch version-number https://github.com/aws/amazon-freertos.git
cd amazon-freertos
git submodule update --checkout —init —recursive
```

Die Liste der Dateien, nach denen die Integritätsprüfung sucht, befindet sich in der `checksums.json` Datei in Ihrem *`freertos`* Verzeichnis. Um einen FreeRTOS-Port ohne Änderungen an den Dateien und der Ordnerstruktur zu qualifizieren, stellen Sie sicher, dass keine der in den Abschnitten " und `exhaustive` '`minimal`' der `checksums.json` Datei aufgelisteten Dateien geändert wurde. Um die Ausführung mit einem konfigurierten SDK durchzuführen, stellen Sie sicher, dass keine der Dateien im Abschnitt `minimal` " geändert wurde.

Wenn Sie IDT mit einem SDK ausführen und einige Dateien in Ihrem *`freertos`* Verzeichnis geändert haben, stellen Sie sicher, dass Sie Ihr SDK in Ihrer `userdata` Datei korrekt konfigurieren. Andernfalls überprüft der Integritätsprüfer alle Dateien im *`freertos`* Verzeichnis.

Fehler bei FullWiFi Testgruppen debuggen

Wenn Sie FRQ 1.x.x verwenden und Fehler in der FullWiFi Testgruppe auftreten und der "`AFQP_WiFiConnectMultipleAP`" Test fehlschlägt, kann dies daran liegen, dass sich beide Access Points nicht im selben Subnetz befinden wie der Host-Computer, auf dem IDT ausgeführt wird. Stellen Sie sicher, dass sich beide Access Points im selben Subnetz befinden wie der Hostcomputer, auf dem IDT ausgeführt wird.

Fehler „Erforderlicher Parameter fehlt“ beheben

Da IDT for FreeRTOS um neue Funktionen erweitert wird, können Änderungen an den Konfigurationsdateien vorgenommen werden. Bei Verwendung einer alten Konfigurationsdatei kann Ihre Konfiguration beschädigt werden. In diesem Fall listet die Datei *`test_group_id__test_case_id`*`.log` im Verzeichnis `results/`*`execution-id`*`/logs` explizit alle fehlenden Parameter auf. IDT for FreeRTOS validiert Ihre JSON-Konfigurationsdateischemas, um sicherzustellen, dass die neueste unterstützte Version verwendet wurde.

Debuggen Sie die Fehler „Test konnte nicht gestartet werden“

Möglicherweise finden Sie Hinweise auf Fehler beim Teststart. Da es mehrere mögliche Ursachen gibt, überprüfen Sie die folgenden Bereiche auf ihre Richtigkeit:

- Stellen Sie sicher, dass der in Ihrem Ausführungsbefehl enthaltene Poolname tatsächlich vorhanden ist. Auf diesen wird direkt über Ihre `device.json`-Datei verwiesen.
- Stellen Sie sicher, dass die Geräte in Ihrem Pool über die richtigen Konfigurationsparameter verfügen.

Debug-Fehler „Beginn der Testergebnisse konnte nicht gefunden werden“

Möglicherweise treten Fehler auf, wenn IDT versucht, die vom zu testenden Gerät ausgegebenen Ergebnisse zu analysieren. Es gibt mehrere mögliche Ursachen. Überprüfen Sie daher die folgenden Bereiche auf Richtigkeit:

- Stellen Sie sicher, dass das zu testende Gerät eine stabile Verbindung zu Ihrem Host-Computer hat. Sie können in der Protokolldatei nach einem Test suchen, der diese Fehler anzeigt, um zu sehen, was IDT empfängt.
- Wenn Sie FRQ 1.x.x verwenden und das zu testende Gerät über ein langsames Netzwerk oder eine andere Schnittstelle verbunden ist oder Sie das Flag „---------STARTING TESTS---------“ in einem FreeRTOS-Testgruppenprotokoll nicht zusammen mit anderen FreeRTOS-Testgruppenausgaben sehen, können Sie versuchen, den Wert von in Ihrer Benutzerdatenkonfiguration zu erhöhen. `testStartDelayms` Weitere Informationen finden Sie unter Konfiguration von Build-, Flash- und Testeinstellungen.

Debuggen Sie die Fehler „Testfehler: erwartete __ Ergebnisse, aber es wurden ___ gesehen“

Möglicherweise werden beim Testen Fehler angezeigt, die auf einen Testfehler hinweisen. Der Test erwartet eine bestimmte Anzahl von Ergebnissen und wird während des Tests nicht angezeigt. Einige FreeRTOS-Tests werden ausgeführt, bevor IDT die Ausgabe des Geräts sieht. Wenn Sie diesen Fehler sehen, können Sie versuchen, den Wert von *`testStartDelayms`* in Ihrer Benutzerdatenkonfiguration zu erhöhen. Weitere Informationen finden Sie unter Konfiguration von Build-, Flash- und Testeinstellungen.

Debuggen Sie den Fehler „_____ wurde aufgrund von Einschränkungen nicht ausgewählt“ ConditionalTests

Das bedeutet, dass Sie einen Test auf einem Gerätepool ausführen, der mit dem Test nicht kompatibel ist. Dies kann bei den OTA E2E-Tests passieren. Beispielsweise haben Sie beim Ausführen der **`OTADataplaneMQTT`** Testgruppe und in Ihrer **`device.json`** Konfigurationsdatei OTA als Nein oder **`OTADataPlaneProtocol`** als HTTP ausgewählt. Die Testgruppe, die Sie ausführen möchten, muss Ihrer `device.json` Funktionsauswahl entsprechen.

Debuggen Sie einen IDT-Timeout während der Überwachung der Geräteausgabe

IDT kann aus verschiedenen Gründen zu einem Timeout führen. Wenn während der Phase der Überwachung der Geräteausgänge eines Tests ein Timeout auftritt und Sie die Ergebnisse im IDT-Testfallprotokoll sehen können, bedeutet dies, dass die Ergebnisse von IDT falsch analysiert wurden. Ein Grund könnten die verschachtelten Protokollnachrichten in der Mitte der Testergebnisse sein. Wenn dies der Fall ist, finden Sie im FreeRTOS Porting Guide weitere Informationen darüber, wie die UNITY-Logs eingerichtet werden sollten.

Ein weiterer Grund für ein Timeout bei der Überwachung der Geräteausgabe könnte sein, dass ein Gerät nach einem einzigen Fehler im TLS-Testfall neu gestartet wird. Das Gerät führt dann das Flash-Image aus und verursacht eine Endlosschleife, die in den Protokollen zu sehen ist. Stellen Sie in diesem Fall sicher, dass Ihr Gerät nach einem fehlgeschlagenen Test nicht neu gestartet wird.

Debuggen Sie den Fehler „Nicht autorisiert, auf die Ressource zuzugreifen“

Möglicherweise wird der Fehler "*`user/role`*ist nicht berechtigt, auf diese Ressource zuzugreifen“ in der Terminalausgabe oder in der `test_manager.log` Datei unter angezeigt/`results/`*`execution-id`*`/logs`. Um dieses Problem zu beheben, fügen Sie die `AWS IoTDeviceTesterForFreeRTOSFullAccess`-verwaltete Richtlinie an Ihren Testbenutzer an. Weitere Informationen finden Sie unter Erstellen und konfigurieren Sie ein Konto AWS.

Fehler bei Netzwerktests debuggen

Für netzwerkbasierte Tests startet IDT einen Echo-Server, der sich an einen nicht reservierten Port auf dem Hostcomputer bindet. Wenn Sie aufgrund von Timeouts oder nicht verfügbaren Verbindungen bei den Tests WiFi oder Secure Sockets auf Fehler stoßen, stellen Sie sicher, dass Ihr Netzwerk so konfiguriert ist, dass Datenverkehr zu konfigurierten Ports im Bereich 1024 bis 49151 zugelassen wird.

Der Secure Sockets-Test verwendet standardmäßig die Ports 33333 und 33334. Die WiFi Tests verwenden standardmäßig Port 33335. Wenn diese drei Ports von einer Firewall oder einem Netzwerk verwendet oder blockiert werden, können Sie verschiedene Ports in userdata.json zum Testen verwenden. Weitere Informationen finden Sie unter Konfiguration von Build-, Flash- und Testeinstellungen. Sie können mit den folgenden Befehle überprüfen, ob ein bestimmter Port verwendet wird:

- Windows: `netsh advfirewall firewall show rule name=all | grep port`
- Linux: `sudo netstat -pan | grep port`
- macOS: `netstat -nat | grep port`

Fehler beim OTA-Update aufgrund der Payload derselben Version

Wenn OTA-Testfälle fehlschlagen, weil sich dieselbe Version auf dem Gerät befindet, nachdem ein OTA ausgeführt wurde, kann dies daran liegen, dass Ihr Build-System (z. B. cmake) die Änderungen von IDT am FreeRTOS-Quellcode nicht bemerkt und keine aktualisierte Binärdatei erstellt. Dies führt dazu, dass OTA mit der gleichen Binärdatei durchgeführt wird, die sich bereits auf dem Gerät befindet, weshalb der Test fehlschlägt. Um OTA-Updatefehler zu beheben, stellen Sie zunächst sicher, dass Sie die neueste unterstützte Version Ihres Build-Systems verwenden.

OTA-Testfehler im **`PresignedUrlExpired`**-Testfall

Eine Voraussetzung für diesen Test ist, dass die OTA-Update-Zeit mehr als 60 Sekunden betragen sollte, da der Test andernfalls fehlschlagen würde. In diesem Fall ist im Protokoll die folgende Fehlermeldung zu finden: „Test takes less than 60 seconds (url expired time) to finish. (Abschluss des Tests dauert weniger als 60 Sekunden (URL-Ablaufzeit). Please reach out to us (Bitte kontaktieren Sie uns).“

Debuggen Sie Geräteschnittstellen- und Portfehler

Dieser Abschnitt enthält Informationen über die Geräteschnittstellen, die IDT zur Verbindung mit Ihren Geräten verwendet.

Unterstützte Plattformen

IDT unterstützt Linux, MacOS und Windows. Auf allen drei Plattformen werden unterschiedliche Benennungen für serielle Geräte verwendet, die mit ihnen verbunden sind:

- Linux: `/dev/tty*`

- macOS: `/dev/tty.*` oder `/dev/cu.*`
- Windows: COM*

So überprüfen Sie den Geräteport:

- Öffnen Sie unter Linux/macOS ein Terminal und führen Sie `ls /dev/tty*` aus.
- Öffnen Sie unter macOS ein Terminal und führen Sie `ls /dev/tty.*` oder `ls /dev/cu.*` aus.
- Öffnen Sie in Windows Sie den Geräte-Manager und erweitern Sie die Gruppe mit den seriellen Geräten.

Um zu überprüfen, welches Gerät mit einem Port verbunden ist:

- Stellen Sie unter Linux sicher, dass das Paket `udev` installiert ist, und führen Sie dann `udevadm info -name=`*`PORT`* aus. Dieses Dienstprogramm gibt die Gerätetreiberinformationen aus, mit deren Hilfe Sie überprüfen können, ob Sie den richtigen Port verwenden.
- Öffnen Sie für macOS Launchpad und suchen Sie nach **`System Information`**.
- Öffnen Sie in Windows Sie den Geräte-Manager und erweitern Sie die Gruppe mit den seriellen Geräten.

Geräteschnittstellen

Jedes eingebettete Gerät ist anders. Dies bedeutet, dass sie einen oder mehrere serielle Ports haben können. Es ist üblich, dass Geräte über zwei Ports verfügen, wenn sie mit einer Maschine verbunden sind:

- Ein Datenport zum Flashen des Geräts.
- Ein Leseport zum Lesen der Ausgabe.

  Sie müssen den richtigen Leseport in Ihrer `device.json`-Datei festlegen. Andernfalls kann das Lesen der Ausgabe vom Gerät fehlschlagen.

  Vergewissern Sie sich bei mehreren Ports, dass Sie den Leseport des Geräts in Ihrer `device.json`-Datei verwenden. Wenn Sie beispielsweise ein WRover Espressif-Gerät anschließen und die beiden ihm zugewiesenen Anschlüsse als `/dev/ttyUSB0` und `/dev/ttyUSB1` `/dev/ttyUSB1` in Ihrer `device.json` Datei verwenden.

Folgen Sie derselben Logik in Windows.

Lesen von Gerätedaten

IDT for FreeRTOS verwendet individuelle Gerätebau- und Flash-Tools, um die Portkonfiguration zu spezifizieren. Wenn Sie Ihr Gerät testen und keine Ausgabe erhalten, versuchen Sie es mit den folgenden Standardeinstellungen:

- Baudrate: 115200
- Datenbits: 8
- Parität: Keine
- Stop-Bits: 1
- Flusssteuerung: Keine

Diese Einstellungen werden von IDT für FreeRTOS verwaltet. Sie müssen sie nicht festlegen. Sie können jedoch dieselbe Methode verwenden, um die Geräteausgabe manuell zu lesen. Unter Linux oder macOS ist dies mit dem Befehl `screen` möglich. Unter Windows können Sie ein Programm wie verwenden. TeraTerm

```
Screen: screen /dev/cu.usbserial 115200
```

```
TeraTerm: Use the above-provided settings to set the fields explicitly in
the GUI.
```

Probleme mit der Entwicklungs-Toolchain

In diesem Abschnitt werden Probleme beschrieben, die mit Ihrer Tool-Chain auftreten können.

Code Composer Studio auf Ubuntu

In den neueren Versionen von Ubuntu (17.10 und 18.04) ist eine Version des `glibc`-Pakets enthalten, die nicht mit den Versionen von Code Composer Studio 7.x kompatibel ist. Wir empfehlen, Code Composer Studio Version 8.2 oder höher zu installieren.

Folgende Anzeichen der Inkompatibilität könnten auftreten:

- FreeRTOS kann nicht erstellt oder auf Ihr Gerät geflasht werden.
- Das Code Composer Studio-Installationsprogramm stürzt ab.

- In der Konsole wird keine Protokollausgabe während des Build- oder Flash-Prozesses angezeigt.
- Der Build-Befehl versucht, im GUI-Modus zu starten, auch wenn er im Headless-Modus aufgerufen wird.

Protokollierung

IDT for FreeRTOS-Protokolle werden an einem einzigen Ort gespeichert. Im IDT-Stammverzeichnis sind diese Dateien unter `results/`*`execution-id`*`/` verfügbar:

- `FRQ_Report.xml`
- `awsiotdevicetester_report.xml`
- `logs/`*`test_group_id`*`__`*`test_case_id`*`.log`

`FRQ_Report.xml` und `logs/`*`test_group_id`*`__`*`test_case_id`*`.log` sind die wichtigsten Protokolle, die Sie untersuchen sollten. `FRQ_Report.xml` enthält Informationen dazu, für welche Testfälle Fehler mit einer bestimmten Fehlermeldung aufgetreten sind. Anschließend können Sie `logs/`*`test_group_id`*`__`*`test_case_id`*`.log` verwenden, um das Problem genauer zu analysieren und so einen besseren Kontext zu erhalten.

Konsolenfehler

Wenn AWS IoT Device Tester es ausgeführt wird, werden Fehler mit kurzen Meldungen an die Konsole gemeldet. Weitere Informationen zum Fehler finden Sie unter `results/`*`execution-id`*`/logs/`*`test_group_id`*`__`*`test_case_id`*`.log`.

Protokollfehler

Jede Ausführung der Testsuite verfügt über eine eindeutige Ausführungs-ID, die zum Erstellen eines Ordners mit dem Namen `results/`*`execution-id`* verwendet wird. Einzelne Testfallprotokolle befinden sich im Verzeichnis `results/`*`execution-id`*`/logs`. Verwenden Sie die Ausgabe der IDT for FreeRTOS-Konsole, um die Ausführungs-ID, die Testfall-ID und die Testgruppen-ID des fehlgeschlagenen Testfalls zu ermitteln. Verwenden Sie dann diese Informationen, um die Protokolldatei für diesen Testfall mit dem Namen zu finden und zu öffnen. `results/`*`execution-id`*`/logs/`*`test_group_id`*`__`*`test_case_id`*`.log` Die Informationen in dieser Datei umfassen die vollständige Build- und Flash-Befehlsausgabe, die Ausgabe der Testausführung und eine AWS IoT Device Tester ausführlichere Konsolenausgabe.

Probleme mit dem S3-Bucket

Wenn Sie CTRL+C während der Ausführung von IDT die Taste drücken, startet IDT einen Bereinigungsprozess. Ein Teil dieser Bereinigung besteht darin, Amazon S3 S3-Ressourcen zu entfernen, die im Rahmen der IDT-Tests erstellt wurden. Wenn die Bereinigung nicht abgeschlossen werden kann, tritt möglicherweise ein Problem auf, bei dem zu viele Amazon S3 S3-Buckets erstellt wurden. Das bedeutet, dass die Tests fehlschlagen werden, wenn Sie IDT das nächste Mal ausführen.

Wenn Sie CTRL+C die Taste drücken, um IDT zu beenden, müssen Sie den Bereinigungsvorgang beenden lassen, um dieses Problem zu vermeiden. Sie können auch die manuell erstellten Amazon S3 S3-Buckets aus Ihrem Konto löschen.

Beheben Sie Timeout-Fehler

Wenn beim Ausführen einer Testsuite Timeout-Fehler auftreten, erhöhen Sie das Timeout, indem Sie einen Timeout-Multiplikatorfaktor angeben. Dieser Faktor wird auf den Standardwert für die Zeitüberschreitung angewendet. Jeder Wert für dieses Kennzeichen muss größer als oder gleich 1,0 sein. Um den Timeout-Multiplikator zu verwenden, verwenden Sie beim Ausführen der Testsuite das Flag `--timeout-multiplier`.

Example

IDT v3.0.0 and later

```
./devicetester_linux run-suite --suite-id FRQ_1.0.1 --pool-id DevicePool1 --timeout-
multiplier 2.5
```

IDT v1.7.0 and earlier

```
./devicetester_linux run-suite --suite-id FRQ_1 --pool-id DevicePool1 --timeout-
multiplier 2.5
```

Mobilfunkfunktion und Gebühren AWS

Wenn die `Cellular` Funktion `Yes` in Ihrer `device.JSON` Datei auf eingestellt ist, FullSecureSockets werden T.micro EC2-Instances für die Ausführung von Tests verwendet, was zu zusätzlichen Kosten für Ihr Konto führen kann. AWS Weitere Informationen dazu finden Sie unter Amazon EC2 – Preise.

Richtlinie zur Generierung von Qualifikationsberichten

Qualifizierungsberichte werden nur von AWS IoT Device Tester (IDT-) Versionen generiert, die FreeRTOS-Versionen unterstützen, die in den letzten zwei Jahren veröffentlicht wurden. Wenn Sie Fragen zu den Support-Richtlinien haben, wenden Sie sich bitte an. AWS Support

Verstehen Sie die AWS verwaltete Richtlinie für AWS IoT Device Tester

Eine AWS verwaltete Richtlinie ist eine eigenständige Richtlinie, die von erstellt und verwaltet wird AWS. AWS Verwaltete Richtlinien sind so konzipiert, dass sie Berechtigungen für viele gängige Anwendungsfälle bereitstellen, sodass Sie damit beginnen können, Benutzern, Gruppen und Rollen Berechtigungen zuzuweisen.

Denken Sie daran, dass AWS verwaltete Richtlinien für Ihre speziellen Anwendungsfälle möglicherweise keine Berechtigungen mit den geringsten Rechten gewähren, da sie für alle AWS Kunden verfügbar sind. Wir empfehlen Ihnen, die Berechtigungen weiter zu reduzieren, indem Sie vom Kunden verwaltete Richtlinien definieren, die speziell auf Ihre Anwendungsfälle zugeschnitten sind.

Sie können die in AWS verwalteten Richtlinien definierten Berechtigungen nicht ändern. Wenn die in einer AWS verwalteten Richtlinie definierten Berechtigungen AWS aktualisiert werden, wirkt sich das Update auf alle Prinzidentitäten (Benutzer, Gruppen und Rollen) aus, denen die Richtlinie zugeordnet ist. AWS aktualisiert eine AWS verwaltete Richtlinie höchstwahrscheinlich, wenn eine neue Richtlinie eingeführt AWS-Service wird oder neue API-Operationen für bestehende Dienste verfügbar werden.

Weitere Informationen finden Sie unter Von AWS verwaltete Richtlinien im IAM-Benutzerhandbuch.

Themen

- AWS verwaltete Richtlinie: AWS Io TDevice TesterForFree RTOSFull Access
- Aktualisierungen der AWS verwalteten Richtlinien

AWS verwaltete Richtlinie: AWS Io TDevice TesterForFree RTOSFull Access

Die `AWSIoTDeviceTesterForFreeRTOSFullAccess` verwaltete Richtlinie enthält die folgenden AWS IoT Device Tester Berechtigungen für die Versionsprüfung, auto Aktualisierungsfunktionen und die Erfassung von Metriken.

Berechtigungsdetails

Diese Richtlinie umfasst die folgenden Berechtigungen:

- `iot-device-tester:SupportedVersion`

  Erteilt die AWS IoT Device Tester Berechtigung zum Abrufen der Liste der unterstützten Produkte, Testsuiten und IDT-Versionen.
- `iot-device-tester:LatestIdt`

  AWS IoT Device Tester Erteilt die Erlaubnis, die neueste IDT-Version abzurufen, die zum Herunterladen verfügbar ist.
- `iot-device-tester:CheckVersion`

  AWS IoT Device Tester Erteilt die Erlaubnis, die Versionskompatibilität für IDT, Testsuiten und Produkte zu überprüfen.
- `iot-device-tester:DownloadTestSuite`

  AWS IoT Device Tester Erteilt die Erlaubnis zum Herunterladen von Testsuite-Updates.
- `iot-device-tester:SendMetrics`

  AWS Erteilt die Erlaubnis, Metriken zur AWS IoT Device Tester internen Nutzung zu sammeln.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
```

```
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::*:role/idt-*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "iot.amazonaws.com"
                }
            }
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "iot:DeleteThing",
                "iot:AttachThingPrincipal",
                "iot:DeleteCertificate",
                "iot:GetRegistrationCode",
                "iot:CreatePolicy",
                "iot:UpdateCACertificate",
                "s3:ListBucket",
                "iot:DescribeEndpoint",
                "iot:CreateOTAUpdate",
                "iot:CreateStream",
                "signer:ListSigningJobs",
                "acm:ListCertificates",
                "iot:CreateKeysAndCertificate",
                "iot:UpdateCertificate",
                "iot:CreateCertificateFromCsr",
                "iot:DetachThingPrincipal",
                "iot:RegisterCACertificate",
                "iot:CreateThing",
                "iam:ListRoles",
                "iot:RegisterCertificate",
                "iot:DeleteCACertificate",
                "signer:PutSigningProfile",
                "s3:ListAllMyBuckets",
                "signer:ListSigningPlatforms",
                "iot-device-tester:SendMetrics",
                "iot-device-tester:SupportedVersion",
                "iot-device-tester:LatestIdt",
                "iot-device-tester:CheckVersion",
                "iot-device-tester:DownloadTestSuite"
            ],
            "Resource": "*"
        },
```

```
        {
            "Sid": "VisualEditor2",
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "signer:StartSigningJob",
                "acm:GetCertificate",
                "signer:DescribeSigningJob",
                "s3:CreateBucket",
                "execute-api:Invoke",
                "s3:DeleteBucket",
                "s3:PutBucketVersioning",
                "signer:CancelSigningProfile"
            ],
            "Resource": [
                "arn:aws:execute-api:us-east-1:098862408343:9xpmnvs5h4/prod/POST/
metrics",
                "arn:aws:signer:*:*:/signing-profiles/*",
                "arn:aws:signer:*:*:/signing-jobs/*",
                "arn:aws:iam::*:role/idt-*",
                "arn:aws:acm:*:*:certificate/*",
                "arn:aws:s3:::idt-*",
                "arn:aws:s3:::afr-ota*"
            ]
        },
        {
            "Sid": "VisualEditor3",
            "Effect": "Allow",
            "Action": [
                "iot:DeleteStream",
                "iot:DeleteCertificate",
                "iot:AttachPolicy",
                "iot:DetachPolicy",
                "iot:DeletePolicy",
                "s3:ListBucketVersions",
                "iot:UpdateCertificate",
                "iot:GetOTAUpdate",
                "iot:DeleteOTAUpdate",
                "iot:DescribeJobExecution"
            ],
            "Resource": [
                "arn:aws:s3:::afr-ota*",
                "arn:aws:iot:*:*:thinggroup/idt*",
                "arn:aws:iam::*:role/idt-*"
```

```
            ]
        },
        {
            "Sid": "VisualEditor4",
            "Effect": "Allow",
            "Action": [
                "iot:DeleteCertificate",
                "iot:AttachPolicy",
                "iot:DetachPolicy",
                "s3:DeleteObjectVersion",
                "iot:DeleteOTAUpdate",
                "s3:PutObject",
                "s3:GetObject",
                "iot:DeleteStream",
                "iot:DeletePolicy",
                "s3:DeleteObject",
                "iot:UpdateCertificate",
                "iot:GetOTAUpdate",
                "s3:GetObjectVersion",
                "iot:DescribeJobExecution"
            ],
            "Resource": [
                "arn:aws:s3:::afr-ota*/*",
                "arn:aws:s3:::idt-*/*",
                "arn:aws:iot:*:*:policy/idt*",
                "arn:aws:iam::*:role/idt-*",
                "arn:aws:iot:*:*:otaupdate/idt*",
                "arn:aws:iot:*:*:thing/idt*",
                "arn:aws:iot:*:*:cert/*",
                "arn:aws:iot:*:*:job/*",
                "arn:aws:iot:*:*:stream/*"
            ]
        },
        {
            "Sid": "VisualEditor5",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::afr-ota*/*",
                "arn:aws:s3:::idt-*/*"
            ]
```

```
        },
        {
            "Sid": "VisualEditor6",
            "Effect": "Allow",
            "Action": [
                "iot:CancelJobExecution"
            ],
            "Resource": [
                "arn:aws:iot:*:*:job/*",
                "arn:aws:iot:*:*:thing/idt*"
            ]
        },
        {
            "Sid": "VisualEditor7",
            "Effect": "Allow",
            "Action": [
                "ec2:TerminateInstances"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:instance/*"
            ],
            "Condition": {
                "StringEquals": {
                    "ec2:ResourceTag/Owner": "IoTDeviceTester"
                }
            }
        },
        {
            "Sid": "VisualEditor8",
            "Effect": "Allow",
            "Action": [
                "ec2:AuthorizeSecurityGroupIngress",
                "ec2:DeleteSecurityGroup"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:security-group/*"
            ],
            "Condition": {
                "StringEquals": {
                    "ec2:ResourceTag/Owner": "IoTDeviceTester"
                }
            }
        },
        {
```

```
            "Sid": "VisualEditor9",
            "Effect": "Allow",
            "Action": [
                "ec2:RunInstances"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:instance/*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/Owner": "IoTDeviceTester"
                }
            }
        },
        {
            "Sid": "VisualEditor10",
            "Effect": "Allow",
            "Action": [
                "ec2:RunInstances"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:image/*",
                "arn:aws:ec2:*:*:security-group/*",
                "arn:aws:ec2:*:*:volume/*",
                "arn:aws:ec2:*:*:key-pair/*",
                "arn:aws:ec2:*:*:placement-group/*",
                "arn:aws:ec2:*:*:snapshot/*",
                "arn:aws:ec2:*:*:network-interface/*",
                "arn:aws:ec2:*:*:subnet/*"
            ]
        },
        {
            "Sid": "VisualEditor11",
            "Effect": "Allow",
            "Action": [
                "ec2:CreateSecurityGroup"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:security-group/*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/Owner": "IoTDeviceTester"
                }
```

```
            }
        },
        {
            "Sid": "VisualEditor12",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeInstances",
                "ec2:DescribeSecurityGroups",
                "ssm:DescribeParameters",
                "ssm:GetParameters"
            ],
            "Resource": "*"
        },
        {
            "Sid": "VisualEditor13",
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:security-group/*",
                "arn:aws:ec2:*:*:instance/*"
            ],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "aws:TagKeys": [
                        "Owner"
                    ]
                },
                "StringEquals": {
                    "ec2:CreateAction": [
                        "RunInstances",
                        "CreateSecurityGroup"
                    ]
                }
            }
        }
    ]
}
```

Aktualisierungen der AWS verwalteten Richtlinien

Sie können sich Details zu Aktualisierungen AWS verwalteter Richtlinien anzeigen lassen, die seit AWS IoT Device Tester dem Zeitpunkt gelten, zu dem dieser Dienst mit der Erfassung dieser Änderungen begonnen hat.

Version	Änderungen	Beschreibung	Date
7 (Aktuell)	Die `ec2:CreateTags` Bedingungen wurden neu strukturiert.	Die Verwendung von wird entfernt. `ForAnyValues`	14.06.2023
6	`freertos:ListHardwarePlatforms` Aus der Richtlinie entfernt.	Das Entfernen von Berechtigungen, da diese Aktion seit dem 1. März 2023 veraltet ist.	02.06.2023
5	Es wurden Berechtigungen zum Ausführen von Echo-Server-Tests mit EC2 hinzugefügt.	Dies dient zum Starten und Stoppen einer EC2-Instance in AWS Kundenkonten.	15.12.2020
4	`iot:CancelJobExecution` hinzugefügt.	Mit dieser Erlaubnis werden OTA-Jobs storniert.	17.07.2020
3	Die folgenden Berechtigungen wurden hinzugefügt: • `iot-device-tester:DownloadTestSuite` ,	• `iot-device-tester:DownloadTestSuite` — AWS IoT Device Tester Erteilt die Erlaubnis zum Herunterladen von Testsuite-Updates,	23.03.2020

Version	Änderungen	Beschreibung	Date
	• `iot-device-tester: CheckVersion` , • `iot-device-tester: LatestIdt` , • `iot-device-tester: SupportedVersion` .	• `iot-device-tester: CheckVersion` — AWS IoT Device Tester Erteilt die Erlaubnis, die Versionskompatibilität für IDT, Testsuiten und Produkte zu überprüfen, • `iot-device-tester: LatestIdt` — AWS IoT Device Tester Erteilt die Erlaubnis, die neueste IDT-Version abzurufen, die zum Herunterladen verfügbar ist, • `iot-device-tester: SupportedVersion` — AWS IoT Device Tester Erteilt die Erlaubnis, die Liste der unterstützten Produkte, Testsuiten und IDT-Versionen abzurufen.	

Version	Änderungen	Beschreibung	Date
2	Berechtigungen hinzugefügt`iot-device-tester:SendMetrics` .	AWS Erteilt die Erlaubnis, Metriken zur AWS IoT Device Tester internen Nutzung zu sammeln.	18.2.2020
1	Erste Version		12.2.2020

Machen Sie sich mit den Support-Richtlinien vertraut für AWS IoT Device Tester

> **Important**
>
> Stand Oktober 2022 generiert AWS IoT FreeRTOS Qualification (FRQ) 1.0 keine signierten Qualifikationsberichte. AWS IoT Device Tester Sie können neue AWS IoT FreeRTOS-Geräte nicht für die Aufnahme in den AWS Partnergerätekatalog über das AWS Gerätequalifizierungsprogramm qualifizieren, wenn Sie IDT FRQ 1.0-Versionen verwenden. Sie können FreeRTOS-Geräte zwar nicht mit IDT FRQ 1.0 qualifizieren, aber Sie können Ihre FreeRTOS-Geräte weiterhin mit FRQ 1.0 testen. Wir empfehlen Ihnen, IDT FRQ 2.0 zu verwenden, um FreeRTOS-Geräte zu qualifizieren und im Partnergerätekatalog aufzulisten.AWS

AWS IoT Device Tester for FreeRTOS ist ein Testautomatisierungstool zur Validierung des FreeRTOS-Ports zu Geräten. Darüber hinaus können Sie Ihre FreeRTOS-Geräte qualifizieren und sie im AWS Partner-Gerätekatalog auflisten. Das AWS IoT Device Tester for FreeRTOS unterstützt die Validierung und Qualifizierung von FreeRTOS Long Term Supported (LTS) -Bibliotheken, die unter FreeRTOS/FreeRTOS-LTS verfügbar sind, und der FreeRTOS-Hauptlinie, die GitHub unter FreeRTOS/FreeRTOS verfügbar ist. Wir empfehlen Ihnen, die neuesten Versionen sowohl von FreeRTOS als auch von FreeRTOS zu verwenden, um Ihre AWS IoT Device Tester Geräte zu validieren und zu qualifizieren.

Für FreeRTOS-LTS unterstützt IDT die Validierung und Qualifizierung der FreeRTOS 202210 LTS-Version. Weitere Informationen zu FreeRTOS LTS-Versionen und deren Wartungszeitplan

finden Sie hier. Sobald der Supportzeitraum für diese LTS-Versionen abgelaufen ist, können Sie die Validierung weiterhin fortsetzen. IDT erstellt jedoch keinen Bericht, anhand dessen Sie Ihr Gerät zur Qualifizierung einreichen können.

Für die Mainline-FreeRTOS, die unter FreeRTOS/FreeRTOS verfügbar sind, unterstützen wir die Validierung und Qualifizierung aller Versionen, die in den letzten sechs Monaten veröffentlicht wurden, oder der vorherigen beiden Versionen von FreeRTOS, wenn sie im Abstand von mehr als sechs Monaten veröffentlicht wurden. Informationen zu den derzeit unterstützten Versionen finden Sie hier. Für nicht unterstützte Versionen von FreeRTOS können Sie die Validierung trotzdem fortsetzen, IDT generiert jedoch keinen Bericht, mit dem Sie Ihr Gerät zur Qualifizierung einreichen können.

Die neuesten unterstützten IDT- und FreeRTOS-Versionen finden Unterstützte Versionen von AWS IoT Device Tester Sie unter. Sie können jede der unterstützten Versionen von AWS IoT Device Tester mit der entsprechenden Version von FreeRTOS verwenden, um Ihr Gerät zu testen oder zu qualifizieren. Wenn Sie das weiterhin verwendenNicht unterstützte IDT-Versionen für FreeRTOS, erhalten Sie nicht die neuesten Bugfixes oder Updates.

Bei Fragen zu den Support-Richtlinien wenden Sie sich an AWS den Kundensupport.

Sicherheit in AWS

Cloud-Sicherheit AWS hat höchste Priorität. Als AWS Kunde profitieren Sie von einer Rechenzentrums- und Netzwerkarchitektur, die darauf ausgelegt sind, die Anforderungen der sicherheitssensibelsten Unternehmen zu erfüllen.

Sicherheit ist eine gemeinsame Verantwortung von Ihnen AWS und Ihnen. Das Modell der geteilten Verantwortung beschreibt dies als Sicherheit der Cloud und Sicherheit in der Cloud:

- Sicherheit der Cloud — AWS ist verantwortlich für den Schutz der Infrastruktur, die AWS Dienste in der AWS Cloud ausführt. AWS bietet Ihnen auch Dienste, die Sie sicher nutzen können. Die Wirksamkeit unserer Sicherheitsfunktionen wird regelmäßig von externen Prüfern im Rahmen des AWS -Compliance-Programms getestet und überprüft. Informationen zu den Compliance-Programmen, die für einen AWS Service gelten, finden Sie unter AWS Services im Umfang nach Compliance-Programmen.
- Sicherheit in der Cloud — Ihre Verantwortung richtet sich nach dem AWS Dienst, den Sie nutzen. In Ihre Verantwortung fallen außerdem weitere Faktoren, wie z. B. die Vertraulichkeit der Daten, die Anforderungen Ihrer Organisation sowie geltende Gesetze und Vorschriften.

Diese Dokumentation hilft Ihnen zu verstehen, wie Sie das Modell der gemeinsamen Verantwortung bei der Nutzung anwenden können AWS. In den folgenden Themen erfahren Sie, wie Sie die Konfiguration vornehmen AWS , um Ihre Sicherheits- und Compliance-Ziele zu erreichen. Außerdem erfahren Sie, wie Sie AWS Dienste nutzen können, die Sie bei der Überwachung und Sicherung Ihrer AWS Ressourcen unterstützen können.

Ausführlichere Informationen zur AWS IoT Sicherheit finden Sie unter Sicherheit und Identität für AWS IoT.

Topics

- Identity and Access Management für FreeRTOS
- Compliance-Validierung
- Resilienz in AWS
- Infrastruktursicherheit in FreeRTOS

Identity and Access Management für FreeRTOS

AWS Identity and Access Management (IAM) hilft einem Administrator AWS-Service , den Zugriff auf Ressourcen sicher zu AWS kontrollieren. IAM-Administratoren kontrollieren, wer authentifiziert (angemeldet) und autorisiert werden kann (über Berechtigungen verfügt), um FreeRTOS-Ressourcen zu verwenden. IAM ist ein Programm AWS-Service , das Sie ohne zusätzliche Kosten nutzen können.

Themen

- Zielgruppe
- Authentifizierung mit Identitäten
- Verwalten des Zugriffs mit Richtlinien
- So funktioniert FreeRTOS mit IAM
- Beispiele für identitätsbasierte Richtlinien für FreeRTOS
- Fehlerbehebung bei FreeRTOS-Identität und -Zugriff

Zielgruppe

Wie Sie AWS Identity and Access Management (IAM) verwenden, hängt von Ihrer Rolle ab:

- Servicebenutzer – Fordern Sie von Ihrem Administrator Berechtigungen an, wenn Sie nicht auf Features zugreifen können (siehe Fehlerbehebung bei FreeRTOS-Identität und -Zugriff).
- Serviceadministrator – Bestimmen Sie den Benutzerzugriff und stellen Sie Berechtigungsanfragen (siehe So funktioniert FreeRTOS mit IAM).
- IAM-Administrator – Schreiben Sie Richtlinien zur Zugriffsverwaltung (siehe Beispiele für identitätsbasierte Richtlinien für FreeRTOS).

Authentifizierung mit Identitäten

Authentifizierung ist die Art und Weise, wie Sie sich AWS mit Ihren Identitätsdaten anmelden. Sie müssen sich als IAM-Benutzer authentifizieren oder eine IAM-Rolle annehmen. Root-Benutzer des AWS-Kontos

Sie können sich als föderierte Identität anmelden, indem Sie Anmeldeinformationen aus einer Identitätsquelle wie AWS IAM Identity Center (IAM Identity Center), Single Sign-On-Authentifizierung oder Anmeldeinformationen verwenden. Google/Facebook Weitere Informationen zum Anmelden finden Sie unter So melden Sie sich bei Ihrem AWS-Konto an im Benutzerhandbuch für AWS-Anmeldung .

AWS Bietet für den programmatischen Zugriff ein SDK und eine CLI zum kryptografischen Signieren von Anfragen. Weitere Informationen finden Sie unter AWS Signature Version 4 for API requests im IAM-Benutzerhandbuch.

AWS-Konto Root-Benutzer

Wenn Sie einen erstellen AWS-Konto, beginnen Sie mit einer Anmeldeidentität, dem sogenannten AWS-Konto Root-Benutzer, der vollständigen Zugriff auf alle AWS-Services Ressourcen hat. Wir raten ausdrücklich davon ab, den Root-Benutzer für Alltagsaufgaben zu verwenden. Eine Liste der Aufgaben, für die Sie sich als Root-Benutzer anmelden müssen, finden Sie unter Tasks that require root user credentials im IAM-Benutzerhandbuch.

Verbundidentität

Es hat sich bewährt, dass menschliche Benutzer für den Zugriff AWS-Services mithilfe temporärer Anmeldeinformationen einen Verbund mit einem Identitätsanbieter verwenden müssen.

Eine föderierte Identität ist ein Benutzer aus Ihrem Unternehmensverzeichnis, Ihrem Directory Service Web-Identitätsanbieter oder der AWS-Services mithilfe von Anmeldeinformationen aus einer Identitätsquelle zugreift. Verbundene Identitäten übernehmen Rollen, die temporäre Anmeldeinformationen bereitstellen.

Für die zentrale Zugriffsverwaltung empfehlen wir AWS IAM Identity Center. Weitere Informationen finden Sie unter Was ist IAM Identity Center? im AWS IAM Identity Center -Benutzerhandbuch.

IAM-Benutzer und -Gruppen

Ein IAM-Benutzer ist eine Identität mit bestimmten Berechtigungen für eine einzelne Person oder Anwendung. Wir empfehlen die Verwendung temporärer Anmeldeinformationen anstelle von IAM-Benutzern mit langfristigen Anmeldeinformationen. Weitere Informationen finden Sie im IAM-Benutzerhandbuch unter Erfordern, dass menschliche Benutzer den Verbund mit einem Identitätsanbieter verwenden müssen, um AWS mithilfe temporärer Anmeldeinformationen darauf zugreifen zu können.

Eine IAM-Gruppe spezifiziert eine Sammlung von IAM-Benutzern und erleichtert die Verwaltung von Berechtigungen für große Gruppen von Benutzern. Weitere Informationen finden Sie unter Anwendungsfälle für IAM-Benutzer im IAM-Benutzerhandbuch.

IAM-Rollen

Eine IAM-Rolle ist eine Identität mit spezifischen Berechtigungen, die temporäre Anmeldeinformationen bereitstellt. Sie können eine Rolle übernehmen, indem Sie von einer Benutzer- zu einer IAM-Rolle (Konsole) wechseln oder indem Sie eine AWS Oder-API-Operation AWS CLI aufrufen. Weitere Informationen finden Sie unter Methoden, um eine Rolle zu übernehmen im IAM-Benutzerhandbuch.

IAM-Rollen sind nützlich für den Verbundbenutzer-Zugriff, temporäre IAM-Benutzerberechtigungen, kontoübergreifenden Zugriff, serviceübergreifenden Zugriff und Anwendungen, die auf Amazon EC2 laufen. Weitere Informationen finden Sie unter Kontoübergreifender Ressourcenzugriff in IAM im IAM-Benutzerhandbuch.

Verwalten des Zugriffs mit Richtlinien

Sie kontrollieren den Zugriff, AWS indem Sie Richtlinien erstellen und diese an AWS Identitäten oder Ressourcen anhängen. Eine Richtlinie definiert Berechtigungen, wenn sie mit einer Identität oder Ressource verknüpft sind. AWS bewertet diese Richtlinien, wenn ein Principal eine Anfrage stellt. Die meisten Richtlinien werden AWS als JSON-Dokumente gespeichert. Weitere Informationen zu JSON-Richtliniendokumenten finden Sie unter Übersicht über JSON-Richtlinien im IAM-Benutzerhandbuch.

Mit Hilfe von Richtlinien legen Administratoren fest, wer Zugriff auf was hat, indem sie definieren, welches Prinzipal welche Aktionen auf welchen Ressourcenund unter welchen Bedingungendurchführen darf.

Standardmäßig haben Benutzer, Gruppen und Rollen keine Berechtigungen. Ein IAM-Administrator erstellt IAM-Richtlinien und fügt sie zu Rollen hinzu, die die Benutzer dann übernehmen können. IAM-Richtlinien definieren Berechtigungen unabhängig von der Methode, die zur Ausführung der Operation verwendet wird.

Identitätsbasierte Richtlinien

Identitätsbasierte Richtlinien sind JSON-Berechtigungsrichtliniendokumente, die Sie einer Identität (Benutzer, Gruppe oder Rolle) anfügen können. Diese Richtlinien steuern, welche Aktionen Identitäten für welche Ressourcen und unter welchen Bedingungen ausführen können. Informationen zum Erstellen identitätsbasierter Richtlinien finden Sie unter Definieren benutzerdefinierter IAM-Berechtigungen mit vom Kunden verwalteten Richtlinien im IAM-Benutzerhandbuch.

Identitätsbasierte Richtlinien können Inline-Richtlinien (direkt in eine einzelne Identität eingebettet) oder verwaltete Richtlinien (eigenständige Richtlinien, die mit mehreren Identitäten verbunden sind)

sein. Informationen dazu, wie Sie zwischen verwalteten und Inline-Richtlinien wählen, finden Sie unter Choose between managed policies and inline policies im IAM-Benutzerhandbuch.

Ressourcenbasierte Richtlinien

Ressourcenbasierte Richtlinien sind JSON-Richtliniendokumente, die Sie an eine Ressource anfügen. Beispiele hierfür sind Vertrauensrichtlinien für IAM-Rollen und Amazon S3-Bucket-Richtlinien. In Services, die ressourcenbasierte Richtlinien unterstützen, können Service-Administratoren sie verwenden, um den Zugriff auf eine bestimmte Ressource zu steuern. Sie müssen in einer ressourcenbasierten Richtlinie einen Prinzipal angeben.

Ressourcenbasierte Richtlinien sind Richtlinien innerhalb dieses Diensts. Sie können AWS verwaltete Richtlinien von IAM nicht in einer ressourcenbasierten Richtlinie verwenden.

Weitere Richtlinientypen

AWS unterstützt zusätzliche Richtlinientypen, mit denen die maximalen Berechtigungen festgelegt werden können, die durch gängigere Richtlinientypen gewährt werden:

- Berechtigungsgrenzen – Eine Berechtigungsgrenze legt die maximalen Berechtigungen fest, die eine identitätsbasierte Richtlinie einer IAM-Entität erteilen kann. Weitere Informationen finden Sie unter Berechtigungsgrenzen für IAM-Entitäten im -IAM-Benutzerhandbuch.
- Richtlinien zur Dienstkontrolle (SCPs) — Geben Sie die maximalen Berechtigungen für eine Organisation oder Organisationseinheit in an AWS Organizations. Weitere Informationen finden Sie unter Service-Kontrollrichtlinien im AWS Organizations -Benutzerhandbuch.
- Richtlinien zur Ressourcenkontrolle (RCPs) — Legen Sie die maximal verfügbaren Berechtigungen für Ressourcen in Ihren Konten fest. Weitere Informationen finden Sie im AWS Organizations Benutzerhandbuch unter Richtlinien zur Ressourcenkontrolle (RCPs).
- Sitzungsrichtlinien – Sitzungsrichtlinien sind erweiterte Richtlinien, die als Parameter übergeben werden, wenn Sie eine temporäre Sitzung für eine Rolle oder einen Verbundbenutzer erstellen. Weitere Informationen finden Sie unter Sitzungsrichtlinien im IAM-Benutzerhandbuch.

Mehrere Richtlinientypen

Wenn für eine Anfrage mehrere Arten von Richtlinien gelten, sind die sich daraus ergebenden Berechtigungen schwieriger zu verstehen. Informationen darüber, wie AWS bestimmt wird, ob eine Anfrage zulässig ist, wenn mehrere Richtlinientypen betroffen sind, finden Sie unter Bewertungslogik für Richtlinien im IAM-Benutzerhandbuch.

So funktioniert FreeRTOS mit IAM

Bevor Sie IAM verwenden, um den Zugriff auf FreeRTOS zu verwalten, sollten Sie sich darüber informieren, welche IAM-Funktionen mit FreeRTOS verwendet werden können.

IAM-Funktionen, die Sie mit FreeRTOS verwenden können

IAM-Feature	FreeRTOS-Unterstützung
Identitätsbasierte Richtlinien	Ja
Ressourcenbasierte Richtlinien	Nein
Richtlinienaktionen	Ja
Richtlinienressourcen	Ja
Richtlinienbedingungsschlüssel (services pezifisch)	Ja
ACLs	Nein
ABAC (Tags in Richtlinien)	Teilweise
Temporäre Anmeldeinformationen	Ja
Prinzipalberechtigungen	Ja
Servicerollen	Ja
Service-verknüpfte Rollen	Nein

Einen allgemeinen Überblick darüber, wie FreeRTOS und andere AWS Dienste mit den meisten IAM-Funktionen funktionieren, finden Sie im AWS IAM-Benutzerhandbuch unter Dienste, die mit IAM funktionieren.

Identitätsbasierte Richtlinien für FreeRTOS

Unterstützt Richtlinien auf Identitätsbasis: Ja

Identitätsbasierte Richtlinien sind JSON-Berechtigungsrichtliniendokumente, die Sie einer Identität anfügen können, wie z. B. IAM-Benutzern, -Benutzergruppen oder -Rollen. Diese Richtlinien steuern, welche Aktionen die Benutzer und Rollen für welche Ressourcen und unter welchen Bedingungen ausführen können. Informationen zum Erstellen identitätsbasierter Richtlinien finden Sie unter Definieren benutzerdefinierter IAM-Berechtigungen mit vom Kunden verwalteten Richtlinien im IAM-Benutzerhandbuch.

Mit identitätsbasierten IAM-Richtlinien können Sie angeben, welche Aktionen und Ressourcen zugelassen oder abgelehnt werden. Darüber hinaus können Sie die Bedingungen festlegen, unter denen Aktionen zugelassen oder abgelehnt werden. Informationen zu sämtlichen Elementen, die Sie in einer JSON-Richtlinie verwenden, finden Sie in der IAM-Referenz für JSON-Richtlinienelemente im IAM-Benutzerhandbuch.

Beispiele für identitätsbasierte Richtlinien für FreeRTOS

Beispiele für identitätsbasierte FreeRTOS-Richtlinien finden Sie unter. Beispiele für identitätsbasierte Richtlinien für FreeRTOS

Ressourcenbasierte Richtlinien innerhalb von FreeRTOS

Unterstützt ressourcenbasierte Richtlinien: Nein

Ressourcenbasierte Richtlinien sind JSON-Richtliniendokumente, die Sie an eine Ressource anfügen. Beispiele für ressourcenbasierte Richtlinien sind IAM-Rollen-Vertrauensrichtlinien und Amazon-S3-Bucket-Richtlinien. In Services, die ressourcenbasierte Richtlinien unterstützen, können Service-Administratoren sie verwenden, um den Zugriff auf eine bestimmte Ressource zu steuern. Für die Ressource, an welche die Richtlinie angehängt ist, legt die Richtlinie fest, welche Aktionen ein bestimmter Prinzipal unter welchen Bedingungen für diese Ressource ausführen kann. Sie müssen in einer ressourcenbasierten Richtlinie einen Prinzipal angeben. Zu den Prinzipalen können Konten, Benutzer, Rollen, Verbundbenutzer oder gehören. AWS-Services

Um kontoübergreifenden Zugriff zu ermöglichen, können Sie ein gesamtes Konto oder IAM-Entitäten in einem anderen Konto als Prinzipal in einer ressourcenbasierten Richtlinie angeben. Weitere Informationen finden Sie unter Kontoübergreifender Ressourcenzugriff in IAM im IAM-Benutzerhandbuch.

Politische Maßnahmen für FreeRTOS

Unterstützt Richtlinienaktionen: Ja

Administratoren können mithilfe von AWS JSON-Richtlinien angeben, wer Zugriff auf was hat. Das heißt, welcher Prinzipal Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen kann.

Das Element `Action` einer JSON-Richtlinie beschreibt die Aktionen, mit denen Sie den Zugriff in einer Richtlinie zulassen oder verweigern können. Nehmen Sie Aktionen in eine Richtlinie auf, um Berechtigungen zur Ausführung des zugehörigen Vorgangs zu erteilen.

Eine Liste der FreeRTOS-Aktionen finden Sie unter Von FreeRTOS definierte Aktionen in der Service Authorization Reference.

Richtlinienaktionen in FreeRTOS verwenden das folgende Präfix vor der Aktion:

```
awes
```

Um mehrere Aktionen in einer einzigen Anweisung anzugeben, trennen Sie sie mit Kommata:

```
"Action": [
      "awes:action1",
      "awes:action2"
         ]
```

Beispiele für identitätsbasierte FreeRTOS-Richtlinien finden Sie unter. Beispiele für identitätsbasierte Richtlinien für FreeRTOS

Politische Ressourcen für FreeRTOS

Unterstützt Richtlinienressourcen: Ja

Administratoren können mithilfe von AWS JSON-Richtlinien angeben, wer auf was Zugriff hat. Das heißt, welcher Prinzipal Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen kann.

Das JSON-Richtlinienelement `Resource` gibt die Objekte an, auf welche die Aktion angewendet wird. Als Best Practice geben Sie eine Ressource mit dem zugehörigen Amazon-Ressourcennamen (ARN) an. Verwenden Sie für Aktionen, die keine Berechtigungen auf Ressourcenebene unterstützen, einen Platzhalter (*), um anzugeben, dass die Anweisung für alle Ressourcen gilt.

```
"Resource": "*"
```

Eine Liste der FreeRTOS-Ressourcentypen und ihrer ARNs Typen finden Sie unter Von FreeRTOS definierte Ressourcen in der Service Authorization Reference. Informationen darüber, mit welchen Aktionen Sie den ARN jeder Ressource angeben können, finden Sie unter Von FreeRTOS definierte Aktionen.

Beispiele für identitätsbasierte FreeRTOS-Richtlinien finden Sie unter. Beispiele für identitätsbasierte Richtlinien für FreeRTOS

Schlüssel zur Richtlinienbedingung für FreeRTOS

Unterstützt servicespezifische Richtlinienbedingungsschlüssel: Ja

Administratoren können AWS JSON-Richtlinien verwenden, um festzulegen, wer Zugriff auf was hat. Das heißt, welcher Prinzipal Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen kann.

Das Element `Condition` gibt an, wann Anweisungen auf der Grundlage definierter Kriterien ausgeführt werden. Sie können bedingte Ausdrücke erstellen, die Bedingungsoperatoren verwenden, z. B. ist gleich oder kleiner als, damit die Bedingung in der Richtlinie mit Werten in der Anforderung übereinstimmt. Eine Übersicht aller AWS globalen Bedingungsschlüssel finden Sie unter Kontextschlüssel für AWS globale Bedingungen im IAM-Benutzerhandbuch.

Eine Liste der FreeRTOS-Bedingungsschlüssel finden Sie unter Bedingungsschlüssel für FreeRTOS in der Service Authorization Reference. Informationen zu den Aktionen und Ressourcen, mit denen Sie einen Bedingungsschlüssel verwenden können, finden Sie unter Von FreeRTOS definierte Aktionen.

Beispiele für identitätsbasierte FreeRTOS-Richtlinien finden Sie unter. Beispiele für identitätsbasierte Richtlinien für FreeRTOS

ACLs in FreeRTOS

Unterstützt ACLs: Nein

Zugriffskontrolllisten (ACLs) steuern, welche Principals (Kontomitglieder, Benutzer oder Rollen) über Zugriffsberechtigungen für eine Ressource verfügen. ACLs ähneln ressourcenbasierten Richtlinien, verwenden jedoch nicht das JSON-Richtliniendokumentformat.

ABAC mit FreeRTOS

Unterstützt ABAC (Tags in Richtlinien): Teilweise

Die attributbasierte Zugriffskontrolle (ABAC) ist eine Autorisierungsstrategie, bei der Berechtigungen basierend auf Attributen, auch als Tags bezeichnet, definiert werden. Sie können Tags an IAM-Entitäten und AWS -Ressourcen anhängen und dann ABAC-Richtlinien entwerfen, die Operationen zulassen, wenn das Tag des Prinzipals mit dem Tag auf der Ressource übereinstimmt.

Um den Zugriff auf der Grundlage von Tags zu steuern, geben Sie im Bedingungselement einer Richtlinie Tag-Informationen an, indem Sie die Schlüssel `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, oder Bedingung `aws:TagKeys` verwenden.

Wenn ein Service alle drei Bedingungsschlüssel für jeden Ressourcentyp unterstützt, lautet der Wert für den Service Ja. Wenn ein Service alle drei Bedingungsschlüssel für nur einige Ressourcentypen unterstützt, lautet der Wert Teilweise.

Weitere Informationen zu ABAC finden Sie unter Definieren von Berechtigungen mit ABAC-Autorisierung im IAM-Benutzerhandbuch. Um ein Tutorial mit Schritten zur Einstellung von ABAC anzuzeigen, siehe Attributbasierte Zugriffskontrolle (ABAC) verwenden im IAM-Benutzerhandbuch.

Temporäre Anmeldeinformationen mit FreeRTOS verwenden

Unterstützt temporäre Anmeldeinformationen: Ja

Temporäre Anmeldeinformationen ermöglichen kurzfristigen Zugriff auf AWS Ressourcen und werden automatisch erstellt, wenn Sie einen Verbund verwenden oder die Rollen wechseln. AWS empfiehlt, temporäre Anmeldeinformationen dynamisch zu generieren, anstatt langfristige Zugriffsschlüssel zu verwenden. Weitere Informationen finden Sie unter Temporäre Anmeldeinformationen in IAM und AWS-Services , die mit IAM funktionieren im IAM-Benutzerhandbuch.

Serviceübergreifende Prinzipalberechtigungen für FreeRTOS

Unterstützt Forward Access Sessions (FAS): Ja

Forward Access Sessions (FAS) verwenden die Berechtigungen des Prinzipals, der einen aufruft AWS-Service, in Kombination mit der Anfrage, Anfragen AWS-Service an nachgelagerte Dienste zu stellen. Einzelheiten zu den Richtlinien für FAS-Anforderungen finden Sie unter Zugriffssitzungen weiterleiten.

Servicerollen für FreeRTOS

Unterstützt Servicerollen: Ja

Eine Servicerolle ist eine IAM-Rolle, die ein Service annimmt, um Aktionen in Ihrem Namen auszuführen. Ein IAM-Administrator kann eine Servicerolle innerhalb von IAM erstellen, ändern und löschen. Weitere Informationen finden Sie unter Erstellen einer Rolle zum Delegieren von Berechtigungen an einen AWS-Service im IAM-Benutzerhandbuch.

> **Warning**
>
> Das Ändern der Berechtigungen für eine Servicerolle kann die FreeRTOS-Funktionalität beeinträchtigen. Bearbeiten Sie Servicerollen nur, wenn FreeRTOS Sie dazu anleitet.

Serviceverknüpfte Rollen für FreeRTOS

Unterstützt serviceverknüpfte Rollen: Ja

Eine dienstbezogene Rolle ist eine Art von Servicerolle, die mit einer AWS-Service verknüpft ist. Der Service kann die Rolle übernehmen, um eine Aktion in Ihrem Namen auszuführen. Dienstbezogene Rollen werden in Ihrem Dienst angezeigt AWS-Konto und gehören dem Dienst. Ein IAM-Administrator kann die Berechtigungen für Service-verknüpfte Rollen anzeigen, aber nicht bearbeiten.

Details zum Erstellen oder Verwalten von serviceverknüpften Rollen finden Sie unter AWS -Services, die mit IAM funktionieren. Suchen Sie in der Tabelle nach einem Service mit einem `Yes` in der Spalte Service-linked role (Serviceverknüpfte Rolle). Wählen Sie den Link Yes (Ja) aus, um die Dokumentation für die serviceverknüpfte Rolle für diesen Service anzuzeigen.

Beispiele für identitätsbasierte Richtlinien für FreeRTOS

Standardmäßig sind Benutzer und Rollen nicht berechtigt, FreeRTOS-Ressourcen zu erstellen oder zu ändern. Ein IAM-Administrator muss IAM-Richtlinien erstellen, die Benutzern die Berechtigung erteilen, Aktionen für die Ressourcen auszuführen, die sie benötigen.

Informationen dazu, wie Sie unter Verwendung dieser beispielhaften JSON-Richtliniendokumente eine identitätsbasierte IAM-Richtlinie erstellen, finden Sie unter Erstellen von IAM-Richtlinien (Konsole) im IAM-Benutzerhandbuch.

Einzelheiten zu den von FreeRTOS definierten Aktionen und Ressourcentypen, einschließlich des Formats von ARNs für jeden der Ressourcentypen, finden Sie unter Aktionen, Ressourcen und Bedingungsschlüssel für FreeRTOS in der Service Authorization Reference.

Themen

- Best Practices für Richtlinien
- Verwenden der FreeRTOS-Konsole
- Gewähren der Berechtigung zur Anzeige der eigenen Berechtigungen für Benutzer

Best Practices für Richtlinien

Identitätsbasierte Richtlinien legen fest, ob jemand FreeRTOS-Ressourcen in Ihrem Konto erstellen, darauf zugreifen oder sie löschen kann. Dies kann zusätzliche Kosten für Ihr verursachen AWS-Konto. Wenn Sie identitätsbasierte Richtlinien erstellen oder bearbeiten, befolgen Sie diese Richtlinien und Empfehlungen:

- Erste Schritte mit AWS verwalteten Richtlinien und Umstellung auf Berechtigungen mit den geringsten Rechten — Verwenden Sie die AWS verwalteten Richtlinien, die Berechtigungen für viele gängige Anwendungsfälle gewähren, um damit zu beginnen, Ihren Benutzern und Workloads Berechtigungen zu gewähren. Sie sind in Ihrem verfügbar. AWS-Konto Wir empfehlen Ihnen, die Berechtigungen weiter zu reduzieren, indem Sie vom AWS Kunden verwaltete Richtlinien definieren, die speziell auf Ihre Anwendungsfälle zugeschnitten sind. Weitere Informationen finden Sie unter Von AWS verwaltete Richtlinien oder Von AWS verwaltete Richtlinien für Auftragsfunktionen im IAM-Benutzerhandbuch.
- Anwendung von Berechtigungen mit den geringsten Rechten – Wenn Sie mit IAM-Richtlinien Berechtigungen festlegen, gewähren Sie nur die Berechtigungen, die für die Durchführung einer Aufgabe erforderlich sind. Sie tun dies, indem Sie die Aktionen definieren, die für bestimmte Ressourcen unter bestimmten Bedingungen durchgeführt werden können, auch bekannt als die geringsten Berechtigungen. Weitere Informationen zur Verwendung von IAM zum Anwenden von Berechtigungen finden Sie unter Richtlinien und Berechtigungen in IAM im IAM-Benutzerhandbuch.
- Verwenden von Bedingungen in IAM-Richtlinien zur weiteren Einschränkung des Zugriffs – Sie können Ihren Richtlinien eine Bedingung hinzufügen, um den Zugriff auf Aktionen und Ressourcen zu beschränken. Sie können beispielsweise eine Richtlinienbedingung schreiben, um festzulegen, dass alle Anforderungen mithilfe von SSL gesendet werden müssen. Sie können auch Bedingungen verwenden, um Zugriff auf Serviceaktionen zu gewähren, wenn diese für einen

bestimmten Zweck verwendet werden AWS-Service, z. CloudFormation B. Weitere Informationen finden Sie unter IAM-JSON-Richtlinienelemente: Bedingung im IAM-Benutzerhandbuch.

- Verwenden von IAM Access Analyzer zur Validierung Ihrer IAM-Richtlinien, um sichere und funktionale Berechtigungen zu gewährleisten – IAM Access Analyzer validiert neue und vorhandene Richtlinien, damit die Richtlinien der IAM-Richtliniensprache (JSON) und den bewährten IAM-Methoden entsprechen. IAM Access Analyzer stellt mehr als 100 Richtlinienprüfungen und umsetzbare Empfehlungen zur Verfügung, damit Sie sichere und funktionale Richtlinien erstellen können. Weitere Informationen finden Sie unter Richtlinienvalidierung mit IAM Access Analyzer im IAM-Benutzerhandbuch.
- Multi-Faktor-Authentifizierung (MFA) erforderlich — Wenn Sie ein Szenario haben, das IAM-Benutzer oder einen Root-Benutzer in Ihrem System erfordert AWS-Konto, aktivieren Sie MFA für zusätzliche Sicherheit. Um MFA beim Aufrufen von API-Vorgängen anzufordern, fügen Sie Ihren Richtlinien MFA-Bedingungen hinzu. Weitere Informationen finden Sie unter Sicherer API-Zugriff mit MFA im IAM-Benutzerhandbuch.

Weitere Informationen zu bewährten Methoden in IAM finden Sie unter Best Practices für die Sicherheit in IAM im IAM-Benutzerhandbuch.

Verwenden der FreeRTOS-Konsole

Um auf die FreeRTOS-Konsole zugreifen zu können, benötigen Sie ein Mindestmaß an Berechtigungen. Diese Berechtigungen müssen es Ihnen ermöglichen, Details zu den FreeRTOS-Ressourcen in Ihrem aufzulisten und anzuzeigen. AWS-Konto Wenn Sie eine identitätsbasierte Richtlinie erstellen, die strenger ist als die mindestens erforderlichen Berechtigungen, funktioniert die Konsole nicht wie vorgesehen für Entitäten (Benutzer oder Rollen) mit dieser Richtlinie.

Sie müssen Benutzern, die nur die API AWS CLI oder die API aufrufen, keine Mindestberechtigungen für die Konsole gewähren. AWS Stattdessen sollten Sie nur Zugriff auf die Aktionen zulassen, die der API-Operation entsprechen, die die Benutzer ausführen möchten.

Um sicherzustellen, dass Benutzer und Rollen die FreeRTOS-Konsole weiterhin verwenden können, fügen Sie den Entitäten auch das FreeRTOS `ConsoleAccess` oder die `ReadOnly` AWS verwaltete Richtlinie hinzu. Weitere Informationen finden Sie unter Hinzufügen von Berechtigungen zu einem Benutzer im IAM-Benutzerhandbuch.

Gewähren der Berechtigung zur Anzeige der eigenen Berechtigungen für Benutzer

In diesem Beispiel wird gezeigt, wie Sie eine Richtlinie erstellen, die IAM-Benutzern die Berechtigung zum Anzeigen der eingebundenen Richtlinien und verwalteten Richtlinien gewährt, die ihrer Benutzeridentität angefügt sind. Diese Richtlinie beinhaltet Berechtigungen zum Ausführen dieser Aktion auf der Konsole oder programmgesteuert mithilfe der API oder. AWS CLI AWS

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Fehlerbehebung bei FreeRTOS-Identität und -Zugriff

Verwenden Sie die folgenden Informationen, um häufig auftretende Probleme zu diagnostizieren und zu beheben, die bei der Arbeit mit FreeRTOS und IAM auftreten können.

Themen

- Ich bin nicht berechtigt, eine Aktion in FreeRTOS durchzuführen
- Ich bin nicht berechtigt, iam auszuführen: PassRole
- Erlaube Leuten außerhalb von mir AWS-Konto den Zugriff auf meine FreeRTOS-Ressourcen

Ich bin nicht berechtigt, eine Aktion in FreeRTOS durchzuführen

Wenn Sie eine Fehlermeldung erhalten, dass Sie nicht zur Durchführung einer Aktion berechtigt sind, müssen Ihre Richtlinien aktualisiert werden, damit Sie die Aktion durchführen können.

Der folgende Beispielfehler tritt auf, wenn der IAM-Benutzer `mateojackson` versucht, über die Konsole Details zu einer fiktiven *`my-example-widget`*-Ressource anzuzeigen, jedoch nicht über `awes:`*`GetWidget`*-Berechtigungen verfügt.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 awes:GetWidget on resource: my-example-widget
```

In diesem Fall muss die Richtlinie für den Benutzer `mateojackson` aktualisiert werden, damit er mit der `awes:`*`GetWidget`*-Aktion auf die *`my-example-widget`*-Ressource zugreifen kann.

Wenn Sie Hilfe benötigen, wenden Sie sich an Ihren AWS Administrator. Ihr Administrator hat Ihnen Ihre Anmeldeinformationen zur Verfügung gestellt.

Ich bin nicht berechtigt, iam auszuführen: PassRole

Wenn Sie eine Fehlermeldung erhalten, dass Sie nicht berechtigt sind, die `iam:PassRole` Aktion auszuführen, müssen Ihre Richtlinien aktualisiert werden, damit Sie eine Rolle an FreeRTOS übergeben können.

Einige AWS-Services ermöglichen es Ihnen, eine bestehende Rolle an diesen Dienst zu übergeben, anstatt eine neue Servicerolle oder eine dienstverknüpfte Rolle zu erstellen. Hierzu benötigen Sie Berechtigungen für die Übergabe der Rolle an den Dienst.

Der folgende Beispielfehler tritt auf, wenn ein IAM-Benutzer mit dem Namen `marymajor` versucht, die Konsole zu verwenden, um eine Aktion in FreeRTOS auszuführen. Die Aktion erfordert jedoch, dass der Service über Berechtigungen verfügt, die durch eine Servicerolle gewährt werden. Mary besitzt keine Berechtigungen für die Übergabe der Rolle an den Dienst.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In diesem Fall müssen die Richtlinien von Mary aktualisiert werden, um die Aktion `iam:PassRole` ausführen zu können.

Wenn Sie Hilfe benötigen, wenden Sie sich an Ihren Administrator. AWS Ihr Administrator hat Ihnen Ihre Anmeldeinformationen zur Verfügung gestellt.

Erlaube Leuten außerhalb von mir AWS-Konto den Zugriff auf meine FreeRTOS-Ressourcen

Sie können eine Rolle erstellen, mit der Benutzer in anderen Konten oder Personen außerhalb Ihrer Organisation auf Ihre Ressourcen zugreifen können. Sie können festlegen, wem die Übernahme der Rolle anvertraut wird. Für Dienste, die ressourcenbasierte Richtlinien oder Zugriffskontrolllisten (ACLs) unterstützen, können Sie diese Richtlinien verwenden, um Personen Zugriff auf Ihre Ressourcen zu gewähren.

Weitere Informationen dazu finden Sie hier:

- Informationen darüber, ob FreeRTOS diese Funktionen unterstützt, finden Sie unter. So funktioniert FreeRTOS mit IAM
- Informationen dazu, wie Sie Zugriff auf Ihre Ressourcen gewähren können, AWS-Konten die Ihnen gehören, finden Sie im IAM-Benutzerhandbuch unter Zugriff auf einen IAM-Benutzer in einem anderen AWS-Konto , den Sie besitzen.
- Informationen dazu, wie Sie Dritten Zugriff auf Ihre Ressourcen gewähren können AWS-Konten, finden Sie AWS-Konten im IAM-Benutzerhandbuch unter Gewähren des Zugriffs für Dritte.
- Informationen dazu, wie Sie über einen Identitätsverbund Zugriff gewähren, finden Sie unter Gewähren von Zugriff für extern authentifizierte Benutzer (Identitätsverbund) im IAM-Benutzerhandbuch.
- Informationen zum Unterschied zwischen der Verwendung von Rollen und ressourcenbasierten Richtlinien für den kontoübergreifenden Zugriff finden Sie unter Kontoübergreifender Ressourcenzugriff in IAM im IAM-Benutzerhandbuch.

Compliance-Validierung

Informationen darüber, ob AWS-Service ein AWS-Services in den Geltungsbereich bestimmter Compliance-Programme fällt, finden Sie unter Umfang nach Compliance-Programm AWS-Services unter . Wählen Sie dort das Compliance-Programm aus, an dem Sie interessiert sind. Allgemeine Informationen finden Sie unter AWS Compliance-Programme AWS .

Sie können Prüfberichte von Drittanbietern unter herunterladen AWS Artifact. Weitere Informationen finden Sie unter Berichte herunterladen unter .

Ihre Verantwortung für die Einhaltung der Vorschriften bei der Nutzung AWS-Services hängt von der Vertraulichkeit Ihrer Daten, den Compliance-Zielen Ihres Unternehmens und den geltenden Gesetzen und Vorschriften ab. Weitere Informationen zu Ihrer Verantwortung für die Einhaltung der Vorschriften bei der Nutzung AWS-Services finden Sie in der AWS Sicherheitsdokumentation.

Resilienz in AWS

Die AWS globale Infrastruktur basiert auf AWS Regionen und Availability Zones. AWS Regionen stellen mehrere physisch getrennte und isolierte Availability Zones bereit, die mit Netzwerken mit geringer Latenz, hohem Durchsatz und hochredundanten Vernetzungen verbunden sind. Mithilfe von Availability Zones können Sie Anwendungen und Datenbanken erstellen und ausführen, die automatisch Failover zwischen Availability Zones ausführen, ohne dass es zu Unterbrechungen kommt. Availability Zones sind besser hoch verfügbar, fehlertoleranter und skalierbarer als herkömmliche Infrastrukturen mit einem oder mehreren Rechenzentren.

Weitere Informationen zu AWS Regionen und Availability Zones finden Sie unter AWS Globale Infrastruktur.

Infrastruktursicherheit in FreeRTOS

AWS Managed Services sind durch die AWS globalen Netzwerksicherheitsverfahren geschützt, die im Whitepaper Amazon Web Services: Sicherheitsprozesse im Überblick beschrieben sind.

Sie verwenden AWS veröffentlichte API-Aufrufe, um über das Netzwerk auf AWS Dienste zuzugreifen. Clients müssen Transport Layer Security (TLS) 1.2 oder höher unterstützen. Wir empfehlen TLS 1.3 oder höher. Clients müssen außerdem Verschlüsselungssammlungen mit PFS (Perfect Forward Secrecy) wie DHE (Ephemeral Diffie-Hellman) oder ECDHE (Elliptic Curve

Ephemeral Diffie-Hellman) unterstützen. Die meisten modernen Systemen wie Java 7 und höher unterstützen diese Modi.

Außerdem müssen Anforderungen mit einer Zugriffsschlüssel-ID und einem geheimen Zugriffsschlüssel signiert sein, der einem IAM-Prinzipal zugeordnet ist. Alternativ können Sie mit AWS -Security-Token-Service (AWS STS) temporäre Sicherheitsanmeldeinformationen erstellen, um die Anforderungen zu signieren.

Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Wenn Sie ein bestehendes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-Freertos-Repository basiert, gehen Sie wie folgt vor:

1. Bleiben Sie über die neuesten, öffentlich verfügbaren Sicherheitsupdates auf dem Laufenden. Suchen Sie auf der Seite mit den FreeRTOS LTS-Bibliotheken nach Updates oder abonnieren Sie das GitHub FreeRTOS-LTS-Repository, um die neuesten LTS-Patches mit kritischen und sicherheitsrelevanten Bugfixes zu erhalten. Sie können die neuesten benötigten FreeRTOS LTS-Patches direkt aus den einzelnen Repositorys herunterladen oder klonen. GitHub
2. Erwägen Sie, die Implementierung der Netzwerktransportschnittstelle umzugestalten, um Ihre Hardwareplattform zu optimieren. Die Zusammenfassung APIs wie Secure Sockets und WLAN APIs werden von der neuesten CoreMQTT-Bibliothek nicht benötigt. Weitere Informationen finden Sie unter Transport Interface.

Anhang

Die folgende Tabelle enthält Empfehlungen für alle Demo-Projekte, Legacy-Bibliotheken und Zusammenfassungen APIs im Amazon-FreeRTOS-Repository.

Migrierte Bibliotheken und Demos

Name	Typ	Empfehlungen	
CoreHTTP	Demos und Bibliothek	Klonen oder laden Sie die CoreHTTP-Bibliothek direkt aus dem CoreHTTP-Repository (Untermodul bei Verwendung von Git) in der FreeRTOS Github-Organisation herunter. Die CoreHTTP-Demos befinden sich in der primären FreeRTOS-Distribution. Weitere Informationen finden Sie auf der CoreHTTP-Seite.	

Name	Typ	Empfehlungen
CoreMQTT	Demos und Bibliothek	Klonen oder laden Sie die CoreMQTT-Bibliothek direkt aus dem CoreMQTT-Repository (Untermodul bei Verwendung von Git) in der FreeRTOS Github-Organisation herunter. Die CoreMQTT-Demos befinden sich in der primären FreeRTOS-Distribution. Weitere Informationen finden Sie auf der CoreMQTT-Seite.
CoreMQTT-Agent	Demos und Bibliothek	Klonen oder laden Sie die CoreMQTT-Agent-Bibliothek direkt aus dem CoreMQTT-Agent-Repository (Untermodul bei Verwendung von Git) in der FreeRTOS Github-Organisation herunter. Die CoreMQTT-Agent-Demos befinden sich im CoreMQTT-Agent-Demos-Repository. Weitere Informationen finden Sie auf der CoreMQTT-Agent-Seite.
device_defender_for_aws	Demos und Bibliothek	Die AWS IoT Device Defender-Bibliothek befindet sich in ihrem Repository in der AWS GitHub Organisation. Klonen oder laden Sie sie direkt aus dem AWS IoT Device Defender-Repository herunter (Untermodul, wenn Sie Git verwenden). Die AWS IoT Device Defender-Demos befinden sich in der primären FreeRTOS-Distribution. Weitere Informationen finden Sie auf der AWS IoT Device Defender-Seite.

Name	Typ	Empfehlungen	
device_shadow_for_aws	Demos und Bibliothek	Die AWS IoT Device Shadow-Bibliothek befindet sich in ihrem Repository in der AWS GitHub Organisation. Klonen oder laden Sie es (Untermodul bei Verwendung von Git) direkt aus dem AWS IoT Device Shadow (Repository) herunter. Die AWS IoT Device Shadow-Demos befinden sich in der primären FreeRTOS-Distribution. Weitere Informationen finden Sie auf der AWS IoT Device Shadow-Seite.	
jobs_for_aws	Demos und Bibliothek	Die AWS IoT Jobs-Bibliothek befindet sich in ihrem Repository in der AWS GitHub Organisation. Klonen oder laden Sie sie direkt aus dem AWS IoT Jobs-Repository herunter (Untermodul, wenn Sie Git verwenden). Die AWS IoT Jobs-Demos befinden sich in der primären FreeRTOS-Distribution. Weitere Informationen finden Sie auf der AWS IoT Jobs-Seite.	

Name	Typ	Empfehlungen
OTA	Demos und Bibliothek	Die AWS IoT Over-The-Air (OTA-) Update-Bibliothek befindet sich in ihrem Repository in der AWS GitHub Organisation. Klonen oder laden Sie sie direkt aus dem AWS IoT OTA-Repository herunter (Untermodul, wenn Sie Git verwenden). Die AWS IoT OTA-Demos befinden sich in der primären FreeRTOS-Distribution. Weitere Informationen finden Sie auf der AWS IoT OTA-Seite.
CLI und FreeRTOS_ Plus_CLI	Demos und Bibliothek	Es läuft ein CLI-Beispiel auf WinSim. Weitere Informationen finden Sie auf der Seite FreeRTOS Plus Command Line Interface. Die Featured FreeRTOS IoT-Refer enzintegrationen auf den Plattformen NXP i.MX RT1 060 und STM32U5 bieten auch CLI-Beispiele auf aktueller Hardware.
Protokollierung	Makro	Es gibt Implementierungen des Logging-Makros für bestimmte Hardwareplattformen, die von einigen FreeRTOS-Bibliotheken verwendet werden. Informationen zur Implement ierung des Logging-Makros finden Sie auf der Logging-Seite. Ein Beispiel, das auf aktueller Hardware läuft, finden Sie in einer der von FreeRTOS empfohlenen IoT-Refer enzen.

Name	Typ	Empfehlungen	
greengrass_connectivity	Demo	[Migration läuft] Bei diesem Demo-Projekt wurde davon ausgegangen, dass Cloud-Konnektivität verfügbar war, bevor eine Verbindung zu einem AWS IoT Greengrass-Gerät hergestellt wurde. Ein neues Projekt, das lokale Authentifizierungs- und Erkennungsfunktionen demonstriert, befindet sich in der Entwicklung. Erwarten Sie, dass das neue Demo-Projekt in Kürze in der FreeRTOS Github-Organisation veröffentlicht wird.	

Veraltete Bibliotheken und Demos

Name	Typ	Empfehlungen	
BLAU	Demos und Bibliotheken	Die FreeRTOS BLE-Bibliothek implementiert das proprietäre MQTT-Protokoll und unterstützt das Veröffentlichen und Abonnieren von MQTT-Themen über Bluetooth Low Energy (BLE) über ein Proxygerät wie ein Mobiltelefon. Dies ist nicht mehr vorgeschrieben. Verwenden Sie entweder Ihren eigenen BLE-Stack oder eine Drittanbieter-Option wie NiMBLE, um Ihr Projekt optimal zu optimieren.	
dev_mode_key_provisioning	Demos	Die Featured FreeRTOS IoT-Referenzintegrationen auf den Plattformen NXP i.MX RT1 060, STM32U5 oder ESP32-C3 bieten Beispiele	

Name	Typ	Empfehlungen
		für wichtige Bereitstellungen mithilfe einer CLI.
Posix	Abstraktion und Demo	Nicht zur Verwendung empfohlen.
wifi_provisioning	Beispiel	In diesem Beispiel wurde gezeigt, wie WiFi Anmeldeinformationen auf einem Gerät mithilfe der Amazon-FreeRTOS BLE-Bibliothek bereitgestellt werden. Ein Beispiel für die WiFi Bereitstellung über BLE finden Sie in der FreeRTOS Featured IoT-Referenz auf der ESP32C3-Plattform.
Zusammenfassung der Vorgängerversionen APIs	Code	Diese wurden entwickelt APIs , um eine abstrakte Schnittstelle für verschiedene Software-Stacks, Konnektivitätsmodule und MCU-Plattformen von Drittanbietern verschiedener Anbieter bereitzustellen. Zum Beispiel gibt es Schnittstellen für WiFi Abstraktion, sichere Sockets usw. Sie werden im Amazon-FreeRTOS-Repository unterstützt und befinden sich im Ordner. `/libraries/abstractions/` Diese APIs sind bei der Verwendung der FreeRTOS LTS-Bibliotheken nicht erforderlich.

Für die Bibliotheken und Demos in der obigen Tabelle werden keine Sicherheitspatches oder Bugfixes bereitgestellt.

Bibliotheken von Drittanbietern

Wenn Demos in Amazon-FreeRTOS Bibliotheken von Drittanbietern verwenden, empfehlen wir, diese direkt aus deren Repositorys von Drittanbietern zu submodulieren.

- CMock: klone es (Submodul, wenn du Git verwendest) direkt aus dem Cmock-Repository.
- jsmn: nicht empfohlen und nicht mehr unterstützt.
- lwip: klone es (Submodul, wenn du Git benutzt) direkt aus dem lwip-tcpip-Repository.
- lwip_osal: Informationen zur Implementierung von lwip_osal auf Ihrer Hardwareplattform/Ihrem Board finden Sie in den FreeRTOS Featured Reference Integrations auf i.MX RT1 060 oder STM32U5.
- mbedtls: klone es (Submodul, wenn du Git verwendest) direkt aus dem Mbed-TLS-Repository. Die mbedtls-Konfiguration und die Hilfsprogramme können wiederverwendet werden. Erstellen Sie in diesem Fall eine lokale Kopie.
- pkcs11: klonen Sie es (Submodul, wenn Sie Git verwenden) direkt aus der PKCS11Kernbibliothek oder dem OASIS PKCS 11-Repository.
- tinycbor: klone es (Submodul, wenn du Git verwendest) direkt aus dem Tinycbor-Repository.
- tinycrypt: Wir empfehlen Ihnen, Kryptobeschleuniger von Ihrer MCU-Plattform zu verwenden, sofern verfügbar. Wenn Sie Tinycrypt weiterhin verwenden möchten, klonen Sie es (Submodul, wenn Sie Git verwenden) direkt aus dem Tinycrypt-Repository.
- tracealyzer_recorder: Klonen Sie es (Submodul, wenn Sie Git verwenden) direkt aus dem Trace-Recorder-Repository von Percepio.
- Unity: klone es (Submodul, wenn du Git verwendest) direkt aus dem /Unity-Repository. ThrowTheSwitch
- win_pcap: win_pcap wird nicht mehr verwaltet. Wir empfehlen, stattdessen libslirp, libpcap (posix) oder npcap zu verwenden.

Portierungstests und Integrationstests

Alle Tests unter dem `/tests` Ordner, die zur Validierung der Integration von FreeRTOS-Bibliotheken erforderlich sind, wurden in das Repository migriert. FreeRTOS-Libraries-Integration-Tests Diese können verwendet werden, um die PAL-Implementierung und die Bibliotheksintegration zu testen. Dieselben Tests werden von AWS IoT Device Tester (IDT) für das AWS Gerätequalifizierungsprogramm für FreeRTOS verwendet.

FreeRTOS Archivierte Dokumentation

Dieser Inhalt ist archiviert und wird nicht mehr aktualisiert. Er kann auf veraltete Informationen oder veraltete Funktionen verweisen. Die neuesten Anleitungen zu FreeRTOS finden Sie in den aktuellsten Informationen in diesem FreeRTOS-Benutzerhandbuch oder in einer anderen relevanten Dokumentation. AWS Aktuelle Informationen finden Sie unter. Was ist FreeRTOS?

Themen

- Archiv des FreeRTOS-Benutzerhandbuches
- Frühere Inhalte des FreeRTOS-Benutzerhandbuchs

Archiv des FreeRTOS-Benutzerhandbuches

Diese früheren Versionen des FreeRTOS-Benutzerhandbuchs sind für die Verwendung mit FreeRTOS LTS-Versionen (Long Term Support) verfügbar.

- FreeRTOS Benutzerhandbuch für FreeRTOS Version 202210.00
- FreeRTOS Benutzerhandbuch für FreeRTOS Version 202012.00

Frühere Inhalte des FreeRTOS-Benutzerhandbuchs

Dieser Inhalt ist veraltet, wird aber hier als Referenz zur Verfügung gestellt.

Links Erste Schritte mit FreeRTOS zu aktuellen Inhalten finden Sie unter.

Erste Schritte mit FreeRTOS

> **Important**
>
> Diese Seite bezieht sich auf das Amazon-FreeRTOS-Repository, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Dieses Tutorial „Erste Schritte mit FreeRTOS“ zeigt Ihnen, wie Sie FreeRTOS auf einen Host-Computer herunterladen und konfigurieren und anschließend eine einfache Demo-Anwendung auf einem qualifizierten Mikrocontroller-Board kompilieren und ausführen.

In diesem Tutorial gehen wir davon aus, dass Sie mit der Konsole vertraut sind. AWS IoT AWS IoT Wenn dies nicht der Fall ist, sollten Sie das Tutorial Erste Schritte mit AWS IoT durcharbeiten, bevor Sie fortfahren.

Themen:

- FreeRTOS-Demo-Anwendung
- Erste Schritte
- Fehlerbehebung – Erste Schritte
- Verwendung CMake mit FreeRTOS
- Schlüsselbereitstellung im Entwicklermodus
- Board-spezifische Handbücher "Erste Schritte"
- Die nächsten Schritte mit FreeRTOS

FreeRTOS-Demo-Anwendung

> **Important**
>
> Diese Seite bezieht sich auf das Amazon-FreeRTOS-Repository, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Die Demo-Anwendung in diesem Tutorial ist die in der Datei definierte CoreMQTT Agent-Demo. *freertos*/demos/coreMQTT_Agent/mqtt_agent_task.c Es verwendet dieCoreMQTT-Bibliothek, um eine Verbindung zur AWS Cloud herzustellen und dann regelmäßig Nachrichten zu einem vom MQTT-Broker gehosteten MQTT-Thema zu veröffentlichen.AWS IoT

Es kann jeweils nur eine einzige FreeRTOS-Demo-Anwendung ausgeführt werden. Wenn Sie ein FreeRTOS-Demo-Projekt erstellen, ist die erste in der *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h Header-Datei aktivierte

Demo die Anwendung, die ausgeführt wird. Für dieses Tutorial müssen Sie keine Demos aktivieren oder deaktivieren. Die CoreMQTT Agent-Demo ist standardmäßig aktiviert.

Weitere Informationen zu den in FreeRTOS enthaltenen Demo-Anwendungen finden Sie unter. FreeRTOS RTOS-Demos

Erste Schritte

> **Important**
>
> Diese Seite bezieht sich auf das Amazon-FreeRTOS-Repository, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Um mit der Nutzung von FreeRTOS zu beginnenAWS IoT, benötigen Sie ein AWS Konto, einen Benutzer mit Zugriffsberechtigungen AWS IoT und FreeRTOS-Cloud-Dienste. Sie müssen auch FreeRTOS herunterladen und das FreeRTOS-Demo-Projekt Ihres Boards so konfigurieren, dass es funktioniert. AWS IoT In den folgenden Abschnitten finden Sie schrittweise Anleitungen, um diese Anforderungen zu erfüllen.

> **Note**
>
> - Wenn du den Espressif ESP32 - DevKit C, oder den ESP32 -WROOM-32SE verwendest ESP-WROVER-KIT, überspringe diese Schritte und gehe zu. Erste Schritte mit dem Espressif ESP32 - DevKit C und dem ESP-WROVER-KIT
> - Wenn Sie den Nordic n RF5284 0-DK verwenden, überspringen Sie diese Schritte und fahren Sie mit. Erste Schritte mit dem Nordic n RF5284 0-DK

1. Richten Sie Ihr AWS Konto und Ihre Berechtigungen ein
2. Registrieren Sie Ihr MCU-Board bei AWS IoT
3. FreeRTOS wird heruntergeladen
4. Konfiguration der FreeRTOS-Demos

Richten Sie Ihr AWS Konto und Ihre Berechtigungen ein

Melde dich an für ein AWS-Konto

Wenn Sie noch keine habenAWS-Konto, führen Sie die folgenden Schritte aus, um eine zu erstellen.

Um sich für eine anzumelden AWS-Konto

1. Öffnen Sie https://portal.aws.amazon.com/billing/die Anmeldung.
2. Folgen Sie den Online-Anweisungen.

   Während der Anmeldung erhalten Sie einen Telefonanruf oder eine Textnachricht und müssen einen Verifizierungscode über die Telefontasten eingeben.

   Wenn Sie sich für eine anmeldenAWS-Konto, Root-Benutzer des AWS-Kontoswird eine erstellt. Der Root-Benutzer hat Zugriff auf alle AWS-Services und Ressourcen des Kontos. Als bewährte Sicherheitsmethode weisen Sie einem Benutzer Administratorzugriff zu und verwenden Sie nur den Root-Benutzer, um Aufgaben auszuführen, die Root-Benutzerzugriff erfordern.

AWSsendet Ihnen nach Abschluss des Anmeldevorgangs eine Bestätigungs-E-Mail. Du kannst jederzeit deine aktuellen Kontoaktivitäten einsehen und dein Konto verwalten, indem du zu https://aws.amazon.com/gehst und Mein Konto auswählst.

Erstellen eines Benutzers mit Administratorzugriff

Nachdem Sie sich für einen angemeldet habenAWS-Konto, sichern Sie Ihren Root-Benutzer des AWS-KontosAWS IAM Identity Center, aktivieren und erstellen Sie einen Administratorbenutzer, sodass Sie den Root-Benutzer nicht für alltägliche Aufgaben verwenden.

Sichern Sie Ihre Root-Benutzer des AWS-Kontos

1. Melden Sie sich AWS-Managementkonsoleals Kontoinhaber an, indem Sie Root-Benutzer auswählen und Ihre AWS-Konto E-Mail-Adresse eingeben. Geben Sie auf der nächsten Seite Ihr Passwort ein.

   Hilfe bei der Anmeldung mit dem Root-Benutzer finden Sie unter Anmelden als Root-Benutzer im AWS-Anmeldung Benutzerhandbuch zu.
2. Aktivieren Sie die Multi-Faktor-Authentifizierung (MFA) für den Root-Benutzer.

Anweisungen finden Sie unter Aktivieren eines virtuellen MFA-Geräts für Ihren AWS-Konto Root-Benutzer (Konsole) im IAM-Benutzerhandbuch.

Erstellen eines Benutzers mit Administratorzugriff

1. Aktivieren Sie das IAM Identity Center.

   Anweisungen finden Sie unter Aktivieren AWS IAM Identity Center im AWS IAM Identity Center Benutzerhandbuch.
2. Gewähren Sie einem Administratorbenutzer im IAM Identity Center Benutzerzugriff.

   Ein Tutorial zur Verwendung von IAM-Identity-Center-Verzeichnis als Identitätsquelle finden Sie IAM-Identity-Center-Verzeichnis im Benutzerhandbuch unter Benutzerzugriff mit der Standardeinstellung konfigurieren. AWS IAM Identity Center

Anmelden als Administratorbenutzer

- Um sich mit Ihrem IAM-Identity-Center-Benutzer anzumelden, verwenden Sie die Anmelde-URL, die an Ihre E-Mail-Adresse gesendet wurde, als Sie den IAM-Identity-Center-Benutzer erstellt haben.

  Hilfe bei der Anmeldung mit einem IAM Identity Center-Benutzer finden Sie im AWS-AnmeldungBenutzerhandbuch unter Anmeldung beim AWS Access-Portal.

Weiteren Benutzern Zugriff zuweisen

1. Erstellen Sie im IAM-Identity-Center einen Berechtigungssatz, der den bewährten Vorgehensweisen für die Anwendung von geringsten Berechtigungen folgt.

   Anweisungen hierzu finden Sie unter Berechtigungssatz erstellen im AWS IAM Identity Center Benutzerhandbuch.
2. Weisen Sie Benutzer einer Gruppe zu und weisen Sie der Gruppe dann Single Sign-On-Zugriff zu.

   Eine genaue Anleitung finden Sie unter Gruppen hinzufügen im AWS IAM Identity Center Benutzerhandbuch.

Um Zugriff zu gewähren, fügen Sie Ihren Benutzern, Gruppen oder Rollen Berechtigungen hinzu:

- Benutzer und Gruppen inAWS IAM Identity Center:

  Erstellen Sie einen Berechtigungssatz. Befolgen Sie die Anweisungen unter Erstellen eines Berechtigungssatzes im AWS IAM Identity Center-Benutzerhandbuch.

- Benutzer, die in IAM über einen Identitätsanbieter verwaltet werden:

  Erstellen Sie eine Rolle für den Identitätsverbund. Befolgen Sie die Anleitung unter Eine Rolle für einen externen Identitätsanbieter (Verbund) erstellen im IAM-Benutzerhandbuch.

- IAM-Benutzer:
  - Erstellen Sie eine Rolle, die Ihr Benutzer annehmen kann. Befolgen Sie die Anleitung unter Eine Rolle für einen IAM-Benutzer erstellen im IAM-Benutzerhandbuch.
  - (Nicht empfohlen) Weisen Sie einem Benutzer eine Richtlinie direkt zu oder fügen Sie einen Benutzer zu einer Benutzergruppe hinzu. Befolgen Sie die Anweisungen unter Hinzufügen von Berechtigungen zu einem Benutzer (Konsole) im IAM-Benutzerhandbuch.

Registrieren Sie Ihr MCU-Board bei AWS IoT

Ihr Board muss registriert sein, AWS IoT um mit der AWS Cloud kommunizieren zu können. Um dein Board zu registrierenAWS IoT, benötigst du:

Eine AWS IoT Richtlinie

Die AWS IoT Richtlinie gewährt Ihrem Gerät Berechtigungen für den Zugriff auf AWS IoT Ressourcen. Es wird in der AWS Cloud gespeichert.

Irgendein AWS IoT Ding

Jedes AWS IoT Ding ermöglicht es Ihnen, Ihre Geräte in zu verwaltenAWS IoT. Es ist in der AWS Cloud gespeichert.

Einen privaten Schlüssel und ein X.509-Zertifikat

Mit dem privaten Schlüssel und dem Zertifikat kann sich Ihr Gerät authentifizieren. AWS IoT

Gehen Sie wie folgt vor, um Ihr Board zu registrieren.

Um eine AWS IoT Richtlinie zu erstellen

1. Um eine IAM-Richtlinie zu erstellen, müssen Sie Ihre AWS Region und AWS Kontonummer kennen.

   Um Ihre AWS Kontonummer zu finden, öffnen Sie die AWSManagement Console, suchen und erweitern Sie das Menü unter Ihrem Kontonamen in der oberen rechten Ecke und wählen Sie Mein Konto aus. Ihre Konto-ID wird unter Kontoeinstellungen angezeigt.

   Um die AWS Region für Ihr AWS Konto zu finden, verwenden Sie die. AWS Command Line Interface Folgen Sie den Anweisungen im AWS Command Line InterfaceBenutzerhandbuch AWS CLI, um das zu installieren. Öffnen Sie nach der AWS CLI Installation von ein Befehlszeilenfenster und geben Sie den folgenden Befehl ein:

   ```
   aws iot describe-endpoint --endpoint-type=iot:Data-ATS
   ```

   Die Ausgabe sollte in etwa wie folgt aussehen:

   ```
   {
       "endpointAddress": "xxxxxxxxxxxxxx-ats.iot.us-west-2.amazonaws.com"
   }
   ```

   In diesem Beispiel ist die Region `us-west-2`.

   > **Note**
   >
   > Wir empfehlen die Verwendung von ATS-Endpunkten, wie im Beispiel gezeigt.

2. Navigieren Sie zur AWS IoT-Konsole.
3. Wählen Sie im Navigationsbereich erst Sicher, dann Richtlinien und anschließend Erstellen aus.
4. Geben Sie einen Namen zur Identifizierung Ihrer Richtlinie ein.
5. Wählen Sie im Abschnitt Anweisungen hinzufügen die Option Erweiterter Modus aus. Kopieren Sie die folgende JSON und fügen Sie sie in das Fenster des Richtlinien-Editors ein. Ersetzen Sie *`aws-region`* und *`aws-account`* durch Ihre AWS Region und Konto-ID.

   JSON

   ```
   {
   ```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:Connect",
            "Resource": "arn:aws:iot:us-east-1:123456789012:*"
        },
        {
            "Effect": "Allow",
            "Action": "iot:Publish",
            "Resource": "arn:aws:iot:us-east-1:123456789012:*"
        },
        {
            "Effect": "Allow",
            "Action": "iot:Subscribe",
            "Resource": "arn:aws:iot:us-east-1:123456789012:*"
        },
        {
            "Effect": "Allow",
            "Action": "iot:Receive",
            "Resource": "arn:aws:iot:us-east-1:123456789012:*"
        }
    ]
}
```

Durch diese Richtlinie werden die folgenden Berechtigungen gewährt:

`iot:Connect`

Erteilt Ihrem Gerät die Erlaubnis, mit einer beliebigen Client-ID eine Verbindung zum AWS IoT Message Broker herzustellen.

`iot:Publish`

Erteilt Ihrem Gerät die Berechtigung für die Veröffentlichung einer MQTT-Nachricht zu jedem MQTT-Thema.

`iot:Subscribe`

Erteilt Ihrem Gerät die Berechtigung für die Abonnierung eines MQTT-Themenfilters.

`iot:Receive`

Erteilt Ihrem Gerät die Berechtigung zum Erhalt von Mitteilungen aus dem AWS IoT-Message Broker zu jedem MQTT-Thema.

6. Wählen Sie Erstellen aus.

So erstellen Sie ein IoT-Objekt, einen privaten Schlüssel und ein Zertifikat für Ihr Gerät:

1. Navigieren Sie zur AWS IoT-Konsole.
2. Wählen Sie im Navigationsbereich Verwalten und dann Objekte aus.
3. Wenn keine IoT-Objekte in Ihrem Konto registriert sind, wird die Seite Sie haben noch keine Objekte angezeigt. Wenn Sie diese Seite sehen, wählen Sie Ein Objekt registrieren aus. Wählen Sie andernfalls Erstellen.
4. Wählen Sie auf der Seite „AWS IoTDinge erstellen“ die Option Eine einzelne Sache erstellen aus.
5. Geben Sie auf der Seite Fügen Sie Ihr Gerät zur Objektregistrierung hinzu einen Namen für Ihr Objekt ein und klicken Sie dann auf Weiter.
6. Wählen Sie auf der Seite Fügen Sie ein Zertifikat für Ihr Objekt hinzu unter Zertifikat mit einem Klick erstellen die Option Zertifikat erstellen aus.
7. Laden Sie Ihren privaten Schlüssel und das Zertifikat herunter, indem Sie die Links zum Herunterladen für jeden Vorgang auswählen.
8. Wählen Sie Aktivieren aus, um Ihr Zertifikat zu aktivieren. Die Zertifikate müssen aktiviert werden, bevor Sie sie verwenden können.
9. Wählen Sie Richtlinie anhängen, um Ihrem Zertifikat eine Richtlinie anzuhängen, die Ihrem Gerät Zugriff auf AWS IoT Vorgänge gewährt.
10. Wählen Sie die soeben erstellte Richtlinie und dann Objekt registrieren aus.

Nachdem dein Board registriert wurdeAWS IoT, kannst du damit fortfahrenFreeRTOS wird heruntergeladen.

FreeRTOS wird heruntergeladen

Sie können FreeRTOS aus dem FreeRTOS-Repository herunterladen. GitHub

Nachdem Sie FreeRTOS heruntergeladen haben, können Sie fortfahren. Konfiguration der FreeRTOS-Demos

Konfiguration der FreeRTOS-Demos

Sie müssen einige Konfigurationsdateien in Ihrem FreeRTOS-Verzeichnis bearbeiten, bevor Sie Demos auf Ihrem Board kompilieren und ausführen können.

Um deinen Endpunkt zu konfigurieren AWS IoT

Sie müssen FreeRTOS Ihren AWS IoT Endpunkt angeben, damit die Anwendung, die auf Ihrem Board läuft, Anfragen an den richtigen Endpunkt senden kann.

1. Navigieren Sie zur AWS IoT-Konsole.
2. Wählen Sie im linken Navigationsbereich die Option Einstellungen aus.

   Ihr AWS IoT Endpunkt wird unter Gerätedatenendpunkt angezeigt. Sie sollte wie folgt aussehen: *1234567890123*`-ats.iot.`*us-east-1*`.amazonaws.com`. Notieren Sie sich diesen Endpunkt.
3. Wählen Sie im Navigationsbereich Verwalten und dann Objekte aus.

   Ihr Gerät sollte einen AWS IoT Namen haben. Notieren Sie sich diesen Namen.
4. Öffnen Sie `demos/include/aws_clientcredential.h`.
5. Geben Sie für die folgenden -Konstanten Werte an:

   - `#define clientcredentialMQTT_BROKER_ENDPOINT "`*YourAWS IoTendpoint*`";`
   - `#define clientcredentialIOT_THING_NAME "`*TheAWS IoTthing name of your board*`"`

So konfigurieren Sie Ihr WLAN

Wenn Ihr Board über eine Wi-Fi-Verbindung eine Verbindung zum Internet herstellt, müssen Sie FreeRTOS Wi-Fi-Anmeldeinformationen zur Verfügung stellen, um eine Verbindung zum Netzwerk herzustellen. Wenn Ihr Board WLAN nicht unterstützt, können Sie diese Schritte überspringen.

1. `demos/include/aws_clientcredential.h`.
2. Geben Sie für die folgenden `#define`-Konstanten Werte an:

   - `#define clientcredentialWIFI_SSID "`*The SSID for your Wi-Fi network*`"`
   - `#define clientcredentialWIFI_PASSWORD "`*The password for your Wi-Fi network*`"`

- `#define clientcredentialWIFI_SECURITY` *`The security type of your Wi-Fi network`*

  Gültige Sicherheitstypen sind:

  - `eWiFiSecurityOpen` (Open, no security (Offen, keine Sicherheit)
  - `eWiFiSecurityWEP` (WEP-Sicherheit)
  - `eWiFiSecurityWPA` (WPA-Sicherheit)
  - `eWiFiSecurityWPA2`(Sicherheit) WPA2

Um Ihre AWS IoT Anmeldedaten zu formatieren

FreeRTOS muss über das AWS IoT Zertifikat und die privaten Schlüssel verfügen, die mit Ihrem registrierten Ding verknüpft sind, sowie über dessen Berechtigungsrichtlinien, um erfolgreich im Namen Ihres Geräts mit AWS IoT FreeRTOS kommunizieren zu können.

> **Note**
>
> Um Ihre AWS IoT Anmeldeinformationen zu konfigurieren, benötigen Sie den privaten Schlüssel und das Zertifikat, die Sie bei der Registrierung Ihres Geräts von der AWS IoT Konsole heruntergeladen haben. Nachdem Sie Ihr Gerät als Objekt registriert habenAWS IoT, können Sie Gerätezertifikate von der AWS IoT Konsole abrufen, aber Sie können keine privaten Schlüssel abrufen.

FreeRTOS ist ein Projekt in C-Sprache, und das Zertifikat und der private Schlüssel müssen speziell formatiert sein, um dem Projekt hinzugefügt zu werden.

1. Öffnen Sie `tools/certificate_configuration/CertificateConfigurator.html` im Browserfenster.
2. Wählen Sie unter Certificate PEM file (PEM-Datei für Zertifikat) die Option *`ID`*`-certificate.pem.crt` aus, die Sie von der AWS IoT-Konsole heruntergeladen haben.
3. Wählen Sie unter Private Key PEM file (PEM-Datei für privaten Schlüssel) die Option *`ID`*`-private.pem.key` aus, die Sie von der AWS IoT-Konsole heruntergeladen haben.
4. Wählen Sie Generate and save aws_clientcredential_keys.h (aws_clientcredential_keys.h generieren und speichern) aus und speichern Sie die Datei in `demos/include`. Diese Einstellung überschreibt die vorhandene Datei im Verzeichnis.

> **Note**
>
> Das Zertifikat und der private Schlüssel sind nur für Demonstrationszwecke hardcodiert. Anwendungen auf Produktionsebene sollten diese Dateien an einem sicheren Ort speichern.

Nachdem Sie FreeRTOS konfiguriert haben, können Sie mit dem Handbuch Erste Schritte für Ihr Board fortfahren, um die Hardware und die Softwareentwicklungsumgebung Ihrer Plattform einzurichten, und dann die Demo kompilieren und auf Ihrem Board ausführen. Board-spezifische Anweisungen finden Sie unter Board-spezifische Handbücher "Erste Schritte". Die Demo-Anwendung, die im Getting Started-Tutorial verwendet wird, ist die CoreMQTT Mutual Authentication-Demo, die sich unter befindet. `demos/coreMQTT/mqtt_demo_mutual_auth.c`

Fehlerbehebung – Erste Schritte

> **Important**
>
> Diese Seite bezieht sich auf das Amazon-FreeRTOS-Repository, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Die folgenden Themen können Ihnen helfen, Probleme zu beheben, die bei den ersten Schritten mit FreeRTOS auftreten:

Themen

- Allgemeine Tipps zur Fehlerbehebung bei den ersten Schritten
- Installieren eines Terminal-Emulators

Board-spezifische Tipps zur Fehlerbehebung finden Sie im Handbuch Erste Schritte mit FreeRTOS für Ihr Board.

Allgemeine Tipps zur Fehlerbehebung bei den ersten Schritten

Nachdem ich das Hello World-Demo-Projekt ausgeführt habe, werden in der AWS IoT Konsole keine Meldungen angezeigt. Was soll ich tun?

Gehen Sie wie folgt vor:

1. Öffnen Sie ein Terminal-Fenster, um die Protokollierungausgabe des Beispiels anzusehen. Dies kann bei der Fehlerbehebung hilfreich sein.
2. Stellen Sie sicher, dass Ihre Netzwerk-Anmeldeinformationen gültig sind.

Die bei der Ausführung einer Demo in meinem Terminal angezeigten Protokolle sind gekürzt. Wie kann ich ihre Länge erhöhen?

Erhöhen Sie den Wert von `configLOGGING_MAX_MESSAGE_LENGTH` 255 in der `FreeRTOSconfig.h` Datei für die Demo, die Sie gerade ausführen:

```
#define configLOGGING_MAX_MESSAGE_LENGTH    255
```

Installieren eines Terminal-Emulators

Ein Terminal-Emulator kann Ihnen dabei helfen, Probleme zu diagnostizieren oder zu überprüfen, ob Ihr Gerätecode korrekt ausgeführt wird. Es sind eine Vielzahl von Terminal-Emulatoren für Windows, macOS und Linux verfügbar.

Sie müssen Ihr Board mit Ihrem Computer verbinden, bevor Sie versuchen, mit einem Terminal-Emulator eine serielle Verbindung zu Ihrem Board herzustellen.

Verwenden Sie die folgenden Einstellungen, um Ihren Terminal-Emulator zu konfigurieren:

Terminal-Einstellung	Wert
BAUDRATE	115200
Daten	8 Bit
Parität	Keine
Stopp	1 Bit

Terminal-Einstellung	Wert
Flusssteuerung	Keine

Suche des seriellen Ports Ihres Boards

Wenn Sie den seriellen Port Ihres Boards nicht kennen, können Sie einen der folgenden Befehle über die Befehlszeile oder das Terminal ausführen, um die seriellen Ports für alle mit Ihrem Host-Computer verbundenen Geräte auszugeben:

Windows

```
chgport
```

Linux

```
ls /dev/tty*
```

macOS

```
ls /dev/cu.*
```

Verwendung CMake mit FreeRTOS

> **Important**
>
> Diese Seite bezieht sich auf das Amazon-FreeRTOS-Repository, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Sie können CMake es verwenden, um Projekt-Build-Dateien aus dem Quellcode der FreeRTOS-Anwendung zu generieren und den Quellcode zu erstellen und auszuführen.

Sie können eine IDE auch verwenden, um Code auf FreeRTOS-qualifizierten Geräten zu bearbeiten, zu debuggen, zu kompilieren, zu flashen und auszuführen. Jeder boardspezifische Erste-Schritte-

Leitfaden enthält Anweisungen zur Einrichtung der IDE für eine bestimmte Plattform. Wenn Sie lieber ohne IDE arbeiten möchten, können Sie andere Codebearbeitungs- und Debugging-Tools von Drittanbietern für die Entwicklung und das Debuggen Ihres Codes verwenden und diese dann zum Erstellen und Ausführen der Anwendungen verwenden CMake .

Die folgenden Boards unterstützen: CMake

- Espressif - C ESP32 DevKit
- Espressif ESP-WROVER-KIT
- Infineon XMC4800 IoT-Konnektivitätskit
- Marvell Einsteigerpaket MW320 AWS IoT
- Marvell Starterpaket MW322 AWS IoT
- Microchip Curiosity MZEF-Paket PIC32
- Nordic n DK Entwicklungskit RF52840
- STMicroelectronicsSTM32L4 Discovery Kit IoT-Knoten
- Texas Instruments CC3220 SF-LAUNCHXL
- Microsoft Windows Simulator

In den folgenden Themen finden Sie weitere Informationen zur Verwendung CMake mit FreeRTOS.

Themen

- Voraussetzungen
- Entwicklung von FreeRTOS-Anwendungen mit Code-Editoren und Debugging-Tools von Drittanbietern
- FreeRTOS erstellen mit CMake

Voraussetzungen

Stellen Sie sicher, dass Ihr Host-Computer die folgenden Voraussetzungen erfüllt, bevor Sie fortfahren:

- Die Kompilierungstoolchain Ihres Geräts muss das Betriebssystem des Geräts unterstützen. CMake unterstützt alle Versionen von Windows, macOS und Linux

  Das Windows Subsystem für Linux (WSL) wird nicht unterstützt. Verwenden Sie Native CMake auf Windows-Computern.

- Sie müssen CMake Version 3.13 oder höher installiert haben.

  Sie können die Binärdistribution von CMake von CMakevon.org herunterladen.

  > **Note**
  >
  > Wenn Sie die Binärdistribution von herunterladen CMake, stellen Sie sicher, dass Sie die CMake ausführbare Datei der Umgebungsvariablen PATH hinzufügen, bevor Sie sie von der Befehlszeile CMake aus verwenden.

  Du kannst auch CMake mit einem Paketmanager wie Homebrew auf macOS und Scoop oder Chocolatey auf Windows herunterladen und installieren.

  > **Note**
  >
  > Die in den CMake Paketmanagern für viele Linux-Distributionen bereitgestellten Paketversionen sind. out-of-date Falls der Paketmanager Ihrer Distribution nicht die neueste Version von bereitstellt CMake, können Sie alternative Paketmanager wie `linuxbrew` oder `nix` ausprobieren.

- Sie müssen über ein kompatibles natives Build-System verfügen.

  CMake kann auf viele native Bausysteme abzielen, einschließlich GNU Make oder Ninja. Sowohl Make als auch Ninja können mit Paketmanagern unter Linux, MacOS und Windows installiert werden. Wenn Sie Make unter Windows verwenden, können Sie eine eigenständige Version von Equation installieren, oder Sie können MinGW installieren, das Make bündelt.

  > **Note**
  >
  > Das ausführbare Make in MinGW heißt `mingw32-make.exe` statt `make.exe`.

  Wir empfehlen Ihnen, Ninja zu verwenden, da es schneller ist als Make und auch nativen Support für alle Desktop-Betriebssysteme bietet.

Entwicklung von FreeRTOS-Anwendungen mit Code-Editoren und Debugging-Tools von Drittanbietern

Sie können einen Code-Editor und eine Debugging-Erweiterung oder ein Debugging-Tool eines Drittanbieters verwenden, um Anwendungen für FreeRTOS zu entwickeln.

Wenn Sie beispielsweise Visual Studio Code als Code-Editor verwenden, können Sie die Cortex-Debug VS Code-Erweiterung als Debugger installieren. Wenn Sie mit der Entwicklung Ihrer Anwendung fertig sind, können Sie das CMake Befehlszeilentool aufrufen, um Ihr Projekt in VS Code zu erstellen. Weitere Hinweise zur Verwendung CMake zum Erstellen von FreeRTOS-Anwendungen finden Sie unter. FreeRTOS erstellen mit CMake

Für das Debugging können Sie einen VS-Code mit einer Debug-Konfiguration ähnlich der folgenden bereitstellen:

```
"configurations": [
    {
        "name": "Cortex Debug",
        "cwd": "${workspaceRoot}",
        "executable": "./build/st/stm32l475_discovery/aws_demos.elf",
        "request": "launch",
        "type": "cortex-debug",
        "servertype": "stutil"
    }
]
```

FreeRTOS erstellen mit CMake

CMake zielt standardmäßig auf Ihr Host-Betriebssystem als Zielsystem ab. Um es für Cross-Compiling zu verwenden, CMake ist eine Toolchain-Datei erforderlich, die den Compiler angibt, den Sie verwenden möchten. In FreeRTOS stellen wir Standard-Toolchain-Dateien bereit. *freertos*`/tools/cmake/toolchains` Die Art und Weise, wie Sie diese Datei bereitstellen, CMake hängt davon ab, ob Sie die CMake Befehlszeilenschnittstelle oder die GUI verwenden. Befolgen Sie die folgenden Generieren von Build-Dateien (CMake Befehlszeilentool)-Anweisungen, um weitere Informationen zu erhalten. Weitere Informationen zum Cross-Compilieren finden Sie CrossCompilingim offiziellen CMake Wiki. CMake

Um ein basiertes Projekt zu erstellen CMake

1. Führen Sie aus CMake , um die Build-Dateien für ein natives Build-System wie Make oder Ninja zu generieren.

Sie können entweder das CMake Befehlszeilentool oder die CMake GUI verwenden, um die Build-Dateien für Ihr natives Build-System zu generieren.

Hinweise zum Generieren von FreeRTOS-Build-Dateien finden Sie unter Generieren von Build-Dateien (CMake Befehlszeilentool) und. Generierung von Build-Dateien (GUI) CMake

2. Rufen Sie das native Build-System auf, um das Projekt in eine ausführbare Datei zu verwandeln.

   Hinweise zur Erstellung von FreeRTOS-Build-Dateien finden Sie unter. FreeRTOS aus generierten Build-Dateien erstellen

Generieren von Build-Dateien (CMake Befehlszeilentool)

Sie können das CMake Befehlszeilentool (cmake) verwenden, um Build-Dateien für FreeRTOS zu generieren. Um die Build-Dateien zu generieren, müssen Sie ein Ziel-Board, einen Compiler und den Speicherort des Quellcodes und des Build-Verzeichnisses angeben.

Sie können die folgenden Optionen für CMake verwenden:

- `-DVENDOR`— Spezifiziert das Zielboard.
- `-DCOMPILER`— Spezifiziert den Compiler.
- `-S`— Gibt den Speicherort des Quellcodes an.
- `-B`— Gibt den Speicherort der generierten Build-Dateien an.

> **Note**
>
> Der Compiler muss sich in der PATH-Variablen des Systems befinden, oder Sie müssen den Speicherort des Compilers angeben.

Wenn der Hersteller beispielsweise Texas Instruments ist und das Board das CC3220 Launchpad ist und der Compiler GCC for ARM ist, können Sie den folgenden Befehl ausführen, um die Quelldateien aus dem aktuellen Verzeichnis in ein Verzeichnis mit dem Namen zu kompilieren: *build-directory*

```
cmake -DVENDOR=ti -DBOARD=cc3220_launchpad -DCOMPILER=arm-ti -S . -B build-directory
```

> **Note**
>
> Wenn Sie Windows verwenden, müssen Sie das native Buildsystem angeben, da standardmäßig Visual Studio CMake verwendet wird. Beispiel:
>
> ```
> cmake -DVENDOR=ti -DBOARD=cc3220_launchpad -DCOMPILER=arm-ti -S . -B build-directory -G Ninja
> ```
>
> Oder:
>
> ```
> cmake -DVENDOR=ti -DBOARD=cc3220_launchpad -DCOMPILER=arm-ti -S . -B build-directory -G "MinGW Makefiles"
> ```

Die regulären Ausdrücke `${VENDOR}.*` und `${BOARD}.*` werden verwendet, um nach einem passenden Board zu suchen, so dass Sie für die Optionen `VENDOR` und `BOARD` nicht den vollständigen Namen des Anbieters und des Boards verwenden müssen. Teilnamen funktionieren, vorausgesetzt, es gibt eine einzige Übereinstimmung. Die folgenden Befehle erzeugen beispielsweise die gleichen Build-Dateien aus derselben Quelle:

```
cmake -DVENDOR=ti -DCOMPILER=arm-ti -S . -B build-directory
```

```
cmake -DBOARD=cc3220 -DCOMPILER=arm-ti -S . -B build-directory
```

```
cmake -DVENDOR=t -DBOARD=cc -DCOMPILER=arm-ti -S . -B build-directory
```

Sie können die Option `CMAKE_TOOLCHAIN_FILE` verwenden, wenn Sie eine Toolchain-Datei verwenden möchten, die sich nicht im Standardverzeichnis `cmake/toolchains` befindet. Beispiel:

```
cmake -DBOARD=cc3220 -DCMAKE_TOOLCHAIN_FILE='/path/to/toolchain_file.cmake' -S . -B build-directory
```

Wenn die Toolchain-Datei keine absoluten Pfade für Ihren Compiler verwendet und Sie Ihren Compiler nicht zur `PATH` Umgebungsvariablen hinzugefügt haben, können Sie CMake ihn möglicherweise nicht finden. Um sicherzustellen, dass Ihre CMake Toolchain-Datei gefunden wird, können Sie die Option verwenden. `AFR_TOOLCHAIN_PATH` Diese Option durchsucht den angegebenen Toolchain-Verzeichnispfad und den Unterordner der Toolchain unter `bin`. Beispiel:

```
cmake -DBOARD=cc3220 -DCMAKE_TOOLCHAIN_FILE='/path/to/toolchain_file.cmake' -
DAFR_TOOLCHAIN_PATH='/path/to/toolchain/' -S . -B build-directory
```

Um das Debugging zu aktivieren, setzen Sie `CMAKE_BUILD_TYPE` auf `debug`. Wenn diese Option aktiviert ist, CMake fügt sie Debug-Flags zu den Kompilierungsoptionen hinzu und erstellt FreeRTOS mit Debug-Symbolen.

```
# Build with debug symbols
cmake -DBOARD=cc3220 -DCOMPILER=arm-ti -DCMAKE_BUILD_TYPE=debug -S . -B build-directory
```

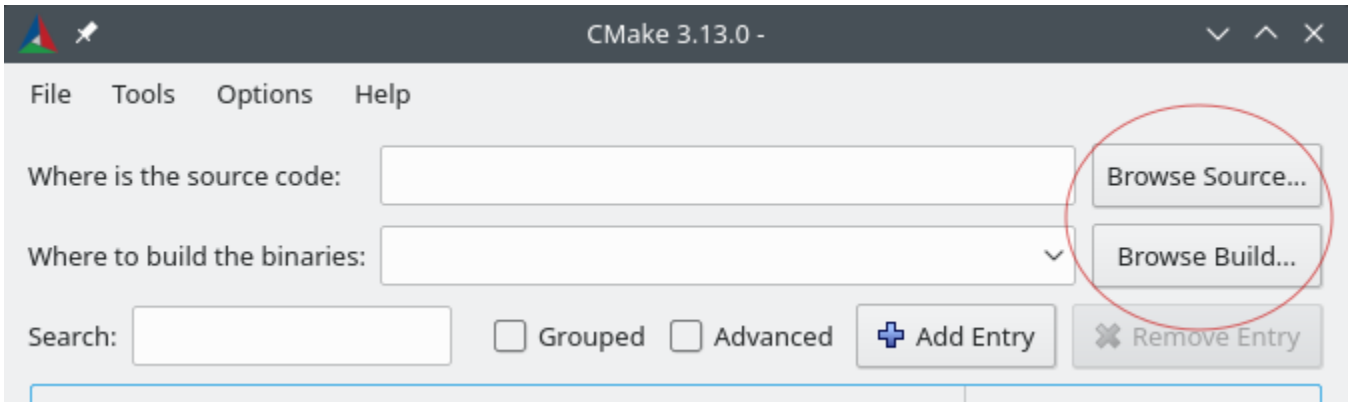
Sie können auch das `CMAKE_BUILD_TYPE` auf `release` setzen, um den Kompilierungsoptionen Optimierungsflags hinzuzufügen.

Generierung von Build-Dateien (GUI) CMake

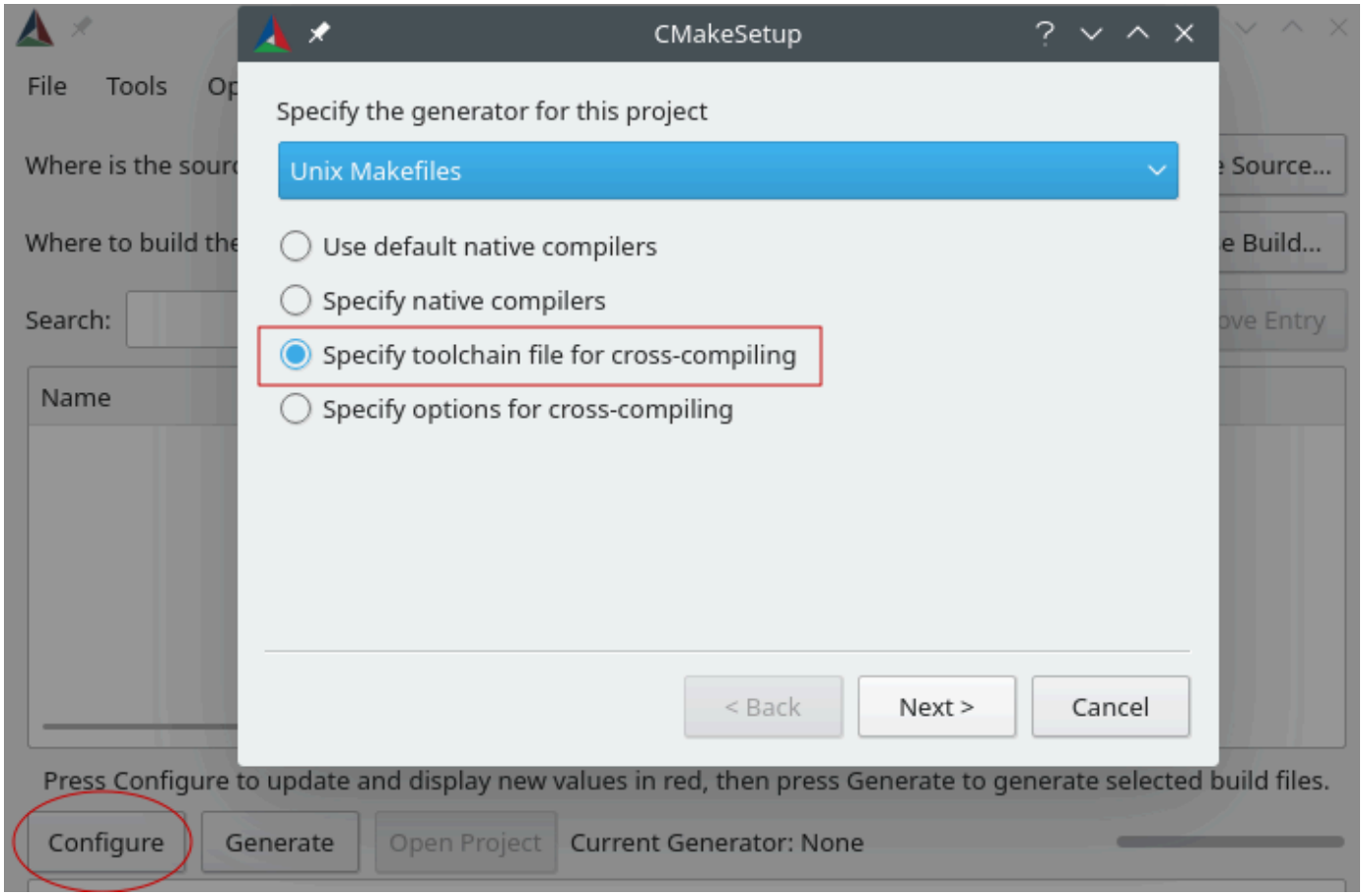
Sie können die CMake GUI verwenden, um FreeRTOS-Build-Dateien zu generieren.

Um Build-Dateien mit der GUI zu generieren CMake

1. Geben Sie in der Befehlszeile `cmake-gui` ein, um die GUI zu starten.
2. Wählen Sie Browse Source (Quelle durchsuchen) und geben Sie die Quelleingabe an und wählen Sie dann Browse Build (Abbild durchsuchen) und geben Sie die Build-Ausgabe an.



3. Wählen Sie Configure (Konfigurieren) und suchen und wählen Sie unter Specify the build generator for this project (Geben Sie die Build-Generator für dieses Projekt an) das Build-System, das Sie verwenden möchten, um die Build-Dateien zu erzeugen. Wenn das Pop-up-Fenster nicht angezeigt wird, verwenden Sie möglicherweise ein vorhandenes Build-Verzeichnis. Löschen Sie in diesem Fall den CMake Cache, indem Sie im Menü Datei die Option Cache löschen wählen.



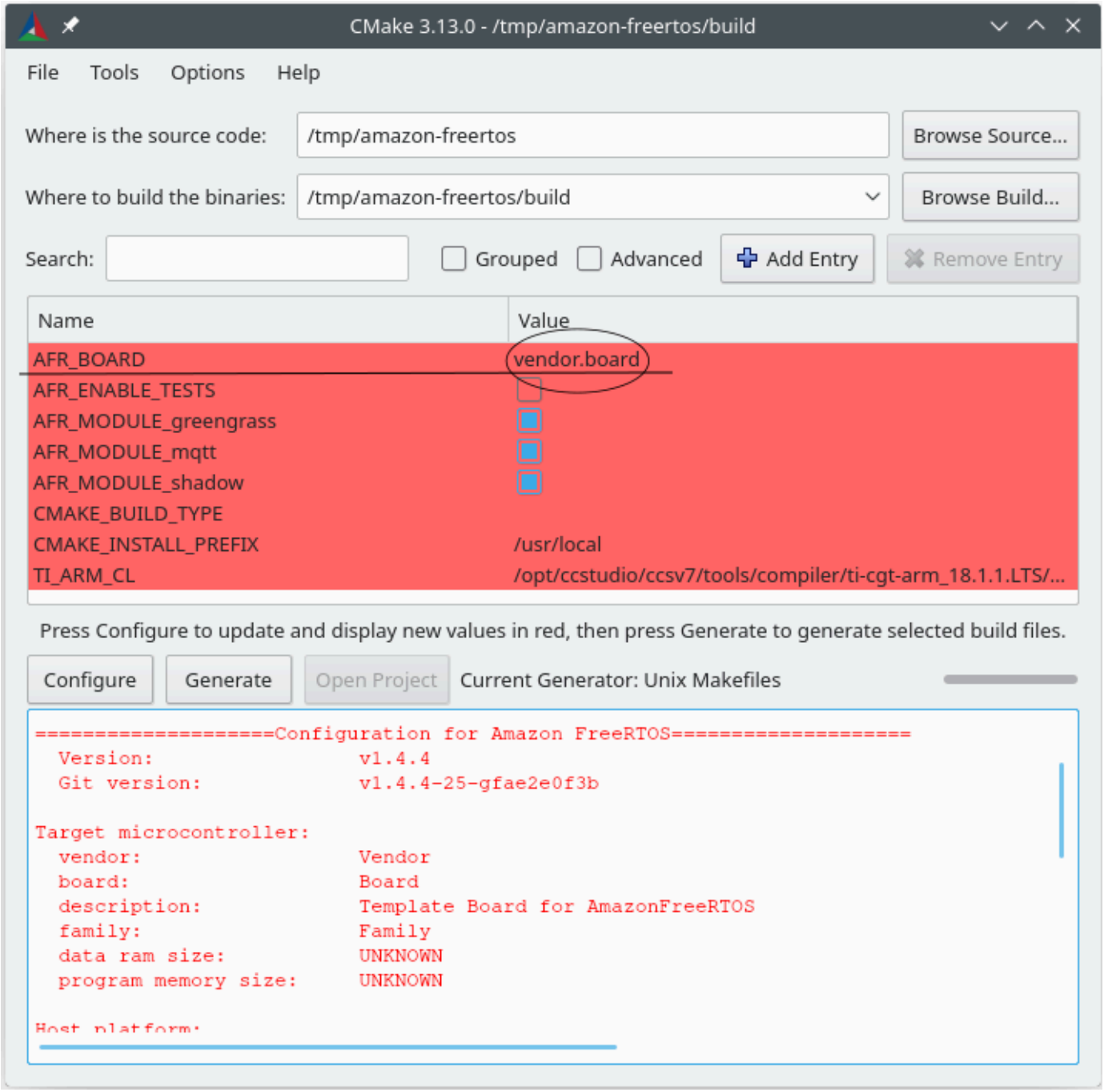
4. Wählen Sie Specify toolchain file for cross-compiling (Geben Sie die Toolchain-Datei für das Cross-Compiling an) aus und klicken Sie dann auf Next (Weiter).
5. Wählen Sie die Toolchain-Datei (z. B. „*freertos*/tools/cmake/toolchains/arm-ti.cmake“) und klicken Sie dann auf Finish (Fertigstellen).

   Die Standardkonfiguration für FreeRTOS ist das Template-Board, das keine portablen Layer-Ziele bietet. Dies hat zur Folge, dass ein Fenster mit der Meldung Error in configuration process angezeigt wird.

> **Note**
>
> Wenn die folgende Fehlermeldung angezeigt wird:
>
> ```
> CMake Error at tools/cmake/toolchains/find_compiler.cmake:23 (message):
> Compiler not found, you can specify search path with AFR_TOOLCHAIN_PATH.
> ```

Dies bedeutet, dass der Compiler sich nicht in Ihrer Umgebungsvariablen `PATH` befindet. Sie können die `AFR_TOOLCHAIN_PATH` Variable in der GUI so einstellen, dass sie angibt, CMake wo Sie Ihren Compiler installiert haben. Wenn die Variable `AFR_TOOLCHAIN_PATH` nicht angezeigt wird, wählen Sie Add Entry (Eintrag hinzufügen). Geben Sie im Pop-up-Fenster unter Name **`AFR_TOOLCHAIN_PATH`** ein. Geben Sie unter Compiler Path (Compiler-Pfad) den Pfad zu Ihrem Compiler ein, z. B.: `C:/toolchains/arm-none-eabi-gcc`.

6. Die GUI sollte jetzt wie folgt aussehen:



Wählen Sie AFR_BOARD, wählen Sie Ihr Board und dann erneut Configure (Konfigurieren).

7. Wählen Sie Generieren. CMake generiert die Build-Systemdateien (z. B. Makefiles oder Ninja-Dateien), und diese Dateien werden in dem Build-Verzeichnis angezeigt, das Sie im ersten Schritt angegeben haben. Befolgen Sie die Anweisungen im nächsten Abschnitt, um das Binärabbild zu generieren.

FreeRTOS aus generierten Build-Dateien erstellen

Build mit dem nativen Build System

Sie können FreeRTOS mit einem nativen Build-System erstellen, indem Sie den Befehl build system aus dem Ausgabe-Binärdateiverzeichnis aufrufen.

Wenn Ihr Ausgabeverzeichnis für die Build-Datei beispielsweise <build_dir> ist und Sie Make als natives Build-System verwenden, führen Sie die folgenden Befehle aus:

```
cd <build_dir>
make -j4
```

Bauen mit CMake

Sie können auch das CMake Befehlszeilentool verwenden, um FreeRTOS zu erstellen. CMake bietet eine Abstraktionsebene zum Aufrufen nativer Build-Systeme. Beispiel:

```
cmake --build build_dir
```

Im Folgenden sind einige andere gebräuchliche Verwendungszwecke des Build-Modus des CMake Befehlszeilentools aufgeführt:

```
# Take advantage of CPU cores.
cmake --build build_dir --parallel 8
```

```
# Build specific targets.
cmake --build build_dir --target afr_kernel
```

```
# Clean first, then build.
cmake --build build_dir --clean-first
```

Weitere Informationen zum CMake Build-Modus finden Sie in der CMake Dokumentation.

Schlüsselbereitstellung im Entwicklermodus

> **Important**
>
> Diese Seite bezieht sich auf das Amazon-FreeRTOS-Repository, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits

ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Einführung

In diesem Abschnitt werden zwei Optionen beschrieben, um ein vertrauenswürdiges X.509-Clientzertifikat auf ein IoT-Gerät für Labortests zu erhalten. Abhängig von den Funktionen des Geräts können verschiedene bereitstellungsbezogene Vorgänge unterstützt werden, einschließlich der integrierten ECDSA-Schlüsselgenerierung, Import privater Schlüssel und X.509-Zertifikatregistrierung. Darüber hinaus erfordern verschiedene Anwendungsfälle unterschiedliche Schutzstufen, angefangen von Onboard-Flash-Speicher bis hin zur Verwendung dedizierter Kryptohardware. Dieser Abschnitt enthält Logik für die Arbeit in den kryptografischen Funktionen Ihres Geräts.

Option #1: Import eines privaten Schlüssels von AWS IoT

Wenn Ihr Gerät den Import von privaten Schlüsseln zulässt, befolgen Sie die Anweisungen unter Konfiguration der FreeRTOS-Demos.

Option 2: Integrierte Generierung eines privaten Schlüssels

Wenn Ihr Gerät über ein sicheres Element verfügt oder Sie Ihr eigenes Geräte-Schlüsselpaar und Zertifikat erstellen möchten, folgen Sie den Anweisungen hier.

Erstkonfiguration

Führen Sie zunächst die Schritte unter ausKonfiguration der FreeRTOS-Demos, überspringen Sie jedoch den letzten Schritt (das heißt, tun Sie nicht, um Ihre AWS IoT Anmeldeinformationen zu formatieren). Das Nettoergebnis sollte sein, dass die Datei `demos/include/aws_clientcredential.h` mit Ihren Einstellungen aktualisiert wurde, jedoch nicht die Datei `demos/include/aws_clientcredential_keys.h`.

Demo-Projekt-Konfiguration

Öffnen Sie die CoreMQTT Mutual Authentication-Demo, wie in der Anleitung für Ihr Board unter beschrieben. Board-spezifische Handbücher "Erste Schritte" Öffnen Sie im Projekt die Datei `aws_dev_mode_key_provisioning.c` und ändern Sie die Definition von `keyprovisioningFORCE_GENERATE_NEW_KEY_PAIR`, die standardmäßig auf Null gesetzt ist, in eins:

```
#define keyprovisioningFORCE_GENERATE_NEW_KEY_PAIR 1
```

Erstellen Sie dann das Demo-Projekt und führen Sie es aus und fahren Sie mit dem nächsten Schritt fort.

Extraktion öffentlicher Schlüssel

Da das Gerät nicht mit einem privaten Schlüssel und einem Client-Zertifikat ausgestattet wurde, kann die Demoversion nicht authentifiziert werden. AWS IoT Die CoreMQTT Mutual Authentication-Demo beginnt jedoch mit der Schlüsselbereitstellung im Entwicklermodus, was zur Erstellung eines privaten Schlüssels führt, falls noch keiner vorhanden war. Sie sollten zu Beginn der seriellen Konsolenausgabe etwas wie das Folgende sehen.

```
7 910 [IP-task] Device public key, 91 bytes:
3059 3013 0607 2a86 48ce 3d02 0106 082a
8648 ce3d 0301 0703 4200 04cd 6569 ceb8
1bb9 1e72 339f e8cf 60ef 0f9f b473 33ac
6f19 1813 6999 3fa0 c293 5fae 08f1 1ad0
41b7 345c e746 1046 228e 5a5f d787 d571
dcb2 4e8d 75b3 2586 e2cc 0c
```

Kopieren Sie die sechs Zeilen von Schlüsselbytes in eine Datei namens `DevicePublicKeyAsciiHex.txt`. Verwenden Sie dann das Befehlszeilentool „xxd“, um die Hex-Bytes binär zu analysieren:

```
xxd -r -ps DevicePublicKeyAsciiHex.txt DevicePublicKeyDer.bin
```

Verwenden Sie „openssl“, um den öffentlichen Schlüssel des binär codierten (DER) Gerätes als PEM zu formatieren:

```
openssl ec -inform der -in DevicePublicKeyDer.bin -pubin -pubout -outform pem -out
 DevicePublicKey.pem
```

Vergessen Sie nicht, die oben aktivierte temporäre Schlüsselgenerierungseinstellung zu deaktivieren. Andernfalls erstellt das Gerät ein weiteres Schlüsselpaar, und Sie müssen die vorherigen Schritte wiederholen:

```
#define keyprovisioningFORCE_GENERATE_NEW_KEY_PAIR 0
```

Einrichtung der öffentlichen Schlüsselinfrastruktur

Folgen Sie den Anweisungen unter Registrieren des Zertifizierungsstellenzertifikats, um eine Zertifikathierarchie für das Gerätelaborzertifikat zu erstellen. Stoppen Sie, bevor Sie die im Abschnitt Erstellen eines Gerätezertifikats mit Ihrem Zertifizierungsstellenzertifikat beschriebene Sequenz ausführen.

In diesem Fall signiert das Gerät die Zertifikatsanforderung (d. h. die Certificate Service Request oder CSR) nicht, da die zum Erstellen und Signieren einer CSR erforderliche X.509-Kodierungslogik aus den FreeRTOS-Demoprojekten ausgeschlossen wurde, um die ROM-Größe zu reduzieren. Erstellen Sie stattdessen für Testzwecke einen privaten Schlüssel auf Ihrer Workstation, und verwenden Sie ihn, um die CSR zu signieren.

```
openssl genrsa -out tempCsrSigner.key 2048
openssl req -new -key tempCsrSigner.key -out deviceCert.csr
```

Sobald Ihre Zertifizierungsstelle erstellt und registriert wurde AWS IoT, verwenden Sie den folgenden Befehl, um ein Client-Zertifikat auszustellen, das auf der Geräte-CSR basiert, die im vorherigen Schritt signiert wurde:

```
openssl x509 -req -in deviceCert.csr -CA rootCA.pem -CAkey rootCA.key
 -CAcreateserial -out deviceCert.pem -days 500 -sha256 -force_pubkey
 DevicePublicKey.pem
```

Obwohl die CSR mit einem temporären privaten Schlüssel signiert wurde, kann das ausgestellte Zertifikat nur mit dem eigentlichen privaten Schlüssel des Geräts verwendet werden. Derselbe Mechanismus kann in der Produktion verwendet werden, wenn Sie den CSR-Signierschlüssel in separater Hardware speichern und Ihre Zertifizierungsstelle so konfigurieren, dass sie nur Zertifikate für Anforderungen ausstellt, die von diesem bestimmten Schlüssel signiert wurden. Dieser Schlüssel sollte auch unter der Kontrolle eines bestimmten Administrators bleiben.

Zertifikatsimport

Wenn das Zertifikat ausgestellt wurde, besteht der nächste Schritt darin, es in Ihr Gerät zu importieren. Sie müssen außerdem Ihr CA-Zertifikat (Certificate Authority) importieren, da es für die erfolgreiche Erstauthentifizierung bei der Verwendung von AWS IoT JITP erforderlich ist. Legen Sie in der Datei `aws_clientcredential_keys.h` in Ihrem Projekt das Makro `keyCLIENT_CERTIFICATE_PEM` auf den Inhalt von deviceCert.pem fest, und legen Sie das Makro `keyJITR_DEVICE_CERTIFICATE_AUTHORITY_PEM` auf den Inhalt von `rootCA.pem` fest.

Geräteautorisierung

Importieren Sie `deviceCert.pem` in die AWS IoT Registrierung, wie unter Verwenden Sie Ihr eigenes Zertifikat beschrieben. Sie müssen ein neues AWS IoT Ding erstellen, das PENDING-Zertifikat und eine Richtlinie an Ihr Ding anhängen und das Zertifikat dann als AKTIV markieren. All diese Schritte können manuell in der AWS IoT Konsole ausgeführt werden.

Sobald das neue Client-Zertifikat AKTIV ist und mit einer Sache und einer Richtlinie verknüpft ist, führen Sie die CoreMQTT Mutual Authentication-Demo erneut aus. Diesmal wird die Verbindung zum AWS IoT MQTT-Broker erfolgreich sein.

Board-spezifische Handbücher "Erste Schritte"

> **Important**
>
> Diese Seite bezieht sich auf das Amazon-FreeRTOS-Repository, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Nachdem du das abgeschlossen hastErste Schritte, findest du in der Anleitung deines Boards spezifische Anweisungen für die ersten Schritte mit FreeRTOS:

- Erste Schritte mit dem Cypress CYW943907 AEVAL1 F Development Kit
- Erste Schritte mit dem Cypress CYW954907 AEVAL1 F Development Kit
- Erste Schritte mit dem Cypress CY8 CKIT-064S0S2-4343W-Kit
- Erste Schritte mit dem Infineon XMC48 00 IoT Connectivity Kit
- Erste Schritte mit dem MW32x AWS IoT Starter Kit
- Erste Schritte mit dem MediaTek MT7697 Hx Development Kit
- Erste Schritte mit dem Microchip Curiosity PIC32 MZ EF
- Erste Schritte mit dem Nuvoton 487 NuMaker-IoT-M
- Erste Schritte mit dem NXP LPC54 018 IoT-Modul
- Erste Schritte mit dem Renesas Starter Kit+ für N-2MB RX65

- Erste Schritte mit dem STMicroelectronics STM32 L4 Discovery Kit IoT Node
- Erste Schritte mit dem Texas Instruments CC322 0SF-LAUNCHXL
- Erste Schritte mit dem Windows-Gerätesimulator
- Erste Schritte mit dem Xilinx MicroZed Avnet Industrial IoT Kit

> **Note**
>
> Sie müssen die Erste Schritte folgenden eigenständigen Anleitungen „Erste Schritte mit FreeRTOS“ nicht ausfüllen:
>
> - Erste Schritte mit dem Microchip ATECC608 A Secure Element with Windows-Simulator
> - Erste Schritte mit dem Espressif ESP32 - DevKit C und dem ESP-WROVER-KIT
> - Erste Schritte mit dem ESP32 Espressif -WROOM-32SE
> - Erste Schritte mit dem Espressif -S2 ESP32
> - Erste Schritte mit dem Infineon OPTIGA Trust X und XMC4800 dem IoT Connectivity Kit
> - Erste Schritte mit dem Nordic n RF5284 0-DK

Erste Schritte mit dem Cypress CYW943907 AEVAL1 F Development Kit

> **Important**
>
> Diese Referenzintegration wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Dieses Tutorial enthält Anweisungen für die ersten Schritte mit dem Cypress F Development Kit. CYW943907 AEVAL1 Wenn Sie das Cypress CYW943907 AEVAL1 F Development Kit nicht haben, besuchen Sie den Gerätekatalog für AWS Partner, um eines von unserem Partner zu erwerben.

> **Note**
>
> Dieses Tutorial führt Sie durch die Einrichtung und Ausführung der CoreMQTT Mutual Authentication Demo. Der FreeRTOS-Port für dieses Board unterstützt derzeit die TCP-Server- und Client-Demos nicht.

Bevor Sie beginnen, müssen Sie FreeRTOS konfigurieren AWS IoT und herunterladen, um Ihr Gerät mit der AWS Cloud zu verbinden. Detaillierte Anweisungen finden Sie unter Erste Schritte.

> **Important**
>
> - In diesem Thema wird der Pfad zum FreeRTOS-Download-Verzeichnis als bezeichnet. *`freertos`*
> - Leerzeichen im *`freertos`*-Pfad können Build-Fehler verursachen. Stellen Sie beim Klonen oder Kopieren des Repositorys sicher, dass der von Ihnen erstellte Pfad keine Leerzeichen enthält.
> - Die maximale Länge eines Dateipfades bei Microsoft Windows ist 260 Zeichen. Lange FreeRTOS-Download-Verzeichnispfade können zu Build-Fehlern führen.
> - Da der Quellcode symbolische Links enthalten kann, müssen Sie, wenn Sie Windows zum Extrahieren des Archivs verwenden, möglicherweise:
>   - Entwicklermodus aktivieren oder
>   - Verwenden Sie eine Konsole mit Administratorberechtigungen.
>
>   Auf diese Weise kann Windows beim Extrahieren des Archivs ordnungsgemäß symbolische Links erstellen. Andernfalls werden symbolische Links als normale Dateien geschrieben, die die Pfade der symbolischen Links als Text enthalten oder leer sind. Weitere Informationen finden Sie im Blogeintrag Symlinks in Windows 10! .
>
>   Wenn Sie Git unter Windows verwenden, müssen Sie den Entwicklermodus aktivieren oder:
>   - Setzen `core.symlinks` Sie ihn mit dem folgenden Befehl auf true:
>
>     ```
>     git config --global core.symlinks true
>     ```

- Verwenden Sie immer dann eine Konsole mit Administratorrechten, wenn Sie einen Git-Befehl verwenden, der in das System schreibt (z. B.git pull,git clone, undgit submodule update --init --recursive).
- Wie bereits erwähntFreeRTOS wird heruntergeladen, sind FreeRTOS-Ports für Cypress derzeit nur auf verfügbar. GitHub

-Übersicht

Dieses Tutorial enthält Anweisungen für die folgenden ersten Schritte:

1. Installieren von Software auf dem Host-Computer zum Entwickeln und Debuggen eingebetteter Anwendungen für Ihr Mikrocontroller-Board.
2. Cross-Compilierung einer FreeRTOS-Demo-Anwendung zu einem Binär-Image.
3. Laden des binären Anwendungs-Image auf Ihr Board und Ausführen der Anwendung.
4. Interaktion mit der Anwendung, die auf Ihrem Board über eine serielle Verbindung ausgeführt wird, zu Überwachungs- und Debuggingzwecken.

Einrichtung der -Entwicklungsumgebung

Herunterladen und Installieren des WICED Studio-SDKs

In diesem Handbuch Erste Schritte verwenden Sie das Cypress WICED Studio SDK, um Ihr Board mit der FreeRTOS-Demo zu programmieren. Besuchen Sie die Website WICED Software, um das WICED Studio-SDK von Cypress herunterzuladen. Sie müssen ein kostenloses Cypress-Konto erstellen, um die Software herunterzuladen. Das WICED Studio-SDK ist für Windows-, macOS- und Linux-Betriebssysteme geeignet.

Note

Bei einigen Betriebssystemen sind möglicherweise zusätzliche Installationsschritte erforderlich. Folgen Sie den Installationsanweisungen für das Betriebssystem und die Version von WICED Studio, die Sie installieren.

Festlegen von Umgebungsvariablen

Bevor Sie WICED Studio zur Programmierung Ihres Boards verwenden, müssen Sie eine Umgebungsvariable für das Installationsverzeichnis des WICED Studio-SDKs erstellen. Wenn WICED Studio während des Erstellens der Variablen ausgeführt wird, müssen Sie die Anwendung nach dem festlegen der variablen neu starten.

> **Note**
>
> Das WICED Studio-Installationsprogramm erstellt zwei separate Ordner mit dem Namen `WICED-Studio-`*`m`*`.`*`n`* auf Ihrem Computer, wobei `m` und `n` die Nummer der Haupt- bzw. der Nebenversion sind. Dieses Dokument setzt den Ordnernamen `WICED-Studio-6.2` voraus. Achten Sie jedoch darauf, dass Sie den korrekten Namen für die von Ihnen installierte Version verwenden. Wenn Sie die Umgebungsvariable `WICED_STUDIO_SDK_PATH` definieren, müssen Sie den vollständigen Installationspfad des WICED Studio-SDKs und nicht den Installationspfad der WICED Studio-IDE angeben. Unter Windows und macOS wird der Ordner `WICED-Studio-`*`m`*`.`*`n`* für das SDK standardmäßig im Ordner `Documents` erstellt.

So erstellen Sie Umgebungsvariablen unter Windows

1. Öffnen Sie die Control Panel (Systemsteuerung) und wählen Sie System und anschließend Advanced System Settings (Erweiterte Systemeinstellungen).
2. Wählen Sie auf der Registerkarte Erweitert die Option Umgebungsvariablen.
3. Wählen Sie unter User variables (Benutzervariablen) die Option New (Neu).
4. Geben Sie als Variablenname ein. **`WICED_STUDIO_SDK_PATH`** Geben Sie unter Variable value (Variablenwert) das Installationsverzeichnis des WICED Studio-SDKs ein.

So erstellen Sie eine Umgebungsvariable unter Linux oder macOS

1. Öffnen Sie die Datei `/etc/profile` auf Ihrem Computer und fügen Sie der letzten Zeile Ihrer Datei Folgendes hinzu:

   ```
   export WICED_STUDIO_SDK_PATH=installation-path/WICED-Studio-6.2
   ```

2. Starten Sie Ihren Computer neu.
3. Öffnen Sie ein Terminal und führen Sie die folgenden Befehle aus:

```
cd freertos/vendors/cypress/WICED_SDK
```

```
perl platform_adjust_make.pl
```

```
chmod +x make
```

Herstellen einer seriellen Verbindung

So stellen Sie eine serielle Verbindung zwischen Ihrem Host-Computer und Ihrem Board her

1. Verbinden Sie das Board mittels eines Standard-USB-Kabels mit A-Stecker auf Micro-B-Stecker mit Ihrem Host-Computer.
2. Identifizieren Sie die USB-Seriennummer für die Verbindung zum Board auf Ihrem Host-Computer.
3. Starten Sie ein serielles Terminal und öffnen Sie eine Verbindung mit den folgenden Einstellungen:
   - Baudrate: 115200
   - Daten: 8 Bit
   - Parität: Keine
   - Stop-Bits: 1
   - Flusssteuerung: Keine

Weitere Informationen zum Installieren eines Terminals und zum Einrichten einer seriellen Verbindung finden Sie unter Installieren eines Terminal-Emulators.

Überwachung von MQTT-Nachrichten in der Cloud

Bevor Sie das FreeRTOS-Demoprojekt ausführen, können Sie den MQTT-Client in der AWS IoT Konsole so einrichten, dass er die Nachrichten überwacht, die Ihr Gerät an die Cloud sendet. AWS

Um das MQTT-Thema mit dem MQTT-Client zu abonnieren AWS IoT

1. Melden Sie sich bei der AWS IoT -Konsole an.

2. Wählen Sie im Navigationsbereich Test und dann MQTT-Testclient, um den MQTT-Client zu öffnen.
3. Geben Sie im Feld Subscription topic (Abonnementthema) die Option ***your-thing-name*/example/topic** ein und wählen Sie dann Subscribe to topic (Thema abonnieren).

Erstellen und starten Sie das FreeRTOS-Demoprojekt

Nachdem Sie eine serielle Verbindung zu Ihrem Board eingerichtet haben, können Sie das FreeRTOS-Demo-Projekt erstellen, die Demo auf Ihr Board flashen und dann die Demo ausführen.

Um das FreeRTOS-Demoprojekt in WICED Studio zu erstellen und auszuführen

1. Starten Sie WICED Studio.
2. Wählen Sie im Menü Datei die Option Import aus. Erweitern Sie den Ordner `General`, wählen Sie Existing Projects into Workspace (Vorhandene Projekte in Arbeitsbereich) und dann Next (Weiter) aus.
3. Wählen Sie unter Select root directory (Stammverzeichnis auswählen) die Option Browse... (Durchsuchen...) aus, navigieren Sie zum Pfad *freertos*`/projects/cypress/CYW943907AEVAL1F/wicedstudio` und wählen Sie dann OK aus.
4. Aktivieren Sie in Projects (Projekte) das Kontrollkästchen nur für das Projekt aws_demo. Wählen Sie Finish (Beenden) aus, um das Projekt zu importieren. Das Zielprojekt aws_demo sollte im Fenster Make Target (Zum Ziel machen) angezeigt werden.
5. Erweitern Sie das Menü WICED Platform (WICED-Plattform) und wählen Sie WICED Filters off (WICED-Filter aus).
6. Erweitern Sie im Fenster Make Target (Zum Ziel machen) den Bereich aws_demo, klicken Sie mit der rechten Maustaste auf die Datei `demo.aws_demo` und wählen Sie dann Build Target (Ziel erstellen) aus, um die Demo zu erstellen und auf Ihr Board herunterzuladen. Die Demo sollte automatisch ausgeführt werden, nachdem sie erstellt und auf Ihr Board heruntergeladen wurde.

Fehlerbehebung

- Unter Windows erhalten Sie möglicherweise folgende Fehlermeldung beim Erstellen und Ausführen des Demoprojekts:

```
: recipe for target 'download_dct' failed
```

```
make.exe[1]: *** [download_dct] Error 1
```

Gehen Sie wie folgt vor, um diesen Fehler zu beheben:

1. Navigieren Sie zu *WICED-Studio-SDK-PATH*\WICED-Studio-6.2\43xxx_Wi-Fi\tools\OpenOCD\Win32 und doppelklicken Sie auf openocd-all-brcm-libftdi.exe.
2. Navigieren Sie zu *WICED-Studio-SDK-PATH*\WICED-Studio-6.2\43xxx_Wi-Fi\tools\drivers\CYW9WCD1EVAL1 und doppelklicken Sie auf InstallDriver.exe.

- Unter Linux oder macOS erhalten Sie möglicherweise folgende Fehlermeldung beim Erstellen und Ausführen des Demoprojekts:

```
make[1]: *** [download_dct] Error 127
```

Verwenden Sie den folgenden Befehl, um das libusb-dev-Paket zu aktualisieren und den Fehler zu beheben.

```
sudo apt-get install libusb-dev
```

Allgemeine Informationen zur Problembehandlung bei Getting Started with FreeRTOS finden Sie unter. Fehlerbehebung – Erste Schritte

Erste Schritte mit dem Cypress CYW954907 AEVAL1 F Development Kit

> **Important**
>
> Diese Referenzintegration wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Dieses Tutorial enthält Anweisungen für die ersten Schritte mit dem Cypress F Development Kit. CYW954907 AEVAL1 Wenn Sie das Cypress CYW954907 AEVAL1 F Development Kit nicht haben, besuchen Sie den Gerätekatalog für AWS Partner, um eines von unserem Partner zu erwerben.

> **Note**
>
> Dieses Tutorial führt Sie durch die Einrichtung und Ausführung der CoreMQTT Mutual Authentication Demo. Der FreeRTOS-Port für dieses Board unterstützt derzeit die TCP-Server- und Client-Demos nicht.

Bevor Sie beginnen, müssen Sie FreeRTOS konfigurieren AWS IoT und herunterladen, um Ihr Gerät mit der AWS Cloud zu verbinden. Detaillierte Anweisungen finden Sie unter Erste Schritte. In diesem Tutorial wird der Pfad zum FreeRTOS-Download-Verzeichnis als bezeichnet. *`freertos`*

> **Important**
>
> - In diesem Thema wird der Pfad zum FreeRTOS-Download-Verzeichnis als bezeichnet. *`freertos`*
> - Leerzeichen im *`freertos`*-Pfad können Build-Fehler verursachen. Stellen Sie beim Klonen oder Kopieren des Repositorys sicher, dass der von Ihnen erstellte Pfad keine Leerzeichen enthält.
> - Die maximale Länge eines Dateipfades bei Microsoft Windows ist 260 Zeichen. Lange FreeRTOS-Download-Verzeichnispfade können zu Build-Fehlern führen.
> - Da der Quellcode symbolische Links enthalten kann, müssen Sie, wenn Sie Windows zum Extrahieren des Archivs verwenden, möglicherweise:
>   - Entwicklermodus aktivieren oder
>   - Verwenden Sie eine Konsole mit Administratorberechtigungen.
>
>   Auf diese Weise kann Windows beim Extrahieren des Archivs ordnungsgemäß symbolische Links erstellen. Andernfalls werden symbolische Links als normale Dateien geschrieben, die die Pfade der symbolischen Links als Text enthalten oder leer sind. Weitere Informationen finden Sie im Blogeintrag Symlinks in Windows 10! .
>
>   Wenn Sie Git unter Windows verwenden, müssen Sie den Entwicklermodus aktivieren oder:
>
>   - Setzen `core.symlinks` Sie ihn mit dem folgenden Befehl auf true:
>
>     ```
>     git config --global core.symlinks true
>     ```

- Verwenden Sie immer dann eine Konsole mit Administratorrechten, wenn Sie einen Git-Befehl verwenden, der in das System schreibt (z. B.git pull,git clone, undgit submodule update --init --recursive).
- Wie bereits erwähntFreeRTOS wird heruntergeladen, sind FreeRTOS-Ports für Cypress derzeit nur auf verfügbar. GitHub

-Übersicht

Dieses Tutorial enthält Anweisungen für die folgenden ersten Schritte:

1. Installieren von Software auf dem Host-Computer zum Entwickeln und Debuggen eingebetteter Anwendungen für Ihr Mikrocontroller-Board.
2. Cross-Compilierung einer FreeRTOS-Demo-Anwendung zu einem Binär-Image.
3. Laden des binären Anwendungs-Image auf Ihr Board und Ausführen der Anwendung.
4. Interaktion mit der Anwendung, die auf Ihrem Board über eine serielle Verbindung ausgeführt wird, zu Überwachungs- und Debuggingzwecken.

Einrichtung der -Entwicklungsumgebung

Herunterladen und Installieren des WICED Studio-SDKs

In diesem Handbuch Erste Schritte verwenden Sie das Cypress WICED Studio SDK, um Ihr Board mit der FreeRTOS-Demo zu programmieren. Besuchen Sie die Website WICED Software, um das WICED Studio-SDK von Cypress herunterzuladen. Sie müssen ein kostenloses Cypress-Konto erstellen, um die Software herunterzuladen. Das WICED Studio-SDK ist für Windows-, macOS- und Linux-Betriebssysteme geeignet.

Note

Bei einigen Betriebssystemen sind möglicherweise zusätzliche Installationsschritte erforderlich. Folgen Sie den Installationsanweisungen für das Betriebssystem und die Version von WICED Studio, die Sie installieren.

Festlegen von Umgebungsvariablen

Bevor Sie WICED Studio zur Programmierung Ihres Boards verwenden, müssen Sie eine Umgebungsvariable für das Installationsverzeichnis des WICED Studio-SDKs erstellen. Wenn WICED Studio während des Erstellens der Variablen ausgeführt wird, müssen Sie die Anwendung nach dem festlegen der variablen neu starten.

> **Note**
>
> Das WICED Studio-Installationsprogramm erstellt zwei separate Ordner mit dem Namen `WICED-Studio-`*`m`*`.`*`n`* auf Ihrem Computer, wobei `m` und `n` die Nummer der Haupt- bzw. der Nebenversion sind. Dieses Dokument setzt den Ordnernamen `WICED-Studio-6.2` voraus. Achten Sie jedoch darauf, dass Sie den korrekten Namen für die von Ihnen installierte Version verwenden. Wenn Sie die Umgebungsvariable `WICED_STUDIO_SDK_PATH` definieren, müssen Sie den vollständigen Installationspfad des WICED Studio-SDKs und nicht den Installationspfad der WICED Studio-IDE angeben. Unter Windows und macOS wird der Ordner `WICED-Studio-`*`m`*`.`*`n`* für das SDK standardmäßig im Ordner `Documents` erstellt.

So erstellen Sie Umgebungsvariablen unter Windows

1. Öffnen Sie die Control Panel (Systemsteuerung) und wählen Sie System und anschließend Advanced System Settings (Erweiterte Systemeinstellungen).
2. Wählen Sie auf der Registerkarte Erweitert die Option Umgebungsvariablen.
3. Wählen Sie unter User variables (Benutzervariablen) die Option New (Neu).
4. Geben Sie als Variablenname ein. **`WICED_STUDIO_SDK_PATH`** Geben Sie unter Variable value (Variablenwert) das Installationsverzeichnis des WICED Studio-SDKs ein.

So erstellen Sie eine Umgebungsvariable unter Linux oder macOS

1. Öffnen Sie die Datei `/etc/profile` auf Ihrem Computer und fügen Sie der letzten Zeile Ihrer Datei Folgendes hinzu:

   ```
   export WICED_STUDIO_SDK_PATH=installation-path/WICED-Studio-6.2
   ```

2. Starten Sie Ihren Computer neu.
3. Öffnen Sie ein Terminal und führen Sie die folgenden Befehle aus:

```
cd freertos/vendors/cypress/WICED_SDK
```

```
perl platform_adjust_make.pl
```

```
chmod +x make
```

Herstellen einer seriellen Verbindung

So stellen Sie eine serielle Verbindung zwischen Ihrem Host-Computer und Ihrem Board her

1. Verbinden Sie das Board mittels eines Standard-USB-Kabels mit A-Stecker auf Micro-B-Stecker mit Ihrem Host-Computer.
2. Identifizieren Sie die USB-Seriennummer für die Verbindung zum Board auf Ihrem Host-Computer.
3. Starten Sie ein serielles Terminal und öffnen Sie eine Verbindung mit den folgenden Einstellungen:
   - Baudrate: 115200
   - Daten: 8 Bit
   - Parität: Keine
   - Stop-Bits: 1
   - Flusssteuerung: Keine

Weitere Informationen zum Installieren eines Terminals und zum Einrichten einer seriellen Verbindung finden Sie unter Installieren eines Terminal-Emulators.

Überwachung von MQTT-Nachrichten in der Cloud

Bevor Sie das FreeRTOS-Demoprojekt ausführen, können Sie den MQTT-Client in der AWS IoT Konsole so einrichten, dass er die Nachrichten überwacht, die Ihr Gerät an die Cloud sendet. AWS

Um das MQTT-Thema mit dem MQTT-Client zu abonnieren AWS IoT

1. Melden Sie sich bei der AWS IoT -Konsole an.

2. Wählen Sie im Navigationsbereich Test und dann MQTT-Testclient, um den MQTT-Client zu öffnen.
3. Geben Sie im Feld Subscription topic (Abonnementthema) die Option ***your-thing-name*/example/topic** ein und wählen Sie dann Subscribe to topic (Thema abonnieren).

Erstellen und starten Sie das FreeRTOS-Demoprojekt

Nachdem Sie eine serielle Verbindung zu Ihrem Board eingerichtet haben, können Sie das FreeRTOS-Demo-Projekt erstellen, die Demo auf Ihr Board flashen und dann die Demo ausführen.

Um das FreeRTOS-Demoprojekt in WICED Studio zu erstellen und auszuführen

1. Starten Sie WICED Studio.
2. Wählen Sie im Menü Datei die Option Import aus. Erweitern Sie den Ordner `General`, wählen Sie Existing Projects into Workspace (Vorhandene Projekte in Arbeitsbereich) und dann Next (Weiter) aus.
3. Wählen Sie unter Select root directory (Stammverzeichnis auswählen) die Option Browse... (Durchsuchen...) aus, navigieren Sie zum Pfad *freertos*`/projects/cypress/CYW954907AEVAL1F/wicedstudio` und wählen Sie dann OK aus.
4. Aktivieren Sie in Projects (Projekte) das Kontrollkästchen nur für das Projekt aws_demo. Wählen Sie Finish (Beenden) aus, um das Projekt zu importieren. Das Zielprojekt aws_demo sollte im Fenster Make Target (Zum Ziel machen) angezeigt werden.
5. Erweitern Sie das Menü WICED Platform (WICED-Plattform) und wählen Sie WICED Filters off (WICED-Filter aus).
6. Erweitern Sie im Fenster Make Target (Zum Ziel machen) den Bereich aws_demo, klicken Sie mit der rechten Maustaste auf die Datei `demo.aws_demo` und wählen Sie dann Build Target (Ziel erstellen) aus, um die Demo zu erstellen und auf Ihr Board herunterzuladen. Die Demo sollte automatisch ausgeführt werden, nachdem sie erstellt und auf Ihr Board heruntergeladen wurde.

Fehlerbehebung

- Unter Windows erhalten Sie möglicherweise folgende Fehlermeldung beim Erstellen und Ausführen des Demoprojekts:

```
: recipe for target 'download_dct' failed
```

```
make.exe[1]: *** [download_dct] Error 1
```

Gehen Sie wie folgt vor, um diesen Fehler zu beheben:

1. Navigieren Sie zu *WICED-Studio-SDK-PATH*\WICED-Studio-6.2\43xxx_Wi-Fi\tools\OpenOCD\Win32 und doppelklicken Sie auf openocd-all-brcm-libftdi.exe.
2. Navigieren Sie zu *WICED-Studio-SDK-PATH*\WICED-Studio-6.2\43xxx_Wi-Fi\tools\drivers\CYW9WCD1EVAL1 und doppelklicken Sie auf InstallDriver.exe.

- Unter Linux oder macOS erhalten Sie möglicherweise folgende Fehlermeldung beim Erstellen und Ausführen des Demoprojekts:

```
make[1]: *** [download_dct] Error 127
```

Verwenden Sie den folgenden Befehl, um das libusb-dev-Paket zu aktualisieren und den Fehler zu beheben.

```
sudo apt-get install libusb-dev
```

Allgemeine Informationen zur Problembehandlung bei Getting Started with FreeRTOS finden Sie unter. Fehlerbehebung – Erste Schritte

Erste Schritte mit dem Cypress CY8 CKIT-064S0S2-4343W-Kit

> **Important**
>
> Diese Referenzintegration wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Dieses Tutorial enthält Anweisungen für die ersten Schritte mit dem CKIT-064S0S2-4343W-Kit. CY8 Wenn Sie noch keines haben, können Sie diesen Link verwenden, um ein Kit zu kaufen. Sie können diesen Link auch verwenden, um auf das Benutzerhandbuch des Kits zuzugreifen.

Erste Schritte

Bevor Sie beginnen, müssen Sie FreeRTOS konfigurieren AWS IoT , um Ihr Gerät mit der AWS Cloud zu verbinden. Detaillierte Anweisungen finden Sie unter Erste Schritte. Nachdem Sie die Voraussetzungen erfüllt haben, erhalten Sie ein FreeRTOS-Paket mit AWS IoT Core Anmeldeinformationen.

> **Note**
>
> In diesem Tutorial wird der Pfad zum FreeRTOS-Download-Verzeichnis, das im Abschnitt „Erste Schritte“ erstellt wurde, als bezeichnet. *freertos*

Einrichten der Entwicklungsumgebung

FreeRTOS funktioniert entweder mit einem CMake oder einem Make-Build-Flow. Sie können es ModusToolbox für Ihren Make-Build-Flow verwenden. Sie können die im Lieferumfang enthaltene Eclipse-IDE ModusToolbox oder eine Partner-IDE wie IAR EW-Arm, Arm MDK oder Microsoft Visual Studio Code verwenden. Die Eclipse-IDE ist mit den Betriebssystemen Windows, MacOS und Linux kompatibel.

Bevor Sie beginnen, laden Sie die neueste ModusToolbox Software herunter und installieren Sie sie. Weitere Informationen finden Sie im ModusToolbox Installationshandbuch.

Tools für ModusToolbox 2.1 oder älter aktualisieren

Wenn Sie die ModusToolbox 2.1 Eclipse IDE verwenden, um dieses Kit zu programmieren, müssen Sie die OpenOCD- und Firmware-Loader-Tools aktualisieren.

In den folgenden Schritten ist standardmäßig der Pfad für: *ModusToolbox*

- Windows ist`C:\Users\`*user_name*`\ModusToolbox`.
- Linux ist *user_home*`/ModusToolbox` oder wo Sie die Archivdatei entpacken möchten.
- macOS befindet sich im Ordner Programme in dem Volume, das Sie im Assistenten auswählen.

OpenOCD aktualisieren

Für dieses Kit ist Cypress OpenOCD 4.0.0 oder höher erforderlich, um den Chip erfolgreich zu löschen und zu programmieren.

Um Cypress OpenOCD zu aktualisieren

1. Gehen Sie zur Cypress OpenOCD-Release-Seite.
2. Laden Sie die Archivdatei für Ihr Betriebssystem herunter (). Windows/Mac/Linux
3. Löschen Sie die vorhandenen Dateien in*ModusToolbox*`/tools_2.x/openocd`.
4. Ersetzen Sie die Dateien in *ModusToolbox*`/tools_2.x/openocd` durch den extrahierten Inhalt des Archivs, das Sie in einem vorherigen Schritt heruntergeladen haben.

Firmware-Loader wird aktualisiert

Für dieses Kit ist Cypress Firmware-Loader 3.0.0 oder höher erforderlich.

Um den Cypress Firmware-Loader zu aktualisieren

1. Gehen Sie zur Versionsseite des Cypress Firmware-Loaders.
2. Laden Sie die Archivdatei für Ihr Betriebssystem herunter (). Windows/Mac/Linux
3. Löschen Sie die vorhandenen Dateien in*ModusToolbox*`/tools_2.x/fw-loader`.
4. Ersetzen Sie die Dateien in *ModusToolbox*`/tools_2.x/fw-loader` durch den extrahierten Inhalt des Archivs, das Sie in einem vorherigen Schritt heruntergeladen haben.

Alternativ können Sie Projekt-Build-Dateien aus dem Quellcode der FreeRTOS-Anwendung generieren, das Projekt mit Ihrem bevorzugten Build-Tool erstellen und dann das Kit mit OpenOCD programmieren. CMake Wenn Sie lieber ein GUI-Tool für die Programmierung mit dem CMake Flow verwenden möchten, laden Sie Cypress Programmer von der Cypress Programming Solutions-Webseite herunter und installieren Sie es. Weitere Informationen finden Sie unter Verwendung CMake mit FreeRTOS.

Einrichten Ihrer Hardware

Gehen Sie wie folgt vor, um die Hardware des Kits einzurichten.

1. Stellen Sie Ihr Kit bereit

   Folgen Sie den Anweisungen im Bereitstellungsleitfaden für das CY8 CKIT-064S0S2-4343W-Kit, um Ihr Kit sicher bereitzustellen für. AWS IoT

   Für CySecureTools dieses Kit ist Version 3.1.0 oder höher erforderlich.

2. Richten Sie eine serielle Verbindung ein

    a. Connect das Kit mit Ihrem Host-Computer.

    b. Der serielle USB-Anschluss für das Kit wird automatisch auf dem Host-Computer aufgelistet. Identifizieren Sie die Portnummer. In Windows können Sie es mithilfe des Geräte-Managers unter Ports (COM & LPT) identifizieren.

    c. Starten Sie ein serielles Terminal und öffnen Sie eine Verbindung mit den folgenden Einstellungen:

    - Baudrate: 115200
    - Daten: 8 Bit
    - Parität: Keine
    - Stop-Bits: 1
    - Flusssteuerung: Keine

Erstellen und starten Sie das FreeRTOS-Demo-Projekt

In diesem Abschnitt erstellen Sie die Demo und führen sie aus.

1. Stellen Sie sicher, dass Sie die Schritte im Bereitstellungshandbuch für das CY8CKIT-064S0S2-4343W Kit befolgen.
2. Erstellen Sie die FreeRTOS-Demo.

    a. Öffnen Sie die Eclipse-IDE für ModusToolbox und wählen oder erstellen Sie einen Arbeitsbereich.

    b. Wählen Sie im Menü Datei die Option Import aus.

    Erweitern Sie Allgemein, wählen Sie Existierendes Projekt in Workspace und klicken Sie dann auf Weiter.

    c. Geben Sie im Stammverzeichnis den Projektnamen ein `freertos/projects/cypress/CY8CKIT-064S0S2-4343W/mtb/aws_demos` und wählen Sie ihn aus`aws_demos`. Er sollte standardmäßig ausgewählt sein.

    d. Wählen Sie Fertig stellen, um das Projekt in Ihren Workspace zu importieren.

    e. Erstellen Sie die Anwendung, indem Sie einen der folgenden Schritte ausführen:

    - Wählen Sie im Quick Panel die Option aws_demos Application erstellen aus.
    - Wählen Sie Project und dann Build All.

Stellen Sie sicher, dass das Projekt fehlerfrei kompiliert wird.

3. Überwachung von MQTT-Nachrichten in der Cloud

   Bevor Sie die Demo ausführen, können Sie den MQTT-Client in der AWS IoT Konsole so einrichten, dass er die Nachrichten überwacht, die Ihr Gerät an die AWS Cloud sendet. Gehen Sie wie folgt vor, um das MQTT-Thema mit dem AWS IoT MQTT-Client zu abonnieren.

   a. Melden Sie sich bei der AWS IoT -Konsole an.

   b. Wählen Sie im Navigationsbereich Test und dann MQTT-Testclient aus, um den MQTT-Client zu öffnen.

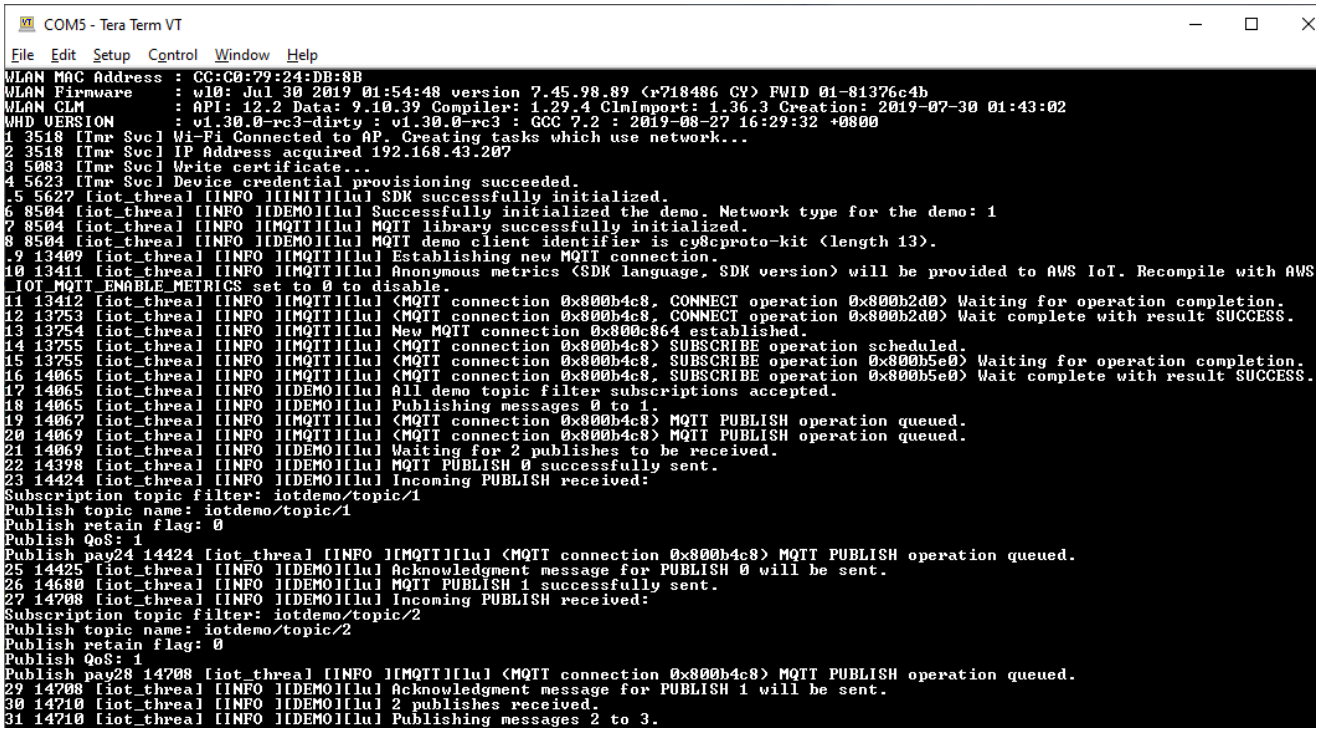
   c. Geben Sie ***your-thing-name*/example/topic** als Abonnementthema den Text ein und wählen Sie dann Thema abonnieren aus.

4. Führen Sie das FreeRTOS-Demo-Projekt aus

   a. Wählen Sie das Projekt `aws_demos` im Workspace aus.

   b. Wählen Sie im Quick Panel aws_demos Program (KitProg3) aus. Dadurch wird das Board programmiert und die Demo-Anwendung wird gestartet, nachdem die Programmierung abgeschlossen ist.

   c. Sie können den Status der laufenden Anwendung im seriellen Terminal einsehen. Die folgende Abbildung zeigt einen Teil der Terminalausgabe.



Die MQTT-Demo veröffentlicht Nachrichten zu vier verschiedenen Themen (`iotdemo/topic/`*n*, wobei n=1 bis 4) und abonniert all diese Themen, um dieselben Nachrichten zurück zu erhalten. Wenn eine Nachricht eingeht, veröffentlicht die Demo eine Bestätigungsnachricht zu diesem Thema. `iotdemo/acknowledgements` In der folgenden Liste werden die Debug-Meldungen beschrieben, die in der Terminalausgabe angezeigt werden, mit Verweisen auf die Seriennummern der Meldungen. In der Ausgabe werden zuerst die Details des WICED-Host-Treibers (WHD) ohne Seriennummerierung gedruckt.

1. 1 bis 4 — Das Gerät stellt eine Verbindung zum konfigurierten Access Point (AP) her und wird bereitgestellt, indem mithilfe des konfigurierten Endpunkts und der Zertifikate eine Verbindung zum AWS Server hergestellt wird.
2. 5 bis 13 — Die CoreMQTT-Bibliothek ist initialisiert und das Gerät stellt eine MQTT-Verbindung her.
3. 14 bis 17 — Das Gerät abonniert alle Themen, um die veröffentlichten Nachrichten zurückzusenden.
4. 18 bis 30 — Das Gerät veröffentlicht zwei Nachrichten und wartet darauf, sie zurück zu erhalten. Wenn jede Nachricht empfangen wird, sendet das Gerät eine Bestätigungsnachricht.

Derselbe Zyklus aus Veröffentlichen, Empfangen und Bestätigen wird fortgesetzt, bis alle Nachrichten veröffentlicht sind. Pro Zyklus werden zwei Nachrichten veröffentlicht, bis die Anzahl der konfigurierten Zyklen abgeschlossen ist.

5. Verwendung CMake mit FreeRTOS

Sie können es auch verwenden CMake , um die Demo-Anwendung zu erstellen und auszuführen. Informationen zur Einrichtung CMake eines systemeigenen Buildsystems finden Sie unterVoraussetzungen.

a. Verwenden Sie den folgenden Befehl, um Build-Dateien zu generieren. Geben Sie das Zielboard mit der `-DBOARD` Option an.

```
cmake -DVENDOR=cypress -DBOARD=CY8CKIT_064S0S2_4343W -DCOMPILER=arm-gcc -
S freertos -B build_dir
```

Wenn Sie Windows verwenden, müssen Sie das native Build-System mit der `-G` Option angeben, da standardmäßig Visual Studio CMake verwendet wird.

Example

```
cmake -DVENDOR=cypress -DBOARD=CY8CKIT_064S0S2_4343W -DCOMPILER=arm-gcc -
S freertos -B build_dir -G Ninja
```

Wenn `arm-none-eabi-gcc` es nicht in Ihrem Shell-Pfad enthalten ist, müssen Sie auch die `AFR_TOOLCHAIN_PATH` CMake Variable festlegen.

Example

```
 -DAFR_TOOLCHAIN_PATH=/home/user/opt/gcc-arm-none-eabi/bin
```

b. Verwenden Sie den folgenden Befehl, um das Projekt mit zu erstellen CMake.

```
cmake --build build_dir
```

c. Programmieren Sie abschließend die `cm4.hex` Dateien `cm0.hex` und, die unter generiert *`build_dir`* wurden, mit Cypress Programmer.

Andere Demos ausführen

Die folgenden Demo-Anwendungen wurden getestet und verifiziert, damit sie mit der aktuellen Version funktionieren. Sie finden diese Demos im *`freertos`*`/demos` Verzeichnis. Informationen zur Ausführung dieser Demos finden Sie unter. FreeRTOS RTOS-Demos

- Demo zu Bluetooth Low Energy
- Over-the-Air Aktualisiert die Demo
- Demo zum Secure Sockets Echo Client
- AWS IoT Device Shadow-Demo

Debuggen

Die KitProg Version 3 im Kit unterstützt das Debuggen über das SWD-Protokoll.

- Um die FreeRTOS-Anwendung zu debuggen, wählen Sie das Projekt aws_demos im Workspace aus und wählen Sie dann aws_demos Debug (3) aus dem Quick Panel. KitProg

OTA-Aktualisierungen

PSoC 64 MCUs hat alle erforderlichen FreeRTOS-Qualifikationstests bestanden. Die optionale Funktion over-the-air (OTA), die in der PSo C 64 Standard Secure AWS Firmware-Bibliothek implementiert ist, muss jedoch noch evaluiert werden. Die OTA-Funktion in der implementierten Form besteht derzeit alle OTA-Qualifizierungstests mit Ausnahme von aws_ota_test_case_rollback_if_unable_to_connect_after_update.py.

Wenn ein erfolgreich validiertes OTA-Image mithilfe der PSo C64 Standard Secure — AWS MCU auf ein Gerät angewendet wird und das Gerät nicht mit dem Gerät kommunizieren kann AWS IoT Core, kann das Gerät nicht automatisch auf das als funktionierend bekannte Originalbild zurückgreifen. Dies kann dazu führen, dass das Gerät für weitere Updates nicht erreichbar ist. AWS IoT Core Diese Funktionalität wird noch vom Cypress-Team entwickelt.

Weitere Informationen finden Sie unter OTA-Updates mit AWS und dem CY8 CKIT-064S0S2-4343W-Kit. Wenn Sie weitere Fragen haben oder technischen Support benötigen, wenden Sie sich an die Cypress Developer Community.

Erste Schritte mit dem Microchip ATECC608 A Secure Element with Windows-Simulator

> **Important**
>
> Diese Referenzintegration wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Dieses Tutorial enthält Anweisungen für die ersten Schritte mit dem Microchip A Secure Element with Windows Simulator. ATECC608

Sie benötigen die folgende Hardware:

- Microchip ATECC608 Ein Clickboard für sichere Elemente
- SAMD21 XPlained Profi
- mikroBUS Xplained Pro-Adapter

Bevor Sie beginnen, müssen Sie FreeRTOS konfigurieren AWS IoT und herunterladen, um Ihr Gerät mit der AWS Cloud zu verbinden. Detaillierte Anweisungen finden Sie unter Erste Schritte. In diesem Tutorial wird der Pfad zum FreeRTOS-Download-Verzeichnis als bezeichnet. *freertos*

-Übersicht

Dieses Tutorial enthält die folgenden Schritte:

1. Verbinden Sie Ihr Board mit einem Host-Rechner.
2. Installieren Sie die Software zum Entwickeln und Debuggen eingebetteter Anwendungen für Ihr Mikrocontroller-Board auf dem Host-Computer.
3. Eine FreeRTOS-Demo-Anwendung zu einem Binär-Image querkompilieren.
4. Laden des binären Anwendungs-Image auf Ihre Platine und Ausführen der Anwendung.

Richten Sie die Microchip A-Hardware ein ATECC608

Bevor Sie mit Ihrem Microchip ATECC608 A-Gerät interagieren können, müssen Sie zuerst das programmieren. SAMD21

So richten Sie das SAMD21 XPlained Pro-Board ein

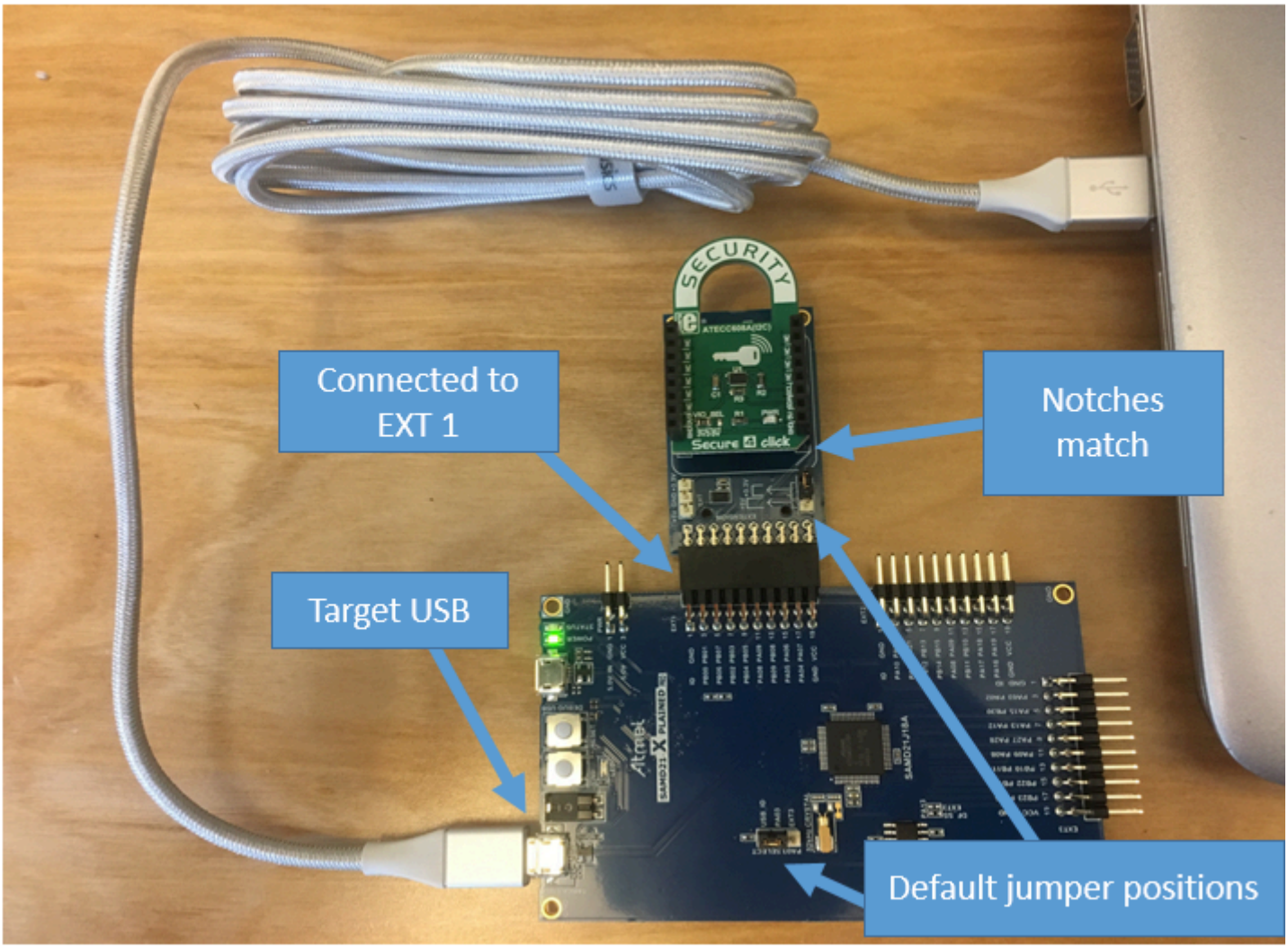
1. Folgen Sie dem Link CryptoAuthSSH-XSTK (DM320109) — Aktuelle Firmware, um eine ZIP-Datei mit Anweisungen (PDF) und einer Binärdatei herunterzuladen, die auf dem D21 programmiert werden kann.
2. Laden Sie das Atmel Studio 7 IDP herunter und installieren Sie es. Stellen Sie sicher, dass Sie während der Installation die SMART ARM MCU-Treiberarchitektur auswählen.
3. Verwenden Sie ein USB 2.0 Micro B-Kabel, um den „Debug USB“-Anschluss an Ihren Computer anzuschließen, und folgen Sie den Anweisungen in der PDF-Datei. (Der „Debug USB“-Anschluss ist der USB-Anschluss, der sich am nächsten zu POWER-LED und den Pins befindet.)

So schließen Sie die Hardware an

1. Trennen Sie das Micro-USB-Kabel vom Debug-USB.
2. Stecken Sie den MikroBus XPlained Pro-Adapter in die SAMD21 Platine am EXT1 Standort.
3. Stecken Sie das ATECC608 A Secure 4 Click-Board in den MikrobusX XPlained Pro-Adapter. Stellen Sie sicher, dass die gekerbte Ecke des Click Boards mit dem eingekerbten Symbol auf der Adapterplatine übereinstimmt.

4. Schließen Sie das Micro-USB-Kabel an das Ziel-USB an.

Ihre Einrichtung sollte wie folgt aussehen.



Einrichten Ihrer Entwicklungsumgebung

Melden Sie sich an für ein AWS-Konto

Wenn Sie noch keine haben AWS-Konto, führen Sie die folgenden Schritte aus, um eine zu erstellen.

Um sich für eine anzumelden AWS-Konto

1. Öffnen Sie https://portal.aws.amazon.com/billing/die Anmeldung.
2. Folgen Sie den Online-Anweisungen.

Während der Anmeldung erhalten Sie einen Telefonanruf oder eine Textnachricht und müssen einen Verifizierungscode über die Telefontasten eingeben.

Wenn Sie sich für eine anmelden AWS-Konto, Root-Benutzer des AWS-Kontoswird eine erstellt. Der Root-Benutzer hat Zugriff auf alle AWS-Services und Ressourcen des Kontos. Als bewährte Sicherheitsmethode weisen Sie einem Benutzer Administratorzugriff zu und verwenden Sie nur den Root-Benutzer, um Aufgaben auszuführen, die Root-Benutzerzugriff erfordern.

AWS sendet Ihnen nach Abschluss des Anmeldevorgangs eine Bestätigungs-E-Mail. Du kannst jederzeit deine aktuellen Kontoaktivitäten einsehen und dein Konto verwalten, indem du zu https://aws.amazon.com/gehst und Mein Konto auswählst.

Erstellen eines Benutzers mit Administratorzugriff

Nachdem Sie sich für einen angemeldet haben AWS-Konto, sichern Sie Ihren Root-Benutzer des AWS-Kontos AWS IAM Identity Center, aktivieren und erstellen Sie einen Administratorbenutzer, sodass Sie den Root-Benutzer nicht für alltägliche Aufgaben verwenden.

Sichern Sie Ihre Root-Benutzer des AWS-Kontos

1. Melden Sie sich AWS-Managementkonsoleals Kontoinhaber an, indem Sie Root-Benutzer auswählen und Ihre AWS-Konto E-Mail-Adresse eingeben. Geben Sie auf der nächsten Seite Ihr Passwort ein.

   Hilfe bei der Anmeldung mit dem Root-Benutzer finden Sie unter Anmelden als Root-Benutzer im AWS-Anmeldung -Benutzerhandbuch zu.
2. Aktivieren Sie die Multi-Faktor-Authentifizierung (MFA) für den Root-Benutzer.

   Anweisungen finden Sie unter Aktivieren eines virtuellen MFA-Geräts für Ihren AWS-Konto Root-Benutzer (Konsole) im IAM-Benutzerhandbuch.

Erstellen eines Benutzers mit Administratorzugriff

1. Aktivieren Sie das IAM Identity Center.

   Anweisungen finden Sie unter Aktivieren AWS IAM Identity Center im AWS IAM Identity Center -Benutzerhandbuch.
2. Gewähren Sie einem Administratorbenutzer im IAM Identity Center Benutzerzugriff.

Ein Tutorial zur Verwendung von IAM-Identity-Center-Verzeichnis als Identitätsquelle finden Sie IAM-Identity-Center-Verzeichnis im Benutzerhandbuch unter Benutzerzugriff mit der Standardeinstellung konfigurieren.AWS IAM Identity Center

Anmelden als Administratorbenutzer

- Um sich mit Ihrem IAM-Identity-Center-Benutzer anzumelden, verwenden Sie die Anmelde-URL, die an Ihre E-Mail-Adresse gesendet wurde, als Sie den IAM-Identity-Center-Benutzer erstellt haben.

  Hilfe bei der Anmeldung mit einem IAM Identity Center-Benutzer finden Sie im AWS-Anmeldung Benutzerhandbuch unter Anmeldung beim AWS Access-Portal.

Weiteren Benutzern Zugriff zuweisen

1. Erstellen Sie im IAM-Identity-Center einen Berechtigungssatz, der den bewährten Vorgehensweisen für die Anwendung von geringsten Berechtigungen folgt.

   Anweisungen hierzu finden Sie unter Berechtigungssatz erstellen im AWS IAM Identity Center - Benutzerhandbuch.
2. Weisen Sie Benutzer einer Gruppe zu und weisen Sie der Gruppe dann Single Sign-On-Zugriff zu.

   Eine genaue Anleitung finden Sie unter Gruppen hinzufügen im AWS IAM Identity Center - Benutzerhandbuch.

Um Zugriff zu gewähren, fügen Sie Ihren Benutzern, Gruppen oder Rollen Berechtigungen hinzu:

- Benutzer und Gruppen in AWS IAM Identity Center:

  Erstellen Sie einen Berechtigungssatz. Befolgen Sie die Anweisungen unter Erstellen eines Berechtigungssatzes im AWS IAM Identity Center -Benutzerhandbuch.
- Benutzer, die in IAM über einen Identitätsanbieter verwaltet werden:

  Erstellen Sie eine Rolle für den Identitätsverbund. Befolgen Sie die Anleitung unter Eine Rolle für einen externen Identitätsanbieter (Verbund) erstellen im IAM-Benutzerhandbuch.
- IAM-Benutzer:

- Erstellen Sie eine Rolle, die Ihr Benutzer annehmen kann. Befolgen Sie die Anleitung unter Eine Rolle für einen IAM-Benutzer erstellen im IAM-Benutzerhandbuch.
- (Nicht empfohlen) Weisen Sie einem Benutzer eine Richtlinie direkt zu oder fügen Sie einen Benutzer zu einer Benutzergruppe hinzu. Befolgen Sie die Anweisungen unter Hinzufügen von Berechtigungen zu einem Benutzer (Konsole) im IAM-Benutzerhandbuch.

Einrichtung

1. Laden Sie das FreeRTOS-Repo aus dem FreeRTOS-Repository herunter. GitHub

   Um FreeRTOS herunterzuladen von: GitHub

   1. Navigieren Sie zum FreeRTOS-Repository GitHub .
   2. Wählen Sie Clone or Download (Klonen oder herunterladen) aus.
   3. Klonen Sie das Repository über die Befehlszeile auf Ihrem Computer in ein Verzeichnis auf Ihrem Host-Computer.

      ```
      git clone https://github.com/aws/amazon-freertos.git -\-recurse-submodules
      ```

      Important

      - In diesem Thema wird der Pfad zum FreeRTOS-Download-Verzeichnis als bezeichnet. *freertos*
      - Leerzeichen im *freertos*-Pfad können Build-Fehler verursachen. Stellen Sie beim Klonen oder Kopieren des Repositorys sicher, dass der von Ihnen erstellte Pfad keine Leerzeichen enthält.
      - Die maximale Länge eines Dateipfades bei Microsoft Windows ist 260 Zeichen. Lange FreeRTOS-Download-Verzeichnispfade können zu Build-Fehlern führen.
      - Da der Quellcode symbolische Links enthalten kann, müssen Sie, wenn Sie Windows zum Extrahieren des Archivs verwenden, möglicherweise:
        - Entwicklermodus aktivieren oder
        - Verwenden Sie eine Konsole mit Administratorberechtigungen.

        Auf diese Weise kann Windows beim Extrahieren des Archivs ordnungsgemäß symbolische Links erstellen. Andernfalls werden symbolische Links als normale Dateien geschrieben, die die Pfade der symbolischen Links als Text enthalten oder

leer sind. Weitere Informationen finden Sie im Blogeintrag Symlinks in Windows 10! .

Wenn Sie Git unter Windows verwenden, müssen Sie den Entwicklermodus aktivieren oder Sie müssen:

- Setzen `core.symlinks` Sie ihn mit dem folgenden Befehl auf true:

```
git config -\-global core.symlinks true
```

- Verwenden Sie immer dann eine Konsole mit Administratorrechten, wenn Sie einen Git-Befehl verwenden, der in das System schreibt (z. B.git pull,git clone, undgit submodule update -\-init -\-recursive).

4. Überprüfen Sie im *`freertos`*-Verzeichnis den Zweig, der verwendet werden soll.

2. Einrichten Ihrer Entwicklungsumgebung

   a. Installieren Sie die neueste Version von Win PCap.

   b. Installieren Sie Microsoft Visual Studio.

      Visual Studio-Versionen 2017 und 2019 funktionieren. Alle Editionen dieser Visual Studio-Versionen werden unterstützt (Community, Professional oder Enterprise).

      Installieren Sie zusätzlich zur IDE die Komponente Desktop-Entwicklung mit C++. Installieren Sie dann unter Optional das neueste Windows 10 SDK.

   c. Stellen Sie sicher, dass Sie über eine aktive kabelgebundene Ethernet-Verbindung verfügen.

Erstellen Sie das FreeRTOS-Demoprojekt und führen Sie es aus

Important

Das Microchip ATECC608 A-Gerät verfügt über eine einmalige Initialisierung, die bei der ersten Ausführung eines Projekts (während des Aufrufs) auf das Gerät beschränkt ist. `C_InitToken` Das FreeRTOS-Demoprojekt und das Testprojekt haben jedoch unterschiedliche Konfigurationen. Wenn das Gerät während der Demo-Projektkonfigurationen gesperrt ist, ist es nicht möglich, dass alle Tests im Testprojekt erfolgreich sind.

Um das FreeRTOS-Demoprojekt mit der Visual Studio-IDE zu erstellen und auszuführen

1. Laden Sie das Projekt in Visual Studio.

   Wählen Sie im Menü File (Datei) die Option Open (Öffnen) aus. Wählen Sie File/Solution (Datei/Lösung), navigieren Sie zur Datei *freertos*`\projects\microchip\ecc608a_plus_winsim\visual_studio\aws_demos\aws_demos.sln` und wählen Sie dann Open (Öffnen).
2. Richten Sie das Demoprojekt neu aus.

   Das bereitgestellte Demoprojekt hängt vom Windows-SDK ab, es wird dafür aber keine Windows-SDK-Version angegeben. Standardmäßig kann die IDE versuchen, die Demo mit einer SDK-Version zu erstellen, die nicht auf Ihrem Computer vorhanden ist. Um die Windows-SDK-Version einzustellen, klicken Sie mit der rechten Maustaste auf aws_demos und wählen Sie dann Retarget Projects (Projekte neu ausrichten). Dadurch wird das Fenster Review solution actions (Lösungsaktionen überprüfen) geöffnet. Wählen Sie eine Windows SDK Version, die auf Ihrem Computer vorhanden ist (verwenden Sie den Anfangswert in der Dropdown-Liste) und wählen Sie dann OK.
3. Erstellen und Ausführen des Projekts.

   Wählen Sie im Menü Build die Option Build Solution aus und stellen Sie sicher, dass die Lösung fehlerfrei erstellt wird. Wählen Sie Debug, Start Debugging, um das Projekt auszuführen. Bei der ersten Ausführung müssen Sie Ihre Geräteschnittstelle konfigurieren und neu kompilieren. Weitere Informationen finden Sie unter Konfigurieren Ihrer Netzwerkschnittstelle.
4. Stellen Sie den Mikrochip A bereit. ATECC608

   Microchip hat mehrere Skriptwerkzeuge bereitgestellt, um bei der Einrichtung der ATECC608 A-Teile zu helfen. Navigieren Sie zu *freertos*`\vendors\microchip\secure_elements\app\example_trust_chain_tool` und öffnen Sie die Datei „README.md“.

   Befolgen Sie die Anweisungen in der `README.md`-Datei, um Ihr Gerät bereitzustellen. Die Schritte umfassen Folgendes:

   1. Erstellen und registrieren Sie eine Zertifizierungsstelle bei. AWS
   2. Generieren Sie Ihre Schlüssel auf dem Mikrochip ATECC608 A und exportieren Sie den öffentlichen Schlüssel und die Seriennummer des Geräts.
   3. Generieren Sie ein Zertifikat für das Gerät und registrieren Sie dieses Zertifikat mit AWS.
   4. Laden Sie das Zertifizierungsstellenzertifikat und das Gerätezertifikat auf das Gerät.

5. Erstellen Sie FreeRTOS-Beispiele und führen Sie sie aus.

   Führen Sie das Demo-Projekt erneut aus. Dieses Mal sollten Sie sich verbinden!

Fehlerbehebung

Informationen zur Problembehebung finden Sie unter Fehlerbehebung – Erste Schritte.

Erste Schritte mit dem Espressif ESP32 - DevKit C und dem ESP-WROVER-KIT

> **Important**
>
> Diese Referenzintegration wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

> **Note**
>
> Um zu erfahren, wie Sie modulare FreeRTOS-Bibliotheken und -Demos in Ihr eigenes Espressif-IDF-Projekt integrieren können, schauen Sie sich unsere vorgestellte Referenzintegration für die -C3-Plattform an. ESP32

Folgen Sie diesem Tutorial, um mit dem DevKit Espressif-C zu beginnen, das mit den Modulen ESP32 -WROOM-32, -SOLO-1 ESP32 oder ESP-WROVER ausgestattet ist, und den. ESP32 ESP-WROVER-KIT-VB Verwenden Sie die folgenden Links, um eines von unserem Partner im Partnergerätekatalog zu erwerben: AWS

- ESP32-WROOM-32 C DevKit
- ESP32-SOLO-1
- ESP32-WROVER-KIT

Diese Versionen von Entwicklungsboards werden auf FreeRTOS unterstützt.

Weitere Informationen zu den neuesten Versionen dieser Boards finden Sie unter ESP32- DevKit C V4 oder ESP-WROVER-KITv4.1 auf der Espressif-Website.

> **Note**
>
> Derzeit unterstützt der FreeRTOS-Port für ESP32 -WROVER-KIT und ESP DevKit C die Funktion Symmetric Multiprocessing (SMP) nicht.

-Übersicht

In diesem Tutorial führen Sie die folgenden Schritte aus:

1. Verbinden Ihres Boards mit einem Host-Computer.
2. Installieren von Software auf dem Host-Computer zum Entwickeln und Debuggen eingebetteter Anwendungen für Ihr Mikrocontroller-Board.
3. Cross-Compilierung einer FreeRTOS-Demo-Anwendung zu einem Binär-Image.
4. Laden des binären Anwendungs-Image auf Ihr Board und Ausführen der Anwendung.
5. Interaktion mit der Anwendung, die auf Ihrem Board über eine serielle Verbindung ausgeführt wird, zu Überwachungs- und Debuggingzwecken.

Voraussetzungen

Bevor Sie mit FreeRTOS auf Ihrem Espressif-Board beginnen, müssen Sie Ihr Konto und Ihre AWS Berechtigungen einrichten.

Melde dich an für ein AWS-Konto

Wenn Sie noch keine haben AWS-Konto, führen Sie die folgenden Schritte aus, um eine zu erstellen.

Um sich für eine anzumelden AWS-Konto

1. Öffnen Sie https://portal.aws.amazon.com/billing/die Anmeldung.
2. Folgen Sie den Online-Anweisungen.

   Während der Anmeldung erhalten Sie einen Telefonanruf oder eine Textnachricht und müssen einen Verifizierungscode über die Telefontasten eingeben.

Wenn Sie sich für eine anmelden AWS-Konto, Root-Benutzer des AWS-Kontoswird eine erstellt. Der Root-Benutzer hat Zugriff auf alle AWS-Services und Ressourcen des Kontos. Als bewährte Sicherheitsmethode weisen Sie einem Benutzer Administratorzugriff zu und verwenden Sie nur den Root-Benutzer, um Aufgaben auszuführen, die Root-Benutzerzugriff erfordern.

AWS sendet Ihnen nach Abschluss des Anmeldevorgangs eine Bestätigungs-E-Mail. Du kannst jederzeit deine aktuellen Kontoaktivitäten einsehen und dein Konto verwalten, indem du zu https://aws.amazon.com/gehst und Mein Konto auswählst.

Erstellen eines Benutzers mit Administratorzugriff

Nachdem Sie sich für einen angemeldet haben AWS-Konto, sichern Sie Ihren Root-Benutzer des AWS-Kontos AWS IAM Identity Center, aktivieren und erstellen Sie einen Administratorbenutzer, sodass Sie den Root-Benutzer nicht für alltägliche Aufgaben verwenden.

Sichern Sie Ihre Root-Benutzer des AWS-Kontos

1. Melden Sie sich AWS-Managementkonsoleals Kontoinhaber an, indem Sie Root-Benutzer auswählen und Ihre AWS-Konto E-Mail-Adresse eingeben. Geben Sie auf der nächsten Seite Ihr Passwort ein.

   Hilfe bei der Anmeldung mit dem Root-Benutzer finden Sie unter Anmelden als Root-Benutzer im AWS-Anmeldung -Benutzerhandbuch zu.
2. Aktivieren Sie die Multi-Faktor-Authentifizierung (MFA) für den Root-Benutzer.

   Anweisungen finden Sie unter Aktivieren eines virtuellen MFA-Geräts für Ihren AWS-Konto Root-Benutzer (Konsole) im IAM-Benutzerhandbuch.

Erstellen eines Benutzers mit Administratorzugriff

1. Aktivieren Sie das IAM Identity Center.

   Anweisungen finden Sie unter Aktivieren AWS IAM Identity Center im AWS IAM Identity Center -Benutzerhandbuch.
2. Gewähren Sie einem Administratorbenutzer im IAM Identity Center Benutzerzugriff.

Ein Tutorial zur Verwendung von IAM-Identity-Center-Verzeichnis als Identitätsquelle finden Sie IAM-Identity-Center-Verzeichnis im Benutzerhandbuch unter Benutzerzugriff mit der Standardeinstellung konfigurieren.AWS IAM Identity Center

Anmelden als Administratorbenutzer

- Um sich mit Ihrem IAM-Identity-Center-Benutzer anzumelden, verwenden Sie die Anmelde-URL, die an Ihre E-Mail-Adresse gesendet wurde, als Sie den IAM-Identity-Center-Benutzer erstellt haben.

  Hilfe bei der Anmeldung mit einem IAM Identity Center-Benutzer finden Sie im AWS-Anmeldung Benutzerhandbuch unter Anmeldung beim AWS Access-Portal.

Weiteren Benutzern Zugriff zuweisen

1. Erstellen Sie im IAM-Identity-Center einen Berechtigungssatz, der den bewährten Vorgehensweisen für die Anwendung von geringsten Berechtigungen folgt.

   Anweisungen hierzu finden Sie unter Berechtigungssatz erstellen im AWS IAM Identity Center - Benutzerhandbuch.
2. Weisen Sie Benutzer einer Gruppe zu und weisen Sie der Gruppe dann Single Sign-On-Zugriff zu.

   Eine genaue Anleitung finden Sie unter Gruppen hinzufügen im AWS IAM Identity Center - Benutzerhandbuch.

Um Zugriff zu gewähren, fügen Sie Ihren Benutzern, Gruppen oder Rollen Berechtigungen hinzu:

- Benutzer und Gruppen in AWS IAM Identity Center:

  Erstellen Sie einen Berechtigungssatz. Befolgen Sie die Anweisungen unter Erstellen eines Berechtigungssatzes im AWS IAM Identity Center -Benutzerhandbuch.
- Benutzer, die in IAM über einen Identitätsanbieter verwaltet werden:

  Erstellen Sie eine Rolle für den Identitätsverbund. Befolgen Sie die Anleitung unter Eine Rolle für einen externen Identitätsanbieter (Verbund) erstellen im IAM-Benutzerhandbuch.
- IAM-Benutzer:

- Erstellen Sie eine Rolle, die Ihr Benutzer annehmen kann. Befolgen Sie die Anleitung unter Eine Rolle für einen IAM-Benutzer erstellen im IAM-Benutzerhandbuch.
- (Nicht empfohlen) Weisen Sie einem Benutzer eine Richtlinie direkt zu oder fügen Sie einen Benutzer zu einer Benutzergruppe hinzu. Befolgen Sie die Anweisungen unter Hinzufügen von Berechtigungen zu einem Benutzer (Konsole) im IAM-Benutzerhandbuch.

Erste Schritte

> **Note**
>
> Für die Linux-Befehle in diesem Tutorial müssen Sie die Bash-Shell verwenden.

1. Richten Sie die Espressif-Hardware ein.

   - Informationen zur Einrichtung der Hardware für das ESP32 DevKit C-Entwicklungsboard finden Sie im ESP32DevKitC V4 Getting Started Guide.
   - Informationen zur Einrichtung der Hardware für das ESP-WROVER-KIT Entwicklungsboard finden Sie im ESP-WROVER-KITV4.1 Getting Started Guide.

   > **Important**
   >
   > Wenn Sie den Abschnitt „Erste Schritte“ der Espressif-Anleitungen erreicht haben, halten Sie an und kehren Sie dann zu den Anweisungen auf dieser Seite zurück.

2. Laden Sie Amazon FreeRTOS von herunter. GitHub (Anweisungen finden Sie in der Datei README.md.)
3. Richten Sie Ihre Entwicklungsumgebung ein.

   Um mit Ihrem Board zu kommunizieren, müssen Sie eine Toolchain installieren. Espressif stellt die ESP-IDF zur Verfügung, um Software für ihre Boards zu entwickeln. Da die ESP-IDF eine eigene Version des FreeRTOS-Kernels als Komponente integriert hat, enthält Amazon FreeRTOS eine benutzerdefinierte Version von ESP-IDF v4.2, bei der der FreeRTOS-Kernel entfernt wurde. Dies behebt Probleme mit doppelten Dateien beim Kompilieren. Um die in Amazon FreeRTOS enthaltene benutzerdefinierte Version von ESP-IDF v4.2 zu verwenden, folgen Sie den nachstehenden Anweisungen für das Betriebssystem Ihres Host-Computers.

Windows

1. Laden Sie den Universal Online Installer von ESP-IDF für Windows herunter.
2. Führen Sie den Universal Online Installer aus.
3. Wenn Sie zum Schritt ESP-IDF herunterladen oder verwenden gelangen, wählen Sie Bestehendes ESP-IDF-Verzeichnis verwenden und setzen Sie Vorhandenes ESP-IDF-Verzeichnis auswählen auf. `freertos/vendors/espressif/esp-idf`
4. Schließen Sie die Installation ab.

macOS

1. Folgen Sie den Anweisungen in den Voraussetzungen für die Standardkonfiguration der Toolchain (ESP-IDF v4.2) für macOS.

   > **Important**
   >
   > Wenn Sie unter Nächste Schritte zu den Anweisungen „Get ESP-IDF“ gelangen, beenden Sie den Vorgang und kehren Sie dann zu den Anweisungen auf dieser Seite zurück.

2. Öffnen Sie ein Befehlszeilenfenster.
3. Navigieren Sie zum FreeRTOS-Download-Verzeichnis und führen Sie dann das folgende Skript aus, um die espressif-Toolchain für Ihre Plattform herunterzuladen und zu installieren.

   ```
   vendors/espressif/esp-idf/install.sh
   ```

4. Fügen Sie die ESP-IDF-Toolketten-Tools mit dem folgenden Befehl zum Pfad Ihres Terminals hinzu.

   ```
   source vendors/espressif/esp-idf/export.sh
   ```

Linux

1. Folgen Sie den Anweisungen in den Voraussetzungen für die Standardkonfiguration der Toolchain (ESP-IDF v4.2) für Linux.

> **Important**
>
> Wenn Sie unter „Nächste Schritte“ zu den Anweisungen „Get ESP-IDF“ gelangen, beenden Sie den Vorgang und kehren Sie dann zu den Anweisungen auf dieser Seite zurück.

2. Öffnen Sie ein Befehlszeilenfenster.
3. Navigieren Sie zum FreeRTOS-Download-Verzeichnis und führen Sie dann das folgende Skript aus, um die Espressif-Toolchain für Ihre Plattform herunterzuladen und zu installieren.

```
vendors/espressif/esp-idf/install.sh
```

4. Fügen Sie die ESP-IDF-Toolketten-Tools mit dem folgenden Befehl zum Pfad Ihres Terminals hinzu.

```
source vendors/espressif/esp-idf/export.sh
```

4. Stellen Sie eine serielle Verbindung her.

   a. Um eine serielle Verbindung zwischen Ihrem Host-Computer und dem ESP32 - DevKit C herzustellen, müssen Sie die CP210x USB-zu-UART-Bridge-VCP-Treiber installieren. Sie können diese Treiber von Silicon Labs herunterladen.

      Um eine serielle Verbindung zwischen Ihrem Host-Computer und dem ESP32 -WROVER-KIT herzustellen, müssen Sie den virtuellen FTDI-COM-Port-Treiber installieren. Sie können diesen Treiber von FTDI herunterladen.

   b. Folgen Sie den Schritten zum Herstellen einer seriellen Verbindung mit ESP32.

   c. Nachdem Sie eine serielle Verbindung hergestellt haben, notieren Sie sich den seriellen Port für Ihre Board-Verbindung. Sie benötigen es, um die Demo zu flashen.

Konfigurieren Sie die FreeRTOS-Demoanwendungen

Für dieses Tutorial befindet sich die FreeRTOS-Konfigurationsdatei unter. *`freertos`*`/vendors/espressif/boards/`*`board-name`*`/aws_demos/config_files/FreeRTOSConfig.h` (Wenn zum Beispiel ausgewählt `AFR_BOARD espressif.esp32_devkitc` ist, befindet sich die Konfigurationsdatei unter*`freertos`*`/vendors/espressif/boards/esp32/aws_demos/config_files/FreeRTOSConfig.h`.)

1. Wenn Sie macOS oder Linux verwenden, öffnen Sie eine Terminal-Eingabeaufforderung. Wenn Sie Windows verwenden, öffnen Sie die App „ESP-IDF 4.x CMD“ (falls Sie diese Option bei der Installation der ESP-IDF-Toolchain angegeben haben) oder andernfalls die App „Command Prompt“.
2. Um zu überprüfen, ob Sie Python3 installiert haben, führen Sie Folgendes aus

   ```
   python --version
   ```

   Die installierte Version wird angezeigt. Wenn Sie Python 3.0.1 oder höher nicht installiert haben, können Sie es von der Python-Website installieren.
3. Sie benötigen die AWS Befehlszeilenschnittstelle (CLI), um AWS IoT Befehle auszuführen. Wenn Sie Windows verwenden, verwenden Sie den `easy_install awscli` Befehl, um die AWS CLI in der App „Command“ oder „ESP-IDF 4.x CMD“ zu installieren.

   Wenn Sie macOS oder Linux verwenden, finden Sie weitere Informationen unter Installation der AWS CLI.
4. Ausführen

   ```
   aws configure
   ```

   und konfigurieren Sie die AWS CLI mit Ihrer AWS Zugriffsschlüssel-ID, Ihrem geheimen Zugriffsschlüssel und Ihrer AWS Standardregion. Weitere Informationen finden Sie unter Konfigurieren der AWS -CLI.
5. Verwenden Sie den folgenden Befehl, um das AWS SDK für Python (boto3) zu installieren:

   - Führen Sie unter Windows in der App „Command“ oder „ESP-IDF 4.x CMD“ den folgenden Befehl aus

     ```
     pip install boto3 --user
     ```

     > **Note**
     >
     > Einzelheiten finden Sie in der Boto3-Dokumentation.

   - Führen Sie unter macOS oder Linux Folgendes aus

```
pip install tornado nose --user
```

und dann ausführen

```
pip install boto3 --user
```

FreeRTOS enthält das `SetupAWS.py` Skript, mit dem Sie Ihr Espressif-Board für die Verbindung einfacher einrichten können. AWS IoT Wenn Sie das Skript konfigurieren möchten, öffnen Sie *`freertos`*`/tools/aws_config_quick_start/configure.json` und legen die folgenden Attribute fest:

`afr_source_dir`

Der vollständige Pfad zum *`freertos`*-Verzeichnis auf Ihrem Computer. Stellen Sie sicher, dass Sie diesen Pfad mit Schrägstrichen angeben.

`thing_name`

Der Name, den Sie dem AWS IoT Ding zuweisen möchten, das Ihr Board repräsentiert.

`wifi_ssid`

Die SSID Ihres WLANs.

`wifi_password`

Das Passwort für Ihr WLAN-Netzwerk

`wifi_security`

Der Sicherheitstyp für Ihr WLAN-Netzwerk

Die folgenden Sicherheitstypen sind gültig:

- `eWiFiSecurityOpen` (Open, no security (Offen, keine Sicherheit)
- `eWiFiSecurityWEP` (WEP-Sicherheit)
- `eWiFiSecurityWPA` (WPA-Sicherheit)
- `eWiFiSecurityWPA2`(WPA2 Sicherheit)

6. Führen Sie das Konfigurationsskript aus.

a. Wenn Sie macOS oder Linux verwenden, öffnen Sie eine Terminal-Eingabeaufforderung. Wenn Sie Windows verwenden, öffnen Sie die App „ESP-IDF 4.x CMD“ oder „Command“.

b. Navigiere zum Verzeichnis und führe es aus *freertos*`/tools/aws_config_quick_start`

```
python SetupAWS.py setup
```

Das -Skript führt folgende Aktionen aus:

- Erstellt ein IoT-Ding, ein Zertifikat und eine Richtlinie.
- Hängt die IoT-Richtlinie an das Zertifikat und das Zertifikat an das AWS IoT Ding an.
- Füllt die `aws_clientcredential.h` Datei mit Ihrem AWS IoT Endpunkt, Ihrer Wi-Fi-SSID und Ihren Anmeldeinformationen auf.
- Formatiert Ihr Zertifikat und Ihren privaten Schlüssel und schreibt sie in die `aws_clientcredential_keys.h` Header-Datei.

> **Note**
>
> Das Zertifikat ist nur zu Demonstrationszwecken hartcodiert. Anwendungen auf Produktionsebene sollten diese Dateien an einem sicheren Ort speichern.

Weitere Informationen dazu `SetupAWS.py` finden Sie `README.md` im *freertos*`/tools/aws_config_quick_start` Verzeichnis.

Überwachung von MQTT-Nachrichten in der Cloud

Bevor Sie das FreeRTOS-Demoprojekt ausführen, können Sie den MQTT-Client in der AWS IoT Konsole so einrichten, dass er die Nachrichten überwacht, die Ihr Gerät an die Cloud sendet. AWS

Um das MQTT-Thema mit dem MQTT-Client zu abonnieren AWS IoT

1. Navigieren Sie zur AWS IoT -Konsole.
2. Wählen Sie im Navigationsbereich Test und dann MQTT Test Client aus.

3. Geben Sie im Feld Subscription topic (Abonnementthema) die Option *your-thing-name*`/example/topic` ein und wählen Sie dann Subscribe to topic (Thema abonnieren).

Wenn das Demo-Projekt erfolgreich auf Ihrem Gerät ausgeführt wird, sehen Sie „Hello World!“ mehrfach zu dem Thema gesendet, das Sie abonniert haben.

Erstellen, flashen und starten Sie das FreeRTOS-Demoprojekt mit dem Skript idf.py

Sie können das IDF-Hilfsprogramm (`idf.py`) von Espressif verwenden, um das Projekt zu erstellen und die Binärdateien auf Ihr Gerät zu flashen.

> **Note**
>
> Bei einigen Setups müssen Sie möglicherweise die Port-Option `"-p port-name"` mit verwenden, `idf.py` um den richtigen Port anzugeben, wie im folgenden Beispiel.
>
> ```
> idf.py -p /dev/cu.usbserial-00101301B flash
> ```

FreeRTOS unter Windows, Linux und macOS erstellen und flashen (ESP-IDF v4.2)

1. Navigieren Sie zum Stammverzeichnis Ihres FreeRTOS-Download-Verzeichnisses.
2. Geben Sie in einem Befehlszeilenfenster den folgenden Befehl ein, um die ESP-IDF-Tools zum PATH Ihres Terminals hinzuzufügen.

   Windows (App „Command“)

   ```
   vendors\espressif\esp-idf\export.bat
   ```

   Windows (App „ESP-IDF 4.x CMD“)

   (Dies wurde bereits getan, als Sie die App geöffnet haben.)

   Linux//macOS

   ```
   source vendors/espressif/esp-idf/export.sh
   ```

3. Konfigurieren Sie cmake im `build` Verzeichnis und erstellen Sie das Firmware-Image mit dem folgenden Befehl.

```
idf.py -DVENDOR=espressif -DBOARD=esp32_wrover_kit -DCOMPILER=xtensa-esp32 build
```

Die Ausgabe sollte ungefähr wie die folgende aussehen.

```
   Running cmake in directory /path/to/hello_world/build
   Executing "cmake -G Ninja --warn-uninitialized /path/to/hello_world"...
   Warn about uninitialized values.
   -- Found Git: /usr/bin/git (found version "2.17.0")
   -- Building empty aws_iot component due to configuration
   -- Component names: ...
   -- Component paths: ...

   ... (more lines of build system output)

   [527/527] Generating hello-world.bin
   esptool.py v2.3.1

   Project build complete. To flash, run this command:
   ../../../components/esptool_py/esptool/esptool.py -p (PORT) -b 921600
 write_flash --flash_mode dio --flash_size detect --flash_freq 40m 0x10000 build/
hello-world.bin  build 0x1000 build/bootloader/bootloader.bin 0x8000 build/
partition_table/partition-table.bin
   or run 'idf.py -p PORT flash'
```

Wenn keine Fehler vorliegen, generiert der Build die binären Firmware-Dateien.

4. Löschen Sie den Flash-Speicher Ihres Entwicklungsboards mit dem folgenden Befehl.

```
idf.py erase_flash
```

5. Verwenden Sie das `idf.py` Skript, um die Binärdatei der Anwendung auf Ihr Board zu flashen.

```
idf.py flash
```

6. Überwachen Sie die Ausgabe von der seriellen Schnittstelle Ihrer Platine mit dem folgenden Befehl.

```
idf.py monitor
```

> **Note**
>
> Sie können diese Befehle wie im folgenden Beispiel kombinieren.
>
> ```
> idf.py erase_flash flash monitor
> ```

Bei bestimmten Host-Rechner-Setups müssen Sie den Port angeben, wenn Sie die Karte flashen, wie im folgenden Beispiel.

```
idf.py erase_flash flash monitor -p /dev/ttyUSB1
```

FreeRTOS erstellen und flashen mit CMake

Zusätzlich zu dem vom IDF SDK bereitgestellten `idf.py` Skript zum Erstellen und Ausführen Ihres Codes können Sie das Projekt auch verwenden. CMake Derzeit unterstützt es entweder Unix-Makefiles oder das Ninja-Build-System.

Um das Projekt zu erstellen und zu flashen

1. Navigieren Sie in einem Befehlszeilenfenster zum Stammverzeichnis Ihres FreeRTOS-Download-Verzeichnisses.
2. Führen Sie das folgende Skript aus, um die ESP-IDF-Tools zum PATH Ihrer Shell hinzuzufügen.

   Windows

   ```
   vendors\espressif\esp-idf\export.bat
   ```

   Linux//macOS

   ```
   source vendors/espressif/esp-idf/export.sh
   ```
3. Geben Sie den folgenden Befehl ein, um die Build-Dateien zu generieren.

Mit Unix-Makefiles

```
cmake -DVENDOR=espressif -DBOARD=esp32_wrover_kit -DCOMPILER=xtensa-esp32 -S . -
B ./YOUR_BUILD_DIRECTORY -DAFR_ENABLE_ALL_MODULES=1 -DAFR_ENABLE_TESTS=0
```

Mit Ninja

```
cmake -DVENDOR=espressif -DBOARD=esp32_wrover_kit -DCOMPILER=xtensa-esp32 -S . -
B ./YOUR_BUILD_DIRECTORY -DAFR_ENABLE_ALL_MODULES=1 -DAFR_ENABLE_TESTS=0 -GNinja
```

4. Erstellen Sie das Projekt.

   Mit Unix-Makefiles

   ```
   make -C ./YOUR_BUILD_DIRECTORY -j8
   ```

   Mit Ninja

   ```
   ninja -C ./YOUR_BUILD_DIRECTORY -j8
   ```

5. Lösche den Blitz und flashe dann die Platine.

   Mit Unix-Makefiles

   ```
   make -C ./YOUR_BUILD_DIRECTORY erase_flash
   ```

   ```
   make -C ./YOUR_BUILD_DIRECTORY flash
   ```

   Mit Ninja

   ```
   ninja -C ./YOUR_BUILD_DIRECTORY erase_flash
   ```

   ```
   ninja -C ./YOUR_BUILD_DIRECTORY flash
   ```

Ausführen der Bluetooth Low Energy-Demos

FreeRTOS unterstützt Bluetooth Low Energy-Bibliothek Konnektivität.

Um das FreeRTOS-Demoprojekt über Bluetooth Low Energy auszuführen, müssen Sie die FreeRTOS Bluetooth Low Energy Mobile SDK-Demoanwendung auf einem iOS- oder Android-Mobilgerät ausführen.

So richten Sie die mobile FreeRTOS Bluetooth Low Energy SDK-Demoanwendung ein

1. Folgen Sie den Anweisungen im Abschnitt Mobil SDKs für FreeRTOS-Bluetooth-Geräte, um das SDK für Ihre mobile Plattform auf Ihren Host-Computer herunterzuladen und dort zu installieren.
2. Folgen Sie den Anweisungen im Abschnitt FreeRTOS Bluetooth Low Energy Mobile SDK-Demoanwendung, um die Demoanwendung für Mobilgeräte auf Ihrem Mobilgerät einzurichten.

Anweisungen zum Ausführen der MQTT over Bluetooth Low Energy-Demo auf Ihrem Board finden Sie unter. MQTT über Bluetooth Low Energy

Anweisungen zur Ausführung der Wi-Fi-Bereitstellungsdemo auf Ihrem Board finden Sie unter. WLAN-Bereitstellung

Verwenden Sie FreeRTOS in Ihrem eigenen CMake Projekt für ESP32

Wenn Sie FreeRTOS in Ihrem eigenen CMake Projekt verwenden möchten, können Sie es als Unterverzeichnis einrichten und zusammen mit Ihrer Anwendung erstellen. Besorgen Sie sich zunächst eine Kopie von FreeRTOS von. GitHub Sie können es auch mit dem folgenden Befehl als Git-Submodul einrichten, damit es in future einfacher aktualisiert werden kann.

```
git submodule add -b release https://github.com/aws/amazon-freertos.git freertos
```

Wenn eine neuere Version veröffentlicht wird, können Sie Ihre lokale Kopie mit diesen Befehlen aktualisieren.

```
# Pull the latest changes from the remote tracking branch.
git submodule update --remote -- freertos
```

```
# Commit the submodule change because it is pointing to a different revision now.
git add freertos
```

```
git commit -m "Update FreeRTOS to a new release"
```

Wenn Ihr Projekt die folgende Verzeichnisstruktur hat:

```
- freertos (the copy that you obtained from GitHub or the AWS IoT console)
- src
  - main.c (your application code)
- CMakeLists.txt
```

Dann ist das Folgende ein Beispiel für die `CMakeLists.txt` Top-Level-Datei, die verwendet werden kann, um Ihre Anwendung zusammen mit FreeRTOS zu erstellen.

```
cmake_minimum_required(VERSION 3.13)

project(freertos_examples)

# Tell IDF build to link against this target.
set(IDF_EXECUTABLE_SRCS "<complete_path>/src/main.c")
set(IDF_PROJECT_EXECUTABLE my_app)

# Add FreeRTOS as a subdirectory. AFR_BOARD tells which board to target.
set(AFR_BOARD espressif.esp32_devkitc CACHE INTERNAL "")
add_subdirectory(freertos)

# Link against the mqtt library so that we can use it. Dependencies are transitively
# linked.
target_link_libraries(my_app PRIVATE AFR::core_mqtt)
```

Führen Sie die folgenden Befehle aus, um das Projekt zu erstellen. CMake Stellen Sie sicher, dass sich der ESP32 Compiler in der Umgebungsvariablen PATH befindet.

```
cmake -S . -B build-directory -DCMAKE_TOOLCHAIN_FILE=freertos/tools/cmake/toolchains/
xtensa-esp32.cmake -GNinja
```

```
cmake --build build-directory
```

Führen Sie den folgenden Befehl aus, um die Anwendung auf Ihr Board zu flashen.

```
cmake --build build-directory --target flash
```

Komponenten von FreeRTOS verwenden

Nach der Ausführung CMake finden Sie alle verfügbaren Komponenten in der Übersichtsausgabe. Es sollte ungefähr wie das folgende Beispiel aussehen.

```
====================Configuration for FreeRTOS====================
  Version:                  202107.00
  Git version:              202107.00-g79ad6defb

Target microcontroller:
  vendor:                   Espressif
  board:                    ESP32-DevKitC
  description:              Development board produced by Espressif that comes in two
                            variants either with ESP-WROOM-32 or ESP32-WROVER module
  family:                   ESP32
  data ram size:            520KB
  program memory size:      4MB

Host platform:
  OS:                       Linux-4.15.0-66-generic
  Toolchain:                xtensa-esp32
  Toolchain path:           /opt/xtensa-esp32-elf
  CMake generator:          Ninja

FreeRTOS modules:
  Modules to build:         backoff_algorithm, common, common_io, core_http,
                            core_http_demo_dependencies, core_json, core_mqtt,
                            core_mqtt_agent, core_mqtt_agent_demo_dependencies,
                            core_mqtt_demo_dependencies, crypto, defender, dev_mode_key_
                            provisioning, device_defender, device_defender_demo_
                            dependencies, device_shadow,
 device_shadow_demo_dependencies,
                            freertos_cli_plus_uart, freertos_plus_cli, greengrass,
                            http_demo_helpers, https, jobs, jobs_demo_dependencies,
                            kernel, logging, mqtt, mqtt_agent_interface, mqtt_demo_
                            helpers, mqtt_subscription_manager, ota, ota_demo_
                            dependencies, ota_demo_version, pkcs11, pkcs11_helpers,
                            pkcs11_implementation, pkcs11_utils, platform,
 secure_sockets,
                            serializer, shadow, tls, transport_interface_secure_sockets,
                            wifi
   Enabled by user:         common_io, core_http_demo_dependencies, core_json,
                            core_mqtt_agent_demo_dependencies, core_mqtt_demo_
                            dependencies, defender, device_defender,
 device_defender_demo_
                            dependencies, device_shadow,
 device_shadow_demo_dependencies,
```

```
                            freertos_cli_plus_uart, freertos_plus_cli, greengrass,
 https,
                            jobs, jobs_demo_dependencies, logging,
 ota_demo_dependencies,
                            pkcs11, pkcs11_helpers, pkcs11_implementation, pkcs11_utils,
                            platform, secure_sockets, shadow, wifi
  Enabled by dependency:    backoff_algorithm, common, core_http, core_mqtt,
                            core_mqtt_agent, crypto, demo_base,
 dev_mode_key_provisioning,
                            freertos, http_demo_helpers, kernel, mqtt, mqtt_agent_
                            interface, mqtt_demo_helpers, mqtt_subscription_manager,
 ota,
                            ota_demo_version, pkcs11_mbedtls, serializer, tls,
                            transport_interface_secure_sockets, utils
  3rdparty dependencies:    jsmn, mbedtls, pkcs11, tinycbor
  Available demos:          demo_cli_uart, demo_core_http, demo_core_mqtt,
 demo_core_mqtt_
                            agent, demo_device_defender, demo_device_shadow,
                            demo_greengrass_connectivity, demo_jobs, demo_ota_core_http,
                            demo_ota_core_mqtt, demo_tcp
  Available tests:
==================================================================================
```

Sie können auf alle Komponenten aus der `Modules to build` Liste verweisen. Um sie mit Ihrer Anwendung zu verknüpfen, stellen Sie den `AFR::` Namespace vor den Namen `AFR::core_mqttAFR::ota`, z. B., usw.

Fügen Sie mithilfe von ESP-IDF benutzerdefinierte Komponenten hinzu

Sie können weitere Komponenten hinzufügen, während Sie ESP-IDF verwenden. Angenommen, Sie möchten eine Komponente mit dem Namen `example_component` hinzufügen und Ihr Projekt sieht folgendermaßen aus:

```
- freertos
- components
  - example_component
    - include
      - example_component.h
    - src
      - example_component.c
    - CMakeLists.txt
- src
  - main.c
```

```
- CMakeLists.txt
```

Im Folgenden finden Sie ein Beispiel für die `CMakeLists.txt` Datei für Ihre Komponente.

```
add_library(example_component src/example_component.c)
target_include_directories(example_component PUBLIC include)
```

Fügen Sie dann in der `CMakeLists.txt` Datei der obersten Ebene die Komponente hinzu, indem Sie direkt danach `add_subdirectory(freertos)` die folgende Zeile einfügen.

```
add_subdirectory(component/example_component)
```

Ändern Sie dann, `target_link_libraries` um Ihre Komponente einzubeziehen.

```
target_link_libraries(my_app PRIVATE AFR::core_mqtt PRIVATE example_component)
```

Diese Komponente ist jetzt standardmäßig automatisch mit Ihrem Anwendungscode verknüpft. Sie können jetzt die zugehörigen Header-Dateien einbeziehen und die darin definierten Funktionen aufrufen.

Überschreiben Sie die Konfigurationen für FreeRTOS

Derzeit gibt es keinen klar definierten Ansatz zur Neudefinition der Konfigurationen außerhalb des FreeRTOS-Quellbaums. Standardmäßig wird nach den Verzeichnissen und CMake gesucht. *freertos*`/vendors/espressif/boards/esp32/aws_demos/config_files/` *freertos*`/demos/include/` Sie können jedoch eine Problemumgehung verwenden, um den Compiler anzuweisen, zuerst andere Verzeichnisse zu durchsuchen. Sie können beispielsweise einen weiteren Ordner für FreeRTOS-Konfigurationen hinzufügen.

```
- freertos
- freertos-configs
  - aws_clientcredential.h
  - aws_clientcredential_keys.h
  - iot_mqtt_agent_config.h
  - iot_config.h
- components
- src
- CMakeLists.txt
```

Die Dateien unter `freertos-configs` werden aus den Verzeichnissen *`freertos`*`/vendors/espressif/boards/esp32/aws_demos/config_files/` und *`freertos`*`/demos/include/` kopiert. Fügen Sie dann in Ihrer `CMakeLists.txt` Datei auf oberster Ebene zuvor diese Zeile hinzu, `add_subdirectory(freertos)` damit der Compiler zuerst dieses Verzeichnis durchsucht.

```
include_directories(BEFORE freertos-configs)
```

Bereitstellen Ihrer eigenen sdkconfig für ESP-IDF

Falls Sie Ihre eigene Variable angeben möchten`sdkconfig.default`, können Sie die CMake Variable `IDF_SDKCONFIG_DEFAULTS` über die Befehlszeile festlegen:

```
cmake -S . -B build-directory -DIDF_SDKCONFIG_DEFAULTS=path_to_your_sdkconfig_defaults
 -DCMAKE_TOOLCHAIN_FILE=freertos/tools/cmake/toolchains/xtensa-esp32.cmake -GNinja
```

Wenn Sie keinen Speicherort für Ihre eigene `sdkconfig.default` Datei angeben, verwendet FreeRTOS die Standarddatei unter. *`freertos`*`/vendors/espressif/boards/esp32/aws_demos/sdkconfig.defaults`

Weitere Informationen finden Sie unter Projektkonfiguration in der Espressif API-Referenz. Falls Sie nach der erfolgreichen Kompilierung auf Probleme stoßen, lesen Sie den Abschnitt über veraltete Optionen und deren Ersatz auf dieser Seite.

Zusammenfassung

Wenn Sie in einem Projekt mit einer Komponente mit dem Namen `example_component` einige Konfigurationen außer Kraft setzen möchten, finden Sie hier ein vollständiges Beispiel für die Datei `CMakeLists.txt` der obersten Ebene.

```
cmake_minimum_required(VERSION 3.13)

project(freertos_examples)

set(IDF_PROJECT_EXECUTABLE my_app)
set(IDF_EXECUTABLE_SRCS "src/main.c")

# Tell IDF build to link against this target.
set(IDF_PROJECT_EXECUTABLE my_app)

# Add some extra components. IDF_EXTRA_COMPONENT_DIRS is a variable used by ESP-IDF
# to collect extra components.
```

```
get_filename_component(
    EXTRA_COMPONENT_DIRS
    "components/example_component" ABSOLUTE
)
list(APPEND IDF_EXTRA_COMPONENT_DIRS ${EXTRA_COMPONENT_DIRS})

# Override the configurations for FreeRTOS.
include_directories(BEFORE freertos-configs)

# Add FreeRTOS as a subdirectory. AFR_BOARD tells which board to target.
set(AFR_BOARD espressif.esp32_devkitc CACHE INTERNAL "")
add_subdirectory(freertos)

# Link against the mqtt library so that we can use it. Dependencies are transitively
# linked.
target_link_libraries(my_app PRIVATE AFR::core_mqtt)
```

Fehlerbehebung

- Wenn Sie macOS verwenden und das Betriebssystem Ihren nicht erkennt, stellen Sie sicher ESP-WROVER-KIT, dass Sie die D2XX-Treiber nicht installiert haben. Befolgen Sie zum Deinstallieren die Anweisungen im FTDI-Treiber-Installationsleitfaden für macOS X.
- Das von ESP-IDF bereitgestellte (und mit make monitor aufgerufene) Monitor-Hilfsprogramm hilft Ihnen beim Dekodieren von Adressen. Aus diesem Grund kann es Ihnen helfen, aussagekräftige Rückverfolgungen für den Fall zu erhalten, dass die Anwendung nicht mehr funktioniert. Weitere Informationen finden Sie unter Automatische Adressdekodierung auf der Espressif-Website.
- Es ist auch möglich, die Kommunikation mit gdb GDBstub zu aktivieren, ohne dass spezielle JTAG-Hardware erforderlich ist. Weitere Informationen finden Sie unter Launching GDB with GDBStub auf der Espressif-Website.
- Informationen zum Einrichten einer OpenOCD-basierten Umgebung, falls hardwarebasiertes JTAG-Debugging erforderlich ist, finden Sie unter JTAG-Debugging auf der Espressif-Website.
- Wenn es nicht unter macOS `pip` installiert werden `pyserial` kann, laden Sie es von der Pyserial-Website herunter.
- Wenn das Board kontinuierlich zurückgesetzt wird, versuchen Sie, den Flash zu löschen, indem Sie den folgenden Befehl am Terminal eingeben.

  ```
  make erase_flash
  ```

- Wenn Fehler bei der Ausführung von `idf_monitor.py` auftreten, verwenden Sie Python 2.7.

- Erforderliche Bibliotheken von ESP-IDF sind in FreeRTOS enthalten, sodass sie nicht extern heruntergeladen werden müssen. Wenn die `IDF_PATH` Umgebungsvariable gesetzt ist, empfehlen wir, dass Sie sie löschen, bevor Sie FreeRTOS erstellen.
- Unter Windows kann es drei bis vier Minuten dauern, bis das Projekt erstellt wird. Um die Build-Zeit zu reduzieren, können Sie den `-j4` Schalter im Befehl make verwenden.

```
make flash monitor -j4
```

- Wenn Ihr Gerät Probleme hat, eine Verbindung herzustellen AWS IoT, öffnen Sie die `aws_clientcredential.h` Datei und überprüfen Sie, ob die Konfigurationsvariablen in der Datei richtig definiert sind. `clientcredentialMQTT_BROKER_ENDPOINT[]`sollte so aussehen`1234567890123-ats.iot.us-east-1.amazonaws.com`.
- Wenn Sie die Schritte in Verwenden Sie FreeRTOS in Ihrem eigenen CMake Projekt für ESP32 ausführen und undefinierte Referenzfehler vom Linker sehen, liegt dies normalerweise an fehlenden abhängigen Bibliotheken oder Demos. Um sie hinzuzufügen, aktualisieren Sie die `CMakeLists.txt` Datei (im Stammverzeichnis) mit der CMake Standardfunktion`target_link_libraries`.
- ESP-IDF v4.2 unterstützt die Verwendung von xtensa\ -esp32\ -elf\ -gcc 8\ .2\ .0\. Werkzeugkette. Wenn Sie eine frühere Version der Xtensa-Toolchain verwenden, laden Sie die erforderliche Version herunter.
- Wenn Sie ein Fehlerprotokoll wie das Folgende über Python-Abhängigkeiten sehen, die für ESP-IDF v4.2 nicht erfüllt sind:

```
The following Python requirements are not satisfied:
  click>=5.0
  pyserial>=3.0
  future>=0.15.2
  pyparsing>=2.0.3,<2.4.0
  pyelftools>=0.22
  gdbgui==0.13.2.0
  pygdbmi<=0.9.0.2
  reedsolo>=1.5.3,<=1.5.4
  bitstring>=3.1.6
  ecdsa>=0.16.0
  Please follow the instructions found in the "Set up the tools" section of ESP-IDF
 Getting Started Guide
```

Installieren Sie die Python-Abhängigkeiten auf Ihrer Plattform mit dem folgenden Python-Befehl:

```
root/vendors/espressif/esp-idf/requirements.txt
```

Weitere Informationen zur Fehlerbehebung finden Sie unterFehlerbehebung – Erste Schritte.

Debuggen

Debuggen von Code auf Espressif ESP32 — DevKit C und ESP-WROVER-KIT (ESP-IDF v4.2)

In diesem Abschnitt erfahren Sie, wie Sie Espressif-Hardware mit ESP-IDF v4.2 debuggen. Sie benötigen ein JTAG zu USB-Kabel. Wir verwenden ein USB-MPSSE-Kabel (z. B. das FTDI C232HM-DDHSL-0).

DevKitESP-C JTAG-Setup

Für das FTDI C232HM-DDHSL-0-Kabel sind dies die Verbindungen zum DevkitC. ESP32

```
| C232HM-DDHSL-0 Wire Color | ESP32 GPIO Pin | JTAG Signal Name |
| ------------------------- | -------------- | ---------------- |
|  Brown (pin 5)            |  IO14          |  TMS             |
|  Yellow (pin 3)           |  IO12          |  TDI             |
|  Black (pin 10)           |  GND           |  GND             |
|  Orange (pin 2)           |  IO13          |  TCK             |
|  Green (pin 4)            |  IO15          |  TDO             |
```

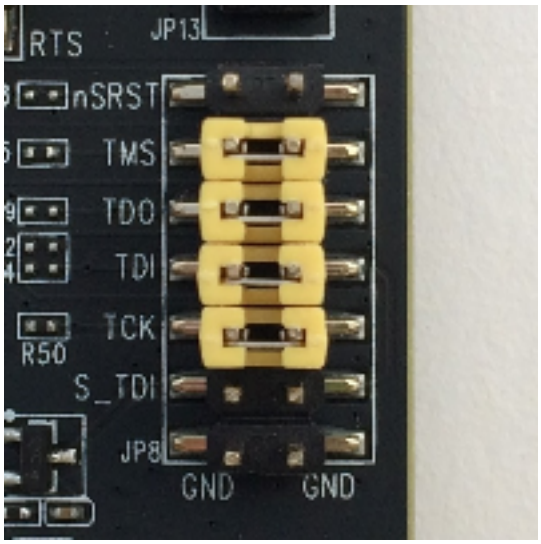
ESP-WROVER-KIT JTAG-Setup

Für das FTDI C232HM-DDHSL-0-Kabel sind dies die Verbindungen zum -WROVER-KIT. ESP32

```
| C232HM-DDHSL-0 Wire Color | ESP32 GPIO Pin | JTAG Signal Name |
| ------------------------- | -------------- | ---------------- |
|  Brown (pin 5)            |  IO14          |  TMS             |
|  Yellow (pin 3)           |  IO12          |  TDI             |
|  Orange (pin 2)           |  IO13          |  TCK             |
|  Green (pin 4)            |  IO15          |  TDO             |
```

Diese Tabellen wurden aus dem FTDI-C232HM-DDHSL-0-Datenblatt entwickelt. Weitere Informationen finden Sie im Abschnitt „C232HM MPSSE-Kabelanschluss und mechanische Details“ im Datenblatt.

Um JTAG auf dem zu aktivieren ESP-WROVER-KIT, platzieren Sie Jumper an den TMS-, TDO-, TDI-, TCK- und S_TDI-Pins, wie hier gezeigt.



Debuggen unter Windows (ESP-IDF v4.2)

So richten Sie das Debugging in Windows ein:

1. Verbinden Sie die USB-Seite von FTDI-C232HM-DDHSL-0 mit Ihrem Computer und der anderen Seite, siehe Debuggen von Code auf Espressif ESP32 — DevKit C und ESP-WROVER-KIT (ESP-IDF v4.2). Das FTDI-C232HM-DDHSL-0-Gerät sollte im Device Manager (Geräte-Manager) unter Universal Serial Bus Controllers (Universal-Serial-Bus-Controller) erscheinen.
2. Klicken Sie in der Liste der Geräte mit dem universellen seriellen Bus mit der rechten Maustaste auf das C232HM-DDHSL-0-Gerät und wählen Sie dann Eigenschaften.

   > **Note**
   >
   > Das Gerät ist möglicherweise als USB-Serial Port (Serieller USB-Anschluss) aufgeführt.

   Um die Eigenschaften des Geräts anzuzeigen, wählen Sie im Eigenschaftenfenster die Registerkarte Details. Wenn das Gerät nicht aufgeführt ist, installieren Sie den Windows-Treiber für FTDI C232HM-DDHSL-0.
3. Wählen Sie auf der Registerkarte Details die Option Eigenschaft und dann Hardware aus. IDs Im Feld Wert sollte etwas Ähnliches angezeigt werden.

```
FTDIBUS\COMPORT&VID_0403&PID_6014
```

In diesem Beispiel lautet die Anbieter-ID 0403 und die Produkt-ID 6014.

Stellen Sie sicher, dass diese IDs mit IDs der Eingabe übereinstimmen`projects/espressif/esp32/make/aws_demos/esp32_devkitj_v1.cfg`. Sie IDs werden in einer Zeile angegeben, die mit beginnt, `ftdi_vid_pid` gefolgt von einer Lieferanten-ID und einer Produkt-ID.

```
ftdi_vid_pid 0x0403 0x6014
```

4. Laden Sie OpenOCD für Windows herunter.
5. Entpacken Sie die Datei in das Verzeichnis `C:\` und fügen Sie `C:\openocd-esp32\bin` Ihrem Systempfad hinzu.
6. OpenOCD erfordert libusb. Dies wird unter Windows nicht standardmäßig installiert. Um libusb zu installieren:
   a. Laden Sie zadig.exe herunter.
   b. Führen Sie `zadig.exe`. Wählen Sie im Menü Optionen die Option List All Devices (Alle Geräte auflisten) aus.
   c. Wählen Sie im Dropdownmenü C232HM-DDHSL-0.
   d. Wählen Sie im Feld für den Zieltreiber rechts neben dem grünen Pfeil WinUSB aus.
   e. Wählen Sie für die Liste unter dem Zieltreiberfeld den Pfeil und dann Treiber installieren aus. Klicken Sie auf Replace Driver (Treiber ersetzen).
7. Öffnen Sie eine Befehlszeile, navigieren Sie zum Stammverzeichnis Ihres FreeRTOS-Download-Verzeichnisses und führen Sie den folgenden Befehl aus.

```
idf.py openocd
```

Lassen Sie dieses Befehlszeilenfenster geöffnet.

8. Öffnen Sie eine neue Eingabeaufforderung, navigieren Sie zum Stammverzeichnis Ihres FreeRTOS-Download-Verzeichnisses und führen Sie Folgendes aus:

```
idf.py flash monitor
```

9. Öffne eine weitere Eingabeaufforderung, navigiere zum Stammverzeichnis deines FreeRTOS-Download-Verzeichnisses und warte, bis die Demo auf deinem Board läuft. Wenn das der Fall ist, starte

   ```
   idf.py gdb
   ```

   Das Programm sollte in der `main`-Funktion stoppen.

   > **Note**
   >
   > Der ESP32 unterstützt maximal zwei Breakpoints.

Debuggen auf macOS (ESP-IDF v4.2)

1. Laden Sie den FTDI-Treiber für macOS herunter.
2. Laden Sie OpenOCD herunter.
3. Extrahieren Sie die heruntergeladen TAR-Datei und legen Sie den Pfad in `.bash_profile` auf `OCD_INSTALL_DIR/openocd-esp32/bin` fest.
4. Verwenden Sie den folgenden Befehl, um die Installation `libusb` auf macOS durchzuführen.

   ```
   brew install libusb
   ```

5. Verwenden Sie den folgenden Befehl, um den Treiber für die serielle Schnittstelle zu entladen.

   ```
   sudo kextunload -b com.FTDI.driver.FTDIUSBSerialDriver
   ```

6. Verwenden Sie den folgenden Befehl, um den Treiber für die serielle Schnittstelle zu entladen.

   ```
   sudo kextunload -b com.FTDI.driver.FTDIUSBSerialDriver
   ```

7. Wenn Sie eine neuere macOS-Version als 10.9 verwenden, verwenden Sie den folgenden Befehl, um den Apple FTDI-Treiber zu entladen.

   ```
   sudo kextunload -b com.apple.driver.AppleUSBFTDI
   ```

8. Verwenden Sie den folgenden Befehl, um die Produkt-ID und Anbieter-ID des FTDI-Kabels zu bekommen. Er listet die angeschlossenen USB-Geräte auf.

   ```
   system_profiler SPUSBDataType
   ```

   Die Ausgabe von `system_profiler` sollte wie folgt aussehen.

   ```
   DEVICE:

   Product ID: product-ID
   Vendor ID: vendor-ID (Future Technology Devices International Limited)
   ```

9. Öffnen Sie die `projects/espressif/esp32/make/aws_demos/esp32_devkitj_v1.cfg` Datei. Die Anbieter-ID und Produkt-ID für Ihr Gerät sind in einer Zeile angegeben, die mit `ftdi_vid_pid` beginnt. Ändern Sie IDs das so, dass es mit der `system_profiler` Ausgabe im vorherigen Schritt übereinstimmt. IDs
10. Öffnen Sie ein Terminalfenster, navigieren Sie zum Stammverzeichnis Ihres FreeRTOS-Download-Verzeichnisses und verwenden Sie den folgenden Befehl, um OpenOCD auszuführen.

    ```
    idf.py openocd
    ```

    Lassen Sie dieses Terminalfenster geöffnet.
11. Öffnen Sie ein neues Terminal und laden Sie mit dem folgenden Befehl den FTDI-Treiber für die serielle Schnittstelle.

    ```
    sudo kextload -b com.FTDI.driver.FTDIUSBSerialDriver
    ```

12. Gehen Sie zum Stammverzeichnis Ihres FreeRTOS-Download-Verzeichnisses und starten Sie

    ```
    idf.py flash monitor
    ```

13. Öffnen Sie ein anderes neues Terminal, navigieren Sie zum Stammverzeichnis Ihres FreeRTOS-Download-Verzeichnisses und starten Sie

    ```
    idf.py gdb
    ```

Das Programm sollte bei `main` stoppen.

Debuggen unter Linux (ESP-IDF v4.2)

1. Laden Sie OpenOCD herunter. Extrahieren Sie den Tarball und befolgen Sie die Installationsanweisungen in der Readme-Datei.
2. Verwenden Sie den folgenden Befehl, um libusb unter Linux zu installieren.

```
sudo apt-get install libusb-1.0
```

3. Öffnen Sie ein Terminal und geben Sie ls -l /dev/ttyUSB* ein, um alle mit Ihrem Computer verbundenen USB-Geräte aufzulisten. Auf diese Weise können Sie überprüfen, ob die USB-Anschlüsse der Platine vom Betriebssystem erkannt werden. Die Ausgabe sollte ungefähr wie die folgende aussehen.

```
   $ls -l /dev/ttyUSB*
   crw-rw----    1    root    dialout    188,    0    Jul    10    19:04    /
dev/ttyUSB0
   crw-rw----    1    root    dialout    188,    1    Jul    10    19:04    /
dev/ttyUSB1
```

4. Melden Sie sich ab und anschließend wieder an und schalten Sie die Stromversorgung zum Board ab und wieder an, damit die Änderungen wirksam werden. Listen Sie in einer Terminal-Eingabeaufforderung die USB-Geräte auf. Stellen Sie sicher, dass der Gruppenbesitzer von `dialout` zu gewechselt hat`plugdev`.

```
   $ls -l /dev/ttyUSB*
   crw-rw----    1    root    plugdev    188,    0    Jul    10    19:04    /
dev/ttyUSB0
   crw-rw----    1    root    plugdev    188,    1    Jul    10    19:04    /
dev/ttyUSB1
```

   Die `/dev/ttyUSBn`-Schnittstelle mit der niedrigeren Zahl wird für die JTAG-Kommunikation verwendet. Die andere Schnittstelle wird an die serielle Schnittstelle (UART) ESP32 von weitergeleitet und dient zum Hochladen von Code in den Flash-Speicher ESP32 von.

5. Navigieren Sie in einem Terminalfenster zum Stammverzeichnis Ihres FreeRTOS-Download-Verzeichnisses und verwenden Sie den folgenden Befehl, um OpenOCD auszuführen.

```
idf.py openocd
```

6. Öffnen Sie ein anderes Terminal, navigieren Sie zum Stammverzeichnis Ihres FreeRTOS-Download-Verzeichnisses und führen Sie den folgenden Befehl aus.

   ```
   idf.py flash monitor
   ```

7. Öffnen Sie ein anderes Terminal, navigieren Sie zum Stammverzeichnis Ihres FreeRTOS-Download-Verzeichnisses und führen Sie den folgenden Befehl aus:

   ```
   idf.py gdb
   ```

   Das Programm sollte bei `main()` stoppen.

Erste Schritte mit dem ESP32 Espressif -WROOM-32SE

Important

Diese Referenzintegration wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Note

- Um zu erfahren, wie Sie modulare FreeRTOS-Bibliotheken und -Demos in Ihr eigenes Espressif-IDF-Projekt integrieren können, schauen Sie sich unsere vorgestellte Referenzintegration für die -C3-Plattform an. ESP32
- Derzeit unterstützt der FreeRTOS-Port für ESP32 -WROOM-32SE die Funktion Symmetric Multiprocessing (SMP) nicht.

Dieses Tutorial zeigt Ihnen, wie Sie mit dem Espressif -WROOM-32SE beginnen. ESP32 Informationen zum Kauf eines Geräts bei unserem Partner im Partnergerätekatalog finden Sie unter -WROOM-32SE AWS . ESP32

-Übersicht

In diesem Tutorial führen Sie die folgenden Schritte aus:

1. Verbinden Sie Ihr Board mit einem Host-Rechner.
2. Installieren Sie Software zum Entwickeln und Debuggen eingebetteter Anwendungen für Ihr Mikrocontroller-Board auf Ihrem Host-Computer.
3. Cross-Kompilieren Sie eine FreeRTOS-Demo-Anwendung zu einem Binär-Image.
4. Laden des binären Anwendungs-Image auf Ihre Platine und Ausführen der Anwendung.
5. Überwachen und debuggen Sie die laufende Anwendung mithilfe einer seriellen Verbindung.

Voraussetzungen

Bevor Sie mit FreeRTOS auf Ihrem Espressif-Board beginnen, müssen Sie Ihr Konto und Ihre AWS Berechtigungen einrichten.

Melde dich an für ein AWS-Konto

Wenn Sie noch keine haben AWS-Konto, führen Sie die folgenden Schritte aus, um eine zu erstellen.

Um sich für eine anzumelden AWS-Konto

1. Öffnen Sie https://portal.aws.amazon.com/billing/die Anmeldung.
2. Folgen Sie den Online-Anweisungen.

   Während der Anmeldung erhalten Sie einen Telefonanruf oder eine Textnachricht und müssen einen Verifizierungscode über die Telefontasten eingeben.

   Wenn Sie sich für eine anmelden AWS-Konto, Root-Benutzer des AWS-Kontoswird eine erstellt. Der Root-Benutzer hat Zugriff auf alle AWS-Services und Ressourcen des Kontos. Als bewährte Sicherheitsmethode weisen Sie einem Benutzer Administratorzugriff zu und verwenden Sie nur den Root-Benutzer, um Aufgaben auszuführen, die Root-Benutzerzugriff erfordern.

AWS sendet Ihnen nach Abschluss des Anmeldevorgangs eine Bestätigungs-E-Mail. Du kannst jederzeit deine aktuellen Kontoaktivitäten einsehen und dein Konto verwalten, indem du zu https://aws.amazon.com/gehst und Mein Konto auswählst.

Erstellen eines Benutzers mit Administratorzugriff

Nachdem Sie sich für einen angemeldet haben AWS-Konto, sichern Sie Ihren Root-Benutzer des AWS-Kontos AWS IAM Identity Center, aktivieren und erstellen Sie einen Administratorbenutzer, sodass Sie den Root-Benutzer nicht für alltägliche Aufgaben verwenden.

Sichern Sie Ihre Root-Benutzer des AWS-Kontos

1. Melden Sie sich AWS-Managementkonsoleals Kontoinhaber an, indem Sie Root-Benutzer auswählen und Ihre AWS-Konto E-Mail-Adresse eingeben. Geben Sie auf der nächsten Seite Ihr Passwort ein.

   Hilfe bei der Anmeldung mit dem Root-Benutzer finden Sie unter Anmelden als Root-Benutzer im AWS-Anmeldung -Benutzerhandbuch zu.
2. Aktivieren Sie die Multi-Faktor-Authentifizierung (MFA) für den Root-Benutzer.

   Anweisungen finden Sie unter Aktivieren eines virtuellen MFA-Geräts für Ihren AWS-Konto Root-Benutzer (Konsole) im IAM-Benutzerhandbuch.

Erstellen eines Benutzers mit Administratorzugriff

1. Aktivieren Sie das IAM Identity Center.

   Anweisungen finden Sie unter Aktivieren AWS IAM Identity Center im AWS IAM Identity Center -Benutzerhandbuch.
2. Gewähren Sie einem Administratorbenutzer im IAM Identity Center Benutzerzugriff.

   Ein Tutorial zur Verwendung von IAM-Identity-Center-Verzeichnis als Identitätsquelle finden Sie IAM-Identity-Center-Verzeichnis im Benutzerhandbuch unter Benutzerzugriff mit der Standardeinstellung konfigurieren.AWS IAM Identity Center

Anmelden als Administratorbenutzer

- Um sich mit Ihrem IAM-Identity-Center-Benutzer anzumelden, verwenden Sie die Anmelde-URL, die an Ihre E-Mail-Adresse gesendet wurde, als Sie den IAM-Identity-Center-Benutzer erstellt haben.

  Hilfe bei der Anmeldung mit einem IAM Identity Center-Benutzer finden Sie im AWS-Anmeldung Benutzerhandbuch unter Anmeldung beim AWS Access-Portal.

Weiteren Benutzern Zugriff zuweisen

1. Erstellen Sie im IAM-Identity-Center einen Berechtigungssatz, der den bewährten Vorgehensweisen für die Anwendung von geringsten Berechtigungen folgt.

   Anweisungen hierzu finden Sie unter Berechtigungssatz erstellen im AWS IAM Identity Center - Benutzerhandbuch.
2. Weisen Sie Benutzer einer Gruppe zu und weisen Sie der Gruppe dann Single Sign-On-Zugriff zu.

   Eine genaue Anleitung finden Sie unter Gruppen hinzufügen im AWS IAM Identity Center - Benutzerhandbuch.

Um Zugriff zu gewähren, fügen Sie Ihren Benutzern, Gruppen oder Rollen Berechtigungen hinzu:

- Benutzer und Gruppen in AWS IAM Identity Center:

  Erstellen Sie einen Berechtigungssatz. Befolgen Sie die Anweisungen unter Erstellen eines Berechtigungssatzes im AWS IAM Identity Center -Benutzerhandbuch.
- Benutzer, die in IAM über einen Identitätsanbieter verwaltet werden:

  Erstellen Sie eine Rolle für den Identitätsverbund. Befolgen Sie die Anleitung unter Eine Rolle für einen externen Identitätsanbieter (Verbund) erstellen im IAM-Benutzerhandbuch.
- IAM-Benutzer:
  - Erstellen Sie eine Rolle, die Ihr Benutzer annehmen kann. Befolgen Sie die Anleitung unter Eine Rolle für einen IAM-Benutzer erstellen im IAM-Benutzerhandbuch.

- (Nicht empfohlen) Weisen Sie einem Benutzer eine Richtlinie direkt zu oder fügen Sie einen Benutzer zu einer Benutzergruppe hinzu. Befolgen Sie die Anweisungen unter Hinzufügen von Berechtigungen zu einem Benutzer (Konsole) im IAM-Benutzerhandbuch.

Erste Schritte

> **Note**
>
> Die Linux-Befehle in diesem Tutorial erfordern, dass Sie die Bash-Shell verwenden.

1. Richten Sie die Espressif-Hardware ein.

   Informationen zur Einrichtung der Hardware für das ESP32 -WROOM-32SE Development-Board finden Sie im Handbuch - C V4 Getting Started Guide. ESP32 DevKit

   > **Important**
   >
   > Wenn Sie den Abschnitt „Installation Schritt für Schritt“ des Handbuchs erreicht haben, folgen Sie diesen Anweisungen, bis Sie Schritt 4 (Umgebungsvariablen einrichten) abgeschlossen haben. Hören Sie auf, nachdem Sie Schritt 4 abgeschlossen haben, und folgen Sie den verbleibenden Schritten hier.

2. Laden Sie Amazon FreeRTOS von herunter. GitHub (Anweisungen finden Sie in der README.md-Datei.)
3. Richten Sie Ihre Entwicklungsumgebung ein.

   Um mit Ihrem Board zu kommunizieren, müssen Sie eine Toolchain installieren. Espressif stellt die ESP-IDF zur Verfügung, um Software für ihre Boards zu entwickeln. Da die ESP-IDF eine eigene Version des FreeRTOS-Kernels als Komponente integriert hat, enthält Amazon FreeRTOS eine benutzerdefinierte Version von ESP-IDF v4.2, bei der der FreeRTOS-Kernel entfernt wurde. Dies behebt Probleme mit doppelten Dateien beim Kompilieren. Um die in Amazon FreeRTOS enthaltene benutzerdefinierte Version von ESP-IDF v4.2 zu verwenden, folgen Sie den nachstehenden Anweisungen für das Betriebssystem Ihres Host-Computers.

   Windows

   1. Laden Sie den Universal Online Installer von ESP-IDF für Windows herunter.

2. Führen Sie den Universal Online Installer aus.
3. Wenn Sie zum Schritt ESP-IDF herunterladen oder verwenden gelangen, wählen Sie Bestehendes ESP-IDF-Verzeichnis verwenden und setzen Sie Vorhandenes ESP-IDF-Verzeichnis auswählen auf. *freertos*/vendors/espressif/esp-idf
4. Schließen Sie die Installation ab.

macOS

1. Folgen Sie den Anweisungen in den Voraussetzungen für die Standardkonfiguration der Toolchain (ESP-IDF v4.2) für macOS.

   > **Important**
   >
   > Wenn Sie unter Nächste Schritte zu den Anweisungen „Get ESP-IDF“ gelangen, beenden Sie den Vorgang und kehren Sie dann zu den Anweisungen auf dieser Seite zurück.

2. Öffnen Sie ein Befehlszeilenfenster.
3. Navigieren Sie zum FreeRTOS-Download-Verzeichnis und führen Sie dann das folgende Skript aus, um die espressif-Toolchain für Ihre Plattform herunterzuladen und zu installieren.

   ```
   vendors/espressif/esp-idf/install.sh
   ```

4. Fügen Sie die ESP-IDF-Toolketten-Tools mit dem folgenden Befehl zum Pfad Ihres Terminals hinzu.

   ```
   source vendors/espressif/esp-idf/export.sh
   ```

Linux

1. Folgen Sie den Anweisungen in den Voraussetzungen für die Standardkonfiguration der Toolchain (ESP-IDF v4.2) für Linux.

> **Important**
>
> Wenn Sie unter „Nächste Schritte“ zu den Anweisungen „Get ESP-IDF“ gelangen, beenden Sie den Vorgang und kehren Sie dann zu den Anweisungen auf dieser Seite zurück.

2. Öffnen Sie ein Befehlszeilenfenster.
3. Navigieren Sie zum FreeRTOS-Download-Verzeichnis und führen Sie dann das folgende Skript aus, um die Espressif-Toolchain für Ihre Plattform herunterzuladen und zu installieren.

```
vendors/espressif/esp-idf/install.sh
```

4. Fügen Sie die ESP-IDF-Toolketten-Tools mit dem folgenden Befehl zum Pfad Ihres Terminals hinzu.

```
source vendors/espressif/esp-idf/export.sh
```

4. Stellen Sie eine serielle Verbindung her.

   a. Um eine serielle Verbindung zwischen Ihrem Host-Computer und dem ESP32 - WROOM-32SE herzustellen, installieren Sie die CP210x USB-zu-UART-Bridge-VCP-Treiber. Sie können diese Treiber von Silicon Labs herunterladen.

   b. Folgen Sie den Schritten zum Herstellen einer seriellen Verbindung mit. ESP32

   c. Nachdem Sie eine serielle Verbindung hergestellt haben, notieren Sie sich den seriellen Port für Ihre Board-Verbindung. Sie benötigen es, um die Demo zu flashen.

Konfigurieren Sie die FreeRTOS-Demoanwendungen

Für dieses Tutorial befindet sich die FreeRTOS-Konfigurationsdatei unter. *freertos*`/vendors/espressif/boards/`*board-name*`/aws_demos/config_files/FreeRTOSConfig.h` (Wenn zum Beispiel ausgewählt `AFR_BOARD espressif.esp32_devkitc` ist, befindet sich die Konfigurationsdatei unter*freertos*`/vendors/espressif/boards/esp32/aws_demos/config_files/FreeRTOSConfig.h`.)

> **Important**
>
> Das ATECC608 A-Gerät verfügt über eine einmalige Initialisierung, die bei der ersten Ausführung eines Projekts (während des Aufrufs von`C_InitToken`) auf dem Gerät gesperrt wird. Das FreeRTOS-Demoprojekt und das Testprojekt haben jedoch unterschiedliche Konfigurationen. Wenn das Gerät während der Demo-Projektkonfigurationen gesperrt ist, sind nicht alle Tests im Testprojekt erfolgreich.

1. Konfigurieren Sie das FreeRTOS-Demo-Projekt, indem Sie den Schritten unter folgen. Konfiguration der FreeRTOS-Demos Wenn Sie mit dem letzten Schritt zum Formatieren Ihrer AWS IoT Anmeldeinformationen fertig sind, beenden Sie den Vorgang und führen Sie die folgenden Schritte aus.
2. Microchip hat mehrere Scripting-Tools zur Verfügung gestellt, die bei der Einrichtung der ATECC608 A-Teile helfen. Navigieren Sie zu dem Verzeichnis *`freertos`*`/vendors/microchip/example_trust_chain_tool` und öffnen Sie die Datei `README.md`.
3. Folgen Sie den Anweisungen in der Datei, um Ihr Gerät bereitzustellen. `README.md` Die Schritte umfassen Folgendes:

    1. Erstellen und registrieren Sie eine Zertifizierungsstelle bei AWS.
    2. Generieren Sie Ihre Schlüssel auf dem ATECC608 A und exportieren Sie den öffentlichen Schlüssel und die Seriennummer des Geräts.
    3. Generieren Sie ein Zertifikat für das Gerät und registrieren Sie dieses Zertifikat bei AWS.
4. Laden Sie das Zertifizierungsstellenzertifikat und das Gerätezertifikat auf das Gerät, indem Sie die Anweisungen für Schlüsselbereitstellung im Entwicklermodus befolgen.

Überwachung von MQTT-Nachrichten in der Cloud AWS

Bevor Sie das FreeRTOS-Demoprojekt ausführen, können Sie den MQTT-Client in der AWS IoT Konsole so einrichten, dass er die Nachrichten überwacht, die Ihr Gerät an die Cloud sendet. AWS

Um das MQTT-Thema mit dem MQTT-Client zu abonnieren AWS IoT

1. Melden Sie sich bei der AWS IoT -Konsole an.
2. Wählen Sie im Navigationsbereich Test und dann MQTT Test Client aus.

3. Geben Sie unter Thema Abonnement den Text ein *your-thing-name*`/example/topic` und wählen Sie dann Thema abonnieren aus.

Erstellen, flashen und starten Sie das FreeRTOS-Demoprojekt mit dem Skript idf.py

Sie können das IDF-Hilfsprogramm (`idf.py`) von Espressif verwenden, um die Build-Dateien zu generieren, die Anwendungs-Binärdatei zu erstellen und die Binärdateien auf Ihr Gerät zu flashen.

> **Note**
>
> Bei einigen Setups müssen Sie möglicherweise die Portoption "`-p port-name`" mit verwenden, `idf.py` um den richtigen Port anzugeben, wie im folgenden Beispiel.
>
> ```
> idf.py -p /dev/cu.usbserial-00101301B flash
> ```

FreeRTOS unter Windows, Linux und macOS erstellen und flashen (ESP-IDF v4.2)

1. Navigieren Sie zum Stammverzeichnis Ihres FreeRTOS-Download-Verzeichnisses.
2. Geben Sie in einem Befehlszeilenfenster den folgenden Befehl ein, um die ESP-IDF-Tools zum PATH Ihres Terminals hinzuzufügen:

   Windows (App „Command“)

   ```
   vendors\espressif\esp-idf\export.bat
   ```

   Windows (App „ESP-IDF 4.x CMD“)

   (Dies wurde bereits getan, als Sie die App geöffnet haben.)

   Linux//macOS

   ```
   source vendors/espressif/esp-idf/export.sh
   ```

3. Konfigurieren Sie cmake im `build` Verzeichnis und erstellen Sie das Firmware-Image mit dem folgenden Befehl.

   ```
   idf.py -DVENDOR=espressif -DBOARD=esp32_ecc608a_devkitc -DCOMPILER=xtensa-esp32
    build
   ```

Sie sollten eine Ausgabe wie das folgende Beispiel sehen.

```
   Running cmake in directory /path/to/hello_world/build
   Executing "cmake -G Ninja --warn-uninitialized /path/to/hello_world"...
   Warn about uninitialized values.
   -- Found Git: /usr/bin/git (found version "2.17.0")
   -- Building empty aws_iot component due to configuration
   -- Component names: ...
   -- Component paths: ...

   ... (more lines of build system output)

   [527/527] Generating hello-world.bin
   esptool.py v2.3.1

   Project build complete. To flash, run this command:
   ../../../components/esptool_py/esptool/esptool.py -p (PORT) -b 921600
 write_flash --flash_mode dio --flash_size detect --flash_freq 40m 0x10000 build/
hello-world.bin  build 0x1000 build/bootloader/bootloader.bin 0x8000 build/
partition_table/partition-table.bin
   or run 'idf.py -p PORT flash'
```

Wenn keine Fehler vorliegen, generiert der Build die binären Firmware-Dateien.

4. Löschen Sie den Flash-Speicher Ihres Entwicklungsboards mit dem folgenden Befehl.

```
idf.py erase_flash
```

5. Verwenden Sie das `idf.py` Skript, um die Binärdatei der Anwendung auf Ihr Board zu flashen.

```
idf.py flash
```

6. Überwachen Sie die Ausgabe von der seriellen Schnittstelle Ihrer Platine mit dem folgenden Befehl.

```
idf.py monitor
```

> **Note**
>
> - Sie können diese Befehle wie im folgenden Beispiel kombinieren.

```
idf.py erase_flash flash monitor
```

- Bei bestimmten Host-Rechner-Setups müssen Sie den Port angeben, wenn Sie die Karte flashen, wie im folgenden Beispiel gezeigt.

```
idf.py erase_flash flash monitor -p /dev/ttyUSB1
```

FreeRTOS erstellen und flashen mit CMake

Sie können nicht nur das vom IDF SDK bereitgestellte `idf.py` Skript verwenden, um Ihren Code zu erstellen und auszuführen, sondern auch das Projekt damit erstellen. CMake Derzeit unterstützt es Unix Makefile und das Ninja Build-System.

Um das Projekt zu erstellen und zu flashen

1. Navigieren Sie in einem Befehlszeilenfenster zum Stammverzeichnis Ihres FreeRTOS-Download-Verzeichnisses.
2. Führen Sie das folgende Skript aus, um die ESP-IDF-Tools zum PATH Ihrer Shell hinzuzufügen.

   Windows

   ```
   vendors\espressif\esp-idf\export.bat
   ```

   Linux//macOS

   ```
   source vendors/espressif/esp-idf/export.sh
   ```

3. Geben Sie den folgenden Befehl ein, um die Build-Dateien zu generieren.

   Mit Unix-Makefiles

   ```
   cmake -DVENDOR=espressif -DBOARD=esp32_plus_ecc608a_devkitc -DCOMPILER=xtensa-
   esp32 -S . -B ./YOUR_BUILD_DIRECTORY -DAFR_ENABLE_ALL_MODULES=1 -
   DAFR_ENABLE_TESTS=0
   ```

Mit Ninja

```
cmake -DVENDOR=espressif -DBOARD=esp32_plus_ecc608a_devkitc -DCOMPILER=xtensa-
esp32 -S . -B ./YOUR_BUILD_DIRECTORY -DAFR_ENABLE_ALL_MODULES=1 -
DAFR_ENABLE_TESTS=0 -GNinja
```

4. Lösche den Blitz und flashe dann die Platine.

   Mit Unix-Makefiles

   ```
   make -C ./YOUR_BUILD_DIRECTORY erase_flash
   ```

   ```
   make -C ./YOUR_BUILD_DIRECTORY flash
   ```

   Mit Ninja

   ```
   ninja -C ./YOUR_BUILD_DIRECTORY erase_flash
   ```

   ```
   ninja -C ./YOUR_BUILD_DIRECTORY flash
   ```

Zusätzliche Informationen

Weitere Informationen zur Verwendung und Problembehebung von ESP32 Espressif-Boards finden Sie in den folgenden Themen:

- Verwenden Sie FreeRTOS in Ihrem eigenen CMake Projekt für ESP32
- Fehlerbehebung
- Debuggen

Erste Schritte mit dem Espressif -S2 ESP32

> **Important**
>
> Diese Referenzintegration wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten

Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

> **Note**
>
> Um zu erfahren, wie Sie modulare FreeRTOS-Bibliotheken und -Demos in Ihr eigenes Espressif-IDF-Projekt integrieren können, schauen Sie sich unsere vorgestellte Referenzintegration für die -C3-Plattform an. ESP32

In diesem Tutorial erfahren Sie, wie Sie mit den Espressif -S2 SoC- und ESP32 -S2-Saola-1-Entwicklungsboards beginnen. ESP32

-Übersicht

In diesem Tutorial führen Sie die folgenden Schritte aus:

1. Verbinden Sie Ihr Board mit einem Host-Rechner.
2. Installieren Sie Software zum Entwickeln und Debuggen eingebetteter Anwendungen für Ihr Mikrocontroller-Board auf Ihrem Host-Computer.
3. Eine FreeRTOS-Demo-Anwendung zu einem Binär-Image querkompilieren.
4. Laden des binären Anwendungs-Image auf Ihre Platine und Ausführen der Anwendung.
5. Überwachen und debuggen Sie die laufende Anwendung über eine serielle Verbindung.

Voraussetzungen

Bevor Sie mit FreeRTOS auf Ihrem Espressif-Board beginnen, müssen Sie Ihr Konto und Ihre AWS Berechtigungen einrichten.

Melde dich an für ein AWS-Konto

Wenn Sie noch keine haben AWS-Konto, führen Sie die folgenden Schritte aus, um eine zu erstellen.

Um sich für eine anzumelden AWS-Konto

1. Öffnen Sie https://portal.aws.amazon.com/billing/die Anmeldung.
2. Folgen Sie den Online-Anweisungen.

Während der Anmeldung erhalten Sie einen Telefonanruf oder eine Textnachricht und müssen einen Verifizierungscode über die Telefontasten eingeben.

Wenn Sie sich für eine anmelden AWS-Konto, Root-Benutzer des AWS-Kontoswird eine erstellt. Der Root-Benutzer hat Zugriff auf alle AWS-Services und Ressourcen des Kontos. Als bewährte Sicherheitsmethode weisen Sie einem Benutzer Administratorzugriff zu und verwenden Sie nur den Root-Benutzer, um Aufgaben auszuführen, die Root-Benutzerzugriff erfordern.

AWS sendet Ihnen nach Abschluss des Anmeldevorgangs eine Bestätigungs-E-Mail. Du kannst jederzeit deine aktuellen Kontoaktivitäten einsehen und dein Konto verwalten, indem du zu https://aws.amazon.com/gehst und Mein Konto auswählst.

Erstellen eines Benutzers mit Administratorzugriff

Nachdem Sie sich für einen angemeldet haben AWS-Konto, sichern Sie Ihren Root-Benutzer des AWS-Kontos AWS IAM Identity Center, aktivieren und erstellen Sie einen Administratorbenutzer, sodass Sie den Root-Benutzer nicht für alltägliche Aufgaben verwenden.

Sichern Sie Ihre Root-Benutzer des AWS-Kontos

1. Melden Sie sich AWS-Managementkonsoleals Kontoinhaber an, indem Sie Root-Benutzer auswählen und Ihre AWS-Konto E-Mail-Adresse eingeben. Geben Sie auf der nächsten Seite Ihr Passwort ein.

   Hilfe bei der Anmeldung mit dem Root-Benutzer finden Sie unter Anmelden als Root-Benutzer im AWS-Anmeldung -Benutzerhandbuch zu.
2. Aktivieren Sie die Multi-Faktor-Authentifizierung (MFA) für den Root-Benutzer.

   Anweisungen finden Sie unter Aktivieren eines virtuellen MFA-Geräts für Ihren AWS-Konto Root-Benutzer (Konsole) im IAM-Benutzerhandbuch.

Erstellen eines Benutzers mit Administratorzugriff

1. Aktivieren Sie das IAM Identity Center.

   Anweisungen finden Sie unter Aktivieren AWS IAM Identity Center im AWS IAM Identity Center -Benutzerhandbuch.
2. Gewähren Sie einem Administratorbenutzer im IAM Identity Center Benutzerzugriff.

Ein Tutorial zur Verwendung von IAM-Identity-Center-Verzeichnis als Identitätsquelle finden Sie IAM-Identity-Center-Verzeichnis im Benutzerhandbuch unter Benutzerzugriff mit der Standardeinstellung konfigurieren.AWS IAM Identity Center

Anmelden als Administratorbenutzer

- Um sich mit Ihrem IAM-Identity-Center-Benutzer anzumelden, verwenden Sie die Anmelde-URL, die an Ihre E-Mail-Adresse gesendet wurde, als Sie den IAM-Identity-Center-Benutzer erstellt haben.

  Hilfe bei der Anmeldung mit einem IAM Identity Center-Benutzer finden Sie im AWS-Anmeldung Benutzerhandbuch unter Anmeldung beim AWS Access-Portal.

Weiteren Benutzern Zugriff zuweisen

1. Erstellen Sie im IAM-Identity-Center einen Berechtigungssatz, der den bewährten Vorgehensweisen für die Anwendung von geringsten Berechtigungen folgt.

   Anweisungen hierzu finden Sie unter Berechtigungssatz erstellen im AWS IAM Identity Center - Benutzerhandbuch.
2. Weisen Sie Benutzer einer Gruppe zu und weisen Sie der Gruppe dann Single Sign-On-Zugriff zu.

   Eine genaue Anleitung finden Sie unter Gruppen hinzufügen im AWS IAM Identity Center - Benutzerhandbuch.

Um Zugriff zu gewähren, fügen Sie Ihren Benutzern, Gruppen oder Rollen Berechtigungen hinzu:

- Benutzer und Gruppen in AWS IAM Identity Center:

  Erstellen Sie einen Berechtigungssatz. Befolgen Sie die Anweisungen unter Erstellen eines Berechtigungssatzes im AWS IAM Identity Center -Benutzerhandbuch.
- Benutzer, die in IAM über einen Identitätsanbieter verwaltet werden:

  Erstellen Sie eine Rolle für den Identitätsverbund. Befolgen Sie die Anleitung unter Eine Rolle für einen externen Identitätsanbieter (Verbund) erstellen im IAM-Benutzerhandbuch.
- IAM-Benutzer:

- Erstellen Sie eine Rolle, die Ihr Benutzer annehmen kann. Befolgen Sie die Anleitung unter Eine Rolle für einen IAM-Benutzer erstellen im IAM-Benutzerhandbuch.
- (Nicht empfohlen) Weisen Sie einem Benutzer eine Richtlinie direkt zu oder fügen Sie einen Benutzer zu einer Benutzergruppe hinzu. Befolgen Sie die Anweisungen unter Hinzufügen von Berechtigungen zu einem Benutzer (Konsole) im IAM-Benutzerhandbuch.

Erste Schritte

> **Note**
>
> Die Linux-Befehle in diesem Tutorial setzen voraus, dass Sie die Bash-Shell verwenden.

1. Richten Sie die Espressif-Hardware ein.

   Informationen zur Einrichtung der Hardware für das ESP32 -S2-Entwicklungsboard finden Sie im -S2-Saola-1 Getting ESP32Started Guide.

   > **Important**
   >
   > Wenn Sie den Abschnitt „Erste Schritte“ der Espressif-Anleitungen erreicht haben, halten Sie an und kehren Sie dann zu den Anweisungen auf dieser Seite zurück.

2. Laden Sie Amazon FreeRTOS von herunter. GitHub (Anweisungen finden Sie in der README.md-Datei.)
3. Richten Sie Ihre Entwicklungsumgebung ein.

   Um mit Ihrem Board zu kommunizieren, müssen Sie eine Toolchain installieren. Espressif stellt die ESP-IDF zur Verfügung, um Software für ihre Boards zu entwickeln. Da die ESP-IDF eine eigene Version des FreeRTOS-Kernels als Komponente integriert hat, enthält Amazon FreeRTOS eine benutzerdefinierte Version von ESP-IDF v4.2, bei der der FreeRTOS-Kernel entfernt wurde. Dies behebt Probleme mit doppelten Dateien beim Kompilieren. Um die in Amazon FreeRTOS enthaltene benutzerdefinierte Version von ESP-IDF v4.2 zu verwenden, folgen Sie den nachstehenden Anweisungen für das Betriebssystem Ihres Host-Computers.

   Windows

   1. Laden Sie den Universal Online Installer von ESP-IDF für Windows herunter.

2. Führen Sie den Universal Online Installer aus.
3. Wenn Sie zum Schritt ESP-IDF herunterladen oder verwenden gelangen, wählen Sie Vorhandenes ESP-IDF-Verzeichnis verwenden und setzen Sie Vorhandenes ESP-IDF-Verzeichnis auswählen auf. *freertos*/vendors/espressif/esp-idf
4. Schließen Sie die Installation ab.

macOS

1. Folgen Sie den Anweisungen in den Voraussetzungen für die Standardkonfiguration der Toolchain (ESP-IDF v4.2) für macOS.

   > **Important**
   >
   > Wenn Sie unter Nächste Schritte zu den Anweisungen „Get ESP-IDF“ gelangen, beenden Sie den Vorgang und kehren Sie dann zu den Anweisungen auf dieser Seite zurück.

2. Öffnen Sie ein Befehlszeilenfenster.
3. Navigieren Sie zum FreeRTOS-Download-Verzeichnis und führen Sie dann das folgende Skript aus, um die espressif-Toolchain für Ihre Plattform herunterzuladen und zu installieren.

   ```
   vendors/espressif/esp-idf/install.sh
   ```

4. Fügen Sie die ESP-IDF-Toolketten-Tools mit dem folgenden Befehl zum Pfad Ihres Terminals hinzu.

   ```
   source vendors/espressif/esp-idf/export.sh
   ```

Linux

1. Folgen Sie den Anweisungen in den Voraussetzungen für die Standardkonfiguration der Toolchain (ESP-IDF v4.2) für Linux.

> **Important**
>
> Wenn Sie unter „Nächste Schritte“ zu den Anweisungen „Get ESP-IDF“ gelangen, beenden Sie den Vorgang und kehren Sie dann zu den Anweisungen auf dieser Seite zurück.

2. Öffnen Sie ein Befehlszeilenfenster.
3. Navigieren Sie zum FreeRTOS-Download-Verzeichnis und führen Sie dann das folgende Skript aus, um die Espressif-Toolchain für Ihre Plattform herunterzuladen und zu installieren.

```
vendors/espressif/esp-idf/install.sh
```

4. Fügen Sie die ESP-IDF-Toolketten-Tools mit dem folgenden Befehl zum Pfad Ihres Terminals hinzu.

```
source vendors/espressif/esp-idf/export.sh
```

4. Stellen Sie eine serielle Verbindung her.

   a. Um eine serielle Verbindung zwischen Ihrem Host-Computer und dem ESP32 - DevKit C herzustellen, installieren Sie die CP210x USB-zu-UART-Bridge-VCP-Treiber. Sie können diese Treiber von Silicon Labs herunterladen.

   b. Folgen Sie den Schritten zum Herstellen einer seriellen Verbindung mit. ESP32

   c. Nachdem Sie eine serielle Verbindung hergestellt haben, notieren Sie sich den seriellen Port für Ihre Board-Verbindung. Sie benötigen es, um die Demo zu flashen.

Konfigurieren Sie die FreeRTOS-Demoanwendungen

Für dieses Tutorial befindet sich die FreeRTOS-Konfigurationsdatei unter. *freertos*`/vendors/espressif/boards/`*board-name*`/aws_demos/config_files/FreeRTOSConfig.h` (Wenn zum Beispiel ausgewählt `AFR_BOARD espressif.esp32_devkitc` ist, befindet sich die Konfigurationsdatei unter*freertos*`/vendors/espressif/boards/esp32/aws_demos/config_files/FreeRTOSConfig.h`.)

1. Wenn Sie macOS oder Linux verwenden, öffnen Sie eine Terminal-Eingabeaufforderung. Wenn Sie Windows verwenden, öffnen Sie die App „ESP-IDF 4.x CMD“ (falls Sie diese Option bei der

Installation der ESP-IDF-Toolchain angegeben haben) oder andernfalls die App „Command Prompt“.

2. Um zu überprüfen, ob Sie Python3 installiert haben, führen Sie Folgendes aus:

```
python --version
```

Die installierte Version wird angezeigt. Wenn Sie Python 3.0.1 oder höher nicht installiert haben, können Sie es von der Python-Website installieren.

3. Sie benötigen die AWS Befehlszeilenschnittstelle (CLI), um AWS IoT Befehle auszuführen. Wenn Sie Windows verwenden, verwenden Sie den `easy_install awscli` Befehl, um die AWS CLI in der App „Command“ oder „ESP-IDF 4.x CMD“ zu installieren.

Wenn Sie macOS oder Linux verwenden, finden Sie weitere Informationen unter Installation der AWS CLI.

4. Ausführen

```
aws configure
```

und konfigurieren Sie die AWS CLI mit Ihrer AWS Zugriffsschlüssel-ID, Ihrem geheimen Zugriffsschlüssel und Ihrer AWS Standardregion. Weitere Informationen finden Sie unter Konfigurieren der AWS -CLI.

5. Verwenden Sie den folgenden Befehl, um das AWS SDK für Python (boto3) zu installieren:

- Führen Sie unter Windows in der App „Command“ oder „ESP-IDF 4.x CMD“ den folgenden Befehl aus

```
easy_install boto3
```

- Führen Sie unter macOS oder Linux Folgendes aus

```
pip install tornado nose --user
```

und dann ausführen

```
pip install boto3 --user
```

FreeRTOS enthält das `SetupAWS.py` Skript, mit dem Sie Ihr Espressif-Board für die Verbindung einfacher einrichten können. AWS IoT

So führen Sie das Konfigurationsskript aus:

1. Wenn Sie das Skript konfigurieren möchten, öffnen Sie *`freertos`*`/tools/aws_config_quick_start/configure.json` und legen die folgenden Attribute fest:

   `afr_source_dir`

   Der vollständige Pfad zum *`freertos`*-Verzeichnis auf Ihrem Computer. Stellen Sie sicher, dass Sie diesen Pfad mit Schrägstrichen angeben.

   `thing_name`

   Der Name, den Sie dem AWS IoT Ding zuweisen möchten, das Ihr Board repräsentiert.

   `wifi_ssid`

   Die SSID Ihres WLANs.

   `wifi_password`

   Das Passwort für Ihr WLAN-Netzwerk

   `wifi_security`

   Der Sicherheitstyp für Ihr WLAN-Netzwerk Die folgenden Sicherheitstypen sind gültig:

   - `eWiFiSecurityOpen` (Open, no security (Offen, keine Sicherheit)
   - `eWiFiSecurityWEP` (WEP-Sicherheit)
   - `eWiFiSecurityWPA` (WPA-Sicherheit)
   - `eWiFiSecurityWPA2`(WPA2 Sicherheit)
2. Wenn Sie macOS oder Linux verwenden, öffnen Sie eine Terminal-Eingabeaufforderung. Wenn Sie Windows verwenden, öffnen Sie die App „ESP-IDF 4.x CMD“ oder „Command“.
3. Navigiere zum Verzeichnis und führe es aus *`freertos`*`/tools/aws_config_quick_start`

   ```
   python SetupAWS.py setup
   ```

   Das -Skript führt folgende Aktionen aus:

   - Erstellt eine AWS IoT Sache, ein Zertifikat und eine Richtlinie.

- Hängt die AWS IoT Richtlinie an das Zertifikat und das Zertifikat an die AWS IoT Sache an.
- Füllt die `aws_clientcredential.h` Datei mit Ihrem AWS IoT Endpunkt, Ihrer Wi-Fi-SSID und Ihren Anmeldeinformationen auf.
- Formatiert Ihr Zertifikat und Ihren privaten Schlüssel und schreibt sie in die `aws_clientcredential_keys.h` Header-Datei.

> **Note**
>
> Das Zertifikat ist nur zu Demonstrationszwecken hartcodiert. Anwendungen auf Produktionsebene sollten diese Dateien an einem sicheren Ort speichern.

Weitere Informationen dazu `SetupAWS.py` finden Sie `README.md` im *`freertos`*`/tools/aws_config_quick_start` Verzeichnis.

Überwachung von MQTT-Nachrichten in der Cloud AWS

Bevor Sie das FreeRTOS-Demoprojekt ausführen, können Sie den MQTT-Client in der AWS IoT Konsole so einrichten, dass er die Nachrichten überwacht, die Ihr Gerät an die Cloud sendet. AWS

Um das MQTT-Thema mit dem MQTT-Client zu abonnieren AWS IoT

1. Melden Sie sich bei der AWS IoT -Konsole an.
2. Wählen Sie im Navigationsbereich Test und dann MQTT Test Client aus.
3. Geben Sie im Feld Subscription topic (Abonnementthema) die Option *`your-thing-name`*`/example/topic` ein und wählen Sie dann Subscribe to topic (Thema abonnieren).

Wenn das Demo-Projekt erfolgreich auf Ihrem Gerät ausgeführt wird, sehen Sie „Hello World!“ mehrfach zu dem Thema gesendet, das Sie abonniert haben.

Erstellen, flashen und starten Sie das FreeRTOS-Demoprojekt mit dem Skript idf.py

Sie können das IDF-Hilfsprogramm von Espressif verwenden, um die Build-Dateien zu generieren, die Anwendungs-Binärdatei zu erstellen und Ihr Board zu flashen.

FreeRTOS unter Windows, Linux und macOS erstellen und flashen (ESP-IDF v4.2)

Verwenden Sie das `idf.py` Skript, um das Projekt zu erstellen und die Binärdateien auf Ihr Gerät zu flashen.

> **Note**
>
> Bei einigen Setups müssen Sie möglicherweise die Port-Option `-p port-name` mit verwenden, `idf.py` um den richtigen Port anzugeben, wie im folgenden Beispiel.
>
> ```
> idf.py -p /dev/cu.usbserial-00101301B flash
> ```

Um das Projekt zu erstellen und zu flashen

1. Navigieren Sie zum Stammverzeichnis Ihres FreeRTOS-Download-Verzeichnisses.
2. Geben Sie in einem Befehlszeilenfenster den folgenden Befehl ein, um die ESP-IDF-Tools zum PATH Ihres Terminals hinzuzufügen:

   Windows (App „Command“)

   ```
   vendors\espressif\esp-idf\export.bat
   ```

   Windows (App „ESP-IDF 4.x CMD“)

   (Dies wurde bereits getan, als Sie die App geöffnet haben.)

   Linux//macOS

   ```
   source vendors/espressif/esp-idf/export.sh
   ```
3. Konfigurieren Sie cmake im `build` Verzeichnis und erstellen Sie das Firmware-Image mit dem folgenden Befehl.

   ```
   idf.py -DVENDOR=espressif -DBOARD=esp32s2_saola_1 -DCOMPILER=xtensa-esp32s2 build
   ```

   Sie sollten eine Ausgabe wie das folgende Beispiel sehen.

   ```
   Executing action: all (aliases: build)
      Running cmake in directory /path/to/hello_world/build
   ```

```
   Executing "cmake -G Ninja -DPYTHON_DEPS_CHECKED=1 -DESP_PLATFORM=1
 -DVENDOR=espressif -DBOARD=esp32s2_saola_1 -DCOMPILER=xtensa-esp32s2 -
DCCACHE_ENABLE=0 /path/to/hello_world"...
   -- The C compiler identification is GNU 8.4.0
   -- The CXX compiler identification is GNU 8.4.0
   -- The ASM compiler identification is GNU

   ... (more lines of build system output)

   [1628/1628] Generating binary image from built executable
   esptool.py v3.0
   Generated /path/to/hello_world/build/aws_demos.bin

   Project build complete. To flash, run this command:
   esptool.py -p (PORT) -b 460800 --before default_reset --after hard_reset --chip
 esp32s2  write_flash --flash_mode dio --flash_size detect --flash_freq 80m 0x1000
 build/bootloader/bootloader.bin 0x8000 build/partition_table/partition-table.bin
 0x16000 build/ota_data_initial.bin 0x20000 build/aws_demos.bin
   or run 'idf.py -p (PORT) flash'
```

Wenn keine Fehler vorliegen, generiert der Build die binären Firmware-Dateien.

4. Löschen Sie den Flash-Speicher Ihres Entwicklungsboards mit dem folgenden Befehl.

```
idf.py erase_flash
```

5. Verwenden Sie das `idf.py` Skript, um die Binärdatei der Anwendung auf Ihr Board zu flashen.

```
idf.py flash
```

6. Überwachen Sie die Ausgabe von der seriellen Schnittstelle Ihrer Platine mit dem folgenden Befehl.

```
idf.py monitor
```

> **Note**
>
> - Sie können diese Befehle wie im folgenden Beispiel kombinieren.
>
>   ```
>   idf.py erase_flash flash monitor
>   ```

- Bei bestimmten Host-Rechner-Setups müssen Sie den Port angeben, wenn Sie die Karte flashen, wie im folgenden Beispiel gezeigt.

```
idf.py erase_flash flash monitor -p /dev/ttyUSB1
```

FreeRTOS erstellen und flashen mit CMake

Sie können nicht nur das vom IDF SDK bereitgestellte `idf.py` Skript verwenden, um Ihren Code zu erstellen und auszuführen, sondern auch das Projekt damit erstellen. CMake Derzeit unterstützt es Unix Makefile und das Ninja Build-System.

Um das Projekt zu erstellen und zu flashen

1. Navigieren Sie in einem Befehlszeilenfenster zum Stammverzeichnis Ihres FreeRTOS-Download-Verzeichnisses.
2. Führen Sie das folgende Skript aus, um die ESP-IDF-Tools zum PATH Ihrer Shell hinzuzufügen.
   - Windows

     ```
     vendors\espressif\esp-idf\export.bat
     ```

   - Linux//macOS

     ```
     source vendors/espressif/esp-idf/export.sh
     ```

3. Geben Sie den folgenden Befehl ein, um die Build-Dateien zu generieren.
   - Mit Unix-Makefiles

     ```
     cmake -DVENDOR=espressif -DBOARD=esp32s2_saola_1 -DCOMPILER=xtensa-esp32s2 -S . -
     B ./YOUR_BUILD_DIRECTORY -DAFR_ENABLE_ALL_MODULES=1 -DAFR_ENABLE_TESTS=0
     ```

   - Mit Ninja

     ```
     cmake -DVENDOR=espressif -DBOARD=esp32s2_saola_1 -DCOMPILER=xtensa-esp32s2 -S . -
     B ./YOUR_BUILD_DIRECTORY -DAFR_ENABLE_ALL_MODULES=1 -DAFR_ENABLE_TESTS=0 -GNinja
     ```

4. Erstellen Sie das Projekt.

- Mit Unix-Makefiles

```
make -C ./YOUR_BUILD_DIRECTORY -j8
```

- Mit Ninja

```
ninja -C ./YOUR_BUILD_DIRECTORY -j8
```

5. Lösche den Blitz und flashe dann die Platine.

- Mit Unix-Makefiles

```
make -C ./YOUR_BUILD_DIRECTORY erase_flash
```

```
make -C ./YOUR_BUILD_DIRECTORY flash
```

- Mit Ninja

```
ninja -C ./YOUR_BUILD_DIRECTORY erase_flash
```

```
ninja -C ./YOUR_BUILD_DIRECTORY flash
```

Zusätzliche Informationen

Weitere Informationen zur Verwendung und Problembehebung von ESP32 Espressif-Boards finden Sie in den folgenden Themen:

- Verwenden Sie FreeRTOS in Ihrem eigenen CMake Projekt für ESP32
- Fehlerbehebung
- Debuggen

Erste Schritte mit dem Infineon XMC48 00 IoT Connectivity Kit

> **Important**
>
> Diese Referenzintegration wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn

Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Dieses Tutorial enthält Anleitungen für die ersten Schritte mit dem Infineon XMC48 00 IoT Connectivity Kit. Wenn Sie das Infineon XMC48 00 IoT Connectivity Kit nicht haben, besuchen Sie den AWS Partnergerätekatalog, um eines von unserem Partner zu erwerben.

Wenn Sie eine serielle Verbindung mit der Platine herstellen möchten, um Protokollierungs- und Debugging-Informationen anzuzeigen, benötigen Sie zusätzlich zum XMC48 00 IoT Connectivity Kit einen USB/Serial 3,3-V-Wandler. Der CP21 04 ist ein gängiger USB/Serial Konverter, der in Boards wie dem 04 Friend von Adafruit weit verbreitet ist. CP21

Bevor Sie beginnen, müssen Sie FreeRTOS konfigurieren AWS IoT und herunterladen, um Ihr Gerät mit der AWS Cloud zu verbinden. Detaillierte Anweisungen finden Sie unter Erste Schritte. In diesem Tutorial wird der Pfad zum FreeRTOS-Download-Verzeichnis als bezeichnet. *`freertos`*

-Übersicht

Dieses Tutorial enthält Anweisungen für die folgenden ersten Schritte:

1. Installieren von Software auf dem Host-Computer zum Entwickeln und Debuggen eingebetteter Anwendungen für Ihr Mikrocontroller-Board.
2. Cross-Compilierung einer FreeRTOS-Demo-Anwendung zu einem Binär-Image.
3. Laden des binären Anwendungs-Image auf Ihr Board und Ausführen der Anwendung.
4. Interaktion mit der Anwendung, die auf Ihrem Board über eine serielle Verbindung ausgeführt wird, zu Überwachungs- und Debuggingzwecken.

Einrichten Ihrer Entwicklungsumgebung

FreeRTOS verwendet die DAVE-Entwicklungsumgebung von Infineon, um den 00 zu programmieren. XMC48 Bevor Sie beginnen, müssen Sie DAVE und einige J-Link-Treiber herunterladen und installieren, um mit dem On-Board-Debugger zu kommunizieren.

Installieren von DAVE

1. Rufen Sie die Seite von Infineon zum Herunterladen der DAVE-Software auf.

2. Wählen Sie das DAVE-Paket für Ihr Betriebssystem aus und senden Sie Ihre Registrierungsinformationen. Nach der Registrierung mit Infineon sollten Sie eine Bestätigungs-E-Mail mit einem Link zum Herunterladen einer ZIP-Datei erhalten.
3. Laden Sie die ZIP-Datei des DAVE-Pakets (`DAVE_`*`version_os_date`*`.zip`) herunter und entpacken Sie es im Speicherort, in dem Sie DAVE installieren möchten (z. B. `C:\DAVE4`).

> **Note**
>
> Einige Windows-Benutzer, die den Windows Explorer nutzen, haben Probleme beim Entpacken der Datei gemeldet. Wir empfehlen die Verwendung eines Drittanbieter-Programms, z. B. 7-Zip.

4. Wenn Sie DAVE starten möchten, müssen Sie die ausführbare Datei ausführen, die Sie aus dem `DAVE_`*`version_os_date`*`.zip`-Ordner extrahiert haben.

Weitere Informationen finden Sie im DAVE-Schnellstartleitfaden.

Installieren von Segger-J-Link-Treibern

Für die Kommunikation mit der integrierten Debugging-Sonde des XMC48 00 Relax EtherCAT-Boards benötigen Sie die Treiber, die im J-Link Software- und Dokumentationspaket enthalten sind. Sie können das J-Link-Software- und Dokumentationspaket von Seggers auf der Seite mit demJ-Link-Software-Download herunterladen.

Herstellen einer seriellen Verbindung

Das Einrichten einer seriellen Verbindung ist zwar optional, wird aber empfohlen. Eine serielle Verbindung ermöglicht Ihrem Board das Senden von Protokollierungs- und Debugging-Informationen in einer Form, die Sie auf Ihrem Entwicklungscomputer ansehen können.

Die XMC48 00-Demoanwendung verwendet eine serielle UART-Verbindung an den Pins P0.0 und P0.1, die auf dem Siebdruck der XMC48 00 Relax EtherCAT-Karte beschriftet sind. So richten Sie eine serielle Verbindung ein:

1. Connect den mit „RX<P0.0“ bezeichneten Pin mit dem „TX“ -Pin Ihres USB/Serial Konverters.
2. Connect den mit „TX>P0.1“ beschrifteten Pin mit dem „RX“ -Pin Ihres USB/Serial Konverters.
3. Verbinden Sie Ihren Ground-Pin des seriellen Konverters mit einem der Pins mit der Bezeichnung "GND". Die Geräte müssen einen gemeinsamen Ground teilen.

Die Energieversorgung erfolgt über den USB-Debugging-Port. Verbinden Sie also nicht den positiven Spannungspin Ihres seriellen Adapters mit dem Board.

> **Note**
>
> Einige serielle Kabel verwenden ein 5 V Signalisierungslevel. Die XMC48 00-Platine und das Wi-Fi Click-Modul benötigen 3,3 V. Verwenden Sie nicht den IOREF-Jumper des Boards, um die Board-Signale in 5 V zu ändern.

Mit dem verbundenen Kabel können Sie eine serielle Verbindung auf einem Terminal-Emulator, wie z. B. dem GNU-Monitor, herstellen. Die Baudrate ist standardmäßig auf 115 200 mit 8 Datenbits, keiner Parität und 1 Stoppbit festgelegt.

Überwachung von MQTT-Nachrichten in der Cloud

Bevor Sie die FreeRTOS-Demo ausführen, können Sie den MQTT-Client in der AWS IoT Konsole so einrichten, dass er die Nachrichten überwacht, die Ihr Gerät an die Cloud sendet. AWS

Um das MQTT-Thema mit dem MQTT-Client zu abonnieren AWS IoT

1. Melden Sie sich bei der AWS IoT-Konsole an.
2. Wählen Sie im Navigationsbereich Test und dann MQTT-Testclient aus, um den MQTT-Client zu öffnen.
3. Geben Sie im Feld Subscription topic (Abonnementthema) die Option ***your-thing-name*/example/topic** ein und wählen Sie dann Subscribe to topic (Thema abonnieren).

Wenn das Demo-Projekt erfolgreich auf Ihrem Gerät ausgeführt wird, sehen Sie „Hello World!“ mehrfach zu dem Thema gesendet, das Sie abonniert haben.

Erstellen Sie das FreeRTOS-Demoprojekt und führen Sie es aus

Importiere die FreeRTOS-Demo in DAVE

1. Starten Sie DAVE.
2. Wählen Sie in DAVE die Optionen File (Datei) und Import (Importieren). Erweitern Sie im Fenster Import (Importieren) den Infineon-Ordner, wählen Sie das DAVE-Projekt aus und klicken Sie dann auf Weiter.

3. Wählen Sie im Fenster DAVE-Projekte importieren die Option Stammverzeichnis auswählen, klicken Sie auf Durchsuchen und wählen Sie dann das XMC48 00-Demoprojekt aus.

   In dem Verzeichnis, in dem Sie Ihren FreeRTOS-Download entpackt haben, befindet sich das Demo-Projekt. `projects/infineon/xmc4800_iotkit/dave4/aws_demos`

   Stellen Sie sicher, dass Copy Projects Into Workspace deaktiviert ist.
4. Wählen Sie Finish (Abschließen).

   Das `aws_demos`-Projekt sollte in Ihren WorkSpace importiert und aktiviert werden.
5. Wählen Sie im Menü Project (Projekt) die Option Build Active Project (Aktive Projekte erstellen) aus.

   Stellen Sie sicher, dass das Projekt ohne Fehler erstellt wird.

Führen Sie das FreeRTOS-Demo-Projekt aus

1. Verwenden Sie ein USB-Kabel, um Ihr XMC48 00 IoT Connectivity Kit mit Ihrem Computer zu verbinden. Das Board verfügt über zwei microUSB-Konnektoren. Verwenden Sie den Konnektor mit der Kennzeichnung "X101", wobei Debug daneben auf dem Silkscreen des Boards erscheint.
2. Wählen Sie im Menü Project (Projekt) die Option Rebuild Active Project (Aktive Projekte neu erstellen) aus, um `aws_demos` neu zu erstellen, und stellen Sie sicher, dass Ihre Konfigurationsänderungen ausgewählt wurden.
3. Klicken Sie im Projekt-Explorer mit der rechten Maustaste auf `aws_demos` und wählen Sie die Option Debug As (Debuggen als) und dann DAVE C/C++ Application (DAVE C/C++-Anwendung) aus.
4. Doppelklicken Sie auf GDB SEGGER J-Link Debugging (GDB-SEGGER-J-Link-Debugging) zum Erstellen einer Debug-Bestätigung. Wählen Sie Debug (Debuggen) aus.
5. Wenn der Debugger am Haltepunkt in `main()` anhält, wählen Sie im Menü Run (Ausführen) die Funktion Fortsetzen aus.

In der AWS IoT Konsole sollte der MQTT-Client aus den Schritten 4 bis 5 die von Ihrem Gerät gesendeten MQTT-Nachrichten anzeigen. Wenn Sie die serielle Verbindung nutzen, werden in der UART-Ausgabe Einträge wie dieser angezeigt:

```
0 0 [Tmr Svc] Starting key provisioning...
1 1 [Tmr Svc] Write root certificate...
```

```
2 4 [Tmr Svc] Write device private key...
3 82 [Tmr Svc] Write device certificate...
4 86 [Tmr Svc] Key provisioning done...
5 291 [Tmr Svc] Wi-Fi module initialized. Connecting to AP...
.6 8046 [Tmr Svc] Wi-Fi Connected to AP. Creating tasks which use network...
7 8058 [Tmr Svc] IP Address acquired [IP Address]
8 8058 [Tmr Svc] Creating MQTT Echo Task...
9 8059 [MQTTEcho] MQTT echo attempting to connect to [MQTT Broker].
...10 23010 [MQTTEcho] MQTT echo connected.
11 23010 [MQTTEcho] MQTT echo test echoing task created.
.12 26011 [MQTTEcho] MQTT Echo demo subscribed to iotdemo/#
13 29012 [MQTTEcho] Echo successfully published 'Hello World 0'
.14 32096 [Echoing] Message returned with ACK: 'Hello World 0 ACK'
.15 37013 [MQTTEcho] Echo successfully published 'Hello World 1'
16 40080 [Echoing] Message returned with ACK: 'Hello World 1 ACK'
.17 45014 [MQTTEcho] Echo successfully published 'Hello World 2'
.18 48091 [Echoing] Message returned with ACK: 'Hello World 2 ACK'
.19 53015 [MQTTEcho] Echo successfully published 'Hello World 3'
.20 56087 [Echoing] Message returned with ACK: 'Hello World 3 ACK'
.21 61016 [MQTTEcho] Echo successfully published 'Hello World 4'
22 64083 [Echoing] Message returned with ACK: 'Hello World 4 ACK'
.23 69017 [MQTTEcho] Echo successfully published 'Hello World 5'
.24 72091 [Echoing] Message returned with ACK: 'Hello World 5 ACK'
.25 77018 [MQTTEcho] Echo successfully published 'Hello World 6'
26 80085 [Echoing] Message returned with ACK: 'Hello World 6 ACK'
.27 85019 [MQTTEcho] Echo successfully published 'Hello World 7'
.28 88086 [Echoing] Message returned with ACK: 'Hello World 7 ACK'
.29 93020 [MQTTEcho] Echo successfully published 'Hello World 8'
.30 96088 [Echoing] Message returned with ACK: 'Hello World 8 ACK'
.31 101021 [MQTTEcho] Echo successfully published 'Hello World 9'
32 104102 [Echoing] Message returned with ACK: 'Hello World 9 ACK'
.33 109022 [MQTTEcho] Echo successfully published 'Hello World 10'
.34 112047 [Echoing] Message returned with ACK: 'Hello World 10 ACK'
.35 117023 [MQTTEcho] Echo successfully published 'Hello World 11'
36 120089 [Echoing] Message returned with ACK: 'Hello World 11 ACK'
.37 122068 [MQTTEcho] MQTT echo demo finished.
38 122068 [MQTTEcho] ----Demo finished----
```

Erstellen Sie die FreeRTOS-Demo mit CMake

Wenn Sie es vorziehen, keine IDE für die FreeRTOS-Entwicklung CMake zu verwenden, können Sie alternativ die Demo-Anwendungen oder Anwendungen, die Sie mit Code-Editoren und Debugging-Tools von Drittanbietern entwickelt haben, erstellen und ausführen.

> **Note**
>
> Dieser Abschnitt behandelt die Verwendung CMake unter Windows mit MingW als nativem Build-System. Weitere Informationen zur Verwendung CMake mit anderen Betriebssystemen und Optionen finden Sie unterVerwendung CMake mit FreeRTOS. (MinGW ist eine minimalistische Entwicklungsumgebung für native Microsoft Windows-Anwendungen.)

Um die FreeRTOS-Demo zu erstellen mit CMake

1. Richten Sie die GNU Arm Embedded-Toolchain ein.

   a. Laden Sie eine Windows-Version der Toolchain von der Arm Embedded Toolchain Downloadseite herunter.

   > **Note**
   >
   > Wir empfehlen Ihnen, eine andere Version als "8-2018-q4-major" herunterzuladen, da für das Dienstprogramm "objcopy" in dieser Version ein Fehler gemeldet ist.

   b. Öffnen Sie das heruntergeladene Toolchain-Installationsprogramm und folgen Sie den Anweisungen des Installationsassistenten, um die Toolchain zu installieren.

   > **Important**
   >
   > Wählen Sie auf der letzten Seite des Installationsassistenten Add path to environment variable, um den Toolchain-Pfad zur Umgebungsvariablen des Systempfads hinzuzufügen.

2. Installieren CMake und MingW.

   Anweisungen finden Sie unter CMake Voraussetzungen.

3. Erstellen Sie einen Ordner, der die generierten Build-Dateien (*`build-folder`*) enthält.
4. Ändern Sie die Verzeichnisse in Ihr FreeRTOS-Download-Verzeichnis (*`freertos`*) und verwenden Sie den folgenden Befehl, um die Build-Dateien zu generieren:

```
cmake -DVENDOR=infineon -DBOARD=xmc4800_iotkit -DCOMPILER=arm-gcc -S . -B build-
folder -G "MinGW Makefiles" -DAFR_ENABLE_TESTS=0
```

5. Ändern Sie die Verzeichnisse in das Build-Verzeichnis (*build-folder*) und verwenden Sie den folgenden Befehl, um die Binärdatei zu erstellen:

   ```
   cmake --build . --parallel 8
   ```

   Dieser Befehl erstellt die `aws_demos.hex`-Ausgabebinärdatei in das Build-Verzeichnis.

6. Flashen und Sie das Image mit JLINK und führen Sie es aus.

   a. Verwenden Sie im Build-Verzeichnis (*build-folder*) die folgenden Befehle, um ein Flash-Skript zu erstellen:

      ```
      echo loadfile aws_demos.hex > flash.jlink
      ```

      ```
      echo r >> flash.jlink
      ```

      ```
      echo g >> flash.jlink
      ```

      ```
      echo q >> flash.jlink
      ```

   b. Flashen Sie das Image mit der ausführbaren Datei JLNIK.

      ```
      JLINK_PATH\JLink.exe  -device XMC4800-2048 -if SWD -speed auto -CommanderScript
       flash.jlink
      ```

      Die Anwendungsprotokolle sollten über die serielle Verbindung sichtbar sein, die Sie mit dem Board hergestellt haben.

Fehlerbehebung

Falls Sie es noch nicht getan haben, stellen Sie sicher, dass Sie Ihr FreeRTOS konfigurieren AWS IoT und herunterladen, um Ihr Gerät mit der AWS Cloud zu verbinden. Detaillierte Anweisungen finden Sie unter Erste Schritte.

Allgemeine Informationen zur Problembehandlung bei Getting Started with FreeRTOS finden Sie unter. Fehlerbehebung – Erste Schritte

Erste Schritte mit dem Infineon OPTIGA Trust X und XMC4800 dem IoT Connectivity Kit

> **Important**
>
> Diese Referenzintegration wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Dieses Tutorial enthält Anleitungen für die ersten Schritte mit dem Infineon OPTIGA Trust X Secure Element and XMC4800 IoT Connectivity Kit. Im Vergleich zum Erste Schritte mit dem Infineon XMC48 00 IoT Connectivity Kit Tutorial zeigt diese Anleitung, wie Sie sichere Anmeldeinformationen mit einem Infineon OPTIGA Trust X Secure Element bereitstellen können.

Sie benötigen die folgende Hardware:

1. Host MCU — Infineon XMC4800 IoT Connectivity Kit. Besuchen Sie den AWS Partner Device Catalog, um eines von unserem Partner zu erwerben.
2. Sicherheits-Erweiterungspaket:
   - Secure Element - Infineon OPTIGA Trust X.

     Besuchen Sie den AWS Partner-Gerätekatalog, um sie bei unserem Partner zu erwerben.
   - Personalization Board - Infineon OPTIGA Personalisation Board.
   - Adapterplatine — Infineon Mylo T-Adapter.

Um die hier beschriebenen Schritte auszuführen, müssen Sie eine serielle Verbindung mit dem Board öffnen, um Protokollierungs- und Debugging-Informationen anzuzeigen. (Einer der Schritte erfordert, dass Sie einen öffentlichen Schlüssel aus der seriellen Debugging-Ausgabe der Platine kopieren und in eine Datei einfügen.) Dazu benötigen Sie zusätzlich zum IoT Connectivity Kit einen 3,3-V-USB/Seriell-Konverter. XMC4800 Es ist bekannt, dass der USB-/Seriell-Konverter JBtek EL-PN-47310126 für diese Demo geeignet ist. Sie benötigen außerdem drei male-to-male Überbrückungskabel (für Empfang (RX), Übertragung (TX) und Masse (GND)), um das serielle Kabel mit der Infineon T-Adapterplatine zu verbinden. Mylo

Bevor Sie beginnen, müssen Sie FreeRTOS konfigurieren AWS IoT und herunterladen, um Ihr Gerät mit der AWS Cloud zu verbinden. Detaillierte Anweisungen finden Sie unter Option 2: Integrierte

Generierung eines privaten Schlüssels. In diesem Tutorial wird der Pfad zum FreeRTOS-Download-Verzeichnis als bezeichnet. *freertos*

-Übersicht

Dieses Tutorial enthält die folgenden Schritte:

1. Installieren Sie Software zum Entwickeln und Debuggen eingebetteter Anwendungen für Ihr Mikrocontroller-Board auf dem Host-Computer.
2. Eine FreeRTOS-Demo-Anwendung zu einem Binär-Image querkompilieren.
3. Laden des binären Anwendungs-Image auf Ihre Platine und Ausführen der Anwendung.
4. Interagieren Sie für Überwachungs- und Debuggingzwecke mit der Anwendung, die auf Ihrem Board über eine serielle Verbindung ausgeführt wird.

Einrichten Ihrer Entwicklungsumgebung

FreeRTOS verwendet die DAVE-Entwicklungsumgebung von Infineon, um das zu programmieren. XMC4800 Bevor Sie beginnen, laden Sie DAVE und einige J-Link-Treiber herunter und installieren Sie sie, um mit dem On-Board-Debugger zu kommunizieren.

Installieren von DAVE

1. Rufen Sie die Seite von Infineon zum Herunterladen der DAVE-Software auf.
2. Wählen Sie das DAVE-Paket für Ihr Betriebssystem aus und senden Sie Ihre Registrierungsinformationen. Nach der Registrierung sollten Sie eine Bestätigungs-E-Mail mit einem Link zum Herunterladen einer ZIP-Datei erhalten.
3. Laden Sie die ZIP-Datei des DAVE-Pakets (`DAVE_`*`version_os_date`*`.zip`) herunter und entpacken Sie es im Speicherort, in dem Sie DAVE installieren möchten (z. B. `C:\DAVE4`).

   > **Note**
   >
   > Einige Windows-Benutzer, die den Windows Explorer nutzen, haben Probleme beim Entpacken der Datei gemeldet. Wir empfehlen die Verwendung eines Drittanbieter-Programms, z. B. 7-Zip.

4. Wenn Sie DAVE starten möchten, müssen Sie die ausführbare Datei ausführen, die Sie aus dem `DAVE_`*`version_os_date`*`.zip`-Ordner extrahiert haben.

Weitere Informationen finden Sie im DAVE-Schnellstartleitfaden.

Installieren von Segger-J-Link-Treibern

Für die Kommunikation mit der integrierten Debugging-Sonde des XMC4800 IoT Connectivity Kits benötigen Sie die Treiber, die im J-Link Software- und Dokumentationspaket enthalten sind. Sie können das J-Link-Software- und Dokumentationspaket von Seggers auf der Seite mit demJ-Link-Software-Download herunterladen.

Herstellen einer seriellen Verbindung

Connect das USB/Serial Konverterkabel mit dem Infineon Shield2Go-Adapter. Dies ermöglicht Ihrem Board das Senden von Protokollierungs- und Debugging-Informationen in einer Form, die Sie auf Ihrem Entwicklungscomputer ansehen können. So richten Sie eine serielle Verbindung ein:

1. Connect den RX-Pin mit dem TX-Pin Ihres USB/Serial Konverters.
2. Connect den TX-Pin mit dem RX-Pin Ihres USB/Serial Konverters.
3. Verbinden Sie den Erdungspin Ihres seriellen Konverters mit einem der GND-Pins auf Ihrem Board. Die Geräte müssen einen gemeinsamen Ground teilen.

Die Energieversorgung erfolgt über den USB-Debugging-Port. Verbinden Sie also nicht den positiven Spannungspin Ihres seriellen Adapters mit dem Board.

> **Note**
>
> Einige serielle Kabel verwenden ein 5 V Signalisierungslevel. Die XMC4800 Platine und das Wi-Fi Click-Modul benötigen 3,3 V. Verwenden Sie nicht den IOREF-Jumper des Boards, um die Board-Signale in 5 V zu ändern.

Mit dem verbundenen Kabel können Sie eine serielle Verbindung auf einem Terminal-Emulator, wie z. B. dem GNU-Monitor, herstellen. Die Baudrate ist standardmäßig auf 115 200 mit 8 Datenbits, keiner Parität und 1 Stoppbit festgelegt.

Überwachung von MQTT-Nachrichten in der Cloud

Bevor Sie das FreeRTOS-Demoprojekt ausführen, können Sie den MQTT-Client in der AWS IoT Konsole so einrichten, dass er die Nachrichten überwacht, die Ihr Gerät an die Cloud sendet. AWS

Um das MQTT-Thema mit dem MQTT-Client zu abonnieren AWS IoT

1. Melden Sie sich bei der AWS IoT -Konsole an.
2. Wählen Sie im Navigationsbereich Test und dann MQTT-Testclient, um den MQTT-Client zu öffnen.
3. Geben Sie im Feld Subscription topic (Abonnementthema) die Option ***your-thing-name*/example/topic** ein und wählen Sie dann Subscribe to topic (Thema abonnieren).

Wenn das Demo-Projekt erfolgreich auf Ihrem Gerät ausgeführt wird, sehen Sie „Hello World!“ mehrfach zu dem Thema gesendet, das Sie abonniert haben.

Erstellen Sie das FreeRTOS-Demoprojekt und führen Sie es aus

Importiere die FreeRTOS-Demo in DAVE

1. Starten Sie DAVE.
2. Wählen Sie in DAVE File (Datei) und dann Import (Importieren) aus. Erweitern Sie den Infineon-Ordner, wählen Sie das DAVE-Project aus und klicken Sie dann auf Next (Weiter).
3. Wählen Sie im Fenster DAVE-Projekte importieren die Option Stammverzeichnis auswählen, klicken Sie auf Durchsuchen und wählen Sie dann das XMC4800 Demo-Projekt aus.

   In dem Verzeichnis, in dem Sie Ihren FreeRTOS-Download entpackt haben, befindet sich das Demo-Projekt. `projects/infineon/xmc4800_plus_optiga_trust_x/dave4/aws_demos/dave4`

   Stellen Sie sicher, dass Copy Projects Into Workspace (Projekte in Arbeitsbereich kopieren) deaktiviert ist.
4. Wählen Sie Finish (Abschließen).

   Das `aws_demos`-Projekt sollte in Ihren WorkSpace importiert und aktiviert werden.
5. Wählen Sie im Menü Project (Projekt) die Option Build Active Project (Aktive Projekte erstellen) aus.

   Stellen Sie sicher, dass das Projekt ohne Fehler erstellt wird.

Führen Sie das FreeRTOS-Demo-Projekt aus

1. Wählen Sie im Menü Project (Projekt) die Option Rebuild Active Project (Aktive Projekte neu erstellen) aus, um `aws_demos` neu zu erstellen, und stellen Sie sicher, dass Ihre Konfigurationsänderungen ausgewählt wurden.
2. Klicken Sie im Projekt-Explorer mit der rechten Maustaste auf `aws_demos` und wählen Sie die Option Debug As (Debuggen als) und dann DAVE C/C++ Application (DAVE C/C++-Anwendung) aus.
3. Doppelklicken Sie auf GDB SEGGER J-Link Debugging (GDB-SEGGER-J-Link-Debugging) zum Erstellen einer Debug-Bestätigung. Wählen Sie Debug (Debuggen) aus.
4. Wenn der Debugger am Haltepunkt in `main()` anhält, wählen Sie im Menü Run (Ausführen) die Funktion Fortsetzen aus.

Fahren Sie an dieser Stelle mit dem in Option 2: Integrierte Generierung eines privaten Schlüssels angegebenen Schritt zur Extraktion des öffentlichen Schlüssels fort. Wenn alle Schritte abgeschlossen sind, gehen Sie zur AWS IoT Konsole. Der zuvor eingerichtete MQTT-Client sollte die von Ihrem Gerät gesendeten MQTT-Nachrichten anzeigen. Sie sollten über die serielle Verbindung des Geräts in etwa Folgendes auf der UART-Ausgabe sehen:

```
0 0 [Tmr Svc] Starting key provisioning...
1 1 [Tmr Svc] Write root certificate...
2 4 [Tmr Svc] Write device private key...
3 82 [Tmr Svc] Write device certificate...
4 86 [Tmr Svc] Key provisioning done...
5 291 [Tmr Svc] Wi-Fi module initialized. Connecting to AP...
.6 8046 [Tmr Svc] Wi-Fi Connected to AP. Creating tasks which use network...
7 8058 [Tmr Svc] IP Address acquired [IP Address]
8 8058 [Tmr Svc] Creating MQTT Echo Task...
9 8059 [MQTTEcho] MQTT echo attempting to connect to [MQTT Broker].
...10 23010 [MQTTEcho] MQTT echo connected.
11 23010 [MQTTEcho] MQTT echo test echoing task created.
.12 26011 [MQTTEcho] MQTT Echo demo subscribed to iotdemo/#
13 29012 [MQTTEcho] Echo successfully published 'Hello World 0'
.14 32096 [Echoing] Message returned with ACK: 'Hello World 0 ACK'
.15 37013 [MQTTEcho] Echo successfully published 'Hello World 1'
16 40080 [Echoing] Message returned with ACK: 'Hello World 1 ACK'
.17 45014 [MQTTEcho] Echo successfully published 'Hello World 2'
.18 48091 [Echoing] Message returned with ACK: 'Hello World 2 ACK'
.19 53015 [MQTTEcho] Echo successfully published 'Hello World 3'
```

```
.20 56087 [Echoing] Message returned with ACK: 'Hello World 3 ACK'
.21 61016 [MQTTEcho] Echo successfully published 'Hello World 4'
22 64083 [Echoing] Message returned with ACK: 'Hello World 4 ACK'
.23 69017 [MQTTEcho] Echo successfully published 'Hello World 5'
.24 72091 [Echoing] Message returned with ACK: 'Hello World 5 ACK'
.25 77018 [MQTTEcho] Echo successfully published 'Hello World 6'
26 80085 [Echoing] Message returned with ACK: 'Hello World 6 ACK'
.27 85019 [MQTTEcho] Echo successfully published 'Hello World 7'
.28 88086 [Echoing] Message returned with ACK: 'Hello World 7 ACK'
.29 93020 [MQTTEcho] Echo successfully published 'Hello World 8'
.30 96088 [Echoing] Message returned with ACK: 'Hello World 8 ACK'
.31 101021 [MQTTEcho] Echo successfully published 'Hello World 9'
32 104102 [Echoing] Message returned with ACK: 'Hello World 9 ACK'
.33 109022 [MQTTEcho] Echo successfully published 'Hello World 10'
.34 112047 [Echoing] Message returned with ACK: 'Hello World 10 ACK'
.35 117023 [MQTTEcho] Echo successfully published 'Hello World 11'
36 120089 [Echoing] Message returned with ACK: 'Hello World 11 ACK'
.37 122068 [MQTTEcho] MQTT echo demo finished.
38 122068 [MQTTEcho] ----Demo finished----
```

Erstellen Sie die FreeRTOS-Demo mit CMake

Dieser Abschnitt behandelt die Verwendung CMake unter Windows mit MingW als nativem Build-System. Weitere Informationen zur Verwendung CMake mit anderen Betriebssystemen und Optionen finden Sie unterVerwendung CMake mit FreeRTOS. (MinGW ist eine minimalistische Entwicklungsumgebung für native Microsoft Windows-Anwendungen.)

Wenn Sie es vorziehen, keine IDE für die FreeRTOS-Entwicklung CMake zu verwenden, können Sie die Demo-Anwendungen oder Anwendungen, die Sie mit Code-Editoren und Debugging-Tools von Drittanbietern entwickelt haben, erstellen und ausführen.

Um die FreeRTOS-Demo zu erstellen mit CMake

1. Richten Sie die GNU Arm Embedded-Toolchain ein.

   a. Laden Sie eine Windows-Version der Toolchain von der Arm Embedded Toolchain-Downloadseite herunter.

> **Note**
>
> Wir empfehlen Ihnen, eine andere Version als „8-2018-q4-major“ herunterzuladen, da ein Fehler gemeldet wurde.

b. Öffnen Sie das heruntergeladene Toolchain-Installationsprogramm und folgen Sie den Anweisungen im Assistenten.

c. Wählen Sie auf der letzten Seite des Installationsassistenten Add path to environment variable, um den Toolchain-Pfad zur Umgebungsvariablen des Systempfads hinzuzufügen.

2. Installieren CMake und MingW.

   Anweisungen finden Sie unter CMake Voraussetzungen.

3. Erstellen Sie einen Ordner, der die generierten Build-Dateien (*build-folder*) enthält.

4. Ändern Sie die Verzeichnisse in Ihr FreeRTOS-Download-Verzeichnis (*freertos*) und verwenden Sie den folgenden Befehl, um die Build-Dateien zu generieren:

```
cmake -DVENDOR=infineon -DBOARD=xmc4800_plus_optiga_trust_x -DCOMPILER=arm-gcc -S .
 -B build-folder -G "MinGW Makefiles" -DAFR_ENABLE_TESTS=0
```

5. Ändern Sie die Verzeichnisse in das Build-Verzeichnis (*build-folder*) und verwenden Sie den folgenden Befehl, um die Binärdatei zu erstellen:

```
cmake --build . --parallel 8
```

   Dieser Befehl erstellt die `aws_demos.hex`-Ausgabebinärdatei in das Build-Verzeichnis.

6. Flashen und Sie das Image mit JLINK und führen Sie es aus.

   a. Verwenden Sie im Build-Verzeichnis (*build-folder*) die folgenden Befehle, um ein Flash-Skript zu erstellen:

```
echo loadfile aws_demos.hex > flash.jlink
echo r >> flash.jlink
echo g >> flash.jlink
echo q >> flash.jlink
```

   b. Flashen Sie das Image mit der ausführbaren Datei JLNIK.

```
JLINK_PATH\JLink.exe  -device XMC4800-2048 -if SWD -speed auto -CommanderScript
 flash.jlink
```

Die Anwendungsprotokolle sollten über die serielle Verbindung sichtbar sein, die Sie mit dem Board hergestellt haben. Fahren Sie mit dem Extraktionsschritt für öffentliche Schlüssel in Option 2: Integrierte Generierung eines privaten Schlüssels fort. Nachdem alle Schritte abgeschlossen sind, wechseln Sie zur AWS IoT Konsole. Der zuvor eingerichtete MQTT-Client sollte die von Ihrem Gerät gesendeten MQTT-Nachrichten anzeigen.

Fehlerbehebung

Informationen zur Problembehebung finden Sie unter Fehlerbehebung – Erste Schritte.

Erste Schritte mit dem MW32x AWS IoT Starter Kit

> **Important**
>
> Diese Referenzintegration wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Das AWS IoT Starter Kit ist ein Entwicklungskit, das auf dem 88 MW320 /88 basiert, dem neuesten integrierten Cortex-M4-Mikrocontroller von NXPMW322, der 802.11b/g/n-Wi-Fi auf einem einzigen Mikrocontroller-Chip integriert. Das Entwicklungskit ist FCC-zertifiziert. Weitere Informationen finden Sie im Gerätekatalog für AWS Partner, wo Sie eines von unserem Partner erwerben können. Die MW320 MW322 88/88-Module sind außerdem FCC-zertifiziert und können individuell angepasst und in großen Stückzahlen verkauft werden.

In diesem Handbuch für die ersten Schritte erfahren Sie, wie Sie Ihre Anwendung zusammen mit dem SDK auf einem Host-Computer Cross-Compilieren kompilieren und dann die generierte Binärdatei mit den im SDK enthaltenen Tools auf das Board laden. Wenn die Anwendung auf dem Motherboard ausgeführt wird, können Sie sie über die serielle Konsole auf Ihrem Host-Computer debuggen oder mit ihr interagieren.

Ubuntu 16.04 ist die Host-Plattform, die für Entwicklung und Debugging unterstützt wird. Möglicherweise können Sie andere Plattformen verwenden, diese werden jedoch nicht offiziell unterstützt. Sie müssen über die erforderlichen Berechtigungen verfügen, um Software auf der Hostplattform zu installieren. Die folgenden externen Tools sind erforderlich, um das SDK zu erstellen:

- Host-Plattform für Ubuntu 16.04
- ARM-Toolchain, Version 4_9_2015q3
- Eclipse 4.9.0 IDE

Die ARM-Toolchain ist für die Cross-Compilierung Ihrer Anwendung und des SDK erforderlich. Das SDK nutzt die neuesten Versionen der Toolchain, um den Image-Footprint zu optimieren und mehr Funktionen auf weniger Platz unterzubringen. In diesem Handbuch wird davon ausgegangen, dass Sie Version 4_9_2015q3 der Toolchain verwenden. Die Verwendung älterer Versionen der Toolchain wird nicht empfohlen. Das Entwicklungskit ist mit der Firmware des Wireless Microcontroller Demo-Projekts vorinstalliert.

Themen

- Einrichten Ihrer Hardware
- Einrichten der Entwicklungsumgebung
- Erstellen und starten Sie das FreeRTOS-Demoprojekt
- Debuggen
- Fehlerbehebung

Einrichten Ihrer Hardware

Connect das MW32x Board mit einem Mini-USB-zu-USB-Kabel mit Ihrem Laptop. Connect das Mini-USB-Kabel mit dem einzigen Mini-USB-Anschluss auf der Platine. Sie benötigen keinen Jumperwechsel.

Wenn das Board an einen Laptop oder Desktop-Computer angeschlossen ist, benötigen Sie keine externe Stromversorgung.

Diese USB-Verbindung bietet Folgendes:

- Konsolenzugriff auf das Board. Auf dem Entwicklungshost ist ein virtueller tty/com Port registriert, über den auf die Konsole zugegriffen werden kann.

- JTAG-Zugriff auf das Board. Dies kann zum Laden oder Entladen von Firmware-Images in den RAM oder Flash der Platine oder zu Debugging-Zwecken verwendet werden.

Einrichten der Entwicklungsumgebung

Für Entwicklungszwecke sind die ARM-Toolchain und die im SDK enthaltenen Tools die Mindestanforderung. Die folgenden Abschnitte enthalten Einzelheiten zur Einrichtung der ARM-Toolchain.

GNU-Toolchain

Das SDK unterstützt offiziell die GCC Compiler-Toolchain. Die Cross-Compiler-Toolchain für GNU ARM ist unter GNU Arm Embedded Toolchain 4.9-2015-q3-update verfügbar.

Das Build-System ist standardmäßig so konfiguriert, dass es die GNU-Toolchain verwendet. Die Makefiles gehen davon aus, dass die Binärdateien der GNU-Compiler-Toolchain im PATH des Benutzers verfügbar sind und von den Makefiles aus aufgerufen werden können. Die Makefiles gehen auch davon aus, dass den Dateinamen der GNU-Toolchain-Binärdateien ein Präfix vorangestellt wird. `arm-none-eabi-`

Die GCC-Toolchain kann mit GDB zum Debuggen mit OpenOCD (im Lieferumfang des SDK enthalten) verwendet werden. Dies stellt die Software zur Verfügung, die mit JTAG verbunden ist.

Wir empfehlen Version 4_9_2015q3 der Toolchain. gcc-arm-embedded

Verfahren zur Einrichtung der Linux-Toolchain

Gehen Sie wie folgt vor, um die GCC-Toolchain unter Linux einzurichten.

1. Laden Sie den Toolchain-Tarball herunter, der unter GNU Arm Embedded Toolchain 4.9-2015-q3-update verfügbar ist. `gcc-arm-none-eabi-4_9-2015q3-20150921-linux.tar.bz2`Die Datei ist.
2. Kopieren Sie die Datei in ein Verzeichnis Ihrer Wahl. Stellen Sie sicher, dass der Verzeichnisname keine Leerzeichen enthält.
3. Verwenden Sie den folgenden Befehl, um die Datei zu entpacken.

```
tar -vxf filename
```

4. Fügen Sie den Pfad der installierten Toolchain zum System-PATH hinzu. Fügen Sie beispielsweise die folgende Zeile am Ende der `.profile` Datei an, die sich im `/home/`*user-name* Verzeichnis befindet.

   ```
   PATH=$PATH:path to gcc-arm-none-eabit-4_9_2015_q3/bin
   ```

> **Note**
>
> Neuere Distributionen von Ubuntu könnten mit einer Debian-Version des GCC Cross Compilers geliefert werden. Falls ja, müssen Sie den systemeigenen Cross Compiler entfernen und das obige Einrichtungsverfahren befolgen.

Arbeiten Sie mit einem Linux-Entwicklungshost

Jede moderne Linux-Desktop-Distribution wie Ubuntu oder Fedora kann verwendet werden. Wir empfehlen jedoch, auf die neueste Version zu aktualisieren. Es wurde verifiziert, dass die folgenden Schritte unter Ubuntu 16.04 funktionieren. Es wird davon ausgegangen, dass Sie diese Version verwenden.

Pakete installieren

Das SDK enthält ein Skript, mit dem Sie Ihre Entwicklungsumgebung auf einem neu eingerichteten Linux-Computer schnell einrichten können. Das Skript versucht, den Maschinentyp auto zu erkennen und die entsprechende Software zu installieren, einschließlich C-Bibliotheken, USB-Bibliothek, FTDI-Bibliothek, Ncurses, Python und Latex. In diesem Abschnitt *amzsdk_bundle-x.y.z* gibt der generische Verzeichnisname das AWS SDK-Stammverzeichnis an. Der tatsächliche Verzeichnisname kann anders sein. Sie müssen über Root-Rechte verfügen.

- Navigieren Sie zum *amzsdk_bundle-x.y.z/* Verzeichnis und führen Sie diesen Befehl aus.

  ```
  ./lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/bin/installpkgs.sh
  ```

Sudo vermeiden

In dieser Anleitung verwendet der `flashprog` Vorgang das `flashprog.py` Skript, um das NAND der Platine zu flashen, wie unten erklärt. In ähnlicher Weise verwendet der `ramload` Vorgang das

`ramload.py` Skript, um das Firmware-Image vom Host direkt in den RAM des Mikrocontrollers zu kopieren, ohne das NAND zu flashen.

Sie können Ihren Linux-Entwicklungshost so konfigurieren, dass er die `flashprog ramload` AND-Operationen ausführt, ohne dass der `sudo` Befehl jedes Mal erforderlich ist. Führen Sie dazu den folgenden Befehl aus.

```
./lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/bin/perm_fix.sh
```

> **Note**
>
> Sie müssen die Berechtigungen Ihres Linux-Entwicklungshosts auf diese Weise konfigurieren, um ein reibungsloses Eclipse-IDE-Erlebnis zu gewährleisten.

Einrichtung der seriellen Konsole

Stecken Sie das USB-Kabel in den USB-Steckplatz des Linux-Hosts. Dies löst die Erkennung des Geräts aus. In der `/var/log/messages` Datei oder nach der Ausführung des `dmesg` Befehls sollten Sie Meldungen wie die folgenden sehen.

```
Jan 6 20:00:51 localhost kernel: usb 4-2: new full speed USB device using uhci_hcd and
 address 127
Jan 6 20:00:51 localhost kernel: usb 4-2: configuration #1 chosen from 1 choice
Jan 6 20:00:51 localhost kernel: ftdi_sio 4-2:1.0: FTDI USB Serial Device converter
 detected
Jan 6 20:00:51 localhost kernel: ftdi_sio: Detected FT2232C
Jan 6 20:00:51 localhost kernel: usb 4-2: FTDI USB Serial Device converter now attached
 to ttyUSB0
Jan 6 20:00:51 localhost kernel: ftdi_sio 4-2:1.1: FTDI USB Serial Device converter
 detected
Jan 6 20:00:51 localhost kernel: ftdi_sio: Detected FT2232C
Jan 6 20:00:51 localhost kernel: usb 4-2: FTDI USB Serial Device converter now attached
 to ttyUSB1
```

Stellen Sie sicher, dass zwei ttyUSB-Geräte erstellt wurden. Der zweite ttyUSB ist die serielle Konsole. Im obigen Beispiel heißt sie „USB1tty“.

In diesem Handbuch verwenden wir Minicom, um die Ausgabe der seriellen Konsole zu sehen. Sie können auch andere serielle Programme verwenden, wie z. `putty` Führen Sie den folgenden Befehl aus, um Minicom im Setup-Modus auszuführen.

```
minicom -s
```

Navigieren Sie in Minicom zu Serial Port Setup und erfassen Sie die folgenden Einstellungen.

```
| A - Serial Device : /dev/ttyUSB1
| B - Lockfile Location : /var/lock
| C - Callin Program :
| D - Callout Program :
| E - Bps/Par/Bits : 115200 8N1
| F - Hardware Flow Control : No
| G - Software Flow Control : No
```

Sie können diese Einstellungen in Minicom zur künftigen Verwendung speichern. Das Minicom-Fenster zeigt jetzt Nachrichten von der seriellen Konsole.

Wählen Sie das serielle Konsolenfenster und drücken Sie die Eingabetaste. Dadurch wird ein Hash (#) auf dem Bildschirm angezeigt.

> **Note**
>
> Die Entwicklungsboards enthalten ein FTDI-Siliziumgerät. Das FTDI-Gerät stellt zwei USB-Schnittstellen für den Host zur Verfügung. Die erste Schnittstelle ist der JTAG-Funktionalität der MCU zugeordnet und die zweite Schnittstelle ist dem physischen UARTx Port der MCU zugeordnet.

OpenOCD installieren

OpenOCD ist eine Software, die Debugging, systeminterne Programmierung und Boundary-Scan-Tests für eingebettete Zielgeräte ermöglicht.

OpenOCD-Version 0.9 ist erforderlich. Sie ist auch für die Eclipse-Funktionalität erforderlich. Wenn eine frühere Version, z. B. Version 0.7, auf Ihrem Linux-Host installiert wurde, entfernen Sie dieses Repository mit dem entsprechenden Befehl für die Linux-Distribution, die Sie derzeit verwenden.

Führen Sie den Standard-Linux-Befehl aus, um OpenOCD zu installieren.

```
apt-get install openocd
```

Wenn der obige Befehl Version 0.9 oder höher nicht installiert, verwenden Sie das folgende Verfahren, um den Openocd-Quellcode herunterzuladen und zu kompilieren.

Um OpenOCD zu installieren

1. Führen Sie den folgenden Befehl aus, um libusb-1.0 zu installieren.

   ```
   sudo apt-get install libusb-1.0
   ```

2. Laden Sie den Openocd 0.9.0-Quellcode von http://openocd.org/ herunter.
3. Extrahieren Sie Openocd und navigieren Sie zu dem Verzeichnis, in das Sie es extrahiert haben.
4. Konfigurieren Sie openocd mit dem folgenden Befehl.

   ```
   ./configure --enable-ftdi --enable-jlink
   ```

5. Führen Sie das Make-Hilfsprogramm aus, um opencd zu kompilieren.

   ```
   make install
   ```

Eclipse einrichten

> **Note**
>
> In diesem Abschnitt wird davon ausgegangen, dass Sie die Schritte in abgeschlossen haben Sudo vermeiden

Eclipse ist die bevorzugte IDE für Anwendungsentwicklung und Debugging. Es bietet eine umfangreiche, benutzerfreundliche IDE mit integrierter Debugging-Unterstützung, einschließlich Thread-fähigem Debugging. In diesem Abschnitt wird das allgemeine Eclipse-Setup für alle unterstützten Entwicklungshosts beschrieben.

Einrichten von Eclipse

1. Laden Sie das Java Run Time Environment (JRE) herunter und installieren Sie es.

Eclipse erfordert, dass Sie die JRE installieren. Wir empfehlen, dass Sie dies zuerst installieren, obwohl es auch nach der Installation von Eclipse installiert werden kann. Die JRE-Version (32/64 Bit) muss mit der Version von Eclipse (32/64 Bit) übereinstimmen. Sie können die JRE von Java SE Runtime Environment 8 Downloads auf der Oracle-Website herunterladen.

2. Laden Sie die „Eclipse IDE for C/C++ Developers“ von http://www.eclipse.org herunter und installieren Sie sie. Eclipse Version 4.9.0 oder höher wird unterstützt. Für die Installation müssen Sie nur das heruntergeladene Archiv entpacken. Sie führen die plattformspezifische Eclipse-Programmdatei aus, um die Anwendung zu starten.

Erstellen und starten Sie das FreeRTOS-Demoprojekt

Es gibt zwei Möglichkeiten, das FreeRTOS-Demo-Projekt auszuführen:

- Verwenden Sie die Befehlszeile.
- Verwenden Sie die Eclipse-IDE.

Dieses Thema behandelt beide Optionen.

Bereitstellung

- Je nachdem, ob Sie die Test- oder die Demo-Anwendung verwenden, legen Sie die Bereitstellungsdaten in einer der folgenden Dateien fest:

  - `./tests/common/include/aws_clientcredential.h`
  - `./demos/common/include/aws_clientcredential.h`

  Beispiel:

```
#define clientcredentialWIFI_SSID "Wi-Fi SSID"
#define clientcredentialWIFI_PASSWORD "Wi-Fi password"
#define clientcredentialWIFI_SECURITY "Wi-Fi security"
```

> **Note**
>
> Sie können die folgenden Wi-Fi-Sicherheitswerte eingeben:
>
> - `eWiFiSecurityOpen`

- `eWiFiSecurityWEP`
- `eWiFiSecurityWPA`
- `eWiFiSecurityWPA2`

Die SSID und das Passwort sollten in doppelte Anführungszeichen gesetzt werden.

Erstellen und starten Sie die FreeRTOS-Demo über die Befehlszeile

1. Verwenden Sie den folgenden Befehl, um mit der Erstellung der Demo-Anwendung zu beginnen.

```
cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S .  -Bbuild -
DAFR_ENABLE_TESTS=0
```

Stellen Sie sicher, dass Sie dieselbe Ausgabe wie im folgenden Beispiel erhalten.

```
marvell@pe-lt586:amzsdk$
marvell@pe-lt586:amzsdk$ cmake -DVENDOR=marvell -DBOARD=mw300_rd -DCOMPILER=arm-gcc -S . -
Bbuild -DAFR_ENABLE_TESTS=0

====================Configuration for Amazon FreeRTOS====================
  Version:                 v1.2.4
  Git version:             AMZSDK_V1.2.r6.p1-12-gdd17d10

Target microcontroller:
  vendor:                  Marvell
  board:                   mw300_rd
  description:             Marvell Board for AmazonFreeRTOS
  family:                  Wireless Microcontroller
  data ram size:           512KB
  program memory size:     2MB

Host platform:
  OS:                      Linux-4.15.0-47-generic
  Toolchain:               arm-gcc
  Toolchain path:          /home/marvell/Software/gcc-arm-none-eabi-4_9-2015q3
  CMake generator:         Unix Makefiles

Amazon FreeRTOS modules:
  Modules to build:        kernel, freertos_plus_tcp, bufferpool, crypto, greengrass, mqtt
, ota, pkcs11, secure_sockets, shadow, tls, wifi
  Disabled by user:
  Disabled by dependency:  posix

  Available demos:         demo_key_provisioning, demo_logging, demo_mqtt_hello_world, dem
o_mqtt_pubsub, demo_tcp, demo_shadow, demo_greengrass, demo_ota
  Available tests:         test_crypto, test_greengrass, test_mqtt, test_ota, test_pkcs11,
 test_secure_sockets, test_tls, test_shadow, test_wifi, test_memory
=========================================================================

-- Configuring done
-- Generating done
-- Build files have been written to: /home/marvell/gerrit/amzsdk/build
marvell@pe-lt586:amzsdk$
```

2. Navigieren Sie zum Build-Verzeichnis.

```
cd build
```

3. Führen Sie das Make-Utility aus, um die Anwendung zu erstellen.

```
make all -j4
```

Stellen Sie sicher, dass Sie dieselbe Ausgabe wie in der folgenden Abbildung erhalten:

4. Verwenden Sie die folgenden Befehle, um eine Testanwendung zu erstellen.

```
cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S .  -Bbuild -
DAFR_ENABLE_TESTS=1
cd build
make all -j4
```

Führen Sie den `cmake` Befehl jedes Mal aus, wenn Sie zwischen dem `aws_demos project` und dem wechseln`aws_tests project`.

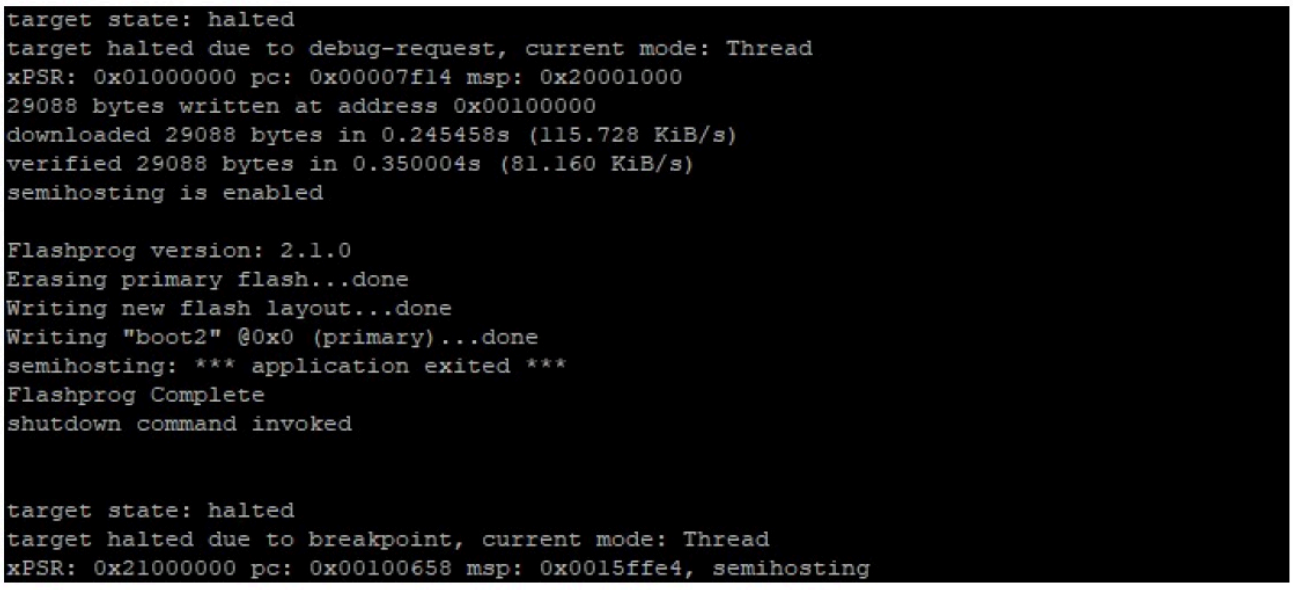
5. Schreiben Sie das Firmware-Image auf den Flash-Speicher des Entwicklungsboards. Die Firmware wird ausgeführt, nachdem das Entwicklungsboard zurückgesetzt wurde. Sie müssen das SDK erstellen, bevor Sie das Image auf den Mikrocontroller flashen.

a. Bevor Sie das Firmware-Image flashen, bereiten Sie den Flash des Entwicklungsboards mit den gemeinsamen Komponenten Layout und Boot2 vor. Verwenden Sie die folgenden Befehle.

```
cd amzsdk_bundle-x.y.z
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py -l ./vendors/
marvell/WMSDK/mw320/sdk/tools/OpenOCD/mw300/layout.txt --boot2 ./vendors/
marvell/WMSDK/mw320/boot2/bin/boot2.bin
```

Der `flashprog` Befehl initiiert Folgendes:

- Layout — Das Flashprog-Hilfsprogramm wird zunächst angewiesen, ein Layout in den Flash zu schreiben. Das Layout ähnelt den Partitionsinformationen für den Flash. Das Standardlayout befindet sich unter`/lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/OpenOCD/mw300/layout.txt`.
- Boot2 — Dies ist der vom WMSDK verwendete Bootloader. Der `flashprog` Befehl schreibt auch einen Bootloader in den Flash. Es ist die Aufgabe des Bootloaders, das Firmware-Image des Mikrocontrollers zu laden, nachdem es geflasht hat. Stellen Sie sicher, dass Sie dieselbe Ausgabe wie in der Abbildung unten erhalten.

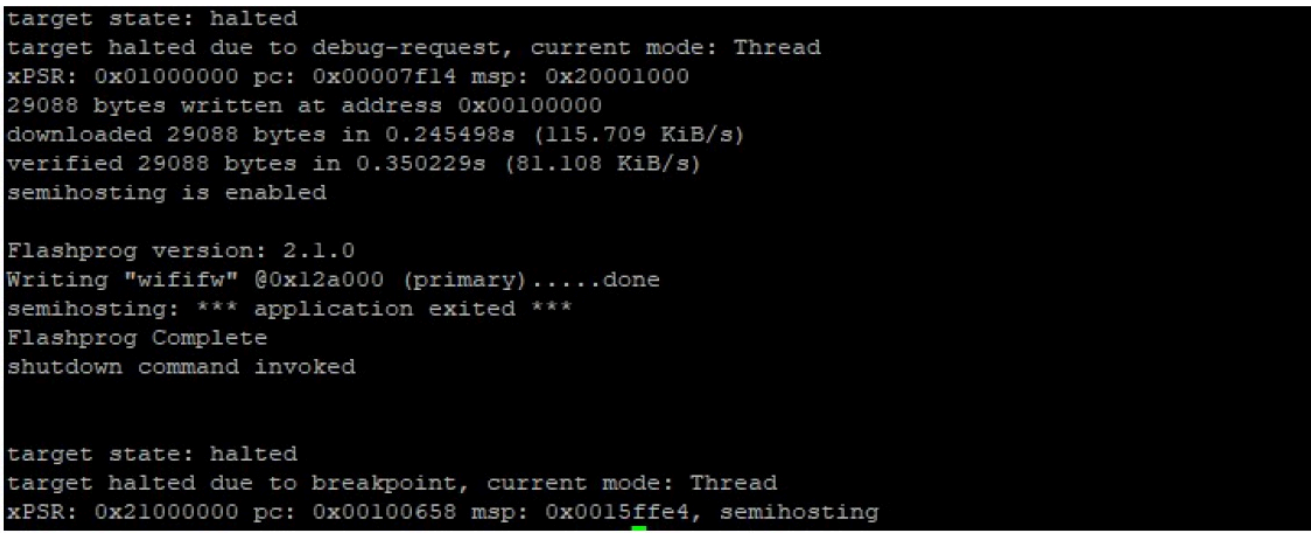


b. Die Firmware verwendet den Wi-Fi-Chipsatz für ihre Funktionalität, und der Wi-Fi-Chipsatz verfügt über eine eigene Firmware, die auch im Flash vorhanden sein muss. Sie verwenden das `flashprog.py` Hilfsprogramm, um die Wi-Fi-Firmware auf die gleiche Weise zu

flashen, wie Sie es beim Flashen des Boot2-Bootloaders und der MCU-Firmware getan haben. Verwenden Sie die folgenden Befehle, um die Wi-Fi-Firmware zu flashen.

```
cd amzsdk_bundle-x.y.z
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py --wififw ./
vendors/marvell/WMSDK/mw320/wifi-firmware/mw30x/mw30x_uapsta_W14.88.36.p135.bin
```

Stellen Sie sicher, dass die Ausgabe des Befehls der folgenden Abbildung ähnelt.



c. Verwenden Sie die folgenden Befehle, um die MCU-Firmware zu flashen.

```
cd amzsdk_bundle-x.y.z
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py --mcufw build/
cmake/vendors/marvell/mw300_rd/aws_demos.bin -r
```

d. Setzen Sie das Board zurück. Sie sollten die Protokolle für die Demo-App sehen.

e. Um die Test-App auszuführen, flashen Sie die `aws_tests.bin` Binärdatei, die sich im selben Verzeichnis befindet.

```
cd amzsdk_bundle-x.y.z
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py --mcufw build/
cmake/vendors/marvell/mw300_rd/aws_tests.bin -r
```

Ihre Befehlsausgabe sollte der in der folgenden Abbildung gezeigten ähneln.

```
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00007f14 msp: 0x20001000
29088 bytes written at address 0x00100000
downloaded 29088 bytes in 0.245499s (115.708 KiB/s)
verified 29088 bytes in 0.350231s (81.107 KiB/s)
semihosting is enabled

Flashprog version: 2.1.0
Writing "mcufw" @0x6a000 (primary)...done
semihosting: *** application exited ***
Flashprog Complete
shutdown command invoked


target state: halted
target halted due to breakpoint, current mode: Thread
xPSR: 0x21000000 pc: 0x00100658 msp: 0x0015ffe4, semihosting
Resetting board...
Using OpenOCD interface file ftdi.cfg
Open On-Chip Debugger 0.9.0 (2015-07-15-15:28)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
adapter speed: 3000 kHz
adapter_nsrst_delay: 100
Info : auto-selecting first available session transport "jtag". To override use 'transport
 select <transport>'.
jtag_ntrst_delay: 100
cortex_m reset_config sysresetreq
sh_load
Info : clock speed 3000 kHz
Info : JTAG tap: wmcore.cpu tap/device found: 0x4ba00477 (mfg: 0x23b, part: 0xba00, ver: 0
x4)
Info : wmcore.cpu: hardware has 6 breakpoints, 4 watchpoints
Info : JTAG tap: wmcore.cpu tap/device found: 0x4ba00477 (mfg: 0x23b, part: 0xba00, ver: 0
x4)
shutdown command invoked
Resetting board done...
```

6. Nachdem Sie die Firmware geflasht und die Platine zurückgesetzt haben, sollte die Demo-App wie in der Abbildung unten gezeigt starten.

```
Network connection successful.

Wi-Fi Connected to AP. Creating tasks which use network...
2 6293 [Startup Hook] Write certificate...
3 6296 [Startup Hook] Write device private key...
4 6362 [Startup Hook] Creating MQTT Echo Task...
6 11668 [MQTTEcho] MQTT echo connected.to connect to a2wtm15blvjjs8-ats.iot.us-east-2.amazonaws.com
7 11668 [MQTTEcho] MQTT echo test echoing task created.
8 11961 [MQTTEcho] MQTT Echo demo subscribed to freertos/demos/echo
9 12248 [MQTTEcho] Echo successfully published 'Hello World 0'
10 12591 [Echoing] Message returned with ACK: 'Hello World 0 ACK'
11 17633 [MQTTEcho] Echo successfully published 'Hello World 1'
12 17927 [Echoing] Message returned with ACK: 'Hello World 1 ACK'
13 22953 [MQTTEcho] Echo successfully published 'Hello World 2'
14 23276 [Echoing] Message returned with ACK: 'Hello World 2 ACK'
15 28245 [MQTTEcho] Echo successfully published 'Hello World 3'
16 28575 [Echoing] Message returned with ACK: 'Hello World 3 ACK'
17 33542 [MQTTEcho] Echo successfully published 'Hello World 4'
18 33980 [Echoing] Message returned with ACK: 'Hello World 4 ACK'
19 38823 [MQTTEcho] Echo successfully published 'Hello World 5'
20 39279 [Echoing] Message returned with ACK: 'Hello World 5 ACK'
21 44139 [MQTTEcho] Echo successfully published 'Hello World 6'
22 44501 [Echoing] Message returned with ACK: 'Hello World 6 ACK'
23 49516 [MQTTEcho] Echo successfully published 'Hello World 7'
24 50270 [Echoing] Message returned with ACK: 'Hello World 7 ACK'
25 54796 [MQTTEcho] Echo successfully published 'Hello World 8'
26 55129 [Echoing] Message returned with ACK: 'Hello World 8 ACK'
27 60080 [MQTTEcho] Echo successfully published 'Hello World 9'
28 60389 [Echoing] Message returned with ACK: 'Hello World 9 ACK'
29 65378 [MQTTEcho] Echo successfully published 'Hello World 10'
30 65998 [Echoing] Message returned with ACK: 'Hello World 10 ACK'
31 70658 [MQTTEcho] Echo successfully published 'Hello World 11'
32 70964 [Echoing] Message returned with ACK: 'Hello World 11 ACK'
33 75958 [MQTTEcho] MQTT echo demo finished.
34 75958 [MQTTEcho] ----Demo finished----
```

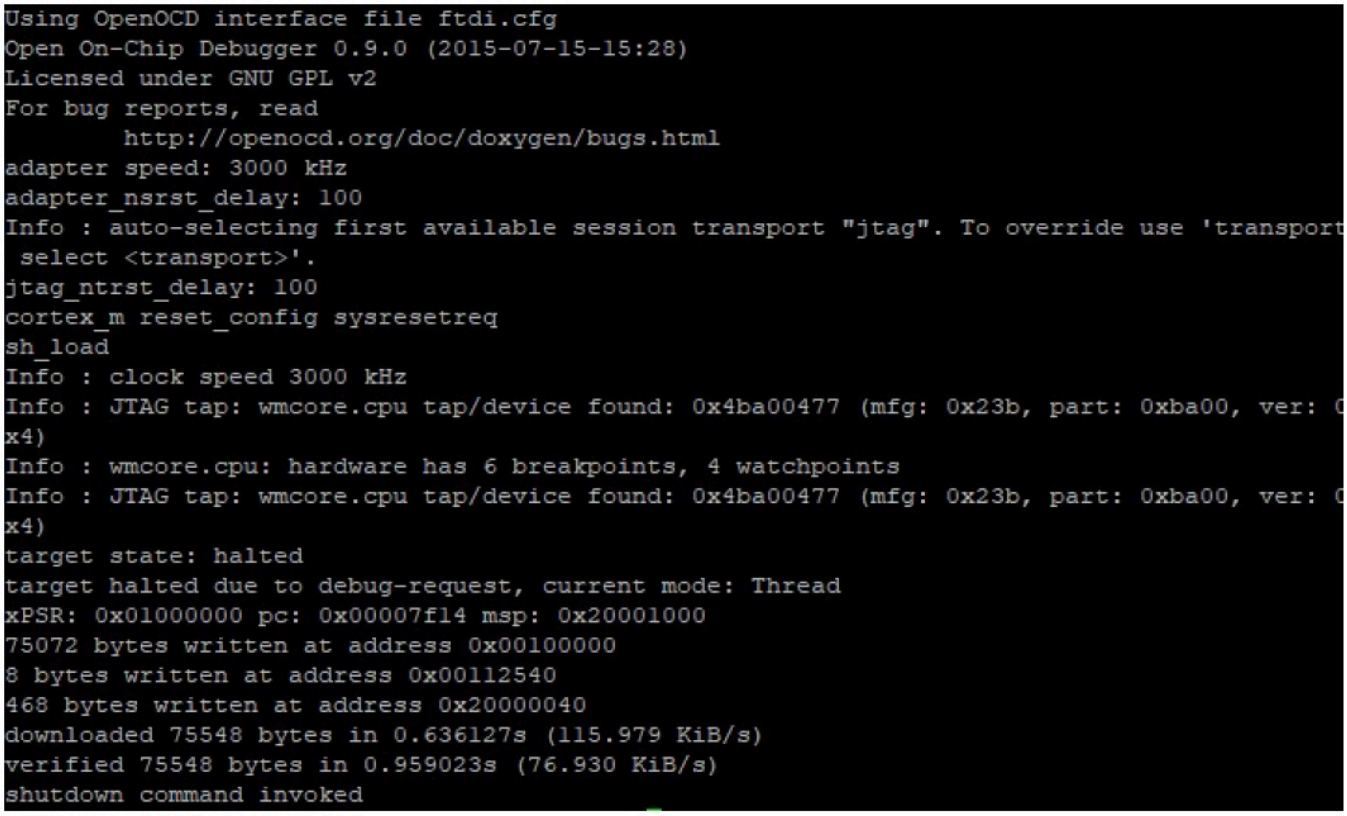
7. (Optional) Verwenden Sie als alternative Methode zum Testen Ihres Images das Flashprog-Hilfsprogramm, um das Mikrocontroller-Image vom Host direkt in den Mikrocontroller-RAM zu kopieren. Das Bild wird nicht im Flash kopiert und geht daher verloren, wenn Sie den Mikrocontroller neu starten.

   Das Laden des Firmware-Images in den SRAM ist ein schnellerer Vorgang, da die Ausführungsdatei sofort gestartet wird. Diese Methode wird hauptsächlich für die iterative Entwicklung verwendet.

   Verwenden Sie die folgenden Befehle, um die Firmware in den SRAM zu laden.

   ```
   cd amzsdk_bundle-x.y.z
   ./lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/OpenOCD/ramload.py
    build/cmake/vendors/marvell/mw300_rd/aws_demos.axf
   ```

   Die Befehlsausgabe ist in der folgenden Abbildung dargestellt.



Wenn die Befehlsausführung abgeschlossen ist, sollten Sie die Protokolle der Demo-App sehen.

Erstellen Sie die FreeRTOS-Demo mit der Eclipse-IDE und führen Sie sie aus

1. Bevor Sie einen Eclipse-Workspace einrichten, müssen Sie den `cmake` Befehl ausführen.

   Führen Sie den folgenden Befehl aus, um mit dem `aws_demos` Eclipse-Projekt zu arbeiten.

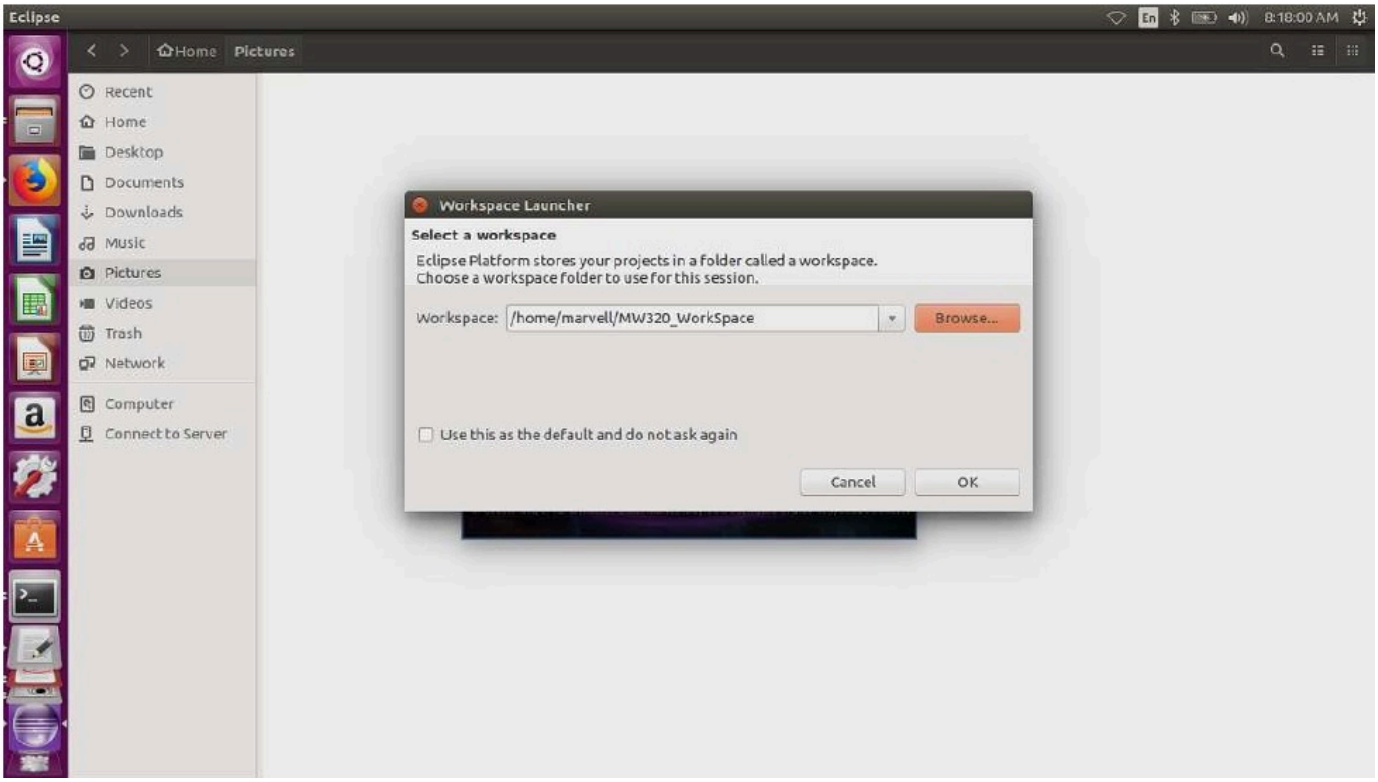
   ```
   cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S .  -Bbuild -
   DAFR_ENABLE_TESTS=0
   ```

   Führen Sie den folgenden Befehl aus, um mit dem `aws_tests` Eclipse-Projekt zu arbeiten.

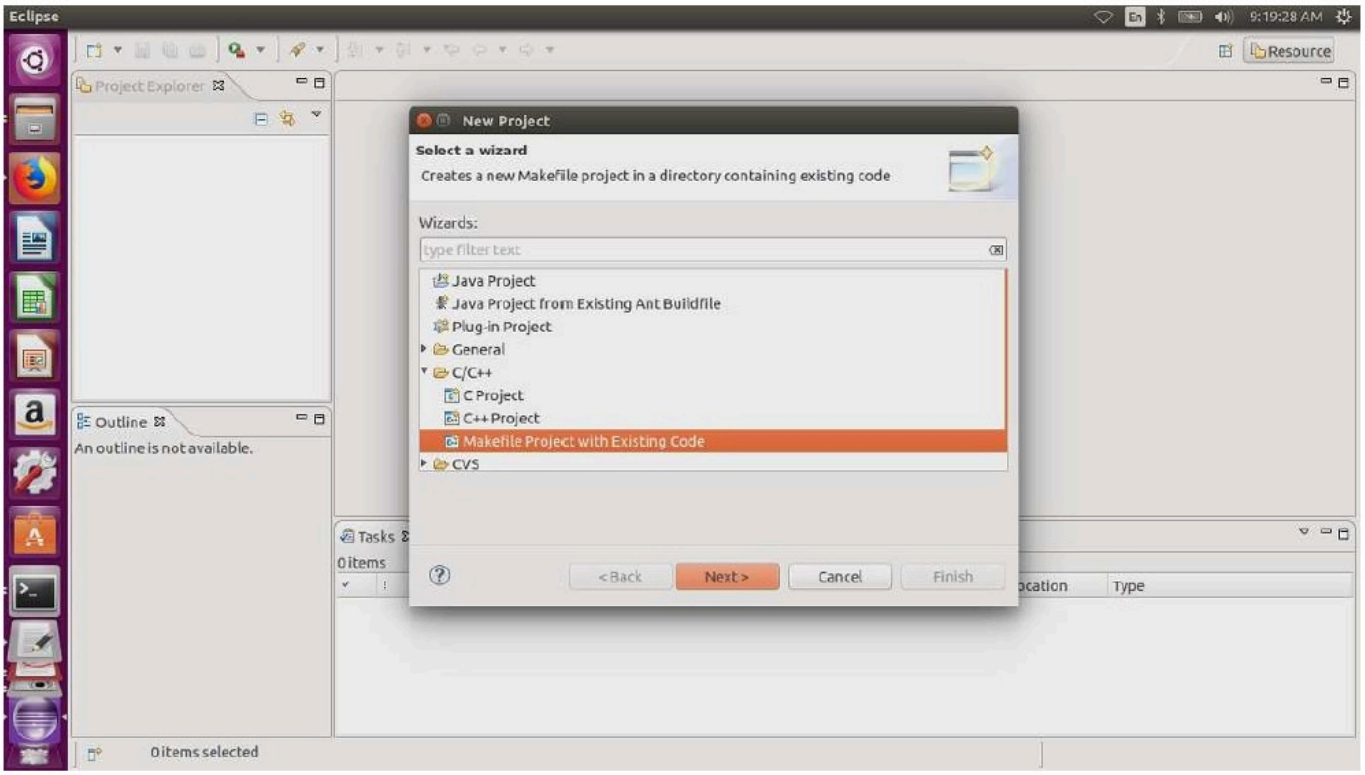
   ```
   cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S .  -Bbuild -
   DAFR_ENABLE_TESTS=1
   ```

> **Tip**
>
> Führen Sie den `cmake` Befehl jedes Mal aus, wenn Sie zwischen dem `aws_demos` Projekt und dem `aws_tests` Projekt wechseln.

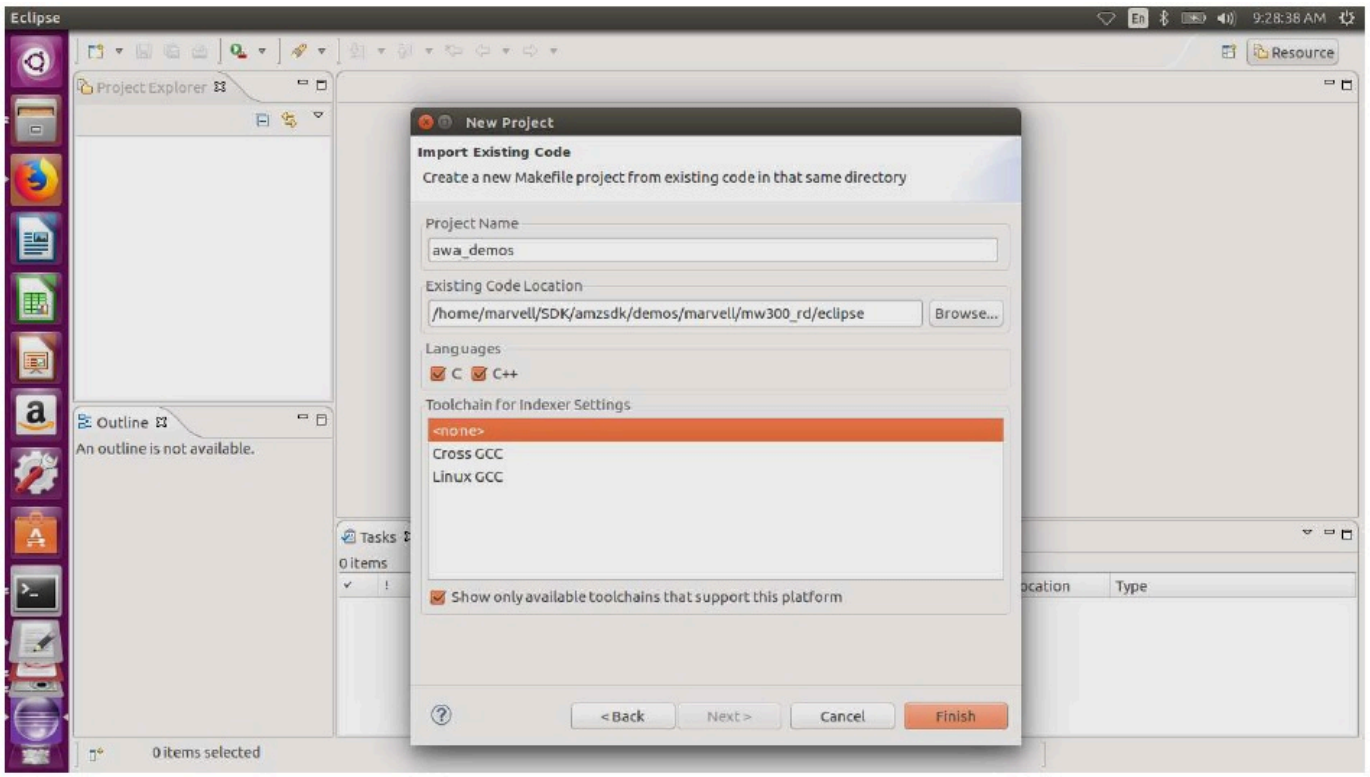
2. Öffnen Sie Eclipse und wählen Sie, wenn Sie dazu aufgefordert werden, Ihren Eclipse-Arbeitsbereich aus, wie in der Abbildung unten gezeigt.



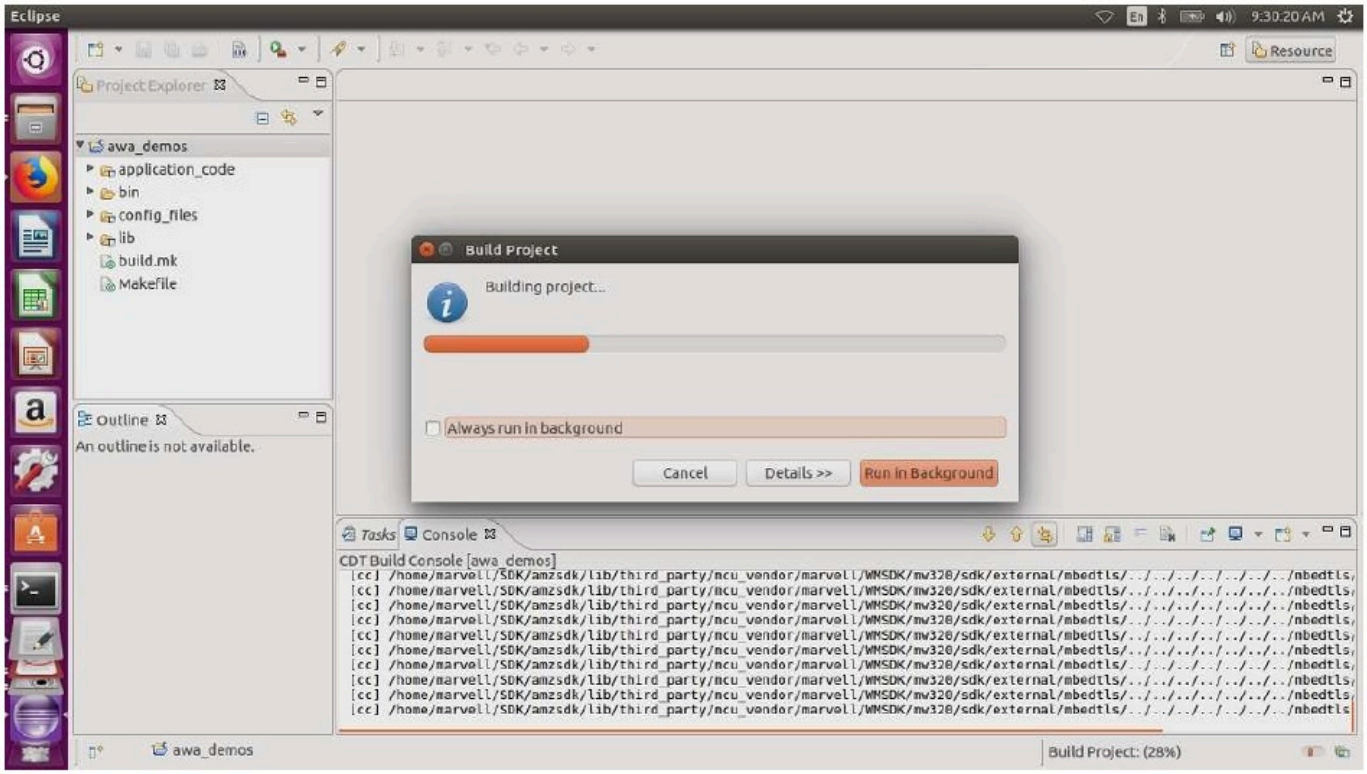
3. Wählen Sie die Option, um ein Makefile-Projekt zu erstellen: mit vorhandenem Code, wie in der Abbildung unten gezeigt.



4. Wählen Sie „Durchsuchen“, geben Sie das Verzeichnis mit dem vorhandenen Code an und wählen Sie dann „Fertig stellen“.



5. Wählen Sie im Navigationsbereich im Projektexplorer aws_demos aus. Klicken Sie mit der rechten Maustaste auf aws_demos, um das Menü zu öffnen, und wählen Sie dann Build.



Wenn der Build erfolgreich ist, wird die Datei generiert. `build/cmake/vendors/marvell/mw300_rd/aws_demos.bin`

6. Verwenden Sie die Befehlszeilentools, um die Layout-Datei (`layout.txt`), die Boot2-Binärdatei (`boot2.bin`), die MCU-Firmware-Binärdatei (`aws_demos.bin`) und die Wi-Fi-Firmware zu flashen.

   a. Bevor Sie das Firmware-Image flashen, bereiten Sie den Flash des Entwicklungsboards mit den gemeinsamen Komponenten Layout und Boot2 vor. Verwenden Sie die folgenden Befehle.

```
cd amzsdk_bundle-x.y.z
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py -l ./vendors/
marvell/WMSDK/mw320/sdk/tools/OpenOCD/mw300/layout.txt --boot2 ./vendors/
marvell/WMSDK/mw320/boot2/bin/boot2.bin
```

   Der `flashprog` Befehl initiiert Folgendes:

- Layout — Das Flashprog-Hilfsprogramm wird zunächst angewiesen, ein Layout in den Flash zu schreiben. Das Layout ähnelt den Partitionsinformationen für den Flash. Das Standardlayout befindet sich unter`/lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/OpenOCD/mw300/layout.txt`.
- Boot2 — Dies ist der vom WMSDK verwendete Bootloader. Der Befehl flashprog schreibt auch einen Bootloader in den Flash. Es ist die Aufgabe des Bootloaders, das Firmware-Image des Mikrocontrollers zu laden, nachdem es geflasht wurde. Stellen Sie sicher, dass Sie dieselbe Ausgabe wie in der Abbildung unten erhalten.

```
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00007f14 msp: 0x20001000
29088 bytes written at address 0x00100000
downloaded 29088 bytes in 0.245458s (115.728 KiB/s)
verified 29088 bytes in 0.350004s (81.160 KiB/s)
semihosting is enabled

Flashprog version: 2.1.0
Erasing primary flash...done
Writing new flash layout...done
Writing "boot2" @0x0 (primary)...done
semihosting: *** application exited ***
Flashprog Complete
shutdown command invoked


target state: halted
target halted due to breakpoint, current mode: Thread
xPSR: 0x21000000 pc: 0x00100658 msp: 0x0015ffe4, semihosting
```

b. Die Firmware verwendet den Wi-Fi-Chipsatz für ihre Funktionalität, und der Wi-Fi-Chipsatz verfügt über eine eigene Firmware, die auch im Flash vorhanden sein muss. Sie verwenden das `flashprog.py` Hilfsprogramm, um die Wi-Fi-Firmware auf die gleiche Weise zu flashen, wie Sie es beim Flashen des Boot2-Bootloaders und der MCU-Firmware getan haben. Verwenden Sie die folgenden Befehle, um die Wi-Fi-Firmware zu flashen.

```
cd amzsdk_bundle-x.y.z
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py --wififw ./
vendors/marvell/WMSDK/mw320/wifi-firmware/mw30x/mw30x_uapsta_W14.88.36.p135.bin
```

Stellen Sie sicher, dass die Ausgabe des Befehls der folgenden Abbildung ähnelt.

```
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00007f14 msp: 0x20001000
29088 bytes written at address 0x00100000
downloaded 29088 bytes in 0.245498s (115.709 KiB/s)
verified 29088 bytes in 0.350229s (81.108 KiB/s)
semihosting is enabled

Flashprog version: 2.1.0
Writing "wififw" @0x12a000 (primary).....done
semihosting: *** application exited ***
Flashprog Complete
shutdown command invoked


target state: halted
target halted due to breakpoint, current mode: Thread
xPSR: 0x21000000 pc: 0x00100658 msp: 0x0015ffe4, semihosting
```

c. Verwenden Sie die folgenden Befehle, um die MCU-Firmware zu flashen.

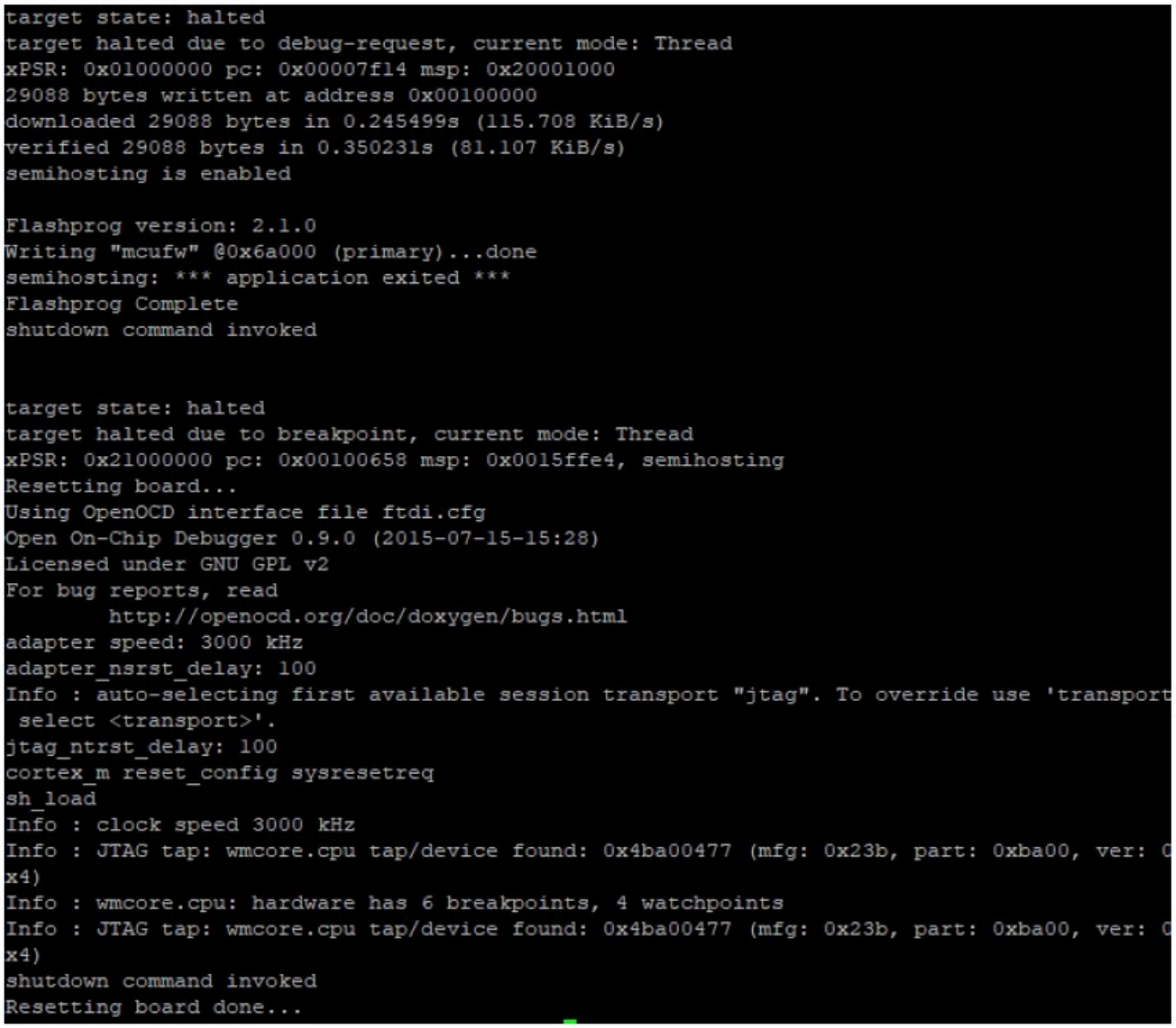
```
cd amzsdk_bundle-x.y.z
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py --mcufw build/
cmake/vendors/marvell/mw300_rd/aws_demos.bin -r
```

d. Setzen Sie das Board zurück. Sie sollten die Protokolle für die Demo-App sehen.

e. Um die Test-App auszuführen, flashen Sie die `aws_tests.bin` Binärdatei, die sich im selben Verzeichnis befindet.

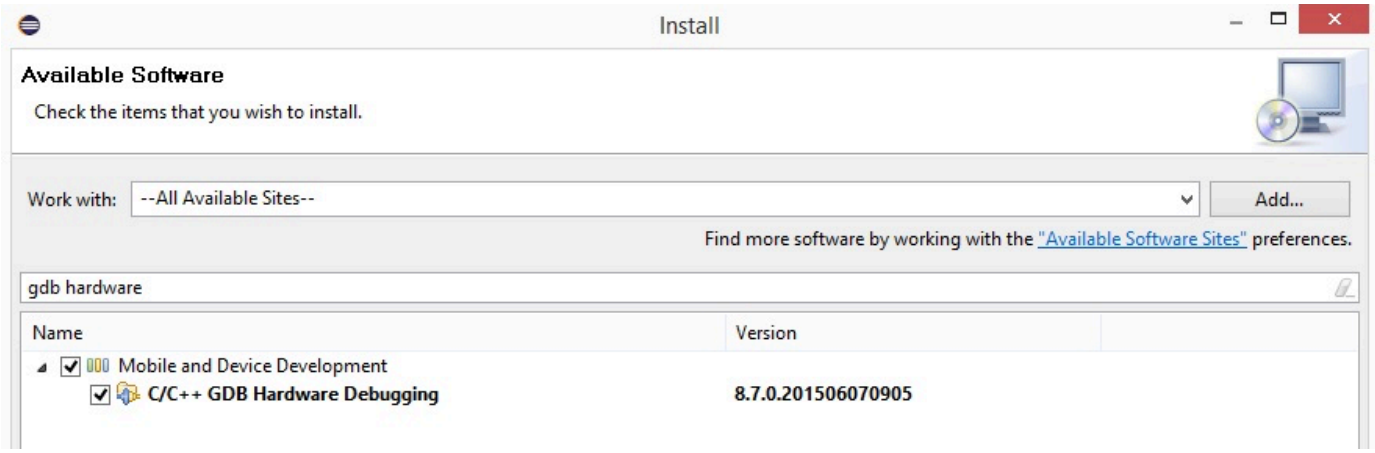
```
cd amzsdk_bundle-x.y.z
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py --mcufw build/
cmake/vendors/marvell/mw300_rd/aws_tests.bin -r
```

Ihre Befehlsausgabe sollte der in der folgenden Abbildung gezeigten ähneln.



Debuggen

- Starten Sie Eclipse und wählen Sie Hilfe und dann Neue Software installieren. Wählen Sie im Menü „Arbeiten mit“ die Option „Alle verfügbaren Websites“. Geben Sie den Filtertext ein`GDB Hardware`. Wählen Sie die C/C++-GDB-Hardware-Debugging-Option und installieren Sie das Plugin.



Fehlerbehebung

Probleme mit dem Netzwerk

Überprüfen Sie Ihre Netzwerkanmeldedaten. Weitere Informationen finden Sie unter „Bereitstellung“. Erstellen und starten Sie das FreeRTOS-Demoprojekt

Zusätzliche Protokolle aktivieren

- Aktiviere platinenspezifische Protokolle.

  Aktiviert Aufrufe der Funktion `prvMiscInitialization` in der `main.c` Datei für Tests oder Demos. `wmstdio_init(UART0_ID, 0)`

- Wi-Fi-Protokolle aktivieren

  Aktivieren Sie das Makro `CONFIG_WLCMGR_DEBUG` in der *`freertos`*`/vendors/marvell/WMSDK/mw320/sdk/src/incl/autoconf.h` Datei.

GDB verwenden

Wir empfehlen, die zusammen mit dem SDK verpackten gdb Befehlsdateien `arm-none-eabi-gdb` und zu verwenden. Navigieren Sie zum Verzeichnis .

```
cd freertos/lib/third_party/mcu_vendor/marvell/WMSDK/mw320
```

Führen Sie den folgenden Befehl (in einer einzigen Zeile) aus, um eine Verbindung zu GDB herzustellen.

```
arm-none-eabi-gdb -x ./sdk/tools/OpenOCD/gdbinit ../../../../../../build/cmake/
vendors/marvell/mw300 _rd/aws_demos.axf
```

Erste Schritte mit dem MediaTek MT7697 Hx Development Kit

> **Important**
>
> Diese Referenzintegration wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Dieses Tutorial enthält Anweisungen für die ersten Schritte mit dem Hx Development Kit. MediaTek MT7697 Wenn Sie das MediaTek MT7697 Hx Development Kit nicht haben, besuchen Sie den AWS Partner-Gerätekatalog, um eines von unserem Partner zu erwerben.

Bevor Sie beginnen, müssen Sie FreeRTOS konfigurieren AWS IoT und herunterladen, um Ihr Gerät mit der AWS Cloud zu verbinden. Detaillierte Anweisungen finden Sie unter Erste Schritte. In diesem Tutorial wird der Pfad zum FreeRTOS-Download-Verzeichnis als bezeichnet. *`freertos`*

-Übersicht

Dieses Tutorial enthält Anweisungen für die folgenden ersten Schritte:

1. Installieren von Software auf dem Host-Computer zum Entwickeln und Debuggen eingebetteter Anwendungen für Ihr Mikrocontroller-Board.
2. Cross-Compilierung einer FreeRTOS-Demo-Anwendung zu einem Binär-Image.
3. Laden des binären Anwendungs-Image auf Ihr Board und Ausführen der Anwendung.
4. Interaktion mit der Anwendung, die auf Ihrem Board über eine serielle Verbindung ausgeführt wird, zu Überwachungs- und Debuggingzwecken.

Einrichten Ihrer Entwicklungsumgebung

Bevor Sie Ihre Umgebung einrichten, schließen Sie Ihren Computer an den USB-Anschluss des MediaTek MT7697 Hx Development Kit an.

Herunterladen und Installieren von Keil MDK

Sie können das GUI-basierte Keil Microcontroller Development Kit (MDK) verwenden, um FreeRTOS-Projekte auf Ihrem Board zu konfigurieren, zu erstellen und auszuführen. Keil MDK beinhaltet die µVision IDE und den µVision Debugger.

> **Note**
>
> Keil MDK wird ausschließlich auf 64-Bit-Computern mit Windows 7, Windows 8 oder Windows 10 unterstützt.

So laden Sie das Keil MDK herunter und installieren es

1. Rufen Sie die Seite Erste Schritte mit Keil MDK auf und wählen Sie Download MDK-Core (MDK-Core herunterladen).
2. Geben Sie Ihre Daten ein und senden Sie sie, um sich bei Keil zu registrieren.
3. Klicken Sie mit der rechten Maustaste auf die ausführbare MDK-Datei und speichern Sie das Keil MDK-Installationsprogramm auf Ihrem Computer.
4. Öffnen Sie das Keil MDK-Installationsprogramm und befolgen Sie die Schritte zum Abschluss der Installation. Stellen Sie sicher, dass Sie das MediaTek Gerätepaket (Serie) installieren. MT76x7

Herstellen einer seriellen Verbindung

Connect das Board mit einem USB-Kabel mit Ihrem Host-Computer. Im Windows-Geräte-Manager wird ein COM-Anschluss angezeigt. Zum Debuggen können Sie mit einem Terminal-Hilfsprogramm wie HyperTerminal oder TeraTerm eine Sitzung für den Port öffnen.

Überwachung von MQTT-Nachrichten in der Cloud

Bevor Sie das FreeRTOS-Demoprojekt ausführen, können Sie den MQTT-Client in der AWS IoT Konsole so einrichten, dass er die Nachrichten überwacht, die Ihr Gerät an die Cloud sendet. AWS

Um das MQTT-Thema mit dem MQTT-Client zu abonnieren AWS IoT

1. Melden Sie sich bei der AWS IoT-Konsole an.
2. Wählen Sie im Navigationsbereich Test und dann MQTT-Testclient aus, um den MQTT-Client zu öffnen.

3. Geben Sie im Feld Subscription topic (Abonnementthema) die Option ***your-thing-name*****example/topic** ein und wählen Sie dann Subscribe to topic (Thema abonnieren).

Wenn das Demo-Projekt erfolgreich auf Ihrem Gerät ausgeführt wird, sehen Sie „Hello World!“ mehrfach zu dem Thema gesendet, das Sie abonniert haben.

Erstellen und starten Sie das FreeRTOS-Demoprojekt mit Keil MDK

Um das FreeRTOS-Demo-Projekt in Keil µVision zu erstellen

1. Öffnen Sie im Startmenü Keil µVision 5.
2. Öffnen Sie die Projektdatei `projects/mediatek/mt7697hx-dev-kit/uvision/aws_demos/aws_demos.uvprojx`.
3. Wählen Sie im Menü Project (Projekt) aus und wählen Sie dann Build target (Ziel erstellen).

   Nachdem der Code erstellt wurde, sehen Sie die ausführbare Datei der Demo als `projects/mediatek/mt7697hx-dev-kit/uvision/aws_demos/out/Objects/aws_demo.axf`.

Um das FreeRTOS-Demo-Projekt auszuführen

1. Stellen Sie das MediaTek MT7697 Hx Development Kit in den PROGRAMM-Modus.

   Um den Modus PROGRAM festzulegen, halten Sie die PROG-Taste gedrückt. Halten Sie die PROG-Taste weiterhin gedrückt, drücken Sie die RESET-Taste und lassen Sie sie wieder los und lassen Sie anschließend die PROG-Taste los.
2. Wählen Sie im Menü Flash und anschließend Configure Flash Tools (Flash-Tools konfigurieren) aus.
3. Wählen Sie in Options for Target (Optionen für Ziel) '**aws_demo**' die Registerkarte Debug (Debuggen) aus. Wählen Sie Use (Verwenden), legen Sie als Debugger CMSIS-DAP Debugger fest und wählen Sie dann OK.
4. Wählen Sie im Menü die Option Flash und anschließend Download (Herunterladen) aus.

   µVision benachrichtigt Sie, wenn der Download abgeschlossen ist.
5. Verwenden Sie ein Terminal-Dienstprogramm, um das Fenster der seriellen Konsole zu öffnen. Legen Sie den seriellen Port auf 115200 bps, keine Parität, 8 Bits und 1 Stoppbit fest.
6. Wählen Sie die RESET-Taste auf Ihrem MediaTek MT7697 Hx Development Kit.

Fehlerbehebung

Debuggen von FreeRTOS-Projekten in Keil µVision

Derzeit müssen Sie das in Keil µVision enthaltene MediaTek Paket bearbeiten, bevor Sie das FreeRTOS-Demoprojekt für mit Keil µVision debuggen können. MediaTek

Um das MediaTek Paket zum Debuggen von FreeRTOS-Projekten zu bearbeiten

1. Suchen und öffnen Sie die Datei `Keil_v5\ARM\PACK\.Web\MediaTek.MTx.pdsc` in Ihrem Keil-MDK-Installationsordner.
2. Ersetzen Sie alle Instances von `flag = Read32(0x20000000);` mit `flag = Read32(0x0010FBFC);`.
3. Ersetzen Sie alle Instances von `Write32(0x20000000, 0x76877697);` mit `Write32(0x0010FBFC, 0x76877697);`.

So starten Sie das Debugging des Projekts

1. Wählen Sie im Menü Flash und anschließend Configure Flash Tools (Flash-Tools konfigurieren) aus.
2. Wählen Sie die Registerkarte Target (Ziel) und wählen Sie dann Read/Write Memory Areas (Speicherbereiche lesen/Auf Speicherbereiche schreiben) aus. Bestätigen Sie das IRAM1 und beide IRAM2 sind ausgewählt.
3. Wählen Sie die Registerkarte Debug (Debuggen) und anschließend CMSIS-DAP Debugger.
4. Öffnen Sie `vendors/mediatek/boards/mt7697hx-dev-kit/aws_demos/application_code/main.c` und legen Sie das Makro `MTK_DEBUGGER` auf 1 fest.
5. Erstellen Sie das Demo-Projekt in µVision neu.
6. Stellen Sie das MediaTek MT7697 Hx Development Kit in den PROGRAMM-Modus.

   Um den Modus PROGRAM festzulegen, halten Sie die PROG-Taste gedrückt. Halten Sie die PROG-Taste weiterhin gedrückt, drücken Sie die RESET-Taste und lassen Sie sie wieder los und lassen Sie anschließend die PROG-Taste los.
7. Wählen Sie im Menü die Option Flash und anschließend Download (Herunterladen) aus.

   µVision benachrichtigt Sie, wenn der Download abgeschlossen ist.
8. Drücken Sie die RESET-Taste an Ihrem MediaTek MT7697 Hx Development Kit.

9. Wählen Sie im µVision-Menü Debug und anschließend Debug-Sitzung starten/beenden. Das Fenster Call Stack + Locals (Aufrufstapel und Lokale) öffnet sich, wenn Sie eine Debug-Sitzung starten.
10. Wählen Sie im Menü Debug, und dann Stop (Anhalten) aus, um die Codeausführung anzuhalten. Der Programmzähler stoppt an der folgenden Zeile:

```
{ volatile int wait_ice = 1 ; while ( wait_ice ) ; }
```

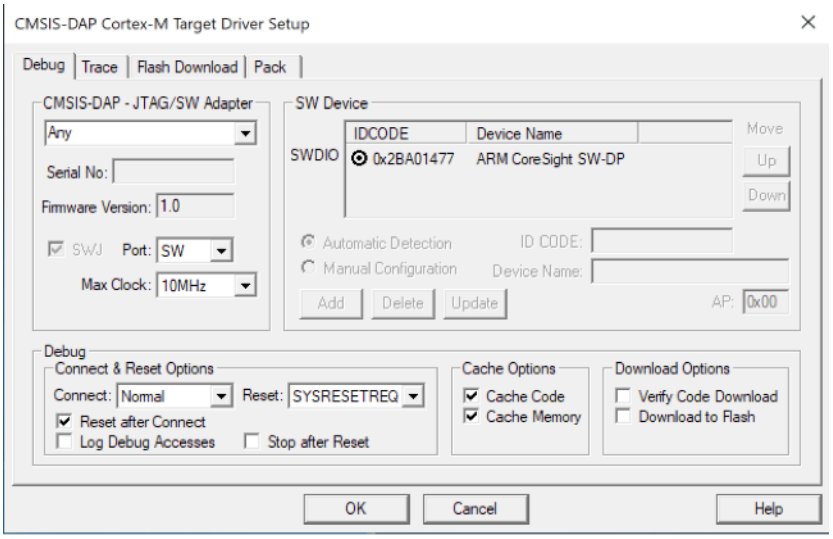
11. Ändern Sie im Fenster Call Stack + Locals (Aufrufstapel und Lokale) den Wert für `wait_ice` in `0`.
12. Legen Sie Haltepunkte im Quellcode Ihres Projekts fest und führen Sie den Code aus.

Fehlerbehebung für die IDE-Debugger-Einstellungen

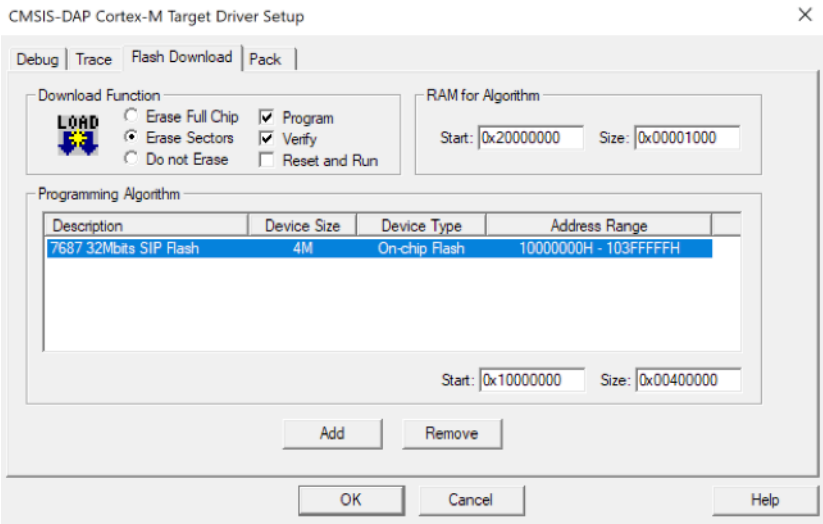
Wenn Sie Probleme beim Debuggen einer Anwendung haben, sind Ihre Debugger-Einstellungen möglicherweise nicht korrekt.

So überprüfen Sie, ob Ihre Debugger-Einstellungen korrekt sind

1. Öffnen Sie Keil µVision.
2. Klicken Sie mit der rechten Maustaste auf das Projekt `aws_demos`, wählen Sie Options (Optionen) und auf der Registerkarte Utilities (Dienstprogramme) Settings (Einstellungen) aus, neben -- Use Debug Driver -- (Debug-Treiber verwenden).
3. Überprüfen Sie, ob die Einstellungen auf der Registerkarte Debug (Debuggen) wie folgt angezeigt werden:



4. Überprüfen Sie, ob die Einstellungen auf der Registerkarte Flash Download (Flash-Download9 wie folgt angezeigt werden:



Allgemeine Informationen zur Problembehandlung bei Getting Started with FreeRTOS finden Sie unter. Fehlerbehebung – Erste Schritte

Erste Schritte mit dem Microchip Curiosity PIC32 MZ EF

Important

Diese Referenzintegration wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Note

In Absprache mit Microchip entfernen wir das Curiosity PIC32 MZEF (DM320104) aus dem Hauptzweig des FreeRTOS Reference Integration-Repositorys und werden es nicht mehr in neuen Releases anbieten. Microchip hat eine offizielle Mitteilung veröffentlicht, dass der PIC32 MZEF (DM320104) nicht mehr für neue Designs empfohlen wird. Auf die PIC32 MZEF-Projekte und den Quellcode kann weiterhin über die Tags der vorherigen Version zugegriffen werden. Microchip empfiehlt Kunden, das Curiosity PIC32MZ-EF-2.0-Entwicklungsboard () für neue Designs zu verwenden. DM320209 Die PIC32 MZv1 Plattform befindet sich immer

noch in Version 202012.00 des FreeRTOS Reference Integration-Repositorys. Die Plattform wird jedoch von v202107.00 der FreeRTOS Reference nicht mehr unterstützt.

Dieses Tutorial enthält Anweisungen für die ersten Schritte mit dem Microchip Curiosity MZ EF. PIC32 Wenn Sie das Microchip Curiosity PIC32 MZ EF-Paket nicht haben, besuchen Sie den AWS Partnergerätekatalog, um eines bei unserem Partner zu kaufen.

Das Bundle enthält die folgenden Elemente:

- Curiosity PIC32 MZ EF Entwicklungsboard
- MikroElectronika USB-UART-Klickboard
- MikroElectronika WiFi 7 klicken Sie auf Board
- PIC32 LAN8720 PHY-Tochterboard

Außerdem benötigen Sie die folgenden Elemente für das Debugging:

- MPLAB Snap In-Circuit Debugger
- (Optional) Kabelsatz mit PICkit 3 Programmierkabeln

Bevor Sie beginnen, müssen Sie FreeRTOS konfigurieren AWS IoT und herunterladen, um Ihr Gerät mit der AWS Cloud zu verbinden. Detaillierte Anweisungen finden Sie unter Erste Schritte.

⚠ Important

- In diesem Thema wird der Pfad zum FreeRTOS-Download-Verzeichnis als bezeichnet. *`freertos`*
- Leerzeichen im *`freertos`*-Pfad können Build-Fehler verursachen. Stellen Sie beim Klonen oder Kopieren des Repositorys sicher, dass der von Ihnen erstellte Pfad keine Leerzeichen enthält.
- Die maximale Länge eines Dateipfades bei Microsoft Windows ist 260 Zeichen. Lange FreeRTOS-Download-Verzeichnispfade können zu Build-Fehlern führen.
- Da der Quellcode symbolische Links enthalten kann, müssen Sie, wenn Sie Windows zum Extrahieren des Archivs verwenden, möglicherweise:
  - Entwicklermodus aktivieren oder

- Verwenden Sie eine Konsole mit Administratorberechtigungen.

Auf diese Weise kann Windows beim Extrahieren des Archivs ordnungsgemäß symbolische Links erstellen. Andernfalls werden symbolische Links als normale Dateien geschrieben, die die Pfade der symbolischen Links als Text enthalten oder leer sind. Weitere Informationen finden Sie im Blogeintrag Symlinks in Windows 10! .

Wenn Sie Git unter Windows verwenden, müssen Sie den Entwicklermodus aktivieren oder Sie müssen:

- Setzen `core.symlinks` Sie ihn mit dem folgenden Befehl auf true:

```
git config --global core.symlinks true
```

- Verwenden Sie immer dann eine Konsole mit Administratorrechten, wenn Sie einen Git-Befehl verwenden, der in das System schreibt (z. B.git pull,git clone, undgit submodule update --init --recursive).

-Übersicht

Dieses Tutorial enthält Anweisungen für die folgenden ersten Schritte:

1. Verbinden Ihres Boards mit einem Host-Computer.
2. Installieren von Software auf dem Host-Computer zum Entwickeln und Debuggen eingebetteter Anwendungen für Ihr Mikrocontroller-Board.
3. Cross-Compilierung einer FreeRTOS-Demo-Anwendung zu einem Binär-Image.
4. Laden des binären Anwendungs-Image auf Ihr Board und Ausführen der Anwendung.
5. Interaktion mit der Anwendung, die auf Ihrem Board über eine serielle Verbindung ausgeführt wird, zu Überwachungs- und Debuggingzwecken.

Richten Sie die Microchip Curiosity MZ EF-Hardware ein PIC32

1. Connect das MikroElectronika USB-UART-Click-Board mit dem MicroBus 1-Anschluss auf dem Microchip Curiosity PIC32 MZ EF.
2. Connect die PIC32 LAN8720 PHY-Tochterplatine mit dem J18-Header auf dem Microchip Curiosity PIC32 MZ EF.

3. Connect das MikroElectronika USB UART Click Board über ein USB-A-auf-USB-Mini-B-Kabel mit Ihrem Computer.
4. Verwenden Sie eine der folgenden Optionen, um Ihr Board mit dem Internet zu verbinden:
   - Um Wi-Fi zu verwenden, verbinden Sie das MikroElectronika Wi-Fi 7-Klick-Board mit dem MicroBus 2-Anschluss auf dem Microchip Curiosity PIC32 MZ EF. Siehe Konfiguration der FreeRTOS-Demos.
   - Um die Microchip Curiosity PIC32 MZ EF-Karte über Ethernet mit dem Internet zu verbinden, verbinden Sie die PIC32 LAN8720 PHY-Tochterplatine mit dem J18-Header auf dem Microchip Curiosity MZ EF. PIC32 Connect ein Ende eines Ethernet-Kabels mit der LAN8720 PHY-Tochterplatine. Schließen Sie das andere Ende an Ihren Router oder an einen anderen Internet-Port an. Sie müssen auch das Präprozessor-Makro `PIC32_USE_ETHERNET` definieren.
5. Falls noch nicht geschehen, löten Sie den Winkelstecker an den ICSP-Header des Microchip Curiosity MZ EF. PIC32
6. Connect ein Ende des ICSP-Kabels des PICkit 3-Programmierkabel-Kits mit dem Microchip Curiosity PIC32 MZ EF.

   Wenn Sie nicht über das Kit mit PICkit 3 Programmierkabeln verfügen, können Sie stattdessen Drahtbrücken von M-F Dupont verwenden, um die Verbindung zu verkabeln. Hinweis: Der weiße Kreis zeigt die Position von Pin 1.
7. Schließen Sie das andere Ende des ICSP-Kabels (oder Jumper) an den MPLAB Snap Debugger an. Pin 1 des 8-Pin-SIL-Programmiersteckverbinders ist mit dem schwarzen Dreieck unten rechts auf dem Board markiert.

   Stellen Sie sicher, dass alle Kabel zu Pin 1 auf dem Microchip Curiosity PIC32 MZ EF, gekennzeichnet durch den weißen Kreis, mit Pin 1 am MPLAB Snap Debugger übereinstimmen.

   Weitere Informationen zum MPLAB Snap Debugger finden Sie im Informationsblatt zum MPLAB Snap In-Circuit Debugger.

Richten Sie die Microchip Curiosity PIC32 MZ EF-Hardware mithilfe von On Board (PKOB) ein PICkit

Es wird empfohlen, das Setup-Verfahren im vorherigen Abschnitt zu befolgen. Sie können jedoch FreeRTOS-Demos mit grundlegendem Debugging mithilfe des integrierten PICkit On Board (PKOB) evaluieren und ausführen, indem Sie die folgenden Schritte ausführen. programmer/debugger

1. Connect das MikroElectronika USB-UART-Click-Board mit dem MicroBus 1-Anschluss auf dem Microchip Curiosity PIC32 MZ EF.
2. Führen Sie einen der folgenden Schritte aus, um Ihr Board mit dem Internet zu verbinden:
   - Um Wi-Fi zu verwenden, verbinden Sie das MikroElectronika Wi-Fi 7-Klick-Board mit dem MicroBus 2-Anschluss auf dem Microchip Curiosity PIC32 MZ EF. (Folgen Sie den Schritten unter „So konfigurieren Sie Ihr WLAN“ in Konfiguration der FreeRTOS-Demos.
   - Um die Microchip Curiosity PIC32 MZ EF-Karte über Ethernet mit dem Internet zu verbinden, verbinden Sie die PIC32 LAN8720 PHY-Tochterplatine mit dem J18-Header auf dem Microchip Curiosity MZ EF. PIC32 Connect ein Ende eines Ethernet-Kabels mit der LAN8720 PHY-Tochterplatine. Schließen Sie das andere Ende an Ihren Router oder an einen anderen Internet-Port an. Sie müssen auch das Präprozessor-Makro `PIC32_USE_ETHERNET` definieren.
3. Connect den USB-Micro-B-Anschluss mit der Bezeichnung „USB DEBUG“ auf der Microchip Curiosity PIC32 MZ EF-Karte über ein USB-Typ-A-auf-USB-Micro-B-Kabel mit Ihrem Computer.
4. Connect das MikroElectronika USB UART Click Board über ein USB-A-auf-USB-Mini-B-Kabel mit Ihrem Computer.

Einrichten Ihrer Entwicklungsumgebung

> **Note**
>
> Das FreeRTOS-Projekt für dieses Gerät basiert auf MPLAB Harmony v2. Um das Projekt zu erstellen, müssen Sie Versionen der MPLAB-Tools verwenden, die mit Harmony v2 kompatibel sind, wie Version 2.10 des XC32 MPLAB-Compilers und Versionen 2.X.X des MPLAB Harmony Configurator (MHC).

1. Installieren Sie Python Version 3.x oder höher.
2. Installieren Sie die MPLAB X-IDE:

   > **Note**
   >
   > FreeRTOS AWS Reference Integrations v202007.00 wird derzeit nur auf Version 3.5 unterstützt. MPLabv5 Frühere Versionen von FreeRTOS AWS Reference Integrations werden auf Version 4.0 unterstützt. MPLabv5

MPLabv53.5 Downloads

- Integrierte Entwicklungsumgebung MPLAB X für Windows
- MPLAB X Integrierte Entwicklungsumgebung für macOS
- Integrierte MPLAB X-Entwicklungsumgebung für Linux

Letzte MPLab Downloads (.40) MPLabv5

- MPLAB-X-IDE (Integrierte Entwicklungsumgebung) für Windows
- MPLAB-X-IDE (Integrierte Entwicklungsumgebung) für macOS
- MPLAB-X-IDE (Integrierte Entwicklungsumgebung) für Linux

3. Installieren Sie den MPLAB-Compiler XC32 :

- MPLAB /32++ Compiler für Windows XC32
- MPLAB XC32 /32++ Compiler für macOS
- MPLAB /32++ Compiler für Linux XC32

4. Starten Sie einen UART-Terminal-Emulator und öffnen Sie eine Verbindung mit den folgenden Einstellungen:

- Baudrate: 115200
- Daten: 8 Bit
- Parität: Keine
- Stop-Bits: 1
- Flusssteuerung: Keine

Überwachung von MQTT-Nachrichten in der Cloud

Bevor Sie das FreeRTOS-Demoprojekt ausführen, können Sie den MQTT-Client in der AWS IoT Konsole so einrichten, dass er die Nachrichten überwacht, die Ihr Gerät an die Cloud sendet. AWS

Um das MQTT-Thema mit dem MQTT-Client zu abonnieren AWS IoT

1. Melden Sie sich bei der AWS IoT -Konsole an.

2. Wählen Sie im Navigationsbereich Test und dann MQTT-Testclient aus, um den MQTT-Client zu öffnen.
3. Geben Sie im Feld Subscription topic (Abonnementthema) die Option ***your-thing-name*/example/topic** ein und wählen Sie dann Subscribe to topic (Thema abonnieren).

Wenn das Demo-Projekt erfolgreich auf Ihrem Gerät ausgeführt wird, sehen Sie „Hello World!“ mehrfach zu dem Thema gesendet, das Sie abonniert haben.

Erstellen Sie das FreeRTOS-Demoprojekt und führen Sie es aus

Öffnen Sie die FreeRTOS-Demo in der MPLAB-IDE

1. Öffnen Sie die MPLAB-IDE. Wenn Sie mehr als eine Version des Compilers installiert haben, müssen Sie den Compiler, den Sie verwenden möchten, innerhalb der IDE auswählen.
2. Wählen Sie im Menü File (Datei) die Option Open project (Projekt öffnen).
3. Navigieren Sie zu `projects/microchip/curiosity_pic32mzef/mplab/aws_demos` und öffnen Sie es.
4. Klicken Sie auf Open project (Projekt öffnen).

> **Note**
>
> Wenn Sie das Projekt zum ersten Mal öffnen, erhalten Sie möglicherweise eine Fehlermeldung über den Compiler. Navigieren Sie in der IDE zu Tools, Options (Optionen), Embedded und wählen Sie dann den Compiler aus, den Sie für Ihr Projekt verwenden.

Um Ethernet für die Verbindung zu verwenden, müssen Sie das Präprozessor-Makro definieren. `PIC32_USE_ETHERNET`

Um Ethernet für die Verbindung mit der MPLAB-IDE zu verwenden

1. Klicken Sie in der MPLAB-IDE mit der rechten Maustaste auf das Projekt und wählen Sie Eigenschaften.
2. Wählen Sie im Dialogfeld Projekteigenschaften ***compiler-name*** (Globale Optionen) aus, um es zu erweitern, und wählen Sie ***compiler-name*** dann -gcc aus.
3. Wählen Sie für Optionenkategorien die Option Vorverarbeitung und Meldungen aus, und fügen Sie dann die **PIC32_USE_ETHERNET** Zeichenfolge zu Präprozessormakros hinzu.

Führen Sie das FreeRTOS-Demo-Projekt aus

1. Erstellen Sie Ihr Projekt neu.
2. Klicken Sie auf der Registerkarte Projects (Projekte) mit der rechten Maustaste auf den übergeordneten Ordner `aws_demos` und wählen Sie dann Debug (Debuggen) aus.
3. Wenn der Debugger am Haltepunkt in `main()` anhält, wählen Sie im Menü Run (Ausführen) die Funktion Fortsetzen aus.

Erstellen Sie die FreeRTOS-Demo mit CMake

Wenn Sie es vorziehen, keine IDE für die FreeRTOS-Entwicklung CMake zu verwenden, können Sie alternativ die Demo-Anwendungen oder Anwendungen, die Sie mit Code-Editoren und Debugging-Tools von Drittanbietern entwickelt haben, erstellen und ausführen.

Um die FreeRTOS-Demo zu erstellen mit CMake

1. Erstellen Sie ein Verzeichnis, das die generierten Build-Dateien enthält, wie z. *`build-directory`*
2. Verwenden Sie den folgenden Befehl, um Build-Dateien aus dem Quellcode zu generieren.

```
cmake -DVENDOR=microchip -DBOARD=curiosity_pic32mzef -DCOMPILER=xc32 -
DMCHP_HEXMATE_PATH=path/microchip/mplabx/version/mplab_platform/bin  -
DAFR_TOOLCHAIN_PATH=path/microchip/xc32/version/bin -S freertos -B build-folder
```

> **Note**
>
> Sie müssen die richtigen Pfade zu den Hexmate- und Toolchain-Binärdateien angeben, z. B. die `C:\Program Files (x86)\Microchip\MPLABX \v5.35\mplab_platform\bin` Pfade und. `C:\Program Files\Microchip \xc32\v2.40\bin`

3. Ändern Sie die Verzeichnisse in das Build-Verzeichnis (*`build-directory`*) und starten Sie es dann von diesem Verzeichnis `make` aus.

Weitere Informationen finden Sie unter Verwendung CMake mit FreeRTOS.

Um Ethernet für die Verbindung zu verwenden, müssen Sie das Präprozessor-Makro `PIC32_USE_ETHERNET` definieren.

Fehlerbehebung

Informationen zur Problembehebung finden Sie unter Fehlerbehebung – Erste Schritte.

Erste Schritte mit dem Nordic n RF5284 0-DK

> **Important**
>
> Diese Referenzintegration wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Dieses Tutorial enthält Anweisungen für die ersten Schritte mit dem Nordic n 0-DK. RF5284 Wenn Sie das Nordic n RF5284 0-DK nicht haben, besuchen Sie den AWS Partner-Gerätekatalog, um eines bei unserem Partner zu kaufen.

Bevor Sie beginnen, müssen Sie die Maßnahmen zum Einrichtung AWS IoT und Amazon Cognito für FreeRTOS Bluetooth Low Energy ausführen.

Um die FreeRTOS Bluetooth Low Energy-Demo ausführen zu können, benötigen Sie außerdem ein iOS- oder Android-Mobilgerät mit Bluetooth- und Wi-Fi-Funktionen.

> **Note**
>
> Wenn Sie ein iOS-Gerät verwenden, benötigen Sie Xcode, um die mobile Demo-Anwendung zu erstellen. Wenn Sie ein Android-Gerät verwenden, können Sie Android Studio verwenden, um die mobile Demo-Anwendung zu erstellen.

Übersicht

Dieses Tutorial enthält Anweisungen für die folgenden ersten Schritte:

1. Verbinden Ihres Boards mit einem Host-Computer.
2. Installieren von Software auf dem Host-Computer zum Entwickeln und Debuggen eingebetteter Anwendungen für Ihr Mikrocontroller-Board.
3. Cross-Compilierung einer FreeRTOS-Demo-Anwendung zu einem Binär-Image.

4. Laden des binären Anwendungs-Image auf Ihr Board und Ausführen der Anwendung.
5. Interaktion mit der Anwendung, die auf Ihrem Board über eine serielle Verbindung ausgeführt wird, zu Überwachungs- und Debuggingzwecken.

Einrichten der Nordic-Hardware

Connect Ihren Host-Computer an den USB-Anschluss mit der Bezeichnung J2 an, der sich direkt über dem Halter für die Knopfzellenbatterie auf Ihrem Nordic n RF5284 0-Board befindet.

Weitere Informationen zur Einrichtung des Nordic n RF5284 0-DK finden Sie im n RF5284 0 Development Kit-Benutzerhandbuch.

Einrichten Ihrer Entwicklungsumgebung

Herunterladen und installieren von Segger Embedded Studio

FreeRTOS unterstützt Segger Embedded Studio als Entwicklungsumgebung für das Nordic n 0-DK. RF5284

Um Ihre Umgebung einzurichten, müssen Sie Segger Embedded Studio auf Ihren Host-Computer herunterladen und dort installieren.

So laden Sie Segger Embedded Studio herunter und installieren es

1. Rufen Sie die Seite Segger Embedded Studio Downloads auf und wählen Sie die Option "Embedded Studio for ARM" für Ihr Betriebssystem aus.
2. Führen Sie das Installationsprogramm aus und befolgen Sie die Eingabeaufforderungen, um die Installation abzuschließen.

Richten Sie die FreeRTOS Bluetooth Low Energy Mobile SDK-Demoanwendung ein

Um das FreeRTOS-Demoprojekt über Bluetooth Low Energy auszuführen, müssen Sie die FreeRTOS Bluetooth Low Energy Mobile SDK-Demoanwendung auf Ihrem Mobilgerät ausführen.

So richten Sie die FreeRTOS Bluetooth Low Energy Mobile SDK Demo-Anwendung ein

1. Folgen Sie den Anweisungen im Abschnitt Mobil SDKs für FreeRTOS-Bluetooth-Geräte, um das SDK für Ihre mobile Plattform auf Ihren Host-Computer herunterzuladen und dort zu installieren.
2. Folgen Sie den Anweisungen im Abschnitt FreeRTOS Bluetooth Low Energy Mobile SDK-Demoanwendung, um die Demoanwendung für Mobilgeräte auf Ihrem Mobilgerät einzurichten.

Herstellen einer seriellen Verbindung

Segger Embedded Studio enthält einen Terminal-Emulator, den Sie verwenden können, um Protokollnachrichten über eine serielle Verbindung zu Ihrem Board zu empfangen.

So stellen Sie eine serielle Verbindung mit Segger Embedded Studio her

1. Öffnen Sie Segger Embedded Studio.
2. Wählen Sie im oberen Menü Target (Ziel), Connect J-Link (J-Link verbinden).
3. Wählen Sie im oberen Menü Tools, Terminal Emulator (Terminal-Emulator), Properties (Eigenschaften) und legen Sie die Eigenschaften gemäß den Anweisungen in Installieren eines Terminal-Emulators fest.
4. Wählen Sie im oberen Menü Tools, Terminal Emulator, Connect ***port*** (115200, N,8,1).

> **Note**
>
> Der integrierte Segger-Studio-Terminal-Emulator unterstützt keine Eingabefunktion. Verwenden Sie dazu einen Terminal-Emulator wie Pu, Tera Term oder GNU TTy Screen. Konfigurieren Sie das Terminal für eine serielle Verbindung zu Ihrem Board gemäß den Anweisungen in Installieren eines Terminal-Emulators.

Laden Sie FreeRTOS herunter und konfigurieren Sie es

Nachdem Sie Ihre Hardware und Umgebung eingerichtet haben, können Sie FreeRTOS herunterladen.

FreeRTOS herunterladen

Um FreeRTOS für das Nordic n RF5284 0-DK herunterzuladen, gehen Sie auf die GitHub FreeRTOS-Seite und klonen Sie das Repository. Anweisungen finden Sie in der Datei README.md.

> **Important**
>
> - In diesem Thema wird der Pfad zum FreeRTOS-Download-Verzeichnis als bezeichnet. *freertos*

- Leerzeichen im *freertos*-Pfad können Build-Fehler verursachen. Stellen Sie beim Klonen oder Kopieren des Repositorys sicher, dass der von Ihnen erstellte Pfad keine Leerzeichen enthält.
- Die maximale Länge eines Dateipfades bei Microsoft Windows ist 260 Zeichen. Lange FreeRTOS-Download-Verzeichnispfade können zu Build-Fehlern führen.
- Da der Quellcode symbolische Links enthalten kann, müssen Sie, wenn Sie Windows zum Extrahieren des Archivs verwenden, möglicherweise:
  - Entwicklermodus aktivieren oder
  - Verwenden Sie eine Konsole mit Administratorberechtigungen.

  Auf diese Weise kann Windows beim Extrahieren des Archivs ordnungsgemäß symbolische Links erstellen. Andernfalls werden symbolische Links als normale Dateien geschrieben, die die Pfade der symbolischen Links als Text enthalten oder leer sind. Weitere Informationen finden Sie im Blogeintrag Symlinks in Windows 10! .

  Wenn Sie Git unter Windows verwenden, müssen Sie den Entwicklermodus aktivieren oder:

  - Setzen `core.symlinks` Sie ihn mit dem folgenden Befehl auf true:

    ```
    git config --global core.symlinks true
    ```

  - Verwenden Sie immer dann eine Konsole mit Administratorrechten, wenn Sie einen Git-Befehl verwenden, der in das System schreibt (z. B.git pull,git clone, undgit submodule update --init --recursive).

Konfigurieren Ihres Projekts

Um die Demo zu aktivieren, müssen Sie Ihr Projekt so konfigurieren, dass Sie damit arbeiten können AWS IoT. Um Ihr Projekt so zu konfigurieren AWS IoT, dass es verwendet werden kann, muss Ihr Gerät als Objekt registriert AWS IoT sein. Sie sollte Ihr Gerät registriert haben, wenn Sie die Maßnahmen zum Einrichtung AWS IoT und Amazon Cognito für FreeRTOS Bluetooth Low Energy ausführen.

Um Ihren AWS IoT Endpunkt zu konfigurieren

1. Melden Sie sich an der AWS IoT -Konsole an.

2. Wählen Sie im Navigationsbereich Settings (Einstellungen).

   Ihr AWS IoT Endpunkt wird im Textfeld Gerätedatenendpunkt angezeigt. Sie sollte wie folgt aussehen: *1234567890123*-ats.iot.*us-east-1*.amazonaws.com. Notieren Sie sich diesen Endpunkt.

3. Wählen Sie im Navigationsbereich Verwalten und dann Objekte aus. Notieren Sie sich den Namen der AWS IoT Sache für Ihr Gerät.

4. Wenn Sie Ihren AWS IoT Endpunkt und den AWS IoT Namen Ihres Dings zur Hand haben, öffnen Sie es *freertos*/demos/include/aws_clientcredential.h in Ihrer IDE und geben Sie Werte für die folgenden #define Konstanten an:

   - clientcredentialMQTT_BROKER_ENDPOINT *Your AWS IoT endpoint*
   - clientcredentialIOT_THING_NAME *Your board's AWS IoT thing name*

So aktivieren Sie die Demo:

1. Überprüfen Sie, ob die Bluetooth Low Energy GATT-Demo aktiviert ist. Gehen Sie zu vendors/nordic/boards/nrf52840-dk/aws_demos/config_files/iot_ble_config.h und fügen Sie #define IOT_BLE_ADD_CUSTOM_SERVICES (1) zur Liste der definierten Anweisungen hinzu.

2. Öffnen Sie vendors/nordic/boards/nrf52840-dk/aws_demos/config_files/aws_demo_config.h und definieren Sie entweder CONFIG_OTA_MQTT_BLE_TRANSPORT_DEMO_ENABLED oder CONFIG_OTA_HTTP_BLE_TRANSPORT_DEMO_ENABLED wie in diesem Beispiel.

```
/* To run a particular demo you need to define one of these.
 * Only one demo can be configured at a time
 *
 *          CONFIG_BLE_GATT_SERVER_DEMO_ENABLED
 *          CONFIG_MQTT_BLE_TRANSPORT_DEMO_ENABLED
 *          CONFIG_SHADOW_BLE_TRANSPORT_DEMO_ENABLED
 *          CONFIG_OTA_MQTT_BLE_TRANSPORT_DEMO_ENABLED
 *          CONFIG_OTA_HTTP_BLE_TRANSPORT_DEMO_ENABLED
 *          CONFIG_POSIX_DEMO_ENABLED
 *
 *  These defines are used in iot_demo_runner.h for demo selection */

#define CONFIG_OTA_MQTT_BLE_TRANSPORT_DEMO_ENABLED
```

3. Da der Nordic Chip mit sehr wenig RAM (250 KB) ausgestattet ist, muss die BLE-Konfiguration möglicherweise geändert werden, um im Vergleich zur Größe jedes Attributs größere GATT-Tabelleneinträge zu ermöglichen. Auf diese Weise können Sie die Menge an Speicher anpassen, die die Anwendung erhält. Überschreiben Sie dazu die Definitionen der folgenden Attribute in der Datei *freertos*`/vendors/nordic/boards/nrf52840-dk/aws_demos/config_files/sdk_config.h`:

   - NRF_SDH_BLE_VS_UUID_COUNT

     Die Anzahl der UUIDs herstellerspezifischen. Erhöhen Sie diese Anzahl um 1, wenn Sie eine neue herstellerspezifische UUID hinzufügen.

   - NRF_SDH_BLE_GATTS_ATTR_TAB_SIZE

     Attributtabellengröße in Byte. Die Größe muss ein Vielfaches von 4 sein. Dieser Wert gibt die festgelegte Speichermenge an, die für die Attributtabelle reserviert ist (einschließlich der charakteristischen Größe). Dieser Wert ist also von Projekt zu Projekt unterschiedlich. Wenn Sie die Größe der Attributtabelle überschreiten, wird ein NRF_ERROR_NO_MEM-Fehler angezeigt. Wenn Sie NRF_SDH_BLE_GATTS_ATTR_TAB_SIZE ändern, müssen Sie normalerweise auch die RAM-Einstellungen neu konfigurieren.

   (Bei Tests ist der Speicherort der Datei *freertos*`/vendors/nordic/boards/nrf52840-dk/aws_tests/config_files/sdk_config.h`.)

Erstellen und starten Sie das FreeRTOS-Demoprojekt

Nachdem Sie FreeRTOS heruntergeladen und Ihr Demo-Projekt konfiguriert haben, können Sie das Demo-Projekt auf Ihrem Board erstellen und ausführen.

> **Important**
>
> Wenn dies das erste Mal ist, dass Sie die Demo auf diesem Board ausführen, müssen Sie einen Bootloader auf das Board flashen, bevor die Demo laufen kann.
> Um den Bootloader zu erstellen und zu flashen, führen Sie die folgenden Schritte aus, aber anstatt die Projektdatei `projects/nordic/nrf52840-dk/ses/aws_demos/aws_demos.emProject` zu verwenden, verwenden Sie `projects/nordic/nrf52840-dk/ses/aws_demos/bootloader/bootloader.emProject`.

Um die FreeRTOS Bluetooth Low Energy-Demo von Segger Embedded Studio zu erstellen und auszuführen

1. Öffnen Sie Segger Embedded Studio. Wählen Sie im oberen Menü File, wählen Sie Open Solution und navigieren Sie dann zu der Projektdatei `projects/nordic/nrf52840-dk/ses/aws_demos/aws_demos.emProject`.
2. Wenn Sie den Segger Embedded Studio Terminal-Emulator verwenden, wählen Sie Tools aus dem oberen Menü und wählen Sie dann Terminal Emulator (Terminal-Emulator), Terminal Emulator (Terminal-Emulator), um Informationen zu Ihrer seriellen Verbindung anzuzeigen.

   Wenn Sie ein anderes Terminal-Tool verwenden, können Sie dieses Tool auf die Ausgabe Ihrer seriellen Verbindung überwachen.
3. Klicken Sie mit der rechten Maustaste auf das Demoprojekt `aws_demos` im Project Explorer (Projekt-Explorer) und wählen Sie dann Build (Erstellen) aus.

   > **Note**
   >
   > Wenn Sie Segger Embedded Studio erstmals verwenden, wird Ihnen möglicherweise die Warnmeldung "Keine Lizenz für kommerzielle Nutzung" angezeigt. Segger Embedded Studio kann für Geräte von Nordic Semiconductor kostenlos verwendet werden. Fordern Sie eine kostenlose Lizenz an, wählen Sie dann während der Installation die Option „Kostenlose Lizenz aktivieren“ und folgen Sie den Anweisungen.
4. Wählen Sie Debug (Debuggen) und anschließend Go (Los) aus.

   Nach dem Start der Demo wartet sie auf die Kopplung mit einem mobilen Gerät über Bluetooth Low Energy.
5. Folgen Sie den Anweisungen für die Demo-Anwendung MQTT over Bluetooth Low Energy, um die Demo mit der FreeRTOS Bluetooth Low Energy Mobile SDK-Demoanwendung als mobilem MQTT-Proxy abzuschließen.

Fehlerbehebung

Allgemeine Informationen zur Problembehandlung bei Getting Started with FreeRTOS finden Sie unter. Fehlerbehebung – Erste Schritte

Erste Schritte mit dem Nuvoton 487 NuMaker-IoT-M

> **Important**
>
> Diese Referenzintegration wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Dieses Tutorial enthält Anweisungen für die ersten Schritte mit dem Nuvoton 487-Entwicklungsboard. NuMaker-IoT-M Der Mikrocontroller der Serie umfasst integrierte RJ45 Ethernet- und Wi-Fi-Module. Wenn Sie den Nuvoton NuMaker-IoT-M 487 nicht haben, besuchen Sie den Gerätekatalog für AWS Partner, um eines von unserem Partner zu kaufen.

Bevor Sie beginnen, müssen Sie Ihre FreeRTOS-Software konfigurieren AWS IoT , um Ihr Entwicklungsboard mit der AWS Cloud zu verbinden. Detaillierte Anweisungen finden Sie unter Erste Schritte. In diesem Tutorial wird der Pfad zum FreeRTOS-Download-Verzeichnis als bezeichnet. *`freertos`*

-Übersicht

In diesem Tutorial führen Sie die folgenden Schritte aus:

1. Installieren von Software auf dem Host-Computer zum Entwickeln und Debuggen eingebetteter Anwendungen für Ihre Mikrocontroller-Platine.
2. Eine FreeRTOS-Demo-Anwendung zu einem Binär-Image querkompilieren.
3. Laden des binären Anwendungs-Image auf Ihre Platine und Ausführen der Anwendung.

Einrichten Ihrer Entwicklungsumgebung

Die Keil MDK Nuvoton Edition wurde für die Entwicklung und das Debugging von Anwendungen für Nuvoton M487-Platinen entwickelt. Die Keil MDK v5 Essential-, Plus- oder Pro-Version ist ebenfalls für die Nuvoton M487 (Cortex-M4 Core) MCU geeignet. Sie können die Keil MDK Nuvoton Edition mit einem Preisnachlass für die Nuvoton Cortex-M4-Serie herunterladen. MCUs Das Keil MDK wird nur unter Windows unterstützt.

Um das NuMaker-IoT-M Entwicklungstool für den 487 zu installieren

1. Laden Sie die Keil MDK Nuvoton Edition von der Keil MDK-Website herunter.
2. Installieren Sie das Keil MDK unter Verwendung Ihrer Lizenz auf Ihrem Hostcomputer. Das Keil MDK enthält die Keil µVision-IDE, eine C/C++-Kompilierungs-Toolchain und den µVision-Debugger.

   Wenn während der Installation Probleme auftreten, wenden Sie sich an Nuvoton, um Unterstützung zu erhalten.
3. Installieren Sie den NU-Link_Keil_Driver_v3.06.7215R (oder die neueste Version), der sich auf der Nuvoton Development Tool-Seite befindet.

Erstellen Sie das FreeRTOS-Demoprojekt und führen Sie es aus

Um das FreeRTOS-Demo-Projekt zu erstellen

1. Öffnen Sie die Keil µVision-IDE.
2. Wählen Sie im Menü File (Datei) die Option Open (Öffnen) aus. Stellen Sie sicher, dass im Dialogfeld Open file (Datei öffnen) die Dateitypauswahl auf Project Files (Projektdateien) festgelegt ist.
3. Wählen Sie entweder das zu erstellende Wi-Fi- oder Ethernet-Demo-Projekt aus.

   - Um das WLAN-Demoprojekt zu öffnen, wählen Sie das Zielprojekt „`aws_demos.uvproj`“ im Verzeichnis „*`freertos`*`\projects\nuvoton\numaker_iot_m487_wifi\uvision\aws_demos`“ aus.
   - Um das Ethernet-Demo-Projekt zu öffnen, wählen Sie das Zielprojekt „`aws_demos_eth.uvproj`“ im Verzeichnis „*`freertos`*`\projects\nuvoton\numaker_iot_m487_wifi\uvision\aws_demos_eth`“ aus.
4. Um sicherzustellen, dass die Einstellungen zum Flashen der Platine korrekt sind, klicken Sie mit der rechten Maustaste auf das `aws_demo`-Projekt in der IDE und wählen Sie dann Options (Optionen). (Weitere Details finden Sie unter Fehlerbehebung.)
5. Stellen Sie sicher, dass auf der Registerkarte Utilities (Dienstprogramme) die Option Use Target Driver for Flash Programming (Zieltreiber für Flash-Programmierung verwenden) ausgewählt ist, und dass Nuvoton Nu-Link Debugger als Zieltreiber festgelegt ist.
6. Wählen Sie auf der Registerkarte Debug (Debuggen) neben Nuvoton Nu-Link Debugger die Option Settings (Einstellungen).

7. Stellen Sie sicher, dass der Chip Type (Chiptyp) auf M480 festgelegt ist.
8. Wählen Sie im Navigationsbereich Project (Projekt) der Keil µVision IDE das Projekt `aws_demos` aus. Wählen Sie im Menü Project (Projekt) die Option Build Target (Ziel erstellen).

Sie können den MQTT-Client in der AWS IoT Konsole verwenden, um die Nachrichten zu überwachen, die Ihr Gerät an die Cloud sendet. AWS

Um das MQTT-Thema mit dem MQTT-Client zu abonnieren AWS IoT

1. Melden Sie sich bei der AWS IoT -Konsole an.
2. Wählen Sie im Navigationsbereich Test und dann MQTT-Testclient aus, um den MQTT-Client zu öffnen.
3. Geben Sie im Feld Subscription topic (Abonnementthema) die Option ***your-thing-name*/example/topic** ein und wählen Sie dann Subscribe to topic (Thema abonnieren).

Um das FreeRTOS-Demo-Projekt auszuführen

1. Connect Sie Ihr Numaker-IoT-M 487-Board mit Ihrem Host-Computer (Computer).
2. Erstellen Sie das Projekt neu.
3. Wählen Sie in der Keil µVision-IDE im Menü Flash die Option Download (Herunterladen).
4. Wählen Sie im Menü Debug (Debuggen) die Option Start/Stop Debug Session (Debug-Sitzung starten/stoppen).
5. Wenn der Debugger am Haltepunkt in `main()` stoppt, öffnen Sie das Menü Run (Ausführen) und wählen Sie dann Run (F5) (Ausführen (F5)).

   Sie sollten die von Ihrem Gerät gesendeten MQTT-Nachrichten im MQTT-Client in der Konsole sehen. AWS IoT

Verwendung CMake mit FreeRTOS

Sie können es auch verwenden CMake , um die FreeRTOS-Demoanwendungen oder Anwendungen, die Sie mit Code-Editoren und Debugging-Tools von Drittanbietern entwickelt haben, zu erstellen und auszuführen.

Stellen Sie sicher, dass Sie das Build-System installiert haben. CMake Befolgen Sie die Anweisungen unter Verwendung CMake mit FreeRTOS und führen Sie anschließend die Schritte in diesem Thema aus.

> **Note**
>
> Stellen Sie sicher, dass sich der Pfad zum Speicherort des Compilers (Keil) in Ihrer Path-Systemvariablen befindet, z. B. `C:\Keil_v5\ARM\ARMCC\bin`.

Sie können auch den MQTT-Client in der AWS IoT Konsole verwenden, um die Nachrichten zu überwachen, die Ihr Gerät an die AWS Cloud sendet.

Um das MQTT-Thema mit dem MQTT-Client zu abonnieren AWS IoT

1. Melden Sie sich bei der AWS IoT -Konsole an.
2. Wählen Sie im Navigationsbereich Test und dann MQTT-Testclient aus, um den MQTT-Client zu öffnen.
3. Geben Sie im Feld Subscription topic (Abonnementthema) die Option ***your-thing-name*/example/topic** ein und wählen Sie dann Subscribe to topic (Thema abonnieren).

So generieren Sie Build-Dateien aus Quelldateien und führen das Demoprojekt aus

1. Öffnen Sie auf Ihrem Host-Computer die Eingabeaufforderung und navigieren Sie zu dem *freertos* Ordner.
2. Erstellen Sie einen Ordner, der die generierten Build-Dateien enthält. Wir werden diesen Ordner als den bezeichnen*BUILD_FOLDER*.
3. Generieren Sie die Build-Dateien für das Wi-Fi- oder Ethernet-Demo.

   - Für WLAN:

     Navigieren Sie zu dem Verzeichnis, das die Quelldateien für das FreeRTOS-Demoprojekt enthält. Generieren Sie dann die Build-Dateien, indem Sie den folgenden Befehl ausführen.

     ```
     cmake -DVENDOR=nuvoton -DBOARD=numaker_iot_m487_wifi -DCOMPILER=arm-keil -S . -B BUILD_FOLDER -G Ninja
     ```

   - Für Ethernet:

Navigieren Sie zu dem Verzeichnis, das die Quelldateien für das FreeRTOS-Demoprojekt enthält. Generieren Sie dann die Build-Dateien, indem Sie den folgenden Befehl ausführen.

```
cmake -DVENDOR=nuvoton -DBOARD=numaker_iot_m487_wifi -DCOMPILER=arm-keil -
DAFR_ENABLE_ETH=1 -S . -B BUILD_FOLDER -G Ninja
```

4. Generieren Sie die Binärdatei, die auf den M487 geflasht werden soll, indem Sie den folgenden Befehl ausführen.

   ```
   cmake --build BUILD_FOLDER
   ```

   An diesem Punkt sollte sich die Binärdatei `aws_demos.bin` im Ordner *BUILD_FOLDER*`/vendors/Nuvoton/boards/numaker_iot_m487_wifi` befinden.

5. Um die Platine für den Flash-Modus zu konfigurieren, stellen Sie sicher, dass der MSG-Schalter (Nr. 4 von ISW1 on ICE) eingeschaltet ist. Wenn Sie die Platine anschließen, wird ein Fenster (und Laufwerk) zugewiesen. (Siehe Fehlerbehebung).

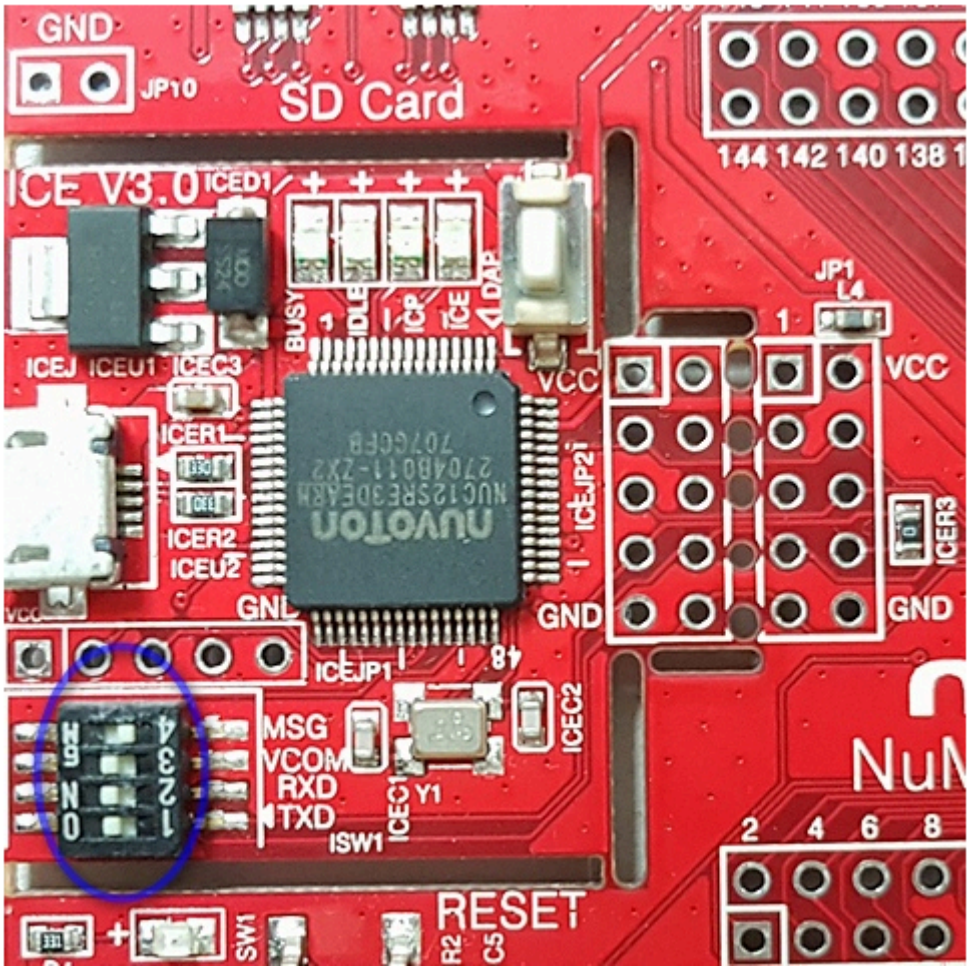
6. Öffnen Sie einen Terminal-Emulator, um die Nachrichten über UART anzuzeigen. Folgen Sie den Anweisungen unter Installieren eines Terminal-Emulators.

7. Führen Sie das Demoprojekt aus, indem Sie die generierte Binärdatei auf das Gerät kopieren.

   Wenn Sie das MQTT-Thema mit dem MQTT-Client abonniert haben, sollten Sie die AWS IoT von Ihrem Gerät gesendeten MQTT-Nachrichten in der Konsole sehen. AWS IoT

Fehlerbehebung

- Wenn Ihr Windows das Gerät nicht erkenntVCOM, installieren Sie den NuMaker Windows-Treiber für die serielle Schnittstelle über den Link Nu-Link USB Driver v1.6.
- Wenn Sie Ihr Gerät über Nu-Link mit dem Keil MDK (IDE) verbinden, stellen Sie sicher, dass der MSG-Schalter (Nr. 4 von on ICE) ausgeschaltet ist, wie in der ISW1 Abbildung gezeigt.



Wenn beim Einrichten Ihrer Entwicklungsumgebung oder beim Herstellen einer Verbindung mit Ihrer Platine Probleme auftreten, wenden Sie sich an Nuvoton.

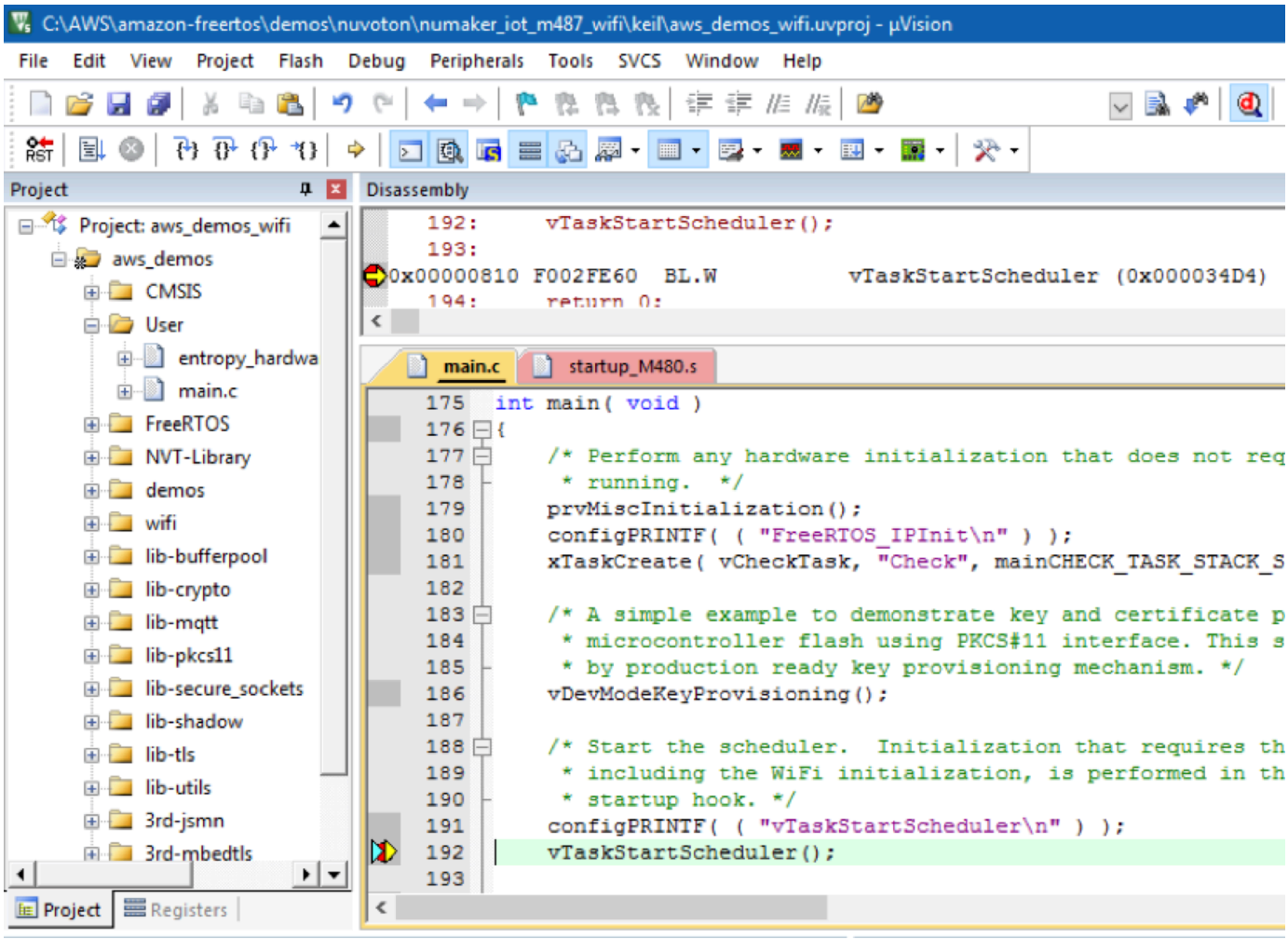
Debuggen von FreeRTOS-Projekten in Keil µVision

So starten Sie eine Debug-Sitzung in Keil µVision

1. Öffnen Sie Keil µVision.
2. Folgen Sie den Schritten, um das FreeRTOS-Demo-Projekt einzubauen. Erstellen Sie das FreeRTOS-Demoprojekt und führen Sie es aus
3. Wählen Sie im Menü Debug (Debuggen) die Option Start/Stop Debug Session (Debug-Sitzung starten/stoppen).

   Das Fenster Call Stack+ Locals (Aufruf-Stack und Lokale) wird angezeigt, wenn Sie eine Debug-Sitzung starten. µVision flasht die Demo auf die Platine, führt die Demo aus und stoppt am Anfang der `main()`-Funktion.

4. Legen Sie Haltepunkte im Quellcode Ihres Projekts fest und führen Sie dann den Code aus. Das Projekt sollte wie folgt aussehen:



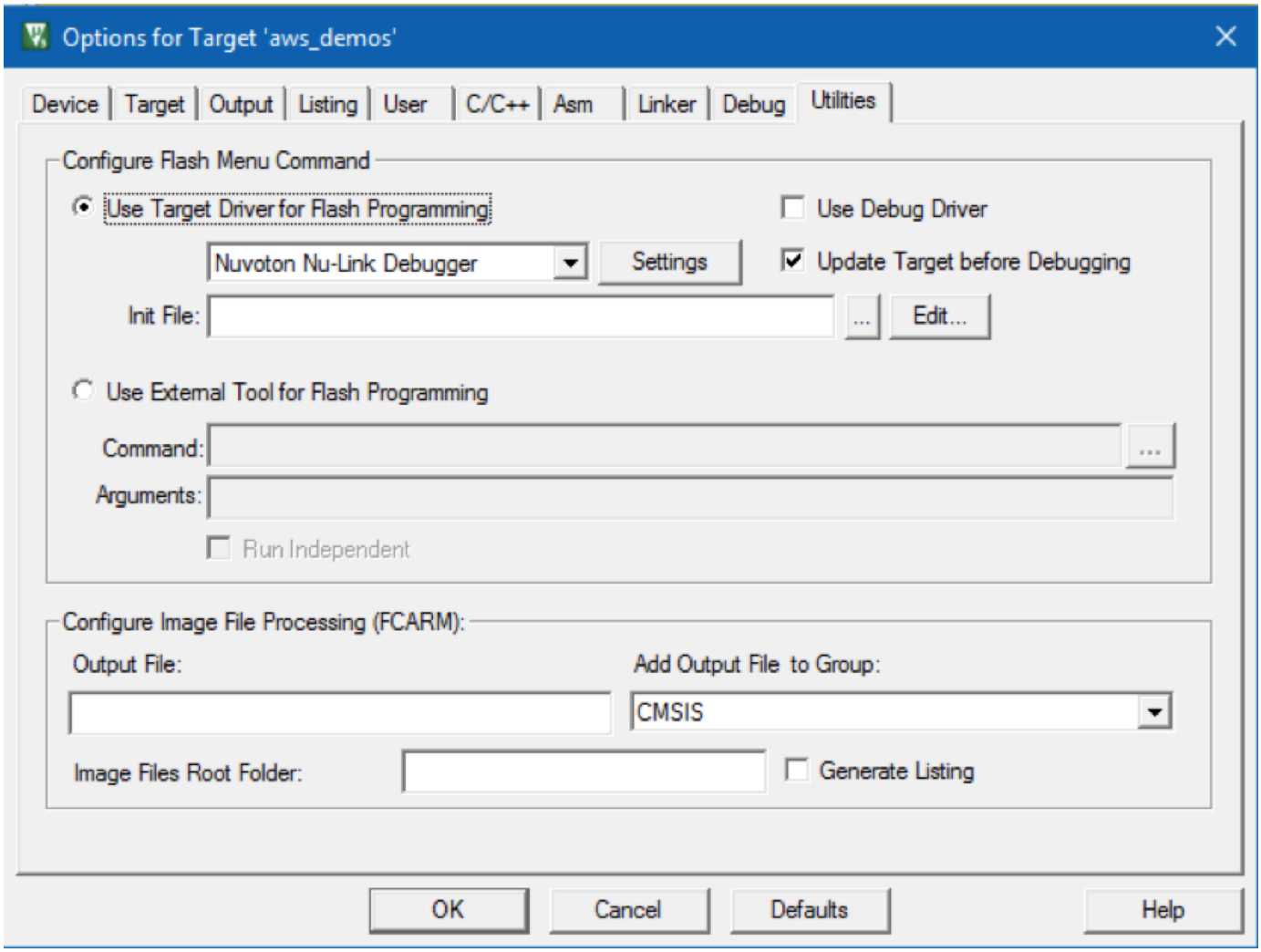
Fehlerbehebung für Debug-Einstellungen von µVision

Wenn beim Debuggen einer Anwendung Probleme auftreten, überprüfen Sie, ob Ihre Debug-Einstellungen in Keil µVision korrekt sind.

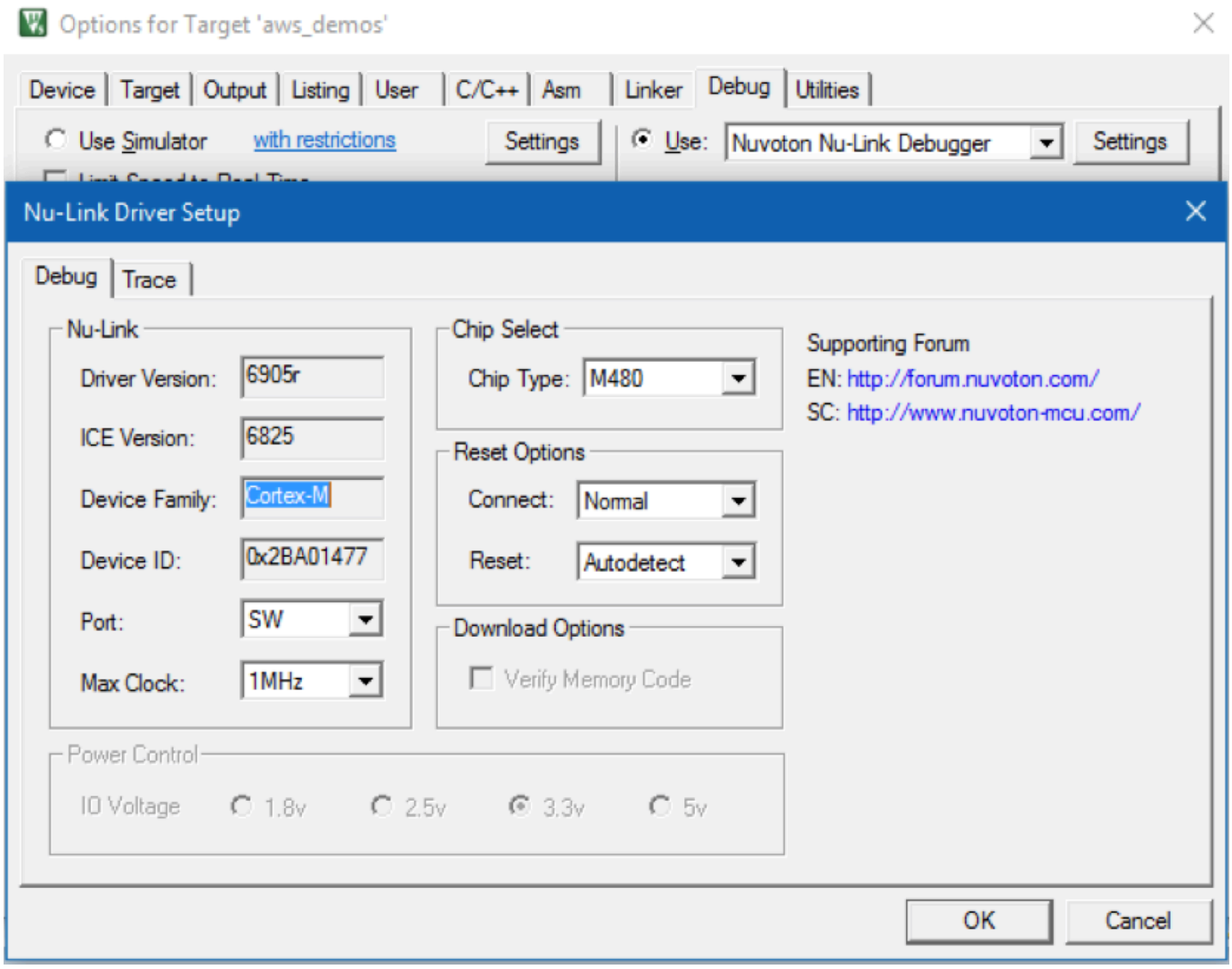
So überprüfen Sie, ob die Debug-Einstellungen von µVision korrekt sind

1. Öffnen Sie Keil µVision.
2. Klicken Sie mit der rechten Maustaste auf das `aws_demo`-Projekt in der IDE und wählen Sie dann Options (Optionen)aus.

3. Stellen Sie sicher, dass auf der Registerkarte Utilities (Dienstprogramme) die Option Use Target Driver for Flash Programming (Zieltreiber für Flash-Programmierung verwenden) ausgewählt ist, und dass Nuvoton Nu-Link Debugger als Zieltreiber festgelegt ist.



4. Wählen Sie auf der Registerkarte Debug (Debuggen) neben Nuvoton Nu-Link Debugger die Option Settings (Einstellungen).



5. Stellen Sie sicher, dass der Chip Type (Chiptyp) auf M480 festgelegt ist.

Erste Schritte mit dem NXP LPC54 018 IoT-Modul

> **Important**
>
> Diese Referenzintegration wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Dieses Tutorial enthält Anweisungen für die ersten Schritte mit dem NXP LPC54 018 IoT-Modul. Wenn Sie kein NXP LPC54 018 IoT-Modul haben, besuchen Sie den AWS Partnergerätekatalog, um eines von unserem Partner zu erwerben. Verwenden Sie ein USB-Kabel, um Ihr NXP LPC54 018 IoT-Modul mit Ihrem Computer zu verbinden.

Bevor Sie beginnen, müssen Sie FreeRTOS konfigurieren AWS IoT und herunterladen, um Ihr Gerät mit der AWS Cloud zu verbinden. Detaillierte Anweisungen finden Sie unter Erste Schritte. In diesem Tutorial wird der Pfad zum FreeRTOS-Download-Verzeichnis als bezeichnet. *freertos*

-Übersicht

Dieses Tutorial enthält Anweisungen für die folgenden ersten Schritte:

1. Verbinden Ihres Boards mit einem Host-Computer.
2. Installieren von Software auf dem Host-Computer zum Entwickeln und Debuggen eingebetteter Anwendungen für Ihr Mikrocontroller-Board.
3. Cross-Compilierung einer FreeRTOS-Demo-Anwendung zu einem Binär-Image.
4. Laden des binären Anwendungs-Image auf Ihr Board und Ausführen der Anwendung.

Einrichten der NXP-Hardware

Um den NXP 018 einzurichten LPC54

- Connect Sie Ihren Computer mit dem USB-Anschluss des NXP LPC54 018.

So richten Sie den JTAG-Debugger ein

Sie benötigen einen JTAG-Debugger, um Ihren Code, der auf dem NXP 018-Board läuft, zu starten und zu debuggen. LPC54 FreeRTOS wurde mit einem OM4 0006 IoT-Modul getestet. Weitere Informationen zu unterstützten Debuggern finden Sie im Benutzerhandbuch für das NXP LPC54 018 IoT-Modul, das auf der Produktseite OM40007 LPC54 018 IoT Module verfügbar ist.

1. Wenn Sie einen OM4 0006 IoT-Modul-Debugger verwenden, verwenden Sie ein Konverterkabel, um den 20-poligen Stecker vom Debugger mit dem 10-poligen Anschluss am NXP IoT-Modul zu verbinden.
2. Connect den NXP LPC54 018 und den OM4 0006 IoT Module Debugger mithilfe von Mini-USB-auf-USB-Kabeln mit den USB-Anschlüssen Ihres Computers.

Einrichten Ihrer Entwicklungsumgebung

FreeRTOS unterstützt zwei IDEs für das NXP LPC54 018 IoT-Modul: IAR Embedded Workbench und. MCUXpresso

Bevor Sie beginnen, installieren Sie eines davon. IDEs

So installieren Sie IAR Embedded Workbench für ARM:

1. Navigieren Sie zu IAR Embedded Workbench for ARM und laden Sie die Software herunter.

   > **Note**
   >
   > IAR Embedded Workbench für ARM erfordert Microsoft Windows.

2. Führen Sie das Installationsprogramm aus und folgen Sie den Anweisungen.
3. Klicken Sie im License Wizard (Lizenz-Assistent) auf Register with IAR Systems to get an evaluation license (Mit IAR Systems registrieren, um eine Evaluierungslizenz zu erhalten).
4. Legen Sie den Bootloader auf dem Gerät ab, bevor Sie versuchen, Demos auszuführen.

Um von NXP MCUXpresso aus zu installieren

1. Laden Sie das MCUXpresso Installationsprogramm von NXP herunter und führen Sie es aus.

   > **Note**
   >
   > Unterstützt werden die Versionen 10.3.x und höher.

2. Navigieren Sie zu MCUXpresso SDK und wählen Sie Build your SDK aus.

   > **Note**
   >
   > Versionen ab Version 2.5 werden unterstützt.

3. Wählen Sie Select Development Board (Entwicklungsplatine auswählen) aus.
4. Unter Select Development Board (Entwicklungsplatine auswählen) geben Sie im Abschnitt Nach Namen durchsuchen die Option **`LPC54018-IoT-Module`** ein.
5. Wählen Sie unter Boards die Option LPC54018-IoT-Module aus.

6. Überprüfen Sie die Hardwaredetails und wählen Sie dann Build SDK aus. MCUXepresso
7. Das SDK für Windows, das die MCUXpresso IDE verwendet, wurde bereits erstellt. Wählen Sie SDK herunterladen aus. Wenn Sie ein anderes Betriebssystem verwenden, wählen Sie unter Host OS (Host-Betriebssystem) Ihr Betriebssystem aus und klicken Sie auf Download SDK (SDK herunterladen).
8. Starten Sie die MCUXpresso IDE und wählen Sie die SDKs Registerkarte Installiert.
9. Ziehen Sie die heruntergeladene SDK-Archivdatei per Drag & Drop in das SDKs Fenster Installiert.

Falls während der Installation Probleme auftreten, finden Sie Informationen in den Bereichen zum NXP-Support oder zu den NXP-Entwicklungsressourcen.

Überwachung von MQTT-Nachrichten in der Cloud

Bevor Sie das FreeRTOS-Demoprojekt ausführen, können Sie den MQTT-Client in der AWS IoT Konsole so einrichten, dass er die Nachrichten überwacht, die Ihr Gerät an die Cloud sendet. AWS

Um das MQTT-Thema mit dem MQTT-Client zu abonnieren AWS IoT

1. Melden Sie sich bei der AWS IoT-Konsole an.
2. Wählen Sie im Navigationsbereich Test und dann MQTT-Testclient aus, um den MQTT-Client zu öffnen.
3. Geben Sie im Feld Subscription topic (Abonnementthema) die Option ***your-thing-name*/example/topic** ein und wählen Sie dann Subscribe to topic (Thema abonnieren).

Wenn das Demo-Projekt erfolgreich auf Ihrem Gerät ausgeführt wird, sehen Sie „Hello World!“ mehrfach zu dem Thema gesendet, das Sie abonniert haben.

Erstellen und starten Sie das FreeRTOS-Demo-Projekt

Importiere die FreeRTOS-Demo in deine IDE

Um den FreeRTOS-Beispielcode in die IAR Embedded Workbench IDE zu importieren

1. Öffnen Sie IAR Embedded Workbench und wählen Sie aus dem Menü Datei die Option Open Workspace (WorkSpace öffnen) aus.
2. Geben Sie in das Textfeld search-directory (Suchverzeichnis) die Option `projects/nxp/lpc54018iotmodule/iar/aws_demos` ein und wählen Sie aws_demos.eww aus.

3. Wählen Sie im Menü Projekt die Option Rebuild All (Alle neu erstellen) aus.

Um den FreeRTOS-Beispielcode in die IDE zu importieren MCUXpresso

1. Öffnen Sie MCUXpresso und wählen Sie im Menü Datei die Option Projekte aus Dateisystem öffnen.
2. Geben Sie in das Textfeld Verzeichnis den Befehl `projects/nxp/lpc54018iotmodule/mcuxpresso/aws_demos` ein und klicken Sie auf Beenden.
3. Wählen Sie im Menü Projekt die Option Build All (Alle erstellen) aus.

Führen Sie das FreeRTOS-Demo-Projekt aus

Um das FreeRTOS-Demoprojekt mit der IAR Embedded Workbench IDE auszuführen

1. Wählen Sie im Menü Project (Projekt) in Ihrer IDE die Option Build (Erstellen) aus.
2. Wählen Sie im Menü Projekt die Option Herunterladen und Debuggen aus.
3. Wählen Sie aus dem Menü Debug (Debuggen) die Option Start Debugging (Debuggen starten) aus.
4. Wenn der Debugger am Haltepunkt in `main` anhält, wählen Sie im Menü Debug (Debuggen) die Option Fortfahren aus.

> **Note**
>
> Wenn sich ein Dialogfeld J-Link Device Selection (J-Link-Geräteauswahl) öffnet, klicken Sie auf OK, um fortzufahren. Wählen Sie im Dialogfeld Target Device Settings (Zielgeräteinstellungen) die Optionen Unspecified (Nicht angegeben) und Cortex-M4 aus und klicken Sie dann auf OK. Sie müssen dies nur einmal tun.

Um das FreeRTOS-Demo-Projekt mit der IDE auszuführen MCUxpresso

1. Wählen Sie im Menü Projekt in Ihrer IDE die Option Build (Erstellen) aus.
2. Wenn Sie zum ersten Mal debuggen, wählen Sie das `aws_demos`-Projekt und aus der Symbolleiste Debug (Debuggen) aus. Kllicken Sie dann auf die blaue Debug-Schaltfläche.
3. Alle erkannten Debug-Proben werden angezeigt. Wählen Sie die Probe aus, die Sie verwenden möchten, und klicken Sie dann auf OK, um das Debugging zu starten.

> **Note**
>
> Wenn der Debugger am Haltepunkt in `main()` anhält,
> klicken Sie einmal auf die Schaltfläche zum Debug-Neustart
>
>
>
> ,
> um die Debugging-Sitzung zurückzusetzen. (Dies ist aufgrund eines Fehlers mit dem MCUXpresso Debugger für das NXP54 018-IoT-Modul erforderlich).

4. Wenn der Debugger am Haltepunkt in `main()` anhält, wählen Sie im Menü Debug (Debuggen) die Option Fortfahren aus.

Fehlerbehebung

Allgemeine Informationen zur Problembehandlung bei Getting Started with FreeRTOS finden Sie unter. Fehlerbehebung – Erste Schritte

Erste Schritte mit dem Renesas Starter Kit+ für N-2MB RX65

> **Important**
>
> Diese Referenzintegration wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Dieses Tutorial enthält Anweisungen für die ersten Schritte mit dem Renesas Starter Kit+ für N-2MB. RX65 Wenn Sie das Renesas RSK+ für RX65 N-2MB nicht haben, besuchen Sie den Partner-Gerätekatalog und kaufen Sie eines bei unseren Partnern. AWS

Bevor Sie beginnen, müssen Sie FreeRTOS konfigurieren AWS IoT und herunterladen, um Ihr Gerät mit der AWS Cloud zu verbinden. Detaillierte Anweisungen finden Sie unter Erste Schritte. In diesem Tutorial wird der Pfad zum FreeRTOS-Download-Verzeichnis als bezeichnet. *`freertos`*

-Übersicht

Dieses Tutorial enthält Anweisungen für die folgenden ersten Schritte:

1. Verbinden Ihres Boards mit einem Host-Computer.
2. Installieren von Software auf dem Host-Computer zum Entwickeln und Debuggen eingebetteter Anwendungen für Ihr Mikrocontroller-Board.
3. Cross-Compilierung einer FreeRTOS-Demo-Anwendung zu einem Binär-Image.
4. Laden des binären Anwendungs-Image auf Ihr Board und Ausführen der Anwendung.

Einrichten der Renesas-Hardware

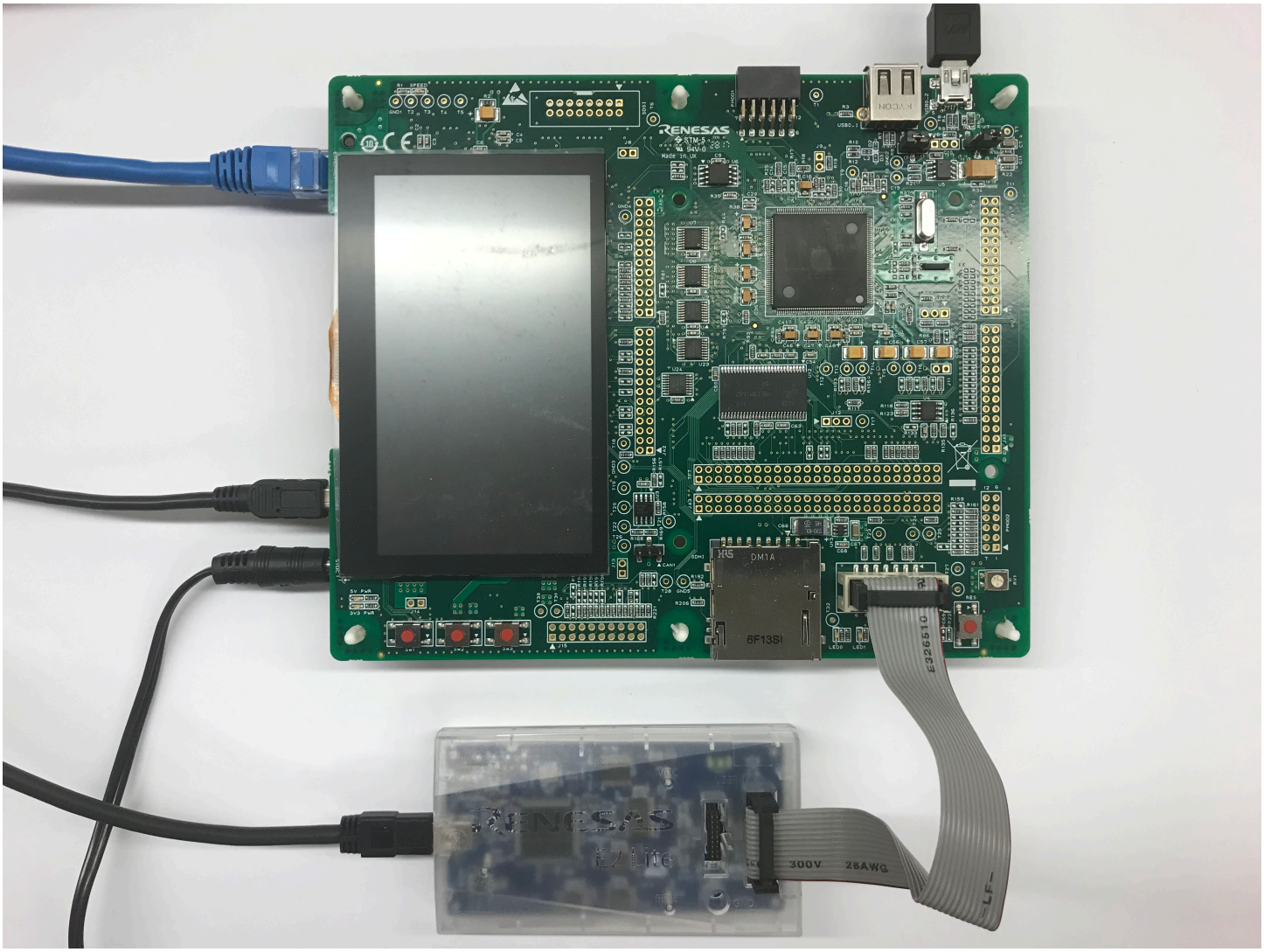
Um das RSK+ für N-2MB einzurichten RX65

1. Connect den positiven +5-V-Netzadapter mit dem PWR-Anschluss am RSK+ für N-2MB. RX65
2. Connect Sie Ihren Computer mit dem USB2 2.0 FS-Port auf dem RSK+ für N-2MB. RX65
3. Connect Sie Ihren Computer mit dem USB-to-serial Anschluss am RSK+ für RX65 N-2MB.
4. Connect einen Router oder einen mit dem Internet verbundenen Ethernet-Port mit dem Ethernet-Port des RSK+ für N-2 MB. RX65

So richten Sie das E2 Lite Debugger-Modul ein

1. Verwenden Sie das 14-polige Flachbandkabel, um das E2 Lite Debugger-Modul mit dem 'E1/E2 Lite'-Anschluss am RSK+ für N-2MB zu verbinden. RX65
2. Verwenden Sie ein USB-Kabel für die Verbindung des E2 Lite Debugger-Moduls mit Ihrem Hostcomputer. Wenn das E2 Lite Debugger-Modul mit dem Board und Ihrem Computer verbunden ist, blinkt eine grüne "ACT"-LED-Anzeige auf dem Debugger.
3. Nachdem der Debugger an Ihren Host-Computer und RSK+ für N-2 MB angeschlossen ist, beginnen die E2 Lite-Debugger-Treiber mit der Installation. RX65

   Beachten Sie, dass zum Installieren der Treiber Administratorrechte erforderlich sind.



Einrichten Ihrer Entwicklungsumgebung

Verwenden Sie die Renesas e 2 Studio-IDE und den CC-RX-Compiler, um FreeRTOS-Konfigurationen für den RSK+ für RX65 N-2 MB einzurichten.

> **Note**
>
> Der Renesas e^{2}studio IDE und CC-RX-Compiler wird nur unter Windows 7, 8 und 10 unterstützt.

So laden Sie e^{2}studio herunter und installieren es

1. Gehen Sie zur Download-Seite für das Renesas e 2 Studio-Installationsprogramm und laden Sie das Offline-Installationsprogramm herunter.
2. Sie werden auf eine Renesas-Anmeldeseite weitergeleitet.

   Wenn Sie ein Konto bei Renesas haben, geben Sie Ihre Anmeldeinformationen ein und wählen Sie dann Anmelden.

   Wenn Sie noch kein Konto haben, klicken Sie auf Register now (Jetzt registrieren) und befolgen Sie die Schritte zur ersten Registrierung. Sie erhalten Sie eine E-Mail mit einem Link zur Aktivierung Ihres Renesas-Kontos. Folgen Sie diesem Link, um Ihre Anmeldung bei Renesas abzuschließen, und melden Sie sich anschließend bei Renesas an.
3. Nachdem Sie sich angemeldet haben, laden Sie das e^{2}studio-Installationsprogramm auf Ihren Computer herunter.
4. Öffnen Sie das Installationsprogramm und befolgen Sie die Schritte bis zum Abschluss der Installation.

Weitere Informationen finden Sie im e ^{2} Studio auf der Renesas-Website.

So laden Sie das RX Family C/C++ Compiler-Paket herunter und installieren es

1. Gehen Sie zur Download-Seite für das C/C++-Compiler-Paket der RX-Familie und laden Sie das Package V3.00.00 herunter.
2. Öffnen Sie die ausführbare Datei und installieren Sie den Compiler.

Weitere Informationen finden Sie unter C/C++ Compiler Package for RX Family auf der Renesas-Website.

> **Note**
>
> Der Compiler ist nur als Testversion kostenlos und 60 Tage gültig. Ab dem 61. Tag benötigen Sie einen Lizenzschlüssel. Weitere Informationen finden Sie unter Evaluation Software Tools.

FreeRTOS-Beispiele erstellen und ausführen

Nachdem Sie jetzt Ihr Demoprojekt konfiguriert haben, können Sie das Projekt auf Ihrem Board erstellen und ausführen.

Erstellen Sie die FreeRTOS-Demo in e 2 Studio

So importieren und erstellen Sie das Demoprojekt in e^{2}studio

1. Starten Sie e^{2}studio im Start-Menü.
2. Wählen Sie im Fenster Select a directory as a workspace (Ein Verzeichnis als WorkSpace auswählen) aus. Navigieren Sie zum Ordner, in dem Sie arbeiten möchten, und klicken Sie auf Launch (Starten).
3. Beim ersten Öffnen von e^{2}studio wird das Fenster Toolchain Registry (Toolchain-Registrierung) aufgerufen. Wählen Sie Renesas Toolchains (Renesas-Toolchains) aus und vergewissern Sie sich, dass **`CC-RX v3.00.00`** ausgewählt ist. Wählen Sie Registrieren und anschließend OK aus.
4. Wenn Sie e^{2}studio zum ersten Mal öffnen, wird das Fenster Code Generator Registration (Registrierung des Code-Generators) angezeigt. Wählen Sie OK aus.
5. Das Fenster Code Generator COM component register (Code-Generator COM-Komponente registrieren) wird angezeigt. Wählen Sie unter Bitte starten Sie e ^{2} Studio neu, um den Codegenerator zu verwenden die Option OK aus.
6. Das Fenster „E ^{2} Studio neu starten“ wird angezeigt. Wählen Sie OK aus.
7. e^{2}studio wird neu gestartet. Wählen Sie im Fenster Select a directory as a workspace (Ein Verzeichnis als WorkSpace auswählen) die Option Launch (Starten) aus.
8. Wählen Sie auf dem Begrüßungsbildschirm von e ^{2} Studio das Pfeilsymbol Gehe zur E ^{2} Studio Workbench.
9. Klicken Sie mit der rechten Maustaste auf das Fenster Project Explorer (Projekt-Explorer) und wählen Sie Import (Importieren) aus.
10. Wählen Sie im Importassistenten die Optionen General (Allgemein) und Existing Projects into Workspace (Vorhandene Projekte in WorkSpace) aus und klicken Sie dann auf Weiter.
11. Klicken Sie auf Browse (Durchsuchen), navigieren Sie zum Verzeichnis `projects/renesas/rx65n-rsk/e2studio/aws_demos` und wählen Sie dann Finish (Fertigstellen) aus.
12. Wählen Sie im Menü Projekt die Optionen Projekt und Build All (Alle erstellen) aus.

Die Build-Konsole gibt eine Warnmeldung aus, dass der License Manager nicht installiert ist. Sie können diese Meldung ignorieren, es sei denn, Sie haben einen Lizenzschlüssel für den CC-RX Compiler. Informationen zum Installieren des License Manager finden Sie auf der Download-Seite License Manager.

Überwachung von MQTT-Nachrichten in der Cloud

Bevor Sie das FreeRTOS-Demoprojekt ausführen, können Sie den MQTT-Client in der AWS IoT Konsole so einrichten, dass er die Nachrichten überwacht, die Ihr Gerät an die Cloud sendet. AWS

Um das MQTT-Thema mit dem MQTT-Client zu abonnieren AWS IoT

1. Melden Sie sich bei der AWS IoT -Konsole an.
2. Wählen Sie im Navigationsbereich Test und dann MQTT-Testclient, um den MQTT-Client zu öffnen.
3. Geben Sie im Feld Subscription topic (Abonnementthema) die Option ***your-thing-name*/example/topic** ein und wählen Sie dann Subscribe to topic (Thema abonnieren).

Wenn das Demo-Projekt erfolgreich auf Ihrem Gerät ausgeführt wird, sehen Sie „Hello World!“ mehrfach zu dem Thema gesendet, das Sie abonniert haben.

Führen Sie das FreeRTOS-Projekt aus

So führen Sie das Projekt in e^{2}studio aus

1. Vergewissern Sie sich, dass Sie das E2 Lite Debugger-Modul für N-2MB an Ihr RSK+ angeschlossen haben RX65
2. Wählen Sie im oberen Menü Ausführen und die Option Debug Configuration (Debug-Konfiguration) aus.
3. Erweitern Sie Renesas GDB Hardware Debugging und wählen Sie aws_demos. HardwareDebug
4. Wählen Sie die Registerkarte Debugger und dann die Registerkarte Connection Settings (Verbindungseinstellungen) aus. Vergewissern Sie sich, dass die Verbindungseinstellungen korrekt sind.
5. Wählen Sie Debug zum Herunterladen des Codes auf Ihr Board aus und beginnen Sie mit dem Debuggen.

Möglicherweise werden Sie von einer Firewall-Warnung zur Eingabe von `e2-server-gdb.exe` aufgefordert. Aktivieren Sie Private networks, such as my home or work network (Private Netzwerke, wie mein Heimnetzwerk oder Arbeitsplatznetzwerk) und klicken Sie dann auf Allow access (Zugriff zulassen).

6. Möglicherweise fordert e^{2}studio Sie zum Ändern der Renesas Debug Perspective (Renesas Debug-Perspektive) auf. Wählen Sie Yes (Ja).

   Die grüne "ACT"-LED-Anzeige auf dem E2 Lite-Debugger leuchtet.

7. Nachdem der Code heruntergeladen wurde, wählen Sie Fortsetzen aus, um den Code bis zur ersten Zeile der Hauptfunktion auszuführen. Wählen Sie erneut Fortsetzen aus, um den Rest des Codes auszuführen.

Die neuesten von Renesas veröffentlichten Projekte finden Sie im Fork des Repositorys unter. `renesas-rx` `amazon-freertos` GitHub

Fehlerbehebung

Allgemeine Informationen zur Problembehandlung bei Getting Started with FreeRTOS finden Sie unter. Fehlerbehebung – Erste Schritte

Erste Schritte mit dem STMicroelectronics STM32 L4 Discovery Kit IoT Node

> **Important**
>
> Diese Referenzintegration wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Dieses Tutorial enthält Anweisungen für die ersten Schritte mit dem STMicroelectronics STM32 L4 Discovery Kit IoT Node. Wenn Sie den STMicroelectronics STM32 L4 Discovery Kit IoT Node noch nicht besitzen, besuchen Sie den AWS Partner Device Catalog, um einen von unserem Partner zu erwerben.

Stellen Sie sicher, dass Sie die neueste WLAN-Firmware installiert haben. Informationen zum Herunterladen der neuesten Wi-Fi-Firmware finden Sie unter STM32L4 Discovery Kit IoT-Knoten,

Low-Power-WLAN, Bluetooth Low Energy, NFC, SubGHz, Wi-Fi. Wählen Sie unter Binary Resources (Binary-Ressourcen) die Option Inventek ISM 43362 Wi-Fi module firmware update (read the readme file for instructions) (Inventek ISM 43362 WLAN-Modul Firmware-Update (siehe Readme-Datei für Anweisungen)) aus.

Bevor Sie beginnen, müssen Sie Ihren FreeRTOS-Download und WLAN konfigurieren AWS IoT, um Ihr Gerät mit der AWS Cloud zu verbinden. Detaillierte Anweisungen finden Sie unter Erste Schritte. In diesem Tutorial wird der Pfad zum FreeRTOS-Download-Verzeichnis als bezeichnet. *`freertos`*

-Übersicht

Dieses Tutorial enthält Anweisungen für die folgenden ersten Schritte:

1. Installieren von Software auf dem Host-Computer zum Entwickeln und Debuggen eingebetteter Anwendungen für Ihr Mikrocontroller-Board.
2. Cross-Compilierung einer FreeRTOS-Demo-Anwendung zu einem Binär-Image.
3. Laden des binären Anwendungs-Image auf Ihr Board und Ausführen der Anwendung.

Einrichten Ihrer Entwicklungsumgebung

Installieren Sie System Workbench für STM32

1. Rufen Sie STM32Open.org auf.
2. Registrieren Sie sich auf der STM32 Open-Webseite. Sie müssen sich anmelden, um das Workbench-System herunterladen zu können.
3. Rufen Sie das STM32 Installationsprogramm von System Workbench auf, um System Workbench herunterzuladen und zu installieren.

Falls bei der Installation Probleme auftreten, finden Sie weitere Informationen FAQs auf der System Workbench-Website.

Erstellen Sie das FreeRTOS-Demoprojekt und führen Sie es aus

Importiere die FreeRTOS-Demo in die System Workbench STM32

1. Öffnen Sie die STM32 System Workbench und geben Sie einen Namen für einen neuen Arbeitsbereich ein.

2. Wählen Sie im Menü Datei die Option Import aus. Erweitern Sie General (Allgemein), wählen Sie Existing Projects into Workspace (Vorhandene Projekte in WorkSpace) aus und klicken Sie dann auf Weiter.
3. Unter Select search-directory (Stammverzeichnis auswählen) geben Sie `projects/st/stm32l475_discovery/ac6/aws_demos` ein.
4. Das Projekt `aws_demos` sollte standardmäßig ausgewählt werden.
5. Wählen Sie Fertig stellen, um das Projekt in STM32 System Workbench zu importieren.
6. Wählen Sie im Menü Projekt die Option Build All (Alle erstellen) aus. Vergewissern Sie sich, dass das Projekt fehlerfrei kompiliert wurde.

Überwachung von MQTT-Nachrichten in der Cloud

Bevor Sie das FreeRTOS-Demoprojekt ausführen, können Sie den MQTT-Client in der AWS IoT Konsole so einrichten, dass er die Nachrichten überwacht, die Ihr Gerät an die Cloud sendet. AWS

Um das MQTT-Thema mit dem MQTT-Client zu abonnieren AWS IoT

1. Melden Sie sich bei der AWS IoT -Konsole an.
2. Wählen Sie im Navigationsbereich Test und dann MQTT-Testclient aus, um den MQTT-Client zu öffnen.
3. Geben Sie im Feld Subscription topic (Abonnementthema) die Option ***your-thing-name*/example/topic** ein und wählen Sie dann Subscribe to topic (Thema abonnieren).

Wenn das Demo-Projekt erfolgreich auf Ihrem Gerät ausgeführt wird, sehen Sie „Hello World!“ mehrfach zu dem Thema gesendet, das Sie abonniert haben.

Führen Sie das FreeRTOS-Demo-Projekt aus

1. Verwenden Sie ein USB-Kabel, um Ihren STMicroelectronics STM32 L4 Discovery Kit IoT Node mit Ihrem Computer zu verbinden. (Informationen zum richtigen USB-Anschluss finden Sie in der Dokumentation des Herstellers, die mit Ihrem Motherboard geliefert wurde.)
2. Klicken Sie im Project Explorer mit der rechten Maustaste`aws_demos`, wählen Sie Debug As und dann Ac6 STM32 C/C++ Application.

   Wenn beim ersten Starten einer Debug-Sitzung ein Debug-Fehler auftritt, führen Sie die folgenden Schritte aus:

1. Wählen Sie in STM32 System Workbench im Menü Ausführen die Option Konfigurationen debuggen.
2. Wählen Sie aws_demos Debuggen aus. (Möglicherweise müssen Sie Ac6 STM32 Debugging erweitern.)
3. Wählen Sie die Registerkarte Debugger aus.
4. Im Configuration Script (Konfigurationsskript) wählen Sie Show Generator Options (Generator-Optionen anzeigen) aus.
5. Setzen Sie unter Mode Setup (Moduseinstellung) die Option Reset Mode (Modus zurücksetzen) auf Software System Reset (Software-System zurückzusetzen). Wählen Sie erst Apply (Anwenden) und anschließend Debug (Debuggen) aus.

3. Wenn der Debugger am Haltepunkt in `main()` anhält, wählen Sie im Menü Run (Ausführen) die Funktion Fortsetzen aus.

Verwendung CMake mit FreeRTOS

Wenn Sie es vorziehen, keine IDE für die FreeRTOS-Entwicklung CMake zu verwenden, können Sie alternativ die Demo-Anwendungen oder Anwendungen, die Sie mit Code-Editoren und Debugging-Tools von Drittanbietern entwickelt haben, erstellen und ausführen.

Erstellen Sie zunächst einen Ordner, der die generierten Build-Dateien () enthält. *`build-folder`*

Verwenden Sie den folgenden Befehl, um Build-Dateien zu erstellen:

```
cmake -DVENDOR=st -DBOARD=stm32l475_discovery -DCOMPILER=arm-gcc -S freertos -B build-folder
```

Wenn `arm-none-eabi-gcc` es sich nicht in Ihrem Shell-Pfad befindet, müssen Sie auch die `AFR_TOOLCHAIN_PATH` CMake Variable setzen. Beispiel:

```
-D AFR_TOOLCHAIN_PATH=/home/user/opt/gcc-arm-none-eabi/bin
```

Weitere Hinweise zur Verwendung CMake mit FreeRTOS finden Sie unter. Verwendung CMake mit FreeRTOS

Fehlerbehebung

Wenn Folgendes in der UART-Ausgabe der Demoanwendung angezeigt wird, müssen Sie die WLAN-Firmware des Moduls aktualisieren:

```
[Tmr Svc] WiFi firmware version is: xxxxxxxxxxxx
[Tmr Svc] [WARN] WiFi firmware needs to be updated.
```

Informationen zum Herunterladen der neuesten Wi-Fi-Firmware finden Sie unter STM32L4 Discovery Kit IoT-Knoten, Low-Power-WLAN, Bluetooth Low Energy, NFC, SubGHz, Wi-Fi. Wählen Sie unter Binary Resources (Binary-Ressourcen) den Download-Link für Inventek ISM 43362 WLAN-Modul-Firmware-Update aus.

Allgemeine Informationen zur Problembehandlung bei Getting Started with FreeRTOS finden Sie unter. Fehlerbehebung – Erste Schritte

Erste Schritte mit dem Texas Instruments CC322 0SF-LAUNCHXL

> **Important**
>
> Diese Referenzintegration wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Dieses Tutorial enthält Anweisungen für die ersten Schritte mit dem 0SF-LAUNCHXL von Texas Instruments. CC322 Wenn Sie nicht über das CC322 0SF-LAUNCHXL Development Kit von Texas Instruments (TI) verfügen, besuchen Sie den Gerätekatalog für Partner, um eines von unserem Partner zu erwerben. AWS

Bevor Sie beginnen, müssen Sie FreeRTOS konfigurieren AWS IoT und herunterladen, um Ihr Gerät mit der AWS Cloud zu verbinden. Detaillierte Anweisungen finden Sie unter Erste Schritte. In diesem Tutorial wird der Pfad zum FreeRTOS-Download-Verzeichnis als bezeichnet. *`freertos`*

-Übersicht

Dieses Tutorial enthält Anweisungen für die folgenden ersten Schritte:

1. Installieren von Software auf dem Host-Computer zum Entwickeln und Debuggen eingebetteter Anwendungen für Ihr Mikrocontroller-Board.
2. Cross-Compilierung einer FreeRTOS-Demo-Anwendung zu einem Binär-Image.
3. Laden des binären Anwendungs-Image auf Ihr Board und Ausführen der Anwendung.

Einrichten Ihrer Entwicklungsumgebung

Gehen Sie wie folgt vor, um Ihre Entwicklungsumgebung für den Einstieg in FreeRTOS einzurichten.

Beachten Sie, dass FreeRTOS zwei IDEs für das TI CC322 0SF-LAUNCHXL Development Kit unterstützt: Code Composer Studio und IAR Embedded Workbench Version 8.32. Sie können beide IDEs für den Einstieg verwenden.

Installieren von Code Composer Studio

1. Navigieren Sie zu TI Code Composer Studio.
2. Wenn Sie den TI ARM-Compiler mit Ihrem TI-Board verwenden, verwenden Sie den folgenden Befehl, um Build-Dateien aus dem Quellcode zu erzeugen:
3. Entpacken Sie das Offline-Installationsprogramm und führen Sie es aus. Folgen Sie den Anweisungen.
4. Wählen Sie für die zu installierenden Produktfamilien die Option Wi-Fi Wireless. SimpleLink CC32xx MCUs
5. Akzeptieren Sie auf der nächsten Seite die Standardeinstellungen für Debugging-Proben und klicken Sie dann auf Beenden.

Wenn bei der Installation von Code Composer Studio Probleme auftreten, wenden Sie sich an den TI Development Tools Support, Code Composer Studio FAQs und Troubleshooting CCS.

Installieren von IAR Embedded Workbench

1. Laden Sie das Windows-Installationsprogramm für Version 8.32 der IAR Embedded Workbench for ARM herunter und führen Sie es aus. Stellen Sie für die Debug probe drivers (Debugging-Proben-Treiber) sicher, dass TI XDS ausgewählt ist:
2. Beenden Sie die Installation und starten Sie das Programm. Wählen Sie auf der Seite License Wizard (Lizenz-Assistent) die Option Register with IAR Systems to get an evaluation license (Mit IAR-System registrieren, um eine Evaluierungslizenz zu bekommen) aus oder verwenden Sie Ihre eigene IAR-Lizenz.

Installieren Sie das SDK SimpleLink CC322 0

Installieren Sie das SDK SimpleLink CC322 0. Das SimpleLink Wi-Fi CC322 0-SDK enthält Treiber für die programmierbare CC322 0SF-MCU, mehr als 40 Beispielanwendungen und die Dokumentation, die für die Verwendung der Beispiele erforderlich ist.

Installieren von Uniflash

Installieren Sie Uniflash. CCS Uniflash ist ein eigenständiges Tool zur Programmierung von On-Chip-Flash-Speichern auf TI. MCUs Uniflash verfügt über eine GUI, eine Befehlszeile und eine Scripting-Schnittstelle.

Installieren des neuesten Service-Packs

1. Platzieren Sie auf Ihrem TI CC322 0SF-LAUNCHXL den SOP-Jumper auf dem mittleren Pinsatz (Position = 1) und setzen Sie die Platine zurück.
2. Starten Sie Uniflash. Wenn Ihre CC322 LaunchPad 0SF-Karte unter Entdeckte Geräte angezeigt wird, wählen Sie Start. Wenn Ihr Board nicht erkannt wird, wählen Sie CC3220SF-LAUNCHXL aus der Liste der Boards unter Neue Konfiguration aus und wählen Sie dann Start Image Creator.
3. Wählen Sie New Project (Neues Projekt) aus.
4. Geben Sie auf der Seite Start new project (Neues Projekt starten) einen Namen für Ihr Projekt ein. Wählen Sie als Gerätetyp 0SF aus. CC322 Wählen Sie für Device Mode (Gerätemodus) erst Develop (Entwickeln) und dann Create Project (Projekt erstellen) aus.
5. Wählen Sie auf der rechten Seite des Uniflash-Anwendungsfensters Connect (Verbinden) aus.
6. Wählen Sie in der linken Spalte Advanced (Erweitert), Files (Dateien) und dann Service Pack (Service-Pack) aus.
7. Wählen Sie Durchsuchen und navigieren Sie dann zu dem Ort, an dem Sie das CC322 SimpleLink 0SF-SDK installiert haben. Die Service-Packs finden Sie unter `ti/simplelink_cc32xx_sdk_`*`VERSION`*`/tools/cc32xx_tools/servicepack-cc3x20/sp_`*`VERSION`*`.bin`.
8. Wählen Sie erst die Schaltfläche Burn (Brennen)

   ( )

   und dann Program Image (Create & Program) (Image programmieren (Erstellen und Programmieren)) zum Installieren des Service-Packs aus. Denken Sie daran, den SOP Jumper zurück an die Position 0 zu setzen und setzen Sie das Board zurück.

Konfigurieren der WLAN-Bereitstellung

So konfigurieren Sie die WLAN-Einstellungen für Ihr Board. Führen Sie einen der folgenden Schritte aus:

- Konfigurieren Sie die FreeRTOS-Demoanwendung, wie unter beschrieben. Konfiguration der FreeRTOS-Demos
- Wird SmartConfigvon Texas Instruments verwendet.

Erstellen Sie das FreeRTOS-Demoprojekt und führen Sie es aus

Erstellen Sie das FreeRTOS-Demoprojekt in TI Code Composer und führen Sie es aus

Um die FreeRTOS-Demo in TI Code Composer zu importieren

1. Öffnen Sie den TI Code Composer und klicken Sie dann auf OK, um den Standard-WorkSpace-Namen zu akzeptieren.
2. Wählen Sie auf der Seite Erste Schritte die Option Import Project (Projekt importieren) aus.
3. Unter Select search-directory (Suchverzeichnis auswählen) geben Sie `projects/ti/cc3220_launchpad/ccs/aws_demos` ein. Das Projekt `aws_demos` sollte standardmäßig ausgewählt werden. Wählen Sie Beenden aus, um das Projekt in den TI Code Composer zu importieren.
4. Doppelklicken Sie im Projekt-Explorer auf die Option aws_demos, um das Projekt zu aktivieren.
5. Wählen Sie im Projekt die Option Build Project (Projekt erstellen) aus, um sicherzustellen, dass das Projekt ohne Fehler oder Warnungen kompiliert wird.

So führen Sie die FreeRTOS-Demo in TI Code Composer aus

1. Stellen Sie sicher, dass sich der Sense On Power (SOP) -Jumper an Ihrem Texas Instruments CC322 0SF-LAUNCHXL in Position 0 befindet. Weitere Informationen finden Sie unter SimpleLink Wi-Fi CC3x2 0, Network Processor User's Guide. CC3x3x
2. Verwenden Sie ein USB-Kabel, um Ihr Texas Instruments CC322 0SF-LAUNCHXL mit Ihrem Computer zu verbinden.
3. Vergewissern Sie sich im Projekt-Explorer, dass `CC3220SF.ccxml` als aktive Zielkonfiguration ausgewählt ist. Klicken Sie zum Aktivieren mit der rechten Maustaste auf die Datei und wählen Sie Set as active target configuration (Als aktive Zielkonfiguration festlegen) aus.

4. Wählen Sie im TI Code Composer unter Run (Ausführen) die Option Debug (Debuggen) aus.
5. Wenn der Debugger am Haltepunkt in `main()` anhält, rufen Sie im Menü Ausführen auf und wählen Sie Fortsetzen aus.

Überwachung von MQTT-Nachrichten in der Cloud

Bevor Sie das FreeRTOS-Demoprojekt ausführen, können Sie den MQTT-Client in der AWS IoT Konsole so einrichten, dass er die Nachrichten überwacht, die Ihr Gerät an die Cloud sendet. AWS

Um das MQTT-Thema mit dem MQTT-Client zu abonnieren AWS IoT

1. Melden Sie sich bei der AWS IoT-Konsole an.
2. Wählen Sie im Navigationsbereich Test und dann MQTT-Testclient aus, um den MQTT-Client zu öffnen.
3. Geben Sie im Feld Subscription topic (Abonnementthema) die Option ***your-thing-name*/example/topic** ein und wählen Sie dann Subscribe to topic (Thema abonnieren).

Wenn das Demo-Projekt erfolgreich auf Ihrem Gerät ausgeführt wird, sehen Sie „Hello World!“ mehrfach zu dem Thema gesendet, das Sie abonniert haben.

FreeRTOS-Demo-Projekt in IAR Embedded Workbench erstellen und ausführen

Um die FreeRTOS-Demo in IAR Embedded Workbench zu importieren

1. Öffnen Sie die IAR Embedded Workbench und wählen Sie Datei und dann Open Workspace (WorkSpace öffnen) aus.
2. Navigieren Sie zu `projects/ti/cc3220_launchpad/iar/aws_demos`, wählen Sie aws_demos.eww aus und klicken Sie anschließend auf OK.
3. Klicken Sie rechts auf den Projektnamen (`aws_demos`) und wählen Sie dann Make (Vornehmen) aus.

So führen Sie die FreeRTOS-Demo in IAR Embedded Workbench aus

1. Stellen Sie sicher, dass sich der Sense On Power (SOP) -Jumper an Ihrem Texas Instruments CC322 0SF-LAUNCHXL in Position 0 befindet. Weitere Informationen finden Sie unter SimpleLink Wi-Fi CC3x2 0, Network Processor User's Guide. CC3x3x

2. Verwenden Sie ein USB-Kabel, um Ihr Texas Instruments CC322 0SF-LAUNCHXL mit Ihrem Computer zu verbinden.
3. Erstellen Sie Ihr Projekt neu.

   Um das Projekt aus dem Menü Projekt neu zu erstellen, wählen Sie Make (Vornehmen) aus.
4. Wählen Sie im Menü Projekt die Option Herunterladen und Debuggen aus. Sie können die Meldung „Warnung: Fehler beim Initialisieren“ ignorieren, falls sie angezeigt EnergyTrace wird. Weitere Informationen zu finden Sie EnergyTrace unter MSP-Technologie EnergyTrace .
5. Wenn der Debugger am Haltepunkt in `main()` anhält, rufen Sie im Menü Debug (Debuggen) auf und wählen Sie Go (Fortfahren) aus.

Verwendung CMake mit FreeRTOS

Wenn Sie es vorziehen, keine IDE für die FreeRTOS-Entwicklung CMake zu verwenden, können Sie alternativ die Demo-Anwendungen oder Anwendungen, die Sie mit Code-Editoren und Debugging-Tools von Drittanbietern entwickelt haben, erstellen und ausführen.

Um die FreeRTOS-Demo zu erstellen mit CMake

1. Erstellen Sie einen Ordner, der die generierten Build-Dateien () *`build-folder`* enthält.
2. Stellen Sie sicher, dass Ihr Suchpfad (Umgebungsvariable $PATH) den Ordner enthält, in dem sich die TI CGT-Compiler-Binärdatei befindet (z. B. `C:\ti\ccs910\ccs\tools\compiler\ti-cgt-arm_18.12.2.LTS\bin`).

   Wenn Sie den TI ARM-Compiler mit Ihrem TI-Board verwenden, verwenden Sie den folgenden Befehl zum Generieren von Build-Dateien aus dem Quellcode:

   ```
   cmake -DVENDOR=ti -DBOARD=cc3220_launchpad -DCOMPILER=arm-ti -S freertos -B build-folder
   ```

Weitere Informationen finden Sie unter Verwendung CMake mit FreeRTOS.

Fehlerbehebung

Wenn Sie im MQTT-Client der AWS IoT Konsole keine Meldungen sehen, müssen Sie möglicherweise die Debug-Einstellungen für das Board konfigurieren.

So konfigurieren Sie Debug-Einstellungen für TI-Boards

1. Wählen Sie im Projekt-Explorer des Code Composers die Option aws_demos aus.
2. Wählen Sie im Menü Ausführen die Option Debug Configurations (Debug-Konfigurationen) aus.
3. Wählen Sie im Navigationsbereich die Option aws_demos aus.
4. Wählen Sie auf der Registerkarte Target (Ziel) unter Connection Options (Verbindungsoptionen) die Option Reset the target on a connect (Ziel auf einer Verbindung zurücksetzen) aus.
5. Wählen Sie Apply und dann Close.

Wenn diese Schritte nicht funktionieren, sollten Sie die Ausgabe des Programms im seriellen Terminal ansehen. Sie sollten einen Text sehen, der die Ursache des Problems angibt.

Allgemeine Informationen zur Problembehandlung bei Getting Started with FreeRTOS finden Sie unter. Fehlerbehebung – Erste Schritte

Erste Schritte mit dem Windows-Gerätesimulator

Dieses Tutorial enthält Anweisungen für die ersten Schritte mit dem FreeRTOS Windows Device Simulator.

Bevor Sie beginnen, müssen Sie FreeRTOS konfigurieren AWS IoT und herunterladen, um Ihr Gerät mit der AWS Cloud zu verbinden. Detaillierte Anweisungen finden Sie unter Erste Schritte. In diesem Tutorial wird der Pfad zum FreeRTOS-Download-Verzeichnis als bezeichnet. *`freertos`*

FreeRTOS wird als Zip-Datei veröffentlicht, die die FreeRTOS-Bibliotheken und Beispielanwendungen für die von Ihnen angegebene Plattform enthält. Um die Beispiele auf einem Windows-Computer auszuführen, laden Sie die Bibliotheken und portierten Beispiele für die Ausführung auf Windows herunter. Diese Dateien werden als FreeRTOS Simulator für Windows bezeichnet.

> **Note**
>
> Dieses Tutorial kann nicht erfolgreich auf Amazon EC2 Windows-Instances ausgeführt werden.

Einrichten Ihrer Entwicklungsumgebung

1. Installieren Sie die neueste Version von Npcap. Wählen Sie während der Installation den "WinPcap API-kompatiblen Modus“.
2. Installieren Sie Microsoft Visual Studio.

   Visual Studio-Versionen 2017 und 2019 funktionieren. Alle Editionen dieser Visual Studio-Versionen werden unterstützt (Community, Professional oder Enterprise).

   Installieren Sie zusätzlich zur IDE die Komponente Desktop-Entwicklung mit C++.

   Installieren Sie das neueste Windows 10 SDK. Sie können dies unter dem Abschnitt Optional der Desktop-Entwicklung mit der Komponente C++ auswählen.
3. Stellen Sie sicher, dass Sie über eine aktive kabelgebundene Ethernet-Verbindung verfügen.
4. (Optional) Wenn Sie das CMake basierte Build-System verwenden möchten, um Ihre FreeRTOS-Projekte zu erstellen, installieren Sie die neueste Version von. CMake FreeRTOS benötigt CMake Version 3.13 oder höher.

Überwachung von MQTT-Nachrichten in der Cloud

Bevor Sie das FreeRTOS-Demoprojekt ausführen, können Sie den MQTT-Client in der AWS IoT Konsole so einrichten, dass er die Nachrichten überwacht, die Ihr Gerät an die Cloud sendet. AWS

Um das MQTT-Thema mit dem MQTT-Client zu abonnieren AWS IoT

1. Melden Sie sich an der AWS IoT -Konsole an.
2. Wählen Sie im Navigationsbereich Test und dann MQTT-Testclient, um den MQTT-Client zu öffnen.
3. Geben Sie im Feld Subscription topic (Abonnementthema) die Option ***your-thing-name*/example/topic** ein und wählen Sie dann Subscribe to topic (Thema abonnieren).

Wenn das Demo-Projekt erfolgreich auf Ihrem Gerät ausgeführt wird, sehen Sie „Hello World!“ mehrfach zu dem Thema gesendet, das Sie abonniert haben.

Erstellen und starten Sie das FreeRTOS-Demoprojekt

Sie können Visual Studio verwenden oder CMake FreeRTOS-Projekte erstellen.

Erstellen und Ausführen des FreeRTOS-Demoprojekts mit der Visual Studio IDE

1. Laden Sie das Projekt in Visual Studio.

   Klicken Sie in Visual Studio im Menü File (Datei) auf die Option Open (Öffnen). Wählen Sie File/Solution (Datei/Lösung), navigieren Sie zur Datei `projects/pc/windows/visual_studio/aws_demos/aws_demos.sln` und wählen Sie dann Open (Öffnen).
2. Richten Sie das Demoprojekt neu aus.

   Das bereitgestellte Demoprojekt hängt vom Windows-SDK ab, hat aber keine Windows-SDK-Version angegeben. Standardmäßig kann die IDE versuchen, die Demo mit einer SDK-Version zu erstellen, die nicht auf Ihrem Computer vorhanden ist. Um die Windows-SDK-Version einzustellen, klicken Sie mit der rechten Maustaste auf `aws_demos` und wählen Sie dann Retarget project (Projekte neu ausrichten). Dadurch wird das Fenster Review solution actions (Lösungsaktionen überprüfen) geöffnet. Wählen Sie eine Windows SDK-Version aus, die auf Ihrem Computer vorhanden ist (der Anfangswert in der Dropdownliste ist in Ordnung), und wählen Sie dann OK.
3. Erstellen und Ausführen des Projekts.

   Wählen Sie im Menü Build (Erstellen) die Option Build Solution (Lösung erstellen) aus und stellen Sie sicher, dass die Lösung ohne Fehler und Warnungen erstellt wird. Wählen Sie Debug, Start debugging (Debugging starten), um das Projekt auszuführen. Bei der ersten Ausführung müssen Sie eine Netzwerkschnittstelle auswählen.

Aufbau und Ausführung des FreeRTOS-Demo-Projekts mit CMake

Wir empfehlen, dass Sie die CMake GUI anstelle des CMake Befehlszeilentools verwenden, um das Demo-Projekt für den Windows Simulator zu erstellen.

Öffnen Sie nach der Installation CMake die CMake GUI. Unter Windows finden Sie dies im Startmenü unter CMakeCMake (cmake-gui).

1. Legt das FreeRTOS-Quellcodeverzeichnis fest.

   Stellen Sie in der GUI das FreeRTOS-Quellcodeverzeichnis (*`freertos`*) für Wo ist der Quellcode ein.

   Stellen Sie ein, wo ***`freertos`*`/build`** die Binärdateien erstellt werden sollen.
2. Konfigurieren Sie das CMake Projekt.

Wählen Sie in der CMake GUI Eintrag hinzufügen und legen Sie im Fenster Cache-Eintrag hinzufügen die folgenden Werte fest:

Name

AFR_BOARD

Typ

STRING

Wert

pc.windows

Beschreibung

(Optional)

3. Wählen Sie Konfigurieren aus. Wenn Sie CMake aufgefordert werden, das Build-Verzeichnis zu erstellen, wählen Sie Ja und wählen Sie dann unter Generator für dieses Projekt angeben einen Generator aus. Wir empfehlen die Verwendung von Visual Studio als Generator, aber auch Ninja wird unterstützt. (Beachten Sie, dass bei der Verwendung von Visual Studio 2019 die Plattform auf Win32 anstatt auf ihre Standardeinstellung eingestellt sein sollte.) Lassen Sie die anderen Generatoroptionen unverändert und wählen Sie Fertig stellen.
4. Generieren und öffnen Sie das CMake Projekt.

   Nachdem Sie das Projekt konfiguriert haben, zeigt die CMake GUI alle verfügbaren Optionen für das generierte Projekt an. Für die Zwecke dieses Tutorials können Sie die Optionen auf ihren Standardwerten belassen.

   Wählen Sie Generate (Generieren), um eine Visual Studio-Lösung zu erstellen, und wählen Sie dann Open Project (Projekt öffnen), um das Projekt in Visual Studio zu öffnen.

   Klicken Sie in Visual Studio mit der rechten Maustaste auf das `aws_demos` Projekt und wählen Sie Als StartUp Projekt festlegen. Dies ermöglicht es Ihnen, das Projekt zu erstellen und auszuführen. Bei der ersten Ausführung müssen Sie eine Netzwerkschnittstelle auswählen.

Weitere Hinweise zur Verwendung CMake mit FreeRTOS finden Sie unter. Verwendung CMake mit FreeRTOS

Konfigurieren Ihrer Netzwerkschnittstelle

Beim ersten Durchlauf des Demoprojekts müssen Sie die zu verwendende Netzwerkschnittstelle auswählen. Das Programm zählt Ihre Netzwerkschnittstellen. Suchen Sie nach der Zahl für Ihre kabelgebundene Ethernet-Schnittstelle. Die Ausgabe sollte in etwa wie folgt aussehen:

```
0 0 [None] FreeRTOS_IPInit
1 0 [None] vTaskStartScheduler
1. rpcap://\Device\NPF_{AD01B877-A0C1-4F33-8256-EE1F4480B70D}
(Network adapter 'Intel(R) Ethernet Connection (4) I219-LM' on local host)

2. rpcap://\Device\NPF_{337F7AF9-2520-4667-8EFF-2B575A98B580}
(Network adapter 'Microsoft' on local host)

The interface that will be opened is set by "configNETWORK_INTERFACE_TO_USE", which
should be defined in FreeRTOSConfig.h

ERROR:  configNETWORK_INTERFACE_TO_USE is set to 0, which is an invalid value.
Please set configNETWORK_INTERFACE_TO_USE to one of the interface numbers listed above,
then re-compile and re-start the application.  Only Ethernet (as opposed to Wi-Fi)
interfaces are supported.
```

Nachdem Sie die Zahl für Ihre kabelgebundene Ethernet-Schnittstelle ermittelt haben, schließen Sie das Anwendungsfenster. Im vorherigen Beispiel lautet die zu verwendende Nummer1.

Öffnen Sie `FreeRTOSConfig.h` und legen Sie `configNETWORK_INTERFACE_TO_USE` auf die Zahl fest, die mit Ihrer fest implementierten Netzwerkschnittstelle übereinstimmt.

> **Important**
>
> Es werden nur Ethernet-Schnittstellen unterstützt. WLAN wird nicht unterstützt.

Fehlerbehebung

Fehlerbehebung bei häufigen Problemen unter Windows

Sie können auf den folgenden Fehler stoßen, wenn Sie versuchen, das Demoprojekt mit Visual Studio zu erstellen:

```
Error "The Windows SDK version X.Y was not found" when building the provided Visual
 Studio solution.
```

Das Projekt muss auf eine Windows SDK-Version ausgerichtet sein, die auf Ihrem Computer vorhanden ist.

Allgemeine Informationen zur Problembehandlung bei den ersten Schritten mit FreeRTOS finden Sie unter. Fehlerbehebung – Erste Schritte

Erste Schritte mit dem Xilinx MicroZed Avnet Industrial IoT Kit

> **Important**
>
> Diese Referenzintegration wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Dieses Tutorial enthält Anweisungen für die ersten Schritte mit dem Xilinx Avnet MicroZed Industrial IoT Kit. Wenn Sie das Xilinx Avnet MicroZed Industrial IoT Kit nicht haben, besuchen Sie den AWS Partner Device Catalog, um eines von unserem Partner zu erwerben.

Bevor Sie beginnen, müssen Sie FreeRTOS konfigurieren AWS IoT und herunterladen, um Ihr Gerät mit der AWS Cloud zu verbinden. Detaillierte Anweisungen finden Sie unter Erste Schritte. In diesem Tutorial wird der Pfad zum FreeRTOS-Download-Verzeichnis als bezeichnet. *`freertos`*

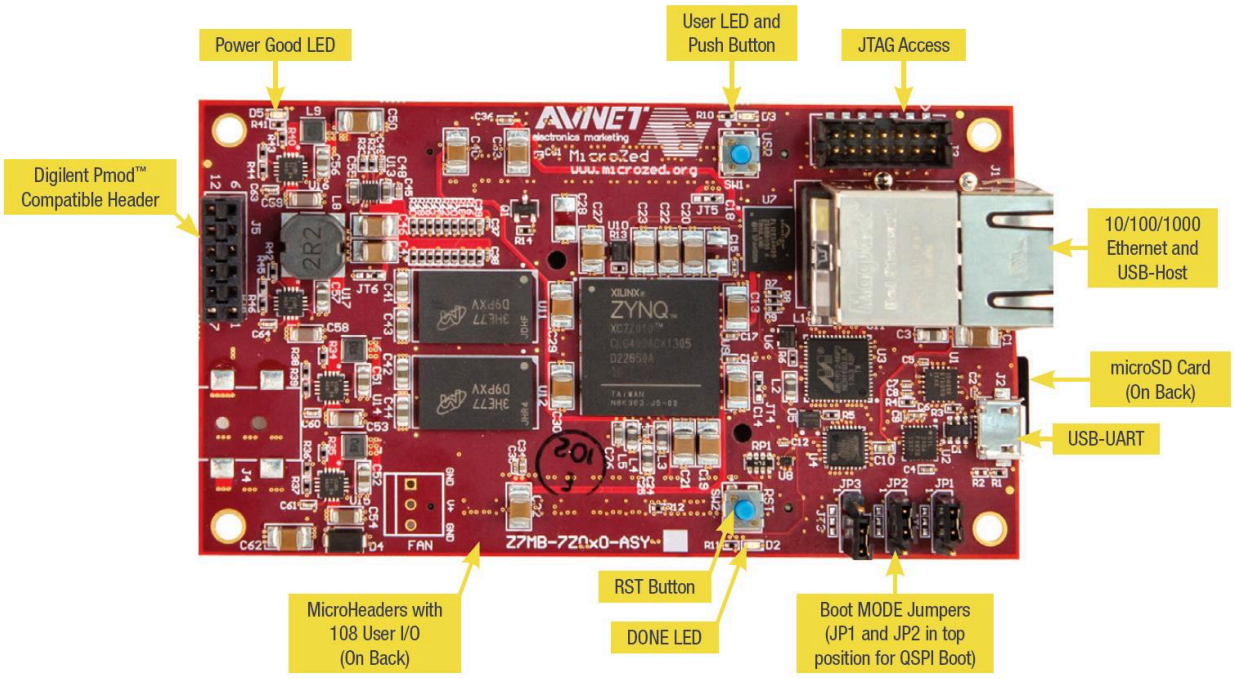
Übersicht

Dieses Tutorial enthält Anweisungen für die folgenden ersten Schritte:

1. Verbinden Ihres Boards mit einem Host-Computer.
2. Installieren von Software auf dem Host-Computer zum Entwickeln und Debuggen eingebetteter Anwendungen für Ihr Mikrocontroller-Board.
3. Cross-Compilierung einer FreeRTOS-Demo-Anwendung zu einem Binär-Image.
4. Laden des binären Anwendungs-Image auf Ihr Board und Ausführen der Anwendung.

Richten Sie die Hardware ein MicroZed

Das folgende Diagramm kann bei der Einrichtung der MicroZed Hardware hilfreich sein:



So richten Sie das Board ein MicroZed

1. Connect Sie Ihren Computer mit dem USB-UART-Anschluss auf Ihrem MicroZed Board.
2. Connect Sie Ihren Computer mit dem JTAG Access-Port auf Ihrem MicroZed Board.
3. Connect einen Router oder einen mit dem Internet verbundenen Ethernet-Port mit dem Ethernet- und USB-Host-Anschluss auf Ihrem Board. MicroZed

Einrichten Ihrer Entwicklungsumgebung

Um FreeRTOS-Konfigurationen für das MicroZed Kit einzurichten, müssen Sie das Xilinx Software Development Kit (XSDK) verwenden. XSDK wird auf Windows und Linux unterstützt.

XSDK herunterladen und installieren

Um die Xilinx-Software zu installieren, benötigen Sie ein kostenloses Xilinx-Konto.

So laden Sie das XDSK herunter:

1. Gehen Sie zur Download-Seite für den Software Development Kit Standalone Client. WebInstall
2. Wählen Sie die geeignete Option für Ihr Betriebssystem.

3. Sie werden auf eine Xilinx Anmeldeseite weitergeleitet.

   Wenn Sie ein Konto bei Xilinx haben, geben Sie Ihre Anmeldeinformationen ein und wählen Sie dann Anmelden.

   Wenn Sie kein Konto haben, wählen Sie Create your account (Konto erstellen). Nachdem Sie sich registriert haben, erhalten Sie eine E-Mail mit einem Link zur Aktivierung Ihres Xilinx-Kontos.
4. Geben Sie auf der Seite Name and Address Verification (Namens- und Adressverifizierung) Ihre Informationen ein und wählen Sie dann Next (Weiter). Der Download ist jetzt zum Start bereit.
5. Speichern Sie die `Xilinx_SDK_version_os`-Datei.

So installieren Sie das XSDK:

1. Öffnen Sie die `Xilinx_SDK_version_os` Datei.
2. Wählen Sie unter Select Edition to Install (Zu installierende Edition auswählen) die Option Xilinx Software Development Kit (XSDK) und anschließend Next (Weiter) aus.
3. Wählen Sie auf der folgenden Seite des Installationsassistenten unter Installation Options (Installationsoptionen) die Option Install Cable Drivers (Kabeltreiber installieren) und wählen Sie anschließend Next (Weiter) aus.

Wenn Ihr Computer die USB-UART-Verbindung nicht erkennt, installieren Sie die MicroZed 0x Bridge VCP-Treiber manuell. CP21 USB-to-UART Anweisungen finden Sie im Silicon Labs CP21 0x-Installationshandbuch. USB-to-UART

Weitere Informationen zu XDSK finden Sie unter Getting Started with the Xilinx SDK auf der Xilinx-Website.

Überwachung von MQTT-Nachrichten in der Cloud

Bevor Sie das FreeRTOS-Demoprojekt ausführen, können Sie den MQTT-Client in der AWS IoT Konsole so einrichten, dass er die Nachrichten überwacht, die Ihr Gerät an die Cloud sendet. AWS

Um das MQTT-Thema mit dem MQTT-Client zu abonnieren AWS IoT

1. Melden Sie sich an der AWS IoT -Konsole an.
2. Wählen Sie im Navigationsbereich Test und dann MQTT-Testclient, um den MQTT-Client zu öffnen.

3. Geben Sie im Feld Subscription topic (Abonnementthema) die Option ***your-thing-name*/example/topic** ein und wählen Sie dann Subscribe to topic (Thema abonnieren).

Erstellen und starten Sie das FreeRTOS-Demoprojekt

Öffnen Sie die FreeRTOS-Demo in der XSDK-IDE

1. Starten Sie die XSDK-IDE mit dem auf *freertos*`/projects/xilinx/microzed/xsdk` festgelegten Workspace-Verzeichnis.
2. Schließen Sie die Willkommenseite. Wählen Sie im Menü Project (Projekt) aus und deaktivieren Sie dann Build Automatically (Automatisch erstellen).
3. Wählen Sie im Menü die Option File (Datei) und anschließend Import aus.
4. Erweitern Sie auf der Seite Select (Auswählen) die Option General (Allgemein), wählen Sie Existing Projects into Workspace (Vorhandene Projekte in WorkSpace) aus und klicken Sie dann auf Next (Weiter).
5. Wählen Sie auf der Seite Projekte importieren die Option Stammverzeichnis auswählen und geben Sie dann das Stammverzeichnis Ihres Demo-Projekts ein:. *freertos*`/projects/xilinx/microzed/xsdk/aws_demos` Um nach dem Verzeichnis zu suchen, wählen Sie Browse (Durchsuchen) aus.

   Nachdem Sie ein Stammverzeichnis angegeben haben, werden die Projekte in diesem Verzeichnis auf der Seite Import Projects (Projekte importieren) angezeigt. Standardmäßig sind alle verfügbaren Projekte ausgewählt.

   > **Note**
   >
   > Wenn Sie oben auf der Seite Import Projects (Projekte importieren) eine Warnung sehen ("Some projects cannot be imported because they already exist in the workspace. (Einige Projekte können nicht importiert werden, da sie bereits im Workspace vorhanden sind.)"), können Sie sie ignorieren.

6. Wenn alle Projekte ausgewählt sind, wählen Sie Finish (Fertigstellen).
7. Wenn Sie die `MicroZed_hw_platform_0` Projekte `aws_bspfsbl`, und nicht im Projektbereich sehen, wiederholen Sie die vorherigen Schritte ab #3, wobei das Stammverzeichnis jedoch auf*freertos*`/vendors/xilinx`, eingestellt ist, und importieren Sie `aws_bspfsbl`, und`MicroZed_hw_platform_0`.

8. Wählen Sie im Menü die Option Window (Fenster) und anschließend Preferences (Einstellungen) aus.
9. Erweitern Sie im Navigationsbereich die Option Run/Debug (Ausführen/Debuggen), wählen Sie String Substitution (Zeichenfolge ersetzen) und anschließend New (Neu) aus.
10. Geben Sie in New String Substitution Variable (Neue Zeichenfolgenersatzvariable) für Name **`AFR_ROOT`** ein. Geben Sie für Value (Wert) den Stammpfad der *`freertos`*`/projects/xilinx/microzed/xsdk/aws_demos` ein. Wählen Sie OK und anschließend OK aus, um die Variable zu speichern, und schließen Sie Preferences (Einstellungen).

Erstellen Sie das FreeRTOS-Demoprojekt

1. Wählen Sie in der XSDK IDE aus dem Menü die Option Project (Projekt) und anschließend Clean (Bereinigen) aus.
2. Belassen Sie in Clean (Bereinigen) die Optionen bei ihren Standardwerten und wählen Sie dann OK aus. XSDK bereinigt und erstellt alle Projekte und generiert dann `.elf`-Dateien.

> **Note**
>
> Um alle Projekte zu erstellen, ohne sie zu bereinigen, wählen Sie Project (Projekt) und anschließend Build All (Alle erstellen) aus.
> Um einzelne Projekte zu erstellen, wählen Sie das Projekt, das Sie erstellen möchten, Project (Projekt) und anschließend Build Project (Projekt erstellen) aus.

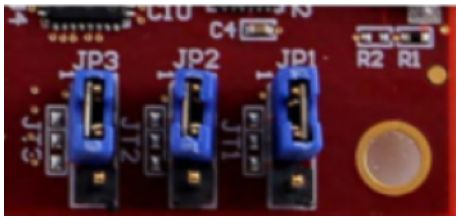
Generieren Sie das Boot-Image für das FreeRTOS-Demo-Projekt

1. Klicken Sie mit der rechten Maustaste in der XSDK IDE auf aws_demos und wählen Sie dann Create Boot Image (Start-Image erstellen) aus.
2. Wählen Sie unter Create Boot Image (Start-Image erstellen) die Option Create new BIF file (Neue BIF-Datei erstellen) aus.
3. Wählen Sie neben Output BIF file path (Ausgabe-BIF-Dateipfad) die Option Browse (Durchsuchen) aus und anschließend in *`<freertos>`*`/vendors/xilinx/microzed/aws_demos/aws_demos.bif` `aws_demos.bif` aus.
4. Wählen Sie Hinzufügen aus.

5. Wählen Sie unter Add new boot image partition (Neue Start-Image-Partition hinzufügen) neben File path (Dateipfad) die Option Browse (Durchsuchen) und anschließend `fsbl.elf` unter `vendors/xilinx/fsbl/Debug/fsbl.elf` aus.
6. Wählen Sie für den Partition type (Partitionstyp) die Option bootloader und anschließend OK aus.
7. Wählen Sie unter Create Boot Image (Start-Image erstellen) die Option Create Image (Image erstellen) aus. Wählen Sie unter Override Files (Dateien überschreiben) die Option OK, aus, um die vorhandene `aws_demos.bif` zu überschreiben und die Datei `BOOT.bin` unter `projects/xilinx/microzed/xsdk/aws_demos/BOOT.bin` zu generieren.

JTAG-Debugging

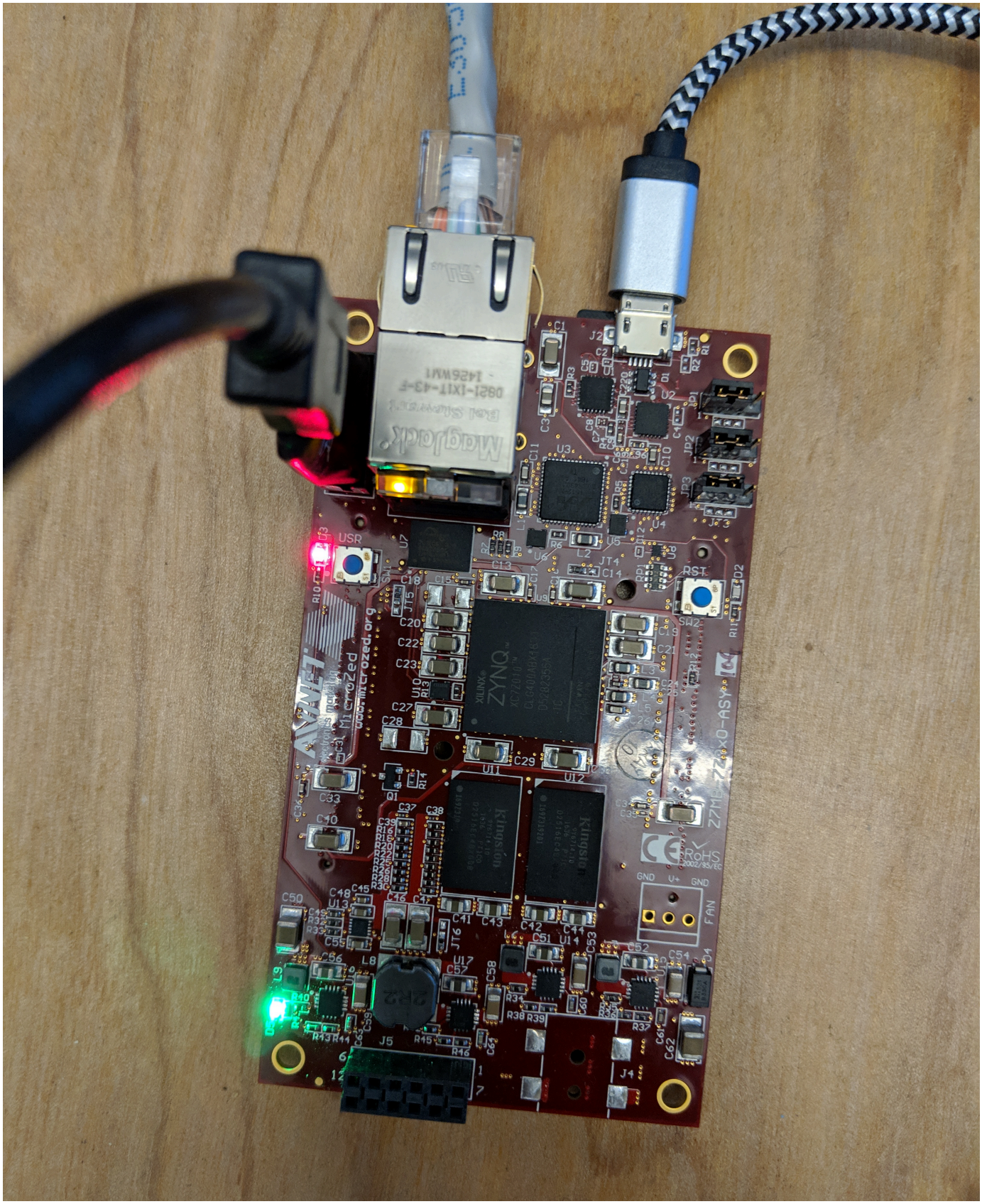
1. Stellen Sie die Startmodus-Jumper Ihres MicroZed Boards auf den JTAG-Boot-Modus ein.



2. Stecken Sie die MicroSD-Karte in den MicroSD-Kartensteckplatz ein, der sich direkt unter dem USB-UART-Port befindet.

> **Note**
>
> Bevor Sie debuggen, sichern Sie unbedingt alle Inhalte, die Sie auf der MicroSD-Karte haben.

Ihre Platine sollte ähnlich wie folgt aussehen:


USR
RST
ZYNQ
XILINX
Kingston
Kingston
GND
V+
GND
FAN
RoHS

3. Klicken Sie mit der rechten Maustaste in der XSDK IDE auf aws_demos, wählen Sie Debug As (Debuggen als) und anschließend 1 Launch on System Hardware (System Debugger) (1 Start auf Systemhardware (System-Debugger)) aus.
4. Wenn der Debugger am Haltepunkt in `main()` anhält, wählen Sie im Menü Run (Ausführen) und anschließend Resume (Fortsetzen) aus.

> **Note**
>
> Wenn Sie die Anwendung zum ersten Mal ausführen, wird ein neues Zertifikat-Schlüsselpaar in nicht flüchtigen Speicher importiert. Für nachfolgende Ausführungen können Sie `vDevModeKeyProvisioning()` in der Datei `main.c` auskommentieren, bevor Sie die Images und die Datei `BOOT.bin` neu erstellen. Dadurch wird verhindert, dass bei jeder Ausführung die Zertifikate und der Schlüssel zur Speicherung kopiert werden.

Sie können wählen, ob Sie Ihr MicroZed Board von einer microSD-Karte oder von QSPI Flash booten möchten, um das FreeRTOS-Demo-Projekt auszuführen. Anweisungen finden Sie unter Generieren Sie das Boot-Image für das FreeRTOS-Demo-Projekt und Führen Sie das FreeRTOS-Demo-Projekt aus.

Führen Sie das FreeRTOS-Demo-Projekt aus

Um das FreeRTOS-Demo-Projekt auszuführen, können Sie Ihr MicroZed Board von einer microSD-Karte oder von QSPI-Flash booten.

Wenn Sie Ihr MicroZed Board für die Ausführung des FreeRTOS-Demo-Projekts einrichten, schauen Sie sich das Diagramm in an. Richten Sie die Hardware ein MicroZed Vergewissern Sie sich, dass Sie Ihr MicroZed Board an Ihren Computer angeschlossen haben.

Booten Sie das FreeRTOS-Projekt von einer microSD-Karte

Formatieren Sie die microSD-Karte, die im Lieferumfang des Xilinx MicroZed Industrial IoT Kit enthalten ist.

1. Kopieren Sie die Datei `BOOT.bin` auf die MicroSD-Karte.
2. Stecken Sie die Karte in den MicroSD-Kartensteckplatz unter dem USB-UART-Port.
3. Stellen Sie die MicroZed Startmodus-Jumper auf den SD-Startmodus ein.

4. Drücken Sie die RST-Taste, um das Gerät zurückzusetzen und die Anwendung zu starten. Sie können das USB-UART-Kabel auch vom USB-UART-Port trennen und das Kabel dann wieder einstecken.

Booten Sie das FreeRTOS-Demo-Projekt von QSPI Flash

1. Stellen Sie die Startmodus-Jumper Ihres MicroZed Boards auf den JTAG-Boot-Modus ein.

2. Vergewissern Sie sich, dass Ihr Computer mit den Zugriffsports USB-UART und JTAG verbunden ist. Die grüne Betriebsbereitschafts-LED sollte aufleuchten.
3. Wählen Sie in der XSDK IDE vom Menü Xilinx und anschließend Program Flash (Flash programmieren) aus.
4. In Program Flash Memory (Flash-Speicher programmieren) sollte die Hardwareplattform automatisch ausgefüllt werden. Wählen Sie unter Verbindung Ihren MicroZed Hardwareserver aus, um Ihre Platine mit Ihrem Host-Computer zu verbinden.

> **Note**
>
> Wenn Sie das Xilinx Smart Lync JTAG-Kabel verwenden, müssen Sie einen Hardwareserver in der XSDK IDE erstellen. Wählen Sie New (Neu) aus und definieren Sie anschließend Ihren Server.

5. Geben Sie unter Image File (Image-Datei) den Verzeichnispfad zu Ihrer `BOOT.bin`-Image-Datei ein. Wählen Sie Browse (Durchsuchen) aus, um stattdessen nach der Datei zu suchen.
6. Geben Sie unter Offset **`0x0`** ein.
7. Geben Sie unter FSBL File (FSBL-Datei) den Verzeichnispfad zu Ihrer `fsbl.elf`-Datei ein. Wählen Sie Browse (Durchsuchen) aus, um stattdessen nach der Datei zu suchen.

8. Wählen Sie Programmieren aus, um Ihre Platine zu programmieren.
9. Nachdem die QSPI-Programmierung abgeschlossen ist, entfernen Sie das USB-UART-Kabel, um die Platine auszuschalten.
10. Stellen Sie die Startmodus-Jumper Ihres MicroZed Boards auf den QSPI-Boot-Modus ein.
11. Stecken Sie die Karte in den MicroSD-Kartensteckplatz ein, der sich direkt unter dem USB-UART-Port befindet.

> **Note**
>
> Sichern Sie unbedingt alle Inhalte, die Sie auf der MicroSD-Karte haben.

12. Drücken Sie die RST-Taste, um das Gerät zurückzusetzen und die Anwendung zu starten. Sie können das USB-UART-Kabel auch vom USB-UART-Port trennen und das Kabel dann wieder einstecken.

Fehlerbehebung

Wenn Sie Build-Fehler feststellen, die sich auf falsche Pfade beziehen, versuchen Sie, das Projekt zu bereinigen und neu zu erstellen, wie in Erstellen Sie das FreeRTOS-Demoprojekt beschrieben.

Wenn Sie Windows verwenden, stellen Sie sicher, dass Sie Schrägstriche (/) verwenden, wenn Sie die Zeichenkettenersetzungsvariablen in der Windows XSDK IDE festlegen.

Allgemeine Informationen zur Problembehandlung bei Getting Started with FreeRTOS finden Sie unter. Fehlerbehebung – Erste Schritte

Die nächsten Schritte mit FreeRTOS

> **Important**
>
> Diese Seite bezieht sich auf das Amazon-FreeRTOS-Repository, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Nachdem du das FreeRTOS-Demoprojekt für dein Board erstellt, flasht und ausgeführt hast, kannst du die FreeRTOS.org-Website besuchen, um mehr über das Erstellen eines neuen FreeRTOS-Projekts zu erfahren. Es gibt auch Demos für viele FreeRTOS-Bibliotheken, die zeigen, wie man wichtige Aufgaben ausführt, mit AWS IoT Diensten interagiert und Board-spezifische Funktionen (wie Mobilfunkmodems) programmiert. Weitere Informationen finden Sie auf der Seite mit den Kategorien der FreeRTOS-Bibliothek.

Die Website FreeRTOS.org bietet auch ausführliche Informationen zum FreeRTOS-Kernel sowie zu grundlegenden Konzepten von Echtzeitbetriebssystemen. Weitere Informationen finden Sie auf den Seiten FreeRTOS Kernel Developer Docs und FreeRTOS Kernel Secondary Docs.

FreeRTOS RTOS-Updates Over-the-Air

> **Note**
>
> Aktuelle Informationen zur Durchführung von AWS IoT Over-the-Air (OTA) -Updates finden Sie unter Over-the-air (OTA) auf der FreeRTOS-Website.

Over-the-air (OTA) -Updates ermöglichen es Ihnen, Firmware-Updates für ein oder mehrere Geräte in Ihrer Flotte bereitzustellen. Obwohl OTA-Updates für die Aktualisierung der Geräte-Firmware entwickelt wurden, können Sie damit beliebige Dateien an eines oder mehrere bei AWS IoT registrierte Geräte senden. Wenn Sie Updates mit Over-the-Air senden, wird empfohlen, sie digital zu signieren, damit die Geräte, die die Dateien empfangen, überprüfen können, dass sie unterwegs nicht manipuliert wurden.

Sie können Code Signing for AWS IoT verwenden, um Ihre Dateien zu signieren. Sie können Ihre Dateien auch mit Ihren eigenen Code-Signing-Tools signieren.

Wenn Sie ein OTA-Update erstellen, erstellt die OTA Update Manager-Service einen AWS IoT -Auftrag, um Ihre Geräte darüber zu informieren, dass ein Update verfügbar ist. Die OTA-Demoanwendung läuft auf Ihrem Gerät und erstellt eine FreeRTOS-Aufgabe, die Benachrichtigungsthemen für AWS IoT Jobs abonniert und auf Aktualisierungsnachrichten wartet. Wenn ein Update verfügbar ist, veröffentlicht der OTA-Agent Anfragen an AWS IoT und empfängt Updates über das HTTP- oder MQTT-Protokoll, je nach den von Ihnen ausgewählten Einstellungen. Der OTA-Agent überprüft die digitale Signatur der heruntergeladenen Dateien. Wenn die Dateien gültig sind, installiert er das Firmware-Update. Wenn Sie die FreeRTOS OTA Update Demo-Anwendung nicht verwenden, müssen Sie sie AWS IoT Over-the-Air-Bibliothek (OTA) in Ihre eigene Anwendung integrieren, um die Firmware-Update-Funktion zu erhalten.

over-the-airFreeRTOS-Updates ermöglichen Ihnen:

- Signieren Sie die Firmware vor der Bereitstellung digital.
- Bereitstellen neuer Firmware-Images auf einem einzelnen Gerät, einer Gruppe von Geräten oder Ihrer gesamten Flotte
- Bereitstellen von Firmware auf Geräten, wenn sie zu Gruppen hinzugefügt, zurückgesetzt oder neu bereitgestellt werden
- Überprüfen der Authentizität und Integrität der neuen Firmware nach der Bereitstellung auf Geräten
- Überwachen des Fortschritts einer Bereitstellung
- Debuggen einer fehlgeschlagenen Bereitstellung

Markieren von OTA-Ressourcen

Zur einfacheren Verwaltung Ihrer OTA-Ressourcen können Sie Updates und Streams optional eigene Metadaten in Form von Tags zuweisen. Mithilfe von Tags können Sie Ihre AWS IoT Ressourcen auf unterschiedliche Weise kategorisieren (z. B. nach Zweck, Eigentümer oder Umgebung). Dies ist nützlich, wenn Sie viele Ressourcen desselben Typs haben. Sie können eine Ressource schnell anhand der ihr zugeordneten Tags identifizieren.

Weitere Informationen finden Sie unter Markieren Ihrer AWS IoT -Ressourcen.

Voraussetzungen für OTA-Updates

Gehen Sie wie folgt vor, um over-the-air (OTA-) Updates zu verwenden:

- Überprüfen Sie das Voraussetzungen für OTA-Updates mit HTTP oder das Voraussetzungen für OTA-Updates mit MQTT.
- Erstellen Sie einen Amazon S3 S3-Bucket, um Ihr Update zu speichern.
- Erstellen einer OTA-Update-Servicerolle.
- Erstellen einer OTA-Benutzerrichtlinie.
- Erstellen eines Zertifikats für die Codesignierung.
- Wenn Sie Code Signing für verwenden AWS IoT,Gewähren Sie Zugriff auf die Codesignatur für AWS IoT.
- Laden Sie FreeRTOS mit der OTA-Bibliothek herunter.

Erstellen Sie einen Amazon S3 S3-Bucket, um Ihr Update zu speichern

OTA-Aktualisierungsdateien werden in Amazon S3 S3-Buckets gespeichert.

Wenn Sie Code Signing for verwenden, verwendet der Befehl AWS IoT, mit dem Sie einen Codesignaturauftrag erstellen, einen Quell-Bucket (in dem sich das unsignierte Firmware-Image befindet) und einen Ziel-Bucket (in den das signierte Firmware-Image geschrieben wird). Sie können denselben Bucket für die Quelle und das Ziel angeben. Die Dateinamen werden GUIDs so geändert, dass die Originaldateien nicht überschrieben werden.

So erstellen Sie einen Amazon-S3-Bucket

1. Melden Sie sich bei der Amazon S3 S3-Konsole unter an https://console.aws.amazon.com/s3/.
2. Wählen Sie Create Bucket (Bucket erstellen) aus.
3. Geben Sie einen Bucket-Namen ein.
4. Lassen Sie unter Bucket-Einstellungen für „Öffentlichen Zugriff blockieren“ die Option Gesamten öffentlichen Zugriff blockieren aktiviert, um die Standardberechtigungen zu akzeptieren.
5. Wählen Sie unter Bucket-Versionierung die Option Aktivieren aus, um alle Versionen im selben Bucket zu behalten.
6. Wählen Sie Create Bucket (Bucket erstellen) aus.

Weitere Informationen zu Amazon S3 finden Sie im Amazon Simple Storage Service User Guide.

Erstellen einer OTA-Update-Servicerolle

Der OTA-Aktualisierungsservice übernimmt diese Rolle, um OTA-Aktualisierungsaufträge in Ihrem Namen zu erstellen und zu verwalten.

So erstellen Sie eine OTA-Service-Rolle:

1. Melden Sie sich an der https://console.aws.amazon.com/iam/ an.
2. Wählen Sie im Navigationsbereich Roles (Rollen) aus.
3. Wählen Sie Create role (Rolle erstellen) aus.
4. Wählen Sie unter Select type of trusted entity (Typ der vertrauenswürdigen Entität auswählen) die Option AWS Service aus.
5. Wählen Sie IoT aus der Liste der AWS Dienste aus.
6. Wählen Sie unter Select your use case (Anwendungsfall auswählen) IoT.

7. Wählen Sie Next: Permissions aus.
8. Wählen Sie Next: Tags (Weiter: Tags) aus.
9. Klicken Sie auf Weiter: Prüfen.
10. Geben Sie einen Rollennamen und eine Beschreibung ein und wählen Sie dann Create role (Rolle erstellen) aus.

Weitere Informationen zu IAM-Rollen finden Sie unter IAM-Rollen.

> **Important**
>
> Um das Sicherheitsproblem „Confused Deputy“ zu lösen, müssen Sie die Anweisungen im AWS IoT CoreLeitfaden befolgen.

So fügen Sie OTA-Update-Berechtigungen zu Ihrer OTA-Service-Rolle hinzu:

1. Geben Sie in das Suchfeld auf der Seite der IAM-Konsole den Namen Ihrer Rolle ein und wählen Sie sie dann aus der Liste aus.
2. Wählen Sie Richtlinien anfügen.
3. Geben Sie in das Suchfeld "AmazonFreeRTOSOTAUpdate„ein, wählen Sie AmazonFreeRTOSOTAUpdateaus der Liste der gefilterten Richtlinien aus und wählen Sie dann Richtlinie anhängen aus, um die Richtlinie an Ihre Servicerolle anzuhängen.

Um die erforderlichen IAM-Berechtigungen zu Ihrer OTA-Servicerolle hinzuzufügen

1. Geben Sie im Suchfeld auf der IAM-Konsolenseite den Namen Ihrer Rolle ein und wählen Sie ihn dann aus der Liste aus.
2. Wählen Sie Inline-Richtlinie hinzufügen.
3. Wählen Sie den Tab JSON.
4. Kopieren Sie das folgende Richtliniendokument und fügen Sie es in das Textfeld ein:

   JSON

```
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "iam:PassRole"
            ],
            "Resource": "arn:aws:iam::111122223333:role/your_role_name"
        }
    ]
}
```

Stellen Sie sicher, dass Sie ihn *your_account_id* durch Ihre AWS Konto-ID und *your_role_name* durch den Namen der OTA-Dienstrolle ersetzen.

5. Wählen Sie Richtlinie prüfen.
6. Geben Sie einen Namen für die Richtlinie ein und wählen Sie dann Create policy (Richtlinie erstellen) aus.

> **Note**
>
> Das folgende Verfahren ist nicht erforderlich, wenn Ihr Amazon S3 S3-Bucket-Name mit „afr-ota“ beginnt. Ist dies der Fall, enthält die AWS verwaltete Richtlinie `AmazonFreeRTOSOTAUpdate` bereits die erforderlichen Berechtigungen.

So fügen Sie Ihrer OTA-Servicerolle die erforderlichen Amazon S3 S3-Berechtigungen hinzu

1. Geben Sie im Suchfeld auf der IAM-Konsolenseite den Namen Ihrer Rolle ein und wählen Sie sie dann aus der Liste aus.
2. Wählen Sie Inline-Richtlinie hinzufügen.
3. Wählen Sie den Tab JSON.
4. Kopieren Sie das folgende Richtliniendokument und fügen Sie es in das Feld ein.

   JSON

```
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObjectVersion",
                "s3:GetObject",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::example-bucket/*"
            ]
        }
    ]
}
```

Diese Richtlinie gewährt Ihrer OTA-Servicerolle die Berechtigung, Amazon S3 S3-Objekte zu lesen. Stellen Sie sicher, dass Sie es *example-bucket* durch den Namen Ihres Buckets ersetzen.

5. Wählen Sie Richtlinie prüfen.
6. Geben Sie einen Namen für die Richtlinie ein und wählen Sie dann Create policy (Richtlinie erstellen) aus.

Erstellen einer OTA-Benutzerrichtlinie

Sie müssen Ihrem Benutzer die Erlaubnis erteilen, over-the-air Updates durchzuführen. Ihr -Benutzer muss über die folgenden Berechtigungen verfügen:

- Zugriff auf den S3-Bucket, in dem Ihre Firmware-Updates gespeichert sind
- Zugreifen auf Zertifikate, die in gespeichert sind AWS Certificate Manager.
- Greifen Sie auf die AWS IoT MQTT-basierte Dateizustellungsfunktion zu.
- Greifen Sie auf FreeRTOS OTA-Updates zu.
- Greifen Sie auf Jobs zu AWS IoT .
- Greifen Sie auf IAM zu.
- Zugriffscodesignatur für AWS IoT. Siehe Gewähren Sie Zugriff auf die Codesignatur für AWS IoT.
- Listet FreeRTOS-Hardwareplattformen auf.
- Ressourcen kennzeichnen und deren Markierung aufheben AWS IoT .

Informationen zum Erteilen der erforderlichen Berechtigungen finden Sie unter IAM-Richtlinien. Weitere Informationen finden Sie unter Autorisieren von Benutzern und Cloud-Diensten zur Nutzung AWS IoT von Jobs.

Um Zugriff zu gewähren, fügen Sie Ihren Benutzern, Gruppen oder Rollen Berechtigungen hinzu:

- Benutzer und Gruppen in AWS IAM Identity Center:

  Erstellen Sie einen Berechtigungssatz. Befolgen Sie die Anweisungen unter Erstellen eines Berechtigungssatzes im AWS IAM Identity Center -Benutzerhandbuch.
- Benutzer, die in IAM über einen Identitätsanbieter verwaltet werden:

  Erstellen Sie eine Rolle für den Identitätsverbund. Befolgen Sie die Anleitung unter Eine Rolle für einen externen Identitätsanbieter (Verbund) erstellen im IAM-Benutzerhandbuch.
- IAM-Benutzer:
  - Erstellen Sie eine Rolle, die Ihr Benutzer annehmen kann. Befolgen Sie die Anleitung unter Eine Rolle für einen IAM-Benutzer erstellen im IAM-Benutzerhandbuch.
  - (Nicht empfohlen) Weisen Sie einem Benutzer eine Richtlinie direkt zu oder fügen Sie einen Benutzer zu einer Benutzergruppe hinzu. Befolgen Sie die Anweisungen unter Hinzufügen von Berechtigungen zu einem Benutzer (Konsole) im IAM-Benutzerhandbuch.

Erstellen eines Zertifikats für die Codesignierung

Um Firmware-Images digital zu signieren, benötigen Sie ein Codesignierungszertifikat und einen privaten Schlüssel. Zu Testzwecken können Sie ein selbstsigniertes Zertifikat und einen privaten Schlüssel erstellen. Für Produktionsumgebungen sollten Sie ein Zertifikat von einer bekannten Zertifizierungsstelle (CA) erwerben.

Verschiedene Plattformen benötigen unterschiedliche Arten von Codesignierungszertifikaten. In den folgenden Abschnitten wird beschrieben, wie Codesignaturzertifikate für verschiedene FreeRTOS-qualifizierte Plattformen erstellt werden.

Themen

- Erstellen eines Codesignaturzertifikats für den SF-LAUNCHXL von Texas Instruments CC3220
- Erstellen eines Codesignaturzertifikats für den Espressif ESP32
- Erstellen eines Zertifikats für die Codesignierung für das Nordic nrf52840-dk
- Erstellen eines Codesignaturzertifikats für den FreeRTOS-Windows-Simulator

- Erstellen eines Zertifikats für die Codesignierung für benutzerdefinierte Hardware

Erstellen eines Codesignaturzertifikats für den SF-LAUNCHXL von Texas Instruments CC3220

> **Important**
>
> Diese Referenzintegration wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Das SimpleLink Wi-Fi CC3220 SF Wireless Microcontroller Launchpad Development Kit unterstützt zwei Zertifikatsketten für die Firmware-Codesignatur:

- Produktion (certificate-catalog)

  Um die Produktions-Zertifikatskette nutzen zu können, müssen Sie ein kommerzielles Codesignierungszertifikat erwerben und das TI Uniflash Tool verwenden, um das Board in den Produktionsmodus zu versetzen.
- Test und Entwicklung (certificate-playground)

  Die Playground-Zertifikatskette ermöglicht es Ihnen, OTA-Updates mit einem selbstsignierten Codesignaturzertifikat auszuprobieren.

Verwenden Sie das AWS Command Line Interface , um Ihr Codesignaturzertifikat, Ihren privaten Schlüssel und Ihre Zertifikatskette zu importieren. AWS Certificate Manager Weitere Informationen finden Sie unter Installation von AWS CLI im AWS Command Line Interface Benutzerhandbuch.

Laden Sie die neueste Version des SimpleLink CC3220 SDK herunter und installieren Sie sie. Standardmäßig befinden sich die benötigten Dateien an den folgenden Speicherorten:

`C:\ti\simplelink_cc32xx_sdk_`*`version`*`\tools\cc32xx_tools\certificate-playground` (Windows)

`/Applications/Ti/simplelink_cc32xx_`*`version`*`/tools/cc32xx_tools/certificate-playground` (macOS)

Die Zertifikate im SimpleLink CC3220 SDK sind im DER-Format. Um ein selbstsigniertes Codesignaturzertifikat zu erstellen, müssen Sie es in das PEM-Format konvertieren.

Gehen Sie wie folgt vor, um ein Codesignaturzertifikat zu erstellen, das mit der Playground-Zertifikatshierarchie von Texas Instruments verknüpft ist und die Code Signing-Kriterien erfüllt. AWS Certificate Manager AWS IoT

> **Note**
>
> Um ein Zertifikat für die Codesignierung erstellen zu können, müssen Sie OpenSSL auf Ihrem Gerät installieren. Stellen Sie nach der Installation von OpenSSL sicher, dass `openssl` der ausführbaren OpenSSL-Datei in Ihrer Eingabeaufforderung oder Terminal-Umgebung zugewiesen ist.

Um ein selbstsigniertes Codesignaturzertifikat zu erstellen

1. Öffnen Sie eine Eingabeaufforderung oder ein Terminal mit Administratorberechtigungen.
2. Verwenden Sie in Ihrem Arbeitsverzeichnis den folgenden Text, um eine Datei mit dem Namen `cert_config.txt` zu erstellen. Ersetze es *`test_signer@amazon.com`* durch deine E-Mail-Adresse.

   ```
   [ req ]
   prompt             = no
   distinguished_name = my dn

   [ my dn ]
   commonName = test_signer@amazon.com

   [ my_exts ]
   keyUsage         = digitalSignature
   extendedKeyUsage = codeSigning
   ```

3. Erstellen Sie einen privaten Schlüssel und eine Zertifikatssignierungsanforderung (Certificate Signing Request, CSR):

   ```
   openssl req -config cert_config.txt -extensions my_exts -nodes -days 365 -newkey
    rsa:2048 -keyout tisigner.key -out tisigner.csr
   ```

4. Konvertieren Sie den privaten Schlüssel der Stamm-CA von Texas Instruments-Playground aus dem DER-Format in das PEM-Format.

   Der private Schlüssel der Stamm-CA von TI-Playground befindet sich hier:

   `C:\ti\simplelink_cc32xx_sdk_version\tools\cc32xx_tools\certificate-playground\dummy-root-ca-cert-key` (Windows)

   `/Applications/Ti/simplelink_cc32xx_sdk_version/tools/cc32xx_tools/certificate-playground/dummy-root-ca-cert-key` (macOS)

   ```
   openssl rsa -inform DER -in dummy-root-ca-cert-key -out dummy-root-ca-cert-key.pem
   ```

5. Konvertieren Sie das Zertifikat der Stamm-CA von Texas Instruments-Playground vom DER-Format in das PEM-Format.

   Das TI-Playground Stammzertifikat befindet sich hier:

   `C:\ti\simplelink_cc32xx_sdk_version\tools\cc32xx_tools\certificate-playground/dummy-root-ca-cert` (Windows)

   `/Applications/Ti/simplelink_cc32xx_sdk_version/tools/cc32xx_tools/certificate-playground/dummy-root-ca-cert` (macOS)

   ```
   openssl x509 -inform DER -in dummy-root-ca-cert -out dummy-root-ca-cert.pem
   ```

6. Signieren Sie die CSR mit der Stamm-CA von Texas Instruments:

   ```
   openssl x509 -extfile cert_config.txt -extensions my_exts  -req -days 365 -in
    tisigner.csr -CA dummy-root-ca-cert.pem -CAkey dummy-root-ca-cert-key.pem -
   set_serial 01 -out tisigner.crt.pem -sha1
   ```

7. Konvertieren Sie Ihr Codesignierungszertifikat (`tisigner.crt.pem`) in das DER-Format:

   ```
   openssl x509 -in tisigner.crt.pem -out tisigner.crt.der -outform DER
   ```

   > **Note**
   >
   > Sie schreiben das `tisigner.crt.der`-Zertifikat später auf das TI-Entwicklungsboard.

8. Importieren Sie das Codesignaturzertifikat, den privaten Schlüssel und die Zertifikatskette in: AWS Certificate Manager

```
aws acm import-certificate --certificate fileb://tisigner.crt.pem --private-key
 fileb://tisigner.key --certificate-chain fileb://dummy-root-ca-cert.pem
```

Dieser Befehl zeigt einen ARN für Ihr Zertifikat an. Sie benötigen diesen ARN, wenn Sie einen OTA-Update-Job anlegen.

> **Note**
>
> Dieser Schritt wurde unter der Annahme geschrieben, dass Sie Code Signing AWS IoT zum Signieren Ihrer Firmware-Images verwenden werden. Obwohl die Verwendung von Code Signing for empfohlen AWS IoT wird, können Sie Ihre Firmware-Images auch manuell signieren.

Erstellen eines Codesignaturzertifikats für den Espressif ESP32

> **Important**
>
> Diese Referenzintegration wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Die ESP32 Espressif-Boards unterstützen ein selbstsigniertes SHA-256 mit ECDSA-Codesignaturzertifikat.

> **Note**
>
> Um ein Zertifikat für die Codesignierung erstellen zu können, müssen Sie OpenSSL auf Ihrem Gerät installieren. Stellen Sie nach der Installation von OpenSSL sicher, dass `openssl` der ausführbaren OpenSSL-Datei in Ihrer Eingabeaufforderung oder Terminal-Umgebung zugewiesen ist.

Verwenden Sie das AWS Command Line Interface , um Ihr Codesignaturzertifikat, Ihren privaten Schlüssel und Ihre Zertifikatskette zu importieren. AWS Certificate Manager Informationen zur Installation von finden Sie AWS CLI unter Installation von. AWS CLI

1. Verwenden Sie in Ihrem Arbeitsverzeichnis den folgenden Text, um eine Datei mit dem Namen `cert_config.txt` zu erstellen. *`test_signer@amazon.com`*Ersetze es durch deine E-Mail-Adresse:

```
[ req ]
prompt             = no
distinguished_name = my_dn

[ my_dn ]
commonName = test_signer@amazon.com

[ my_exts ]
keyUsage         = digitalSignature
extendedKeyUsage = codeSigning
```

2. Erstellen Sie einen privaten ECDSA-Code-Signaturschlüssel:

```
openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -pkeyopt
 ec_param_enc:named_curve -outform PEM -out ecdsasigner.key
```

3. Erstellen Sie ein ECDSA-Codesignierungszertifikat:

```
openssl req -new -x509 -config cert_config.txt -extensions my_exts -nodes -days 365
 -key ecdsasigner.key -out ecdsasigner.crt
```

4. Importieren Sie das Codesignaturzertifikat, den privaten Schlüssel und die Zertifikatskette in: AWS Certificate Manager

```
aws acm import-certificate --certificate fileb://ecdsasigner.crt --private-key
 fileb://ecdsasigner.key
```

Dieser Befehl zeigt einen ARN für Ihr Zertifikat an. Sie benötigen diesen ARN, wenn Sie einen OTA-Update-Job anlegen.

> **Note**
>
> Dieser Schritt wurde unter der Annahme geschrieben, dass Sie Code Signing AWS IoT zum Signieren Ihrer Firmware-Images verwenden werden. Obwohl die Verwendung von Code Signing for empfohlen AWS IoT wird, können Sie Ihre Firmware-Images auch manuell signieren.

Erstellen eines Zertifikats für die Codesignierung für das Nordic nrf52840-dk

> **Important**
>
> Diese Referenzintegration wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Das nordische nrf52840-dk unterstützt ein selbstsigniertes Zertifikat mit ECDSA-Codesignaturen. SHA256

> **Note**
>
> Um ein Zertifikat für die Codesignierung erstellen zu können, müssen Sie OpenSSL auf Ihrem Gerät installieren. Stellen Sie nach der Installation von OpenSSL sicher, dass `openssl` der ausführbaren OpenSSL-Datei in Ihrer Eingabeaufforderung oder Terminal-Umgebung zugewiesen ist.
> Verwenden Sie das, AWS Command Line Interface um Ihr Codesignaturzertifikat, Ihren privaten Schlüssel und Ihre Zertifikatskette zu importieren. AWS Certificate Manager Informationen zur Installation von finden Sie AWS CLI unter Installation von. AWS CLI

1. Verwenden Sie in Ihrem Arbeitsverzeichnis den folgenden Text, um eine Datei mit dem Namen `cert_config.txt` zu erstellen. *`test_signer@amazon.com`*Ersetze es durch deine E-Mail-Adresse:

```
[ req ]
prompt             = no
distinguished_name = my_dn

[ my_dn ]
commonName = test_signer@amazon.com

[ my_exts ]
keyUsage         = digitalSignature
extendedKeyUsage = codeSigning
```

2. Erstellen Sie einen privaten ECDSA-Code-Signaturschlüssel:

```
openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -pkeyopt
 ec_param_enc:named_curve -outform PEM -out ecdsasigner.key
```

3. Erstellen Sie ein ECDSA-Codesignierungszertifikat:

```
openssl req -new -x509 -config cert_config.txt -extensions my_exts -nodes -days 365
 -key ecdsasigner.key -out ecdsasigner.crt
```

4. Importieren Sie das Codesignaturzertifikat, den privaten Schlüssel und die Zertifikatskette in: AWS Certificate Manager

```
aws acm import-certificate --certificate fileb://ecdsasigner.crt --private-key
 fileb://ecdsasigner.key
```

Dieser Befehl zeigt einen ARN für Ihr Zertifikat an. Sie benötigen diesen ARN, wenn Sie einen OTA-Update-Job anlegen.

> **Note**
>
> Dieser Schritt wurde unter der Annahme geschrieben, dass Sie Code Signing AWS IoT zum Signieren Ihrer Firmware-Images verwenden werden. Obwohl die Verwendung von Code Signing for empfohlen AWS IoT wird, können Sie Ihre Firmware-Images auch manuell signieren.

Erstellen eines Codesignaturzertifikats für den FreeRTOS-Windows-Simulator

Der FreeRTOS Windows-Simulator benötigt ein Codesignaturzertifikat mit einem ECDSA P-256-Schlüssel und einem SHA-256-Hash, um OTA-Updates durchzuführen. Wenn Sie kein Codesignierungszertifikat besitzen, führen Sie die folgenden Schritte aus, um ein Codesignierungszertifikat zu erstellen.

> **Note**
>
> Um ein Zertifikat für die Codesignierung erstellen zu können, müssen Sie OpenSSL auf Ihrem Gerät installieren. Stellen Sie nach der Installation von OpenSSL sicher, dass `openssl` der ausführbaren OpenSSL-Datei in Ihrer Eingabeaufforderung oder Terminal-Umgebung zugewiesen ist.
> Verwenden Sie das AWS Command Line Interface , um Ihr Codesignaturzertifikat, Ihren privaten Schlüssel und Ihre Zertifikatskette in zu importieren. AWS Certificate Manager Informationen zur Installation von finden Sie AWS CLI unter Installation von. AWS CLI

1. Verwenden Sie in Ihrem Arbeitsverzeichnis den folgenden Text, um eine Datei mit dem Namen `cert_config.txt` zu erstellen. *`test_signer@amazon.com`*Ersetze es durch deine E-Mail-Adresse:

```
[ req ]
prompt             = no
distinguished_name = my_dn

[ my_dn ]
commonName = test_signer@amazon.com

[ my_exts ]
keyUsage         = digitalSignature
extendedKeyUsage = codeSigning
```

2. Erstellen Sie einen privaten ECDSA-Code-Signaturschlüssel:

```
openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -pkeyopt
 ec_param_enc:named_curve -outform PEM -out ecdsasigner.key
```

3. Erstellen Sie ein ECDSA-Codesignierungszertifikat:

```
openssl req -new -x509 -config cert_config.txt -extensions my_exts -nodes -days 365
 -key ecdsasigner.key -out ecdsasigner.crt
```

4. Importieren Sie das Codesignaturzertifikat, den privaten Schlüssel und die Zertifikatskette in: AWS Certificate Manager

```
aws acm import-certificate --certificate fileb://ecdsasigner.crt --private-key
 fileb://ecdsasigner.key
```

Dieser Befehl zeigt einen ARN für Ihr Zertifikat an. Sie benötigen diesen ARN, wenn Sie einen OTA-Update-Job anlegen.

> **Note**
>
> Dieser Schritt wurde unter der Annahme geschrieben, dass Sie Code Signing AWS IoT zum Signieren Ihrer Firmware-Images verwenden werden. Obwohl die Verwendung von Code Signing for empfohlen AWS IoT wird, können Sie Ihre Firmware-Images auch manuell signieren.

Erstellen eines Zertifikats für die Codesignierung für benutzerdefinierte Hardware

Erstellen Sie mit den geeigneten Tools ein selbstsigniertes Zertifikat und einen privaten Schlüssel für Ihre Hardware.

Verwenden Sie den AWS Command Line Interface , um Ihr Codesignaturzertifikat, Ihren privaten Schlüssel und Ihre Zertifikatskette in zu importieren. AWS Certificate Manager Informationen zur Installation von finden Sie AWS CLI unter Installation von. AWS CLI

Nachdem Sie Ihr Codesignaturzertifikat erstellt haben, können Sie es verwenden, um es in AWS CLI ACM zu importieren:

```
aws acm import-certificate --certificate fileb://code-sign.crt --private-key fileb://
code-sign.key
```

Die Ausgabe dieses Befehls zeigt einen ARN für Ihr Zertifikat an. Sie benötigen diesen ARN, wenn Sie einen OTA-Update-Job anlegen.

ACM erfordert, dass Zertifikate bestimmte Algorithmen und Schlüsselgrößen verwenden. Weitere Informationen finden Sie unter Voraussetzungen für den Import von Zertifikaten. Weitere Informationen zu ACM finden Sie unter Zertifikate importieren in. AWS Certificate Manager

Sie müssen den Inhalt Ihres Codesignaturzertifikats in die `vendors/vendor/boards/board/aws_demos/config_files/ota_demo_config.h` Datei kopieren, einfügen und formatieren, die Teil des FreeRTOS-Codes ist, den Sie später herunterladen.

Gewähren Sie Zugriff auf die Codesignatur für AWS IoT

Um Zugriff zu gewähren, fügen Sie Ihren Benutzern, Gruppen oder Rollen Berechtigungen hinzu:

- Benutzer und Gruppen in AWS IAM Identity Center:

  Erstellen Sie einen Berechtigungssatz. Befolgen Sie die Anweisungen unter Erstellen eines Berechtigungssatzes im AWS IAM Identity Center -Benutzerhandbuch.
- Benutzer, die in IAM über einen Identitätsanbieter verwaltet werden:

  Erstellen Sie eine Rolle für den Identitätsverbund. Befolgen Sie die Anleitung unter Eine Rolle für einen externen Identitätsanbieter (Verbund) erstellen im IAM-Benutzerhandbuch.
- IAM-Benutzer:
  - Erstellen Sie eine Rolle, die Ihr Benutzer annehmen kann. Befolgen Sie die Anleitung unter Eine Rolle für einen IAM-Benutzer erstellen im IAM-Benutzerhandbuch.
  - (Nicht empfohlen) Weisen Sie einem Benutzer eine Richtlinie direkt zu oder fügen Sie einen Benutzer zu einer Benutzergruppe hinzu. Befolgen Sie die Anweisungen unter Hinzufügen von Berechtigungen zu einem Benutzer (Konsole) im IAM-Benutzerhandbuch.

Laden Sie FreeRTOS mit der OTA-Bibliothek herunter

Sie können FreeRTOS von klonen oder herunterladen. GitHub Anweisungen finden Sie in der Datei README.md.

Informationen zum Einrichten und Ausführen der OTA-Demoanwendung finden Sie unter Over-the-air aktualisiert die Demo-Anwendung.

> **Important**
>
> - In diesem Thema wird der Pfad zum FreeRTOS-Download-Verzeichnis als bezeichnet. *freertos*

- Leerzeichen im *freertos*-Pfad können Build-Fehler verursachen. Stellen Sie beim Klonen oder Kopieren des Repositorys sicher, dass der Pfad, den Sie erstellen, keine Leerzeichen enthält.
- Die maximale Länge eines Dateipfades bei Microsoft Windows ist 260 Zeichen. Lange FreeRTOS-Download-Verzeichnispfade können zu Build-Fehlern führen.
- Da der Quellcode symbolische Links enthalten kann, müssen Sie, wenn Sie Windows zum Extrahieren des Archivs verwenden, möglicherweise:
  - Entwicklermodus aktivieren oder
  - Verwenden Sie eine Konsole mit Administratorberechtigungen.

  Auf diese Weise kann Windows beim Extrahieren des Archivs ordnungsgemäß symbolische Links erstellen. Andernfalls werden symbolische Links als normale Dateien geschrieben, die die Pfade der symbolischen Links als Text enthalten oder leer sind. Weitere Informationen finden Sie im Blogeintrag Symlinks in Windows 10! .

  Wenn Sie Git unter Windows verwenden, müssen Sie den Entwicklermodus aktivieren oder Sie müssen:

  - Setzen `core.symlinks` Sie ihn mit dem folgenden Befehl auf true:

    ```
    git config --global core.symlinks true
    ```

  - Verwenden Sie immer dann eine Konsole mit Administratorrechten, wenn Sie einen Git-Befehl verwenden, der in das System schreibt (z. B.git pull,git clone, undgit submodule update --init --recursive).

Voraussetzungen für OTA-Updates mit MQTT

In diesem Abschnitt werden die allgemeinen Anforderungen für die Verwendung von MQTT zur Ausführung over-the-air (OTA-Updates) beschrieben.

Mindestanforderungen

- Die Gerätefirmware muss die erforderlichen FreeRTOS-Bibliotheken (CoreMQTT-Agent, OTA-Update und deren Abhängigkeiten) enthalten.
- FreeRTOS Version 1.4.0 oder höher ist erforderlich. Wir empfehlen jedoch, wenn möglich die neueste Version zu verwenden.

Konfigurationen

Ab Version 201912.00 kann FreeRTOS OTA entweder das HTTP- oder das MQTT-Protokoll verwenden, um Firmware-Update-Images von auf Geräte zu übertragen. AWS IoT Wenn Sie beide Protokolle angeben, wenn Sie ein OTA-Update in FreeRTOS erstellen, bestimmt jedes Gerät das Protokoll, das für die Übertragung des Images verwendet wird. Weitere Informationen finden Sie unter Voraussetzungen für OTA-Updates mit HTTP.

Standardmäßig verwendet die Konfiguration der OTA-Protokolle in `ota_config.h`das MQTT-Protokoll.

Gerätespezifische Konfigurationen

Keine.

Speicherauslastung

Wenn MQTT für die Datenübertragung verwendet wird, ist kein zusätzlicher Speicher für die MQTT-Verbindung erforderlich, da sie von Steuerungs- und Datenvorgängen gemeinsam genutzt wird.

Geräterichtlinie

Jedes Gerät, das mithilfe von MQTT ein OTA-Update erhält, muss als Ding in registriert sein AWS IoT und dem Ding muss eine Richtlinie wie die hier aufgeführte beigefügt sein. Weitere Informationen zu den Elementen finden Sie unter den `"Action"`- und `"Resource"`-Objekten der AWS IoT Core-Richtlinienaktionen und AWS IoT Core-Aktionsressourcen.

Hinweise

- Die `iot:Connect` Berechtigungen ermöglichen es Ihrem Gerät, eine Verbindung AWS IoT über MQTT herzustellen.
- Die `iot:Subscribe` `iot:Publish` Berechtigungen zu den Themen AWS IoT Jobs (`.../jobs/*`) ermöglichen es dem verbundenen Gerät, Job-Benachrichtigungen und Job-Dokumente zu empfangen und den Abschlussstatus einer Job-Ausführung zu veröffentlichen.
- Die `iot:Publish` Berechtigungen `iot:Subscribe` und zu den Themen AWS IoT OTA-Streams (`.../streams/*`) ermöglichen es dem verbundenen Gerät, OTA-Aktualisierungsdaten von AWS IoT abzurufen. Diese Berechtigungen sind zum Ausführen von Firmware-Updates über MQTT erforderlich.
- Die `iot:Receive` Berechtigungen ermöglichen es AWS IoT Core , Nachrichten zu diesen Themen auf dem angeschlossenen Gerät zu veröffentlichen. Diese Berechtigung wird bei jeder

Zustellung einer MQTT-Nachricht überprüft. Sie können diese Berechtigung verwenden, um den Zugriff auf Clients zu widerrufen, die derzeit ein Thema abonniert haben.

Voraussetzungen für OTA-Updates mit HTTP

In diesem Abschnitt werden die allgemeinen Anforderungen für die Verwendung von HTTP zur Durchführung von over-the-air (OTA-) Updates beschrieben. Ab Version 201912.00 kann FreeRTOS OTA entweder das HTTP- oder das MQTT-Protokoll verwenden, um Firmware-Update-Images von auf Geräte zu übertragen. AWS IoT

> **Note**
>
> - Obwohl das HTTP-Protokoll für die Übertragung des Firmware-Images verwendet werden kann, ist die CoreMQTT-Agentenbibliothek dennoch erforderlich, da andere Interaktionen die CoreMQTT-Agentenbibliothek AWS IoT Core verwenden, einschließlich des Sendens oder Empfangens von Benachrichtigungen zur Jobausführung, Jobdokumenten und Ausführungsstatus-Updates.
> - Wenn Sie für den OTA-Aktualisierungsauftrag sowohl das MQTT- als auch das HTTP-Protokoll festlegen, bestimmt das Setup der OTA-Agent-Software auf jedem einzelnen Gerät, welches Protokoll für die Übertragung des Firmware-Images verwendet wird. Um für den OTA-Agent von der standardmäßigen MQTT-Protokollmethode zum HTTP-Protokoll zu wechseln, können Sie die Header-Dateien ändern, die zum Kompilieren des FreeRTOS-Quellcodes für das Gerät verwendet werden.

Mindestanforderungen

- Die Gerätefirmware muss die erforderlichen FreeRTOS-Bibliotheken (CoreMQTT Agent, HTTP, OTA Agent und deren Abhängigkeiten) enthalten.
- FreeRTOS Version 201912.00 oder höher ist erforderlich, um die Konfiguration der OTA-Protokolle zu ändern, um die OTA-Datenübertragung über HTTP zu ermöglichen.

Konfigurationen

In der Datei `\vendors\boards\board\aws_demos\config_files\ota_config.h` finden Sie die folgende Konfiguration der OTA-Protokolle.

So aktivieren Sie die OTA-Datenübertragung über HTTP

1. Ändern Sie `configENABLED_DATA_PROTOCOLS` zu `OTA_DATA_OVER_HTTP`.
2. Bei OTA-Updates können Sie beide Protokolle angeben, so dass als Protokoll entweder MQTT oder HTTP verwendet werden kann. Sie können als primäres Protokoll für das Gerät HTTP festlegen, indem Sie `configOTA_PRIMARY_DATA_PROTOCOL` in `OTA_DATA_OVER_HTTP` ändern.

> **Note**
>
> HTTP wird nur für OTA-Datenvorgänge unterstützt. Für Steuerungsvorgänge müssen Sie MQTT verwenden.

Gerätespezifische Konfigurationen

ESP32

Aufgrund einer begrenzten Menge an RAM müssen Sie BLE deaktivieren, wenn Sie HTTP als OTA-Datenprotokoll aktivieren. Ändern Sie in der Datei vendors/espressif/boards/esp32/aws_demos/config_files/aws_iot_network_config.h nur `configENABLED_NETWORKS` zu `AWSIOT_NETWORK_TYPE_WIFI`.

```
/**
     * @brief Configuration flag which is used to enable one or more network
 interfaces for a board.
     *
     * The configuration can be changed any time to keep one or more network enabled
 or disabled.
     * More than one network interfaces can be enabled by using 'OR' operation with
 flags for
     * each network types supported. Flags for all supported network types can be
 found
     * in "aws_iot_network.h"
     *
     */
   #define configENABLED_NETWORKS      ( AWSIOT_NETWORK_TYPE_WIFI )
```

Speicherauslastung

Wenn MQTT für die Datenübertragung verwendet wird, ist kein zusätzlicher Heap-Speicher für die MQTT-Verbindung erforderlich, da sie von Steuerungs- und Datenvorgängen gemeinsam genutzt wird. Das Aktivieren von Daten über HTTP erfordert jedoch zusätzlichen Heap-Speicher. Im Folgenden finden Sie die Daten zur Heap-Speichernutzung für alle unterstützten Plattformen, die mit der `xPortGetFreeHeapSize` FreeRTOS-API berechnet wurden. Sie müssen sicherstellen, dass genügend RAM zur Verwendung der OTA-Bibliothek vorhanden ist.

Texas Instruments SF-LAUNCHXL CC3220

- Steuerungsvorgänge (MQTT): 12 KB
- Datenvorgänge (HTTP): 10 KB

> **Note**
>
> TI verbraucht deutlich weniger RAM, da es SSL auf Hardware ausführt und somit die mbedtls-Bibliothek nicht verwendet.

Mikrochip PIC32 Curiosity MZEF

- Verwaltungsvorgänge (MQTT): 65 KB
- Datenvorgänge (HTTP): 43 KB

Espressif ESP32

- Verwaltungsvorgänge (MQTT): 65 KB
- Datenvorgänge (HTTP): 45 KB

> **Note**
>
> BLE on ESP32 benötigt etwa 87 KB RAM. Das RAM reicht nicht aus, um alle von ihnen zu aktivieren, was in den gerätespezifischen Konfigurationen oben erwähnt wird.

Windows Simulator

- Verwaltungsvorgänge (MQTT): 82 KB

Datenvorgänge (HTTP): 63 KB

Nordic nrf52840-dk

HTTP wird nicht unterstützt.

Geräterichtlinie

Mit dieser Richtlinie können Sie entweder MQTT oder HTTP für OTA-Updates verwenden.

Jedes Gerät, das ein OTA-Update über HTTP erhält, muss als Objekt in AWS IoT registriert sein, und das Objekt muss über eine angehängte Richtlinie wie die hier aufgelistete verfügen. Weitere Informationen zu den Elementen finden Sie unter den `"Action"`- und `"Resource"`-Objekten der AWS IoT Core-Richtlinienaktionen und AWS IoT Core-Aktionsressourcen.

Hinweise

- Mit den `iot:Connect`-Berechtigungen kann Ihr Gerät über MQTT eine Verbindung mit AWS IoT herstellen.
- Die `iot:Publish` Berechtigungen `iot:Subscribe` und zu den Themen AWS IoT Jobs (`.../jobs/*`) ermöglichen es dem verbundenen Gerät, Auftragsbenachrichtigungen und Auftragsdokumente zu empfangen und den Abschlussstatus einer Auftragsausführung zu veröffentlichen.
- Die `iot:Receive` Berechtigungen ermöglichen es AWS IoT Core , Nachrichten zu diesen Themen auf dem aktuell verbundenen Gerät zu veröffentlichen. Diese Berechtigung wird bei jeder Zustellung einer MQTT-Nachricht überprüft. Sie können diese Berechtigung verwenden, um den Zugriff auf Clients zu widerrufen, die derzeit ein Thema abonniert haben.

OTA-Tutorial

Dieser Abschnitt enthält ein Tutorial zum Aktualisieren der Firmware auf Geräten, auf denen FreeRTOS ausgeführt wird, mithilfe von OTA-Updates. Zusätzlich zu Firmware-Images können Sie ein OTA-Update verwenden, um jede Art von Datei an ein Gerät zu senden, das mit AWS IoT verbunden ist.

Sie können die AWS IoT Konsole oder die verwenden, um ein OTA-Update AWS CLI zu erstellen. Die Konsole ist der einfachste Weg für den Einstig in OTA. Sie erledigt viel Arbeit für Sie. Dies AWS CLI ist nützlich, wenn Sie OTA-Aktualisierungsjobs automatisieren, mit einer großen Anzahl von Geräten arbeiten oder Geräte verwenden, die nicht für FreeRTOS qualifiziert sind. Weitere

Informationen zu Geräten, die für FreeRTOS in Frage kommen, finden Sie auf der Website der FreeRTOS-Partner.

So erstellen Sie einen OTA-Update

1. Stellen Sie eine erste Version Ihrer Firmware auf einem oder mehreren Geräten bereit.
2. Überprüfen Sie, ob die Firmware korrekt läuft.
3. Wenn ein Firmware-Update erforderlich ist, nehmen Sie die Code-Änderungen vor und erstellen Sie das neue Image.
4. Wenn Sie Ihre Firmware manuell signieren, signieren Sie das signierte Firmware-Image und laden Sie es dann in Ihren Amazon S3 S3-Bucket hoch. Wenn Sie Code Signing for verwenden AWS IoT, laden Sie Ihr unsigniertes Firmware-Image in einen Amazon S3 S3-Bucket hoch.
5. Erstellen Sie ein OTA-Update.

Wenn Sie ein OTA-Update erstellen, geben Sie das Image-Delivery Protocol (MQTT oder HTTP oder beides) an, um dem Gerät die Auswahl zu überlassen. Der FreeRTOS OTA-Agent auf dem Gerät empfängt das aktualisierte Firmware-Image und verifiziert die digitale Signatur, Prüfsumme und Versionsnummer des neuen Images. Wenn das Firmware-Update verifiziert wird, wird das Gerät zurückgesetzt. Dann wird das Update auf Basis einer von der Anwendung definierten Logik angewendet. Wenn auf Ihren Geräten FreeRTOS nicht ausgeführt wird, müssen Sie einen OTA-Agenten implementieren, der auf Ihren Geräten ausgeführt wird.

Installieren der ersten Firmware

Um die Firmware zu aktualisieren, müssen Sie eine erste Version der Firmware installieren, die die OTA-Agent-Bibliothek zum Empfang von OTA-Update-Aufgaben verwendet. Wenn Sie FreeRTOS nicht ausführen, überspringen Sie diesen Schritt. Sie müssen stattdessen Ihre OTA-Agent-Implementierung auf Ihre Geräte kopieren.

Themen

- Installieren Sie die erste Version der Firmware auf dem Texas Instruments SF-LAUNCHXL CC3220
- Installieren Sie die erste Version der Firmware auf dem Espressif ESP32
- Installieren Sie die erste Version der Firmware auf dem Nordic n RF52840 DK
- Erste Firmware auf dem Windows Simulator
- Installieren der ersten Version der Firmware auf einem benutzerdefinierten Board

Installieren Sie die erste Version der Firmware auf dem Texas Instruments SF-LAUNCHXL CC3220

> **Important**
>
> Diese Referenzintegration wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Diese Schritte gehen davon aus, dass Sie das Projekt `aws_demos`, wie unter Laden Sie die FreeRTOS OTA-Demo auf dem Texas Instruments SF-LAUNCHXL herunter, erstellen Sie sie, flashen Sie sie und führen Sie sie aus CC3220 beschrieben, bereits erstellt haben.

1. Platzieren Sie auf Ihrem Texas Instruments CC3220 SF-LAUNCHXL den SOP-Jumper auf dem mittleren Satz von Pins (Position = 1) und setzen Sie die Platine zurück.
2. Laden Sie das TI Uniflash Tool herunter und installieren Sie es.
3. Starten Sie Uniflash. Wählen Sie aus der Liste der Konfigurationen CC3220SF-LAUNCHXL und anschließend Start Image Creator aus.
4. Wählen Sie New Project (Neues Projekt) aus.
5. Geben Sie auf der Seite Start new project (Neues Projekt starten) einen Namen für Ihr Projekt ein. Wählen Sie als Gerätetyp die Option SF aus. CC3220 Wählen Sie für Device Mode (Gerätemodus) die Option Develop (Entwicklung) aus. Wählen Sie Projekt erstellen aus.
6. Trennen Sie Ihren Terminalemulator.
7. Wählen Sie auf der rechten Seite des Uniflash-Anwendungsfensters Connect (Verbinden) aus.
8. Wählen Sie unter Advanced (Erweitert) Files (Dateien) die Option User Files (Benutzerdateien) aus.
9. Wählen Sie im Auswahlbereich File (Datei) das Symbol Add File (Datei hinzufügen)

   aus.
10. Wechseln Sie zum Verzeichnis `/Applications/Ti/simplelink_cc32xx_sdk_`*`version`*`/tools/cc32xx_tools/certificate-playground` und wählen Sie `dummy-root-ca-cert`, Open (Öffnen) und dann Write (Schreiben) aus.

11. Wählen Sie im Auswahlbereich File (Datei) das Symbol Add File (Datei hinzufügen)

    aus.

12. Wechseln Sie zu dem Arbeitsverzeichnis, in dem Sie das Codesignierungszertifikat und den privaten Schlüssel erstellt haben, wählen Sie `tisigner.crt.der` aus, wählen Sie Open (Öffnen) aus und wählen Sie dann Write (Schreiben) aus.
13. Wählen Sie aus der Dropdown-Liste Action (Aktion) die Option Select MCU Image (MCU-Image auswählen) aus und wählen Sie dann Browse (Durchsuchen) aus, um das Firmware-Image auszuwählen, das Sie auf Ihr Gerät schreiben möchten (aws_demos.bin). Diese Datei befindet sich im Verzeichnis *`freertos`*`/vendors/ti/boards/cc3220_launchpad/aws_demos/ccs/Debug`. Klicken Sie auf Open.

    a. Überprüfen Sie im Dialogfeld „Datei“, ob der Dateiname auf `mcuflashimg.bin` festgelegt ist.

    b. Aktivieren Sie das Kontrollkästchen Vendor (Anbieter) aus.

    c. Geben Sie unter File Token (Datei-Token) **`1952007250`** ein.

    d. Wählen Sie in Private Key File Name (Dateiname für den privaten Schlüssel) die Option Browse (Durchsuchen) und anschließend in dem Arbeitsverzeichnis, in dem Sie das Codesignierungszertifikat und den privaten Schlüssel erstellt haben, `tisigner.key` aus.

    e. Wählen Sie unter Certification File Name (Zertifizierungsdateiname) die Option `tisigner.crt.der` aus.

    f. Wählen Sie Write (Schreiben) aus.

14. Wählen Sie im linken Bereich unter Files (Dateien) die Option Service Pack aus.
15. Wählen Sie unter Service Pack File Name (Service-Pack-Dateiname) die Option Browse (Durchsuchen) aus, navigieren Sie zu `simplelink_cc32x_sdk_`*`version`*`/tools/cc32xx_tools/servicepack-cc3x20` und wählen Sie `sp_3.7.0.1_2.0.0.0_2.2.0.6.bin` und dann Open (Öffnen) aus.
16. Wählen Sie im linken Bereich unter Files (Dateien) die Option Trusted Root-Certificate Catalog (Katalog für vertrauenswürdige Stammzertifikate) aus.
17. Deaktivieren Sie das Kontrollkästchen Use default Trusted Root-Certificate Catalog (Standardmäßigen Katalog für vertrauenswürdige Stammzertifikate verwenden) .
18. Wählen Sie unter Quelldatei die Option Durchsuchen, wählen Sie ***`version`***simplelink_cc32xx_sdk_/20160911.lst und wählen Sie dann Öffnen aus. tools/cc32xx_tools/certificate-playground/certcatalogPlayGround

19. Wählen Sie unter Signaturquelldatei die Option Durchsuchen, wählen Sie simplelink_cc32xx_sdk_/20160911.lst.signed_3220.bin und wählen Sie dann Öffnen aus. ***version*** tools/cc32xx_tools/certificate-playground/certcatalogPlayGround
20. Wählen Sie die Schaltfläche

    aus, um Ihr Projekt zu speichern.
21. Klicken Sie auf die Schaltfläche

    .
22. Wählen Sie Program Image (Create and Program) (Image programmieren (Erstellen und Programmieren)) aus.
23. Nachdem der Programmiervorgang abgeschlossen ist, setzen Sie den SOP-Jumper auf die erste Position (Position = 0) und setzen Sie das Board zurück. Verbinden Sie Ihren Terminalemulator wieder, um sicherzustellen, dass die Ausgabe mit der beim Debuggen der Demo mit Code Composer Studio identisch ist. Notieren Sie sich die Versionsnummer der Anwendung in der Terminalausgabe. Sie verwenden diese Versionsnummer später, um sicherzustellen, dass Ihre Firmware durch ein OTA-Update aktualisiert wurde.

    Das Terminal sollte die folgende Ausgabe anzeigen:

```
0 0 [Tmr Svc] Simple Link task created

Device came up in Station mode

1 369 [Tmr Svc] Starting key provisioning...
2 369 [Tmr Svc] Write root certificate...
3 467 [Tmr Svc] Write device private key...
4 568 [Tmr Svc] Write device certificate...
SL Disconnect...

5 664 [Tmr Svc] Key provisioning done...
Device came up in Station mode

Device disconnected from the AP on an ERROR..!!

[WLAN EVENT] STA Connected to the AP: Guest , BSSID: 11:22:a1:b2:c3:d4

[NETAPP EVENT] IP acquired by the device
```

```
Device has connected to Guest

Device IP Address is 111.222.3.44


6 1716 [OTA] OTA demo version 0.9.0
7 1717 [OTA] Creating MQTT Client...
8 1717 [OTA] Connecting to broker...
9 1717 [OTA] Sending command to MQTT task.
10 1717 [MQTT] Received message 10000 from queue.
11 2193 [MQTT] MQTT Connect was accepted. Connection established.
12 2193 [MQTT] Notifying task.
13 2194 [OTA] Command sent to MQTT task passed.
14 2194 [OTA] Connected to broker.
15 2196 [OTA Task] Sending command to MQTT task.
16 2196 [MQTT] Received message 20000 from queue.
17 2697 [MQTT] MQTT Subscribe was accepted. Subscribed.
18 2697 [MQTT] Notifying task.
19 2698 [OTA Task] Command sent to MQTT task passed.
20 2698 [OTA Task] [OTA] Subscribed to topic: $aws/things/TI-LaunchPad/jobs/$next/
get/accepted

21 2699 [OTA Task] Sending command to MQTT task.
22 2699 [MQTT] Received message 30000 from queue.
23 2800 [MQTT] MQTT Subscribe was accepted. Subscribed.
24 2800 [MQTT] Notifying task.
25 2801 [OTA Task] Command sent to MQTT task passed.
26 2801 [OTA Task] [OTA] Subscribed to topic: $aws/things/TI-LaunchPad/jobs/notify-
next

27 2814 [OTA Task] [OTA] Check For Update #0
28 2814 [OTA Task] Sending command to MQTT task.
29 2814 [MQTT] Received message 40000 from queue.
30 2916 [MQTT] MQTT Publish was successful.
31 2916 [MQTT] Notifying task.
32 2917 [OTA Task] Command sent to MQTT task passed.
33 2917 [OTA Task] [OTA] Set job doc parameter [ clientToken: 0:TI-LaunchPad ]
34 2917 [OTA Task] [OTA] Missing job parameter: execution
35 2917 [OTA Task] [OTA] Missing job parameter: jobId
36 2918 [OTA Task] [OTA] Missing job parameter: jobDocument
37 2918 [OTA Task] [OTA] Missing job parameter: ts_ota
38 2918 [OTA Task] [OTA] Missing job parameter: files
```

```
39 2918 [OTA Task] [OTA] Missing job parameter: streamname
40 2918 [OTA Task] [OTA] Missing job parameter: certfile
41 2918 [OTA Task] [OTA] Missing job parameter: filepath
42 2918 [OTA Task] [OTA] Missing job parameter: filesize
43 2919 [OTA Task] [OTA] Missing job parameter: sig-sha1-rsa
44 2919 [OTA Task] [OTA] Missing job parameter: fileid
45 2919 [OTA Task] [OTA] Missing job parameter: attr
47 3919 [OTA] [OTA] Queued: 1   Processed: 1   Dropped: 0
48 4919 [OTA] [OTA] Queued: 1   Processed: 1   Dropped: 0
49 5919 [OTA] [OTA] Queued: 1   Processed: 1   Dropped: 0
```

Installieren Sie die erste Version der Firmware auf dem Espressif ESP32

> **Important**
>
> Diese Referenzintegration wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Bei der Erstellung dieses Handbuchs wurde davon ausgegangen, dass Sie die Schritte unter Erste Schritte mit ESP32 Espressif — C und den Voraussetzungen für Updates bereits ausgeführt haben. DevKit ESP-WROVER-KIT Over-the-Air Bevor Sie versuchen, ein OTA-Update durchzuführen, sollten Sie das unter Erste Schritte mit FreeRTOS beschriebene MQTT-Demoprojekt ausführen, um sicherzustellen, dass Ihr Board und Ihre Toolchain korrekt eingerichtet sind.

So flashen Sie ein erstes Fabric-Image auf das Board:

1. Öffne*freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h, kommentiere und #define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED definiere oder. CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED
2. Kopieren Sie das SHA-256/ECDSA PEM-formatierte Codesignierungszertifikat, das Sie in Voraussetzungen für OTA-Updates erstellt haben, zu vendors/*vendor*/boards/*board*/aws_demos/config_files/ota_demo_config.h. Es sollte folgendermaßen formatiert sein:

   ```
   #define otapalconfigCODE_SIGNING_CERTIFICATE  \
   ```

```
"-----BEGIN CERTIFICATE-----\n" \
"...base64 data...\n" \
"-----END CERTIFICATE-----\n";
```

3. Wenn die OTA-Update-Demo ausgewählt ist, folgen Sie den unter Erste Schritte mit beschriebenen Schritten, um das Image ESP32 zu erstellen und zu flashen. Wenn Sie das Projekt zuvor erstellt und geflasht haben, müssen Sie möglicherweise zuerst `make clean` ausführen. Nachdem Sie `make flash monitor` ausgeführt haben, sollten Sie eine Ausgabe ähnlich der folgenden sehen. Die Reihenfolge einiger Nachrichten kann variieren, da die Demoanwendung mehrere Aufgaben gleichzeitig ausführt:

```
I (28) boot: ESP-IDF v3.1-dev-322-gf307f41-dirty 2nd stage bootloader
I (28) boot: compile time 16:32:33
I (29) boot: Enabling RNG early entropy source...
I (34) boot: SPI Speed : 40MHz
I (38) boot: SPI Mode : DIO
I (42) boot: SPI Flash Size : 4MB
I (46) boot: Partition Table:
I (50) boot: ## Label Usage Type ST Offset Length
I (57) boot: 0 nvs WiFi data 01 02 00010000 00006000
I (64) boot: 1 otadata OTA data 01 00 00016000 00002000
I (72) boot: 2 phy_init RF data 01 01 00018000 00001000
I (79) boot: 3 ota_0 OTA app 00 10 00020000 00100000
I (87) boot: 4 ota_1 OTA app 00 11 00120000 00100000
I (94) boot: 5 storage Unknown data 01 82 00220000 00010000
I (102) boot: End of partition table
I (106) esp_image: segment 0: paddr=0x00020020 vaddr=0x3f400020 size=0x14784
 ( 83844) map
I (144) esp_image: segment 1: paddr=0x000347ac vaddr=0x3ffb0000 size=0x023ec
 ( 9196) load
I (148) esp_image: segment 2: paddr=0x00036ba0 vaddr=0x40080000 size=0x00400
 ( 1024) load
I (151) esp_image: segment 3: paddr=0x00036fa8 vaddr=0x40080400 size=0x09068
 ( 36968) load
I (175) esp_image: segment 4: paddr=0x00040018 vaddr=0x400d0018 size=0x719b8
 (465336) map
I (337) esp_image: segment 5: paddr=0x000b19d8 vaddr=0x40089468 size=0x04934
 ( 18740) load
I (345) esp_image: segment 6: paddr=0x000b6314 vaddr=0x400c0000 size=0x00000 ( 0)
 load
I (353) boot: Loaded app from partition at offset 0x20000
I (353) boot: ota rollback check done
I (354) boot: Disabling RNG early entropy source...
```

```
I (360) cpu_start: Pro cpu up.
I (363) cpu_start: Single core mode
I (368) heap_init: Initializing. RAM available for dynamic allocation:
I (375) heap_init: At 3FFAE6E0 len 00001920 (6 KiB): DRAM
I (381) heap_init: At 3FFC0748 len 0001F8B8 (126 KiB): DRAM
I (387) heap_init: At 3FFE0440 len 00003BC0 (14 KiB): D/IRAM
I (393) heap_init: At 3FFE4350 len 0001BCB0 (111 KiB): D/IRAM
I (400) heap_init: At 4008DD9C len 00012264 (72 KiB): IRAM
I (406) cpu_start: Pro cpu start user code
I (88) cpu_start: Starting scheduler on PRO CPU.
I (113) wifi: wifi firmware version: f79168c
I (113) wifi: config NVS flash: enabled
I (113) wifi: config nano formating: disabled
I (113) system_api: Base MAC address is not set, read default base MAC address from
 BLK0 of EFUSE
I (123) system_api: Base MAC address is not set, read default base MAC address from
 BLK0 of EFUSE
I (133) wifi: Init dynamic tx buffer num: 32
I (143) wifi: Init data frame dynamic rx buffer num: 32
I (143) wifi: Init management frame dynamic rx buffer num: 32
I (143) wifi: wifi driver task: 3ffc73ec, prio:23, stack:4096
I (153) wifi: Init static rx buffer num: 10
I (153) wifi: Init dynamic rx buffer num: 32
I (163) wifi: wifi power manager task: 0x3ffcc028 prio: 21 stack: 2560
0 6 [main] WiFi module initialized. Connecting to AP <Your_WiFi_SSID>...
I (233) phy: phy_version: 383.0, 79a622c, Jan 30 2018, 15:38:06, 0, 0
I (233) wifi: mode : sta (30:ae:a4:80:0a:04)
I (233) WIFI: SYSTEM_EVENT_STA_START
I (363) wifi: n:1 0, o:1 0, ap:255 255, sta:1 0, prof:1
I (1343) wifi: state: init -> auth (b0)
I (1343) wifi: state: auth -> assoc (0)
I (1353) wifi: state: assoc -> run (10)
I (1373) wifi: connected with <Your_WiFi_SSID>, channel 1
I (1373) WIFI: SYSTEM_EVENT_STA_CONNECTED
1 302 [IP-task] vDHCPProcess: offer c0a86c13ip
I (3123) event: sta ip: 192.168.108.19, mask: 255.255.224.0, gw: 192.168.96.1
I (3123) WIFI: SYSTEM_EVENT_STA_GOT_IP
2 302 [IP-task] vDHCPProcess: offer c0a86c13ip
3 303 [main] WiFi Connected to AP. Creating tasks which use network...
4 304 [OTA] OTA demo version 0.9.6
5 304 [OTA] Creating MQTT Client...
6 304 [OTA] Connecting to broker...
I (4353) wifi: pm start, type:0
```

```
I (8173) PKCS11: Initializing SPIFFS
I (8183) PKCS11: Partition size: total: 52961, used: 0
7 1277 [OTA] Connected to broker.
8 1280 [OTA Task] [prvSubscribeToJobNotificationTopics] OK: $aws/things/
<Your_Thing_Name>/jobs/$next/get/accepted
I (12963) ota_pal: prvPAL_GetPlatformImageState
I (12963) esp_ota_ops: [0] aflags/seq:0x2/0x1, pflags/seq:0xffffffff/0x0
9 1285 [OTA Task] [prvSubscribeToJobNotificationTopics] OK: $aws/things/
<Your_Thing_Name>/jobs/notify-next
10 1286 [OTA Task] [OTA_CheckForUpdate] Request #0
11 1289 [OTA Task] [prvParseJSONbyModel] Extracted parameter [ clientToken:
 0:<Your_Thing_Name> ]
12 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: execution
13 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: jobId
14 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: jobDocument
15 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: afr_ota
16 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: streamname
17 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: files
18 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: filepath
19 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: filesize
20 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: fileid
21 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: certfile
22 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: sig-sha256-ecdsa
23 1289 [OTA Task] [prvParseJobDoc] Ignoring job without ID.
24 1289 [OTA Task] [prvOTA_Close] Context->0x3ffbb4a8
25 1290 [OTA] [OTA_AgentInit] Ready.
26 1390 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
27 1490 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
28 1590 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
29 1690 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
[ ... ]
```

4. Das ESP32 Board wartet jetzt auf OTA-Updates. Der ESP-IDF-Monitor wird durch den Befehl `make flash monitor` gestartet. Zum Beenden können Sie Strg+] drücken. Sie können außerdem Ihr bevorzugtes TTY-Terminalprogramm (z. B. PuTTY, Tera Term oder GNU Screen) verwenden, um die serielle Ausgabe des Boards anzuzeigen. Beachten Sie, dass eine Verbindung mit der seriellen Schnittstelle des Boards zu einem Neustart führen kann.

Installieren Sie die erste Version der Firmware auf dem Nordic n RF52840 DK

> **Important**
>
> Diese Referenzintegration wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Dieses Handbuch wurde unter der Annahme verfasst, dass Sie die Schritte unter Voraussetzungen für das Update bereits ausgeführt haben. Erste Schritte mit dem Nordic n RF5284 0-DK Over-the-Air Bevor Sie versuchen, ein OTA-Update durchzuführen, sollten Sie das unter Erste Schritte mit FreeRTOS beschriebene MQTT-Demoprojekt ausführen, um sicherzustellen, dass Ihr Board und Ihre Toolchain korrekt eingerichtet sind.

So flashen Sie ein erstes Fabric-Image auf das Board:

1. Öffnen Sie *freertos*`/vendors/nordic/boards/nrf52840-dk/aws_demos/config_files/aws_demo_config.h`.
2. Ersetzen Sie `#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED` durch `CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` oder `CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`.
3. Wenn Sie die OTA Update-Demo ausgewählt haben, führen Sie die gleichen Schritte wie unter Erste Schritte mit dem Nordic n RF5284 0-DK aus, um das Image zu erstellen und zu flashen.

   Die Ausgabe sollte in etwa wie folgt aussehen:

```
9 1285 [OTA Task] [prvSubscribeToJobNotificationTopics] OK: $aws/things/your-thing-name/jobs/notify-next
10 1286 [OTA Task] [OTA_CheckForUpdate] Request #0
11 1289 [OTA Task] [prvParseJSONbyModel] Extracted parameter [ clientToken: 0:your-thing-name ]
12 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: execution
13 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: jobId
14 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: jobDocument
15 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: afr_ota
16 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: streamname
```

```
17 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: files
18 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: filepath
19 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: filesize
20 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: fileid
21 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: certfile
22 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: sig-sha256-ecdsa
23 1289 [OTA Task] [prvParseJobDoc] Ignoring job without ID.
24 1289 [OTA Task] [prvOTA_Close] Context->0x3ffbb4a8
25 1290 [OTA] [OTA_AgentInit] Ready.
26 1390 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
27 1490 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
28 1590 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
29 1690 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
```

Das Board wartet nun auf OTA-Updates.

Erste Firmware auf dem Windows Simulator

Wenn Sie den Windows Simulator verwenden, ist es nicht erforderlich, eine erste Version der Firmware zu flashen. Der Windows Simulator ist Teil der `aws_demos`-Anwendung, die auch die Firmware beinhaltet.

Installieren der ersten Version der Firmware auf einem benutzerdefinierten Board

Erstellen Sie mit Ihrer IDE das Projekt `aws_demos` und stellen Sie sicher, dass die OTA-Bibliothek enthalten ist. Weitere Hinweise zur Struktur des FreeRTOS-Quellcodes finden Sie unter. FreeRTOS RTOS-Demos

Stellen Sie sicher, dass Sie Ihr Codesignaturzertifikat, Ihren privaten Schlüssel und Ihre Zertifikatsvertrauenskette entweder im FreeRTOS-Projekt oder auf Ihrem Gerät angeben.

Flashen Sie die Anwendung mit dem entsprechenden Tool auf Ihr Board und stellen Sie sicher, dass sie korrekt läuft.

Aktualisieren der Version Ihrer Firmware

Der in FreeRTOS enthaltene OTA-Agent überprüft die Version eines Updates und installiert es nur, wenn es neuer ist als die vorhandene Firmware-Version. Die folgenden Schritte zeigen Ihnen, wie Sie die Firmware-Version der OTA-Demo-Anwendung erhöhen können.

1. Öffnen Sie das Projekt `aws_demos` in Ihrer IDE.

2. Suchen Sie die Datei **`/vendors/vendor/boards/board/aws_demos/config_files/ota_demo_config.h`** und erhöhen Sie den Wert von APP_VERSION_BUILD.
3. Um ein Update für eine Renesas rx65n-Plattform mit einem anderen Dateityp als 0 (Nicht-Firmware-Dateien) zu planen, müssen Sie die Datei mit dem Renesas Secure Flash Programmer-Tool signieren, andernfalls schlägt die Signaturprüfung auf dem Gerät fehl. Das Tool erstellt ein signiertes Dateipaket mit der Erweiterung, bei der es sich um einen proprietären Dateityp für Renesas handelt. `.rsu` Das Tool ist auf Github zu finden. Sie können den folgenden Beispielbefehl verwenden, um das Bild zu generieren:

   ```
   "Renesas Secure Flash Programmer.exe" CUI Update "RX65N(ROM 2MB)/Secure
    Bootloader=256KB" "sig-sha256-ecdsa" 1 "file_name" "output_file_name.rsu"
   ```
4. Erstellen Sie das Projekt neu.

Sie müssen Ihr Firmware-Update in den Amazon S3 S3-Bucket kopieren, den Sie wie unter beschrieben erstellt habenErstellen Sie einen Amazon S3 S3-Bucket, um Ihr Update zu speichern. Der Name der Datei, die Sie nach Amazon S3 kopieren müssen, hängt von der verwendeten Hardwareplattform ab:

- Texas Instruments CC3220 SF-LAUNCHXL: `vendors/ti/boards/cc3220_launchpad/aws_demos/ccs/debug/aws_demos.bin`
- ESP32Ausdrucksvoll: `vendors/espressif/boards/esp32/aws_demos/make/build/aws_demos.bin`

Ein OTA-Update erstellen (Konsole)AWS IoT

1. Wählen Sie im Navigationsbereich der AWS IoT Konsole unter Verwalten die Option Remote-Aktionen und dann Jobs aus.
2. Wählen Sie Job erstellen aus.
3. Wählen Sie unter Jobtyp die Option FreeRTOS OTA-Aktualisierungsjob erstellen und dann Weiter aus.
4. Geben Sie in den Jobeigenschaften einen Jobnamen und (optional) eine Beschreibung des Jobs ein, und wählen Sie dann Weiter aus.
5. Sie können ein OTA-Update auf einem einzelnen Gerät oder einer Gruppe von Geräten bereitstellen. Wählen Sie unter Zu aktualisierende Geräte ein oder mehrere Dinge oder Dinggruppen aus der Dropdownliste aus.

6. Wählen Sie unter Wählen Sie das Protokoll für die Dateiübertragung entweder HTTP oder MQTT aus, oder wählen Sie beide aus, damit jedes Gerät das zu verwendende Protokoll bestimmen kann.
7. Wählen Sie unter Signieren und Ihre Datei auswählen die Option Neue Datei für mich signieren aus.
8. Wählen Sie unter Codesignaturprofil die Option Neues Profil erstellen aus.
9. Geben Sie unter Create a code signing profile (Erstellen eines Codesignierungsprofils) einen Namen für Ihr Codesignierungsprofil ein.

    a. Wählen Sie unter Device hardware platform (Geräte-Hardwareplattform) Ihre Hardwareplattform aus.

    > **Note**
    >
    > In dieser Liste werden nur Hardwareplattformen angezeigt, die für FreeRTOS qualifiziert wurden. Wenn Sie eine nicht qualifizierte Plattform testen und die ECDSA P-256 SHA-256-Ciphersuite zum Signieren verwenden, können Sie das Codesignierungsprofil von Windows Simulator verwenden, um eine kompatible Signatur zu erstellen. Wenn Sie eine nicht qualifizierte Plattform verwenden und eine andere Ciphersuite als ECDSA P-256 SHA-256 zum Signieren verwenden, können Sie Code Signing for verwenden oder Sie können Ihr Firmware-Update selbst signieren. AWS IoT Weitere Informationen finden Sie unter Firmware-Updates digital signieren.

    b. Wählen Sie unter Codesignaturzertifikat die Option Bestehendes Zertifikat auswählen und dann ein zuvor importiertes Zertifikat aus, oder wählen Sie Neues Codesignaturzertifikat importieren, wählen Sie Ihre Dateien aus und wählen Sie Importieren aus, um ein neues Zertifikat zu importieren.

    c. Geben Sie unter Pathname of code signing certificate on device (Pfadname des Codesignierungszertifikats auf dem Gerät) den vollständig qualifizierten Pfadnamen zum Codesignierungszertifikat auf Ihrem Gerät ein. Bei den meisten Geräten können Sie dieses Feld leer lassen. Geben Sie für den Windows-Simulator und für Geräte, die das Zertifikat an einem bestimmten Speicherort speichern, den Pfadnamen hier ein.

> **Important**
>
> Fügen Sie auf dem CC3220 SF-LAUNCHXL von Texas Instruments keinen Schrägstrich (`/`) vor dem Dateinamen ein, wenn Ihr Codesignaturzertifikat im Stammverzeichnis des Dateisystems vorhanden ist. Andernfalls schlägt das OTA-Update während der Authentifizierung mit einem `file not found`-Fehler fehl.

d. Wählen Sie Erstellen aus.

10. Wählen Sie unter Datei die Option Wählen Sie eine vorhandene Datei aus und wählen Sie dann Browse S3 aus. Eine Liste Ihrer Amazon S3 S3-Buckets wird angezeigt. Wählen Sie den Bucket aus, der Ihr Firmware-Update enthält und wählen Sie dann Ihr Firmware-Update im Bucket aus.

> **Note**
>
> Die PIC32 MZEF-Demoprojekte von Microchip Curiosity erzeugen zwei Binärbilder mit den Standardnamen und. `mplab.production.bin` `mplab.production.ota.bin` Verwenden Sie die zweite Datei, wenn Sie ein Image für das OTA-Update hochladen.

11. Geben Sie unter Pfadname der Datei auf dem Gerät den vollqualifizierten Pfadnamen zu dem Speicherort auf Ihrem Gerät ein, an den der OTA-Job das Firmware-Image kopiert. Dieser Ort ist von der Plattform abhängig.

> **Important**
>
> Auf dem CC3220 SF-LAUNCHXL von Texas Instruments muss aus Sicherheitsgründen der Pfadname für das Firmware-Image lauten. `/sys/mcuflashimg.bin`

12. Öffnen Sie den Dateityp und geben Sie eine Ganzzahl im Bereich von 0 bis 255 ein. Der von Ihnen eingegebene Dateityp wird dem Job-Dokument hinzugefügt, das an die MCU geliefert wird. Der firmware/software MCU-Entwickler hat die volle Verantwortung dafür, was mit diesem Wert geschehen soll. Mögliche Szenarien beinhalten eine MCU mit einem Sekundärprozessor, dessen Firmware unabhängig vom Primärprozessor aktualisiert werden kann. Wenn das Gerät einen OTA-Aktualisierungsauftrag erhält, kann es anhand des Dateityps ermitteln, für welchen Prozessor das Update bestimmt ist.
13. Wählen Sie unter IAM-Rolle eine Rolle gemäß den Anweisungen unter aus. Erstellen einer OTA-Update-Servicerolle

14. Wählen Sie Weiter aus.
15. Geben Sie eine ID und eine Beschreibung für Ihre OTA-Aktualisierungsaufgabe ein.
16. Wählen Sie unter Jobtyp die Option Your job will complete after deployment for the selected devices/groups (snapshot).
17. Wählen Sie geeignete optionale Konfigurationen für Ihre Aufgabe (Job executions rollout (Rollout der Aufgabenausführungen), Job abort (Aufgabenabbruch), Job executions timeout (Aufgabenausführungstimeout) und Tags).
18. Wählen Sie Erstellen aus.

So verwenden Sie ein zuvor signiertes Firmware-Image:

1. Wählen Sie unter Select and sign your firmware image (Ihr Firmware-Image auswählen und signieren) die Option Select a previously signed firmware image (Ein zuvor signiertes Firmware-Image auswählen) aus.
2. Geben Sie unter Pathname of firmware image on device (Pfadname des Firmware-Images auf dem Gerät) den vollqualifizierten Pfadnamen zu dem Speicherort auf Ihrem Gerät ein, an dem der OTA-Auftrag das Firmware-Image kopieren wird. Dieser Ort ist von der Plattform abhängig.
3. Wählen Sie unter Previous code signing job (Vorheriger Codesignierungsjob) die Option Select (Auswählen) und dann den vorherigen Codesignierungsjob aus, mit dem das Firmware-Image signiert wird, das Sie für das OTA-Update verwenden.

Verwendung eines benutzerdefinierten, signierten Firmware-Images

1. Wählen Sie unter Select and sign your firmware image (Ihr Firmware-Image auswählen und signieren) die Option Use my custom signed firmware image (Mein benutzerdefiniertes Firmware-Image verwenden) aus.
2. Geben Sie unter Pathname of code signing certificate on device (Pfadname des Codesignierungszertifikats auf dem Gerät) den vollständig qualifizierten Pfadnamen zum Codesignierungszertifikat auf Ihrem Gerät ein. Bei den meisten Geräten können Sie dieses Feld leer lassen. Geben Sie für den Windows-Simulator und für Geräte, die das Zertifikat an einem bestimmten Speicherort speichern, den Pfadnamen hier ein.
3. Geben Sie unter Pathname of firmware image on device (Pfadname des Firmware-Images auf dem Gerät) den vollqualifizierten Pfadnamen zu dem Speicherort auf Ihrem Gerät ein, an dem der OTA-Auftrag das Firmware-Image kopieren wird. Dieser Ort ist von der Plattform abhängig.
4. Fügen Sie unter Signatur Ihre Signatur im PEM-Format ein.

5. Wählen Sie in Original hash algorithm (Ursprünglicher Hash-Algorithmus) den Hash-Algorithmus aus, der bei der Erstellung Ihrer Dateisignatur verwendet wurde.
6. Wählen Sie in Original encryption algorithm (Ursprünglicher Verschlüsselungsalgorithmus) den Algorithmus aus, der bei der Erstellung Ihrer Dateisignatur verwendet wurde.
7. Wählen Sie unter Wählen Sie Ihr Firmware-Image in Amazon S3 den Amazon S3 S3-Bucket und das signierte Firmware-Image im Amazon S3 S3-Bucket aus.

Nachdem Sie die Code-Signing-Informationen angegeben haben, geben Sie den OTA-Update-Jobtyp, die Service-Rolle und eine ID für das Update an.

> **Note**
>
> Verwenden Sie keine personenbezogenen Daten in der Auftrags-ID für Ihr OTA-Update. Beispiele für personenbezogene Daten sind:
>
> - Namen.
> - IP-Adressen.
> - E-Mail-Adressen.
> - Speicherorte.
> - Bankverbindung
> - Medizinische Informationen

1. Wählen Sie unter Job type (Job-Typ) die Option Your job will complete after deploying to the selected devices/groups (snapshot) (Job wird nach der Bereitstellung auf den ausgewählten Geräten/Gruppen abgeschlossen (Snapshot)) aus.
2. Wählen Sie unter IAM role for OTA update job (IAM-Rolle für OTA-Update-Job) die OTA-Service-Rolle aus.
3. Geben Sie eine alphanumerische ID für Ihre Aufgabe ein und wählen Sie dann Create (Erstellen) aus.

Der Job wird in der AWS IoT Konsole mit dem Status IN BEARBEITUNG angezeigt.

> **Note**
>
> - Die AWS IoT Konsole aktualisiert den Status von Aufträgen nicht automatisch. Aktualisieren Sie Ihren Browser, um Updates zu anzuzeigen.

Verbinden Sie Ihr serielles UART-Terminal mit Ihrem Gerät. Sie sollten eine Ausgabe sehen, die anzeigt, dass das Gerät die aktualisierte Firmware herunterlädt.

Nachdem das Gerät die aktualisierte Firmware heruntergeladen hat, startet es neu und installiert die Firmware. Sie können im UART-Terminal sehen was passiert.

Ein Tutorial, das die Verwendung der Konsole zum Erstellen eines OTA-Updates zeigt, finden Sie unter Over-the-air aktualisiert die Demo-Anwendung.

Erstellen eines OTA-Updates mit dem AWS CLI

Wenn Sie das verwenden AWS CLI , um ein OTA-Update zu erstellen, gehen Sie wie folgt vor:

1. Signieren Sie Ihr Firmware-Image digital.
2. Erstellen Sie einen Stream Ihres digital signierten Firmware-Images.
3. Starten Sie einen OTA-Update-Job.

Firmware-Updates digital signieren

Wenn Sie das AWS CLI zur Durchführung von OTA-Updates verwenden, können Sie Code Signing für verwenden AWS IoT, oder Sie können Ihr Firmware-Update selbst signieren. Eine Liste der kryptografischen Signaturen und Hashing-Algorithmen, die von Code Signing for unterstützt werden AWS IoT, finden Sie unter. SigningConfigurationOverrides Wenn Sie einen kryptografischen Algorithmus verwenden möchten, der von Code Signing for nicht unterstützt wird AWS IoT, müssen Sie Ihre Firmware-Binärdatei signieren, bevor Sie sie auf Amazon S3 hochladen.

Signieren Sie Ihr Firmware-Image mit Code Signing für AWS IoT

Um Ihr Firmware-Image mit Code Signing for zu signieren AWS IoT, können Sie eines der AWS SDKs Befehlszeilentools oder verwenden. Weitere Informationen zu Code Signing für AWS IoT finden Sie unter Code Signing für AWS IoT.

Nachdem Sie die Codesignatur-Tools installiert und konfiguriert haben, kopieren Sie Ihr unsigniertes Firmware-Image in Ihren Amazon S3 S3-Bucket und starten Sie einen Codesignatur-Job mit den folgenden Befehlen. AWS CLI Der Befehl put-signing-profile erstellt ein wiederverwendbares Codesignierungsprofil. Der Befehl start-signing-job startet die Signierungsaufgabe.

```
aws signer put-signing-profile \
    --profile-name your_profile_name \
    --signing-material certificateArn=arn:aws:acm::your-region:your-aws-account-
id:certificate/your-certificate-id \
    --platform your-hardware-platform \
    --signing-parameters certname=your_certificate_path_on_device
```

```
aws signer start-signing-job \
    --source
 's3={bucketName=your_s3_bucket,key=your_s3_object_key,version=your_s3_object_version_id}'
 \
    --destination 's3={bucketName=your_destination_bucket}' \
    --profile-name your_profile_name
```

> **Note**
>
> *your-source-bucket-name*und *your-destination-bucket-name* kann derselbe Amazon S3 S3-Bucket sein.

Dies sind die Parameter für die Befehle put-signing-profile und start-signing-job:

`source`

Gibt den Speicherort der unsignierten Firmware in einem S3-Bucket an.

- `bucketName`: Der Name des S3-Buckets.
- `key`: Der Schlüssel (Dateiname) Ihrer Firmware in Ihrem S3-Bucket.
- `version`: Die S3-Version Ihrer Firmware in Ihrem S3-Bucket. Diese unterscheidet sich von Ihrer Firmware-Version. Sie finden es, indem Sie zur Amazon S3 S3-Konsole gehen, Ihren Bucket auswählen und oben auf der Seite neben Versionen die Option Anzeigen auswählen.

`destination`

Das Ziel auf dem Gerät, auf das die signierte Firmware im S3-Bucket kopiert wird. Das Format dieses Parameters ist mit dem von Parameter `source` identisch.

`signing-material`

Der ARN Ihres Codesignierungszertifikats. Dieser ARN wird generiert, wenn Sie Ihr Zertifikat in ACM importieren.

`signing-parameters`

Eine Zuordnung aus Schlüssel-Wert-Paaren für die Signierung. Diese können alle Informationen beinhalten, die Sie beim Signieren verwenden möchten.

> **Note**
>
> Dieser Parameter ist erforderlich, wenn Sie ein Codesignierungsprofil für das Signieren von OTA-Aktualisierungen mit Code Signing for AWS IoT verwenden.

`platform`

Der `platformId` der Hardwareplattform, auf der Sie das OTA-Update verteilen.

Um eine Liste der verfügbaren Plattformen und ihrer `platformId`-Werte zurückzugeben, verwenden Sie den Befehl `aws signer list-signing-platforms`.

Der Signaturauftrag wird gestartet und das signierte Firmware-Image wird in den Amazon S3-Ziel-Bucket geschrieben. Der Dateiname für das signierte Firmware-Image ist eine GUID. Sie benötigen diesen Dateinamen, wenn Sie einen Stream erstellen. Sie finden den Dateinamen, indem Sie zur Amazon S3-Konsole gehen und Ihren Bucket auswählen. Wenn Sie keine Datei mit einem GUID-Dateinamen sehen, aktualisieren Sie Ihren Browser.

Der Befehl zeigt einen Aufgaben-ARN und eine Aufgaben-ID an. Diese Werte benötigen Sie später. Weitere Informationen zu Code Signing für AWS IoT finden Sie unter Code Signing for AWS IoT.

Firmware-Images manuell signieren

Signieren Sie Ihr Firmware-Image digital und laden Sie Ihr signiertes Firmware-Image in Ihren Amazon S3 S3-Bucket hoch.

Erstellen eines Streams Ihres Firmware-Updates

Ein Stream ist eine abstrakte Schnittstelle zu Daten, die von einem Gerät verarbeitet werden können. Ein Stream kann die Komplexität des Zugriffs auf Daten verbergen, die an verschiedenen

Speicherorten oder in verschiedenen cloudbasierten Services gespeichert sind. Mit dem OTA Update Manager-Service können Sie mehrere Daten verwenden, die an verschiedenen Orten in Amazon S3 gespeichert sind, um ein OTA-Update durchzuführen.

Wenn Sie ein AWS IoT OTA-Update erstellen, können Sie auch einen Stream erstellen, der Ihr signiertes Firmware-Update enthält. Erstellen Sie eine JSON-Datei (`stream.json`), die Ihr signiertes Firmware-Image identifiziert. Die JSON-Datei sollte Folgendes enthalten:

```
[
  {
    "fileId":"your_file_id",
    "s3Location":{
      "bucket":"your_bucket_name",
      "key":"your_s3_object_key"
    }
  }
]
```

Dies sind die Attribute in der JSON-Datei:

`fileId`

Eine beliebige Ganzzahl zwischen 0 und 255, die Ihr Firmware-Image identifiziert.

`s3Location`

Der Bucket und der Schlüssel für die zu streamende Firmware.

`bucket`

Der Amazon S3 S3-Bucket, in dem Ihr unsigniertes Firmware-Image gespeichert ist.

`key`

Der Dateiname Ihres signierten Firmware-Images im Amazon S3 S3-Bucket. Sie finden diesen Wert in der Amazon S3 S3-Konsole, indem Sie sich den Inhalt Ihres Buckets ansehen.

Wenn Sie Code Signing for verwenden AWS IoT, ist der Dateiname eine GUID, die von Code Signing for AWS IoT generiert wurde.

Verwenden Sie den Befehl create-stream AWS CLI , um einen Stream zu erstellen.

```
aws iot create-stream \
    --stream-id your_stream_id \
```

```
    --description your_description \
    --files file://stream.json \
    --role-arn your_role_arn
```

Dies sind die Argumente für den create-stream AWS CLI Befehl:

`stream-id`

Eine beliebige Zeichenfolge zur Identifizierung des Streams.

`description`

Eine optionale Beschreibung des Streams.

`files`

Einer oder mehrere Verweise auf JSON-Dateien, die Daten über Firmware-Images zum Streamen enthalten. Die JSON-Datei muss die folgenden Attribute enthalten:

`fileId`

Eine beliebige Datei-ID.

`s3Location`

Der Bucket-Name, unter dem das signierte Firmware-Image gespeichert ist und der Schlüssel (Dateiname) des signierten Firmware-Images.

`bucket`

Der Amazon S3 S3-Bucket, in dem das signierte Firmware-Image gespeichert ist.

`key`

Der Schlüssel (Dateiname) des signierten Firmware-Images.

Wenn Sie Code Signing for verwenden AWS IoT, ist dieser Schlüssel eine GUID.

Im Folgenden sehen Sie ein Beispiel für eine `stream.json`-Datei.

```
[
    {
        "fileId":123,
        "s3Location": {
            "bucket":"codesign-ota-bucket",
            "key":"48c67f3c-63bb-4f92-a98a-4ee0fbc2bef6"
        }
```

```
    }
]
```

`role-arn`

Die OTA-Servicerolle, die auch Zugriff auf den Amazon S3 S3-Bucket gewährt, in dem das Firmware-Image gespeichert ist.

Um den Amazon S3 S3-Objektschlüssel Ihres signierten Firmware-Images zu finden, verwenden Sie den aws signer describe-signing-job --job-id ***my-job-id*** Befehl where `my-job-id` is the job id displayed by the create-signing-job AWS CLI command. Die Ausgabe des Befehls describe-signing-job enthält den Schlüssel des signierten Firmware-Images.

```
... text deleted for brevity ...
  "signedObject": {
    "s3": {
      "bucketName": "ota-bucket",
      "key": "7309da2c-9111-48ac-8ee4-5a4262af4429"
    }
  }
... text deleted for brevity ...
```

Erstellen eines OTA-Updates

Verwenden Sie den create-ota-update AWS CLI Befehl, um einen OTA-Aktualisierungsauftrag zu erstellen.

> **Note**
>
> Verwenden Sie keine personenbezogenen Daten in Ihrer OTA-Aktualsierungsauftrags-ID. Beispiele für personenbezogene Daten sind:
>
> - Namen.
> - IP-Adressen.
> - E-Mail-Adressen.
> - Speicherorte.
> - Bankverbindung
> - Medizinische Informationen

```
aws iot  create-ota-update \
    --ota-update-id value \
    [--description value] \
    --targets value \
    [--protocols value] \
    [--target-selection value] \
    [--aws-job-executions-rollout-config value] \
    [--aws-job-presigned-url-config value] \
    [--aws-job-abort-config value] \
    [--aws-job-timeout-config value] \
    --files value \
    --role-arn value \
    [--additional-parameters value] \
    [--tags value]  \
    [--cli-input-json value] \
    [--generate-cli-skeleton]
```

`cli-input-json`-Format

```
{
  "otaUpdateId": "string",
  "description": "string",
  "targets": [
    "string"
  ],
  "protocols": [
    "string"
  ],
  "targetSelection": "string",
  "awsJobExecutionsRolloutConfig": {
    "maximumPerMinute": "integer",
    "exponentialRate": {
      "baseRatePerMinute": "integer",
      "incrementFactor": "double",
      "rateIncreaseCriteria": {
        "numberOfNotifiedThings": "integer",
        "numberOfSucceededThings": "integer"
      }
    }
  },
  "awsJobPresignedUrlConfig": {
    "expiresInSec": "long"
  },
```

```
  "awsJobAbortConfig": {
    "abortCriteriaList": [
      {
        "failureType": "string",
        "action": "string",
        "thresholdPercentage": "double",
        "minNumberOfExecutedThings": "integer"
      }
    ]
  },
  "awsJobTimeoutConfig": {
    "inProgressTimeoutInMinutes": "long"
  },
  "files": [
    {
      "fileName": "string",
      "fileType": "integer",
      "fileVersion": "string",
      "fileLocation": {
        "stream": {
          "streamId": "string",
          "fileId": "integer"
        },
        "s3Location": {
          "bucket": "string",
          "key": "string",
          "version": "string"
        }
      },
      "codeSigning": {
        "awsSignerJobId": "string",
        "startSigningJobParameter": {
          "signingProfileParameter": {
            "certificateArn": "string",
            "platform": "string",
            "certificatePathOnDevice": "string"
          },
          "signingProfileName": "string",
          "destination": {
            "s3Destination": {
              "bucket": "string",
              "prefix": "string"
            }
          }
```

```
        },
        "customCodeSigning": {
          "signature": {
            "inlineDocument": "blob"
          },
          "certificateChain": {
            "certificateName": "string",
            "inlineDocument": "string"
          },
          "hashAlgorithm": "string",
          "signatureAlgorithm": "string"
        }
      },
      "attributes": {
        "string": "string"
      }
    }
  ],
  "roleArn": "string",
  "additionalParameters": {
    "string": "string"
  },
  "tags": [
    {
      "Key": "string",
      "Value": "string"
    }
  ]
}
```

cli-input-json-Felder

Name	Typ	Description
`otaUpdateId`	Zeichenfolge (max:128 min:1)	Die ID des zu erstellenden OTA-Updates.
`description`	Zeichenfolge (max:2028)	Die Beschreibung des OTA-Updates.

Name	Typ	Description
`targets`	list	Die Zielgeräte für den Empfang der OTA-Updates.
`protocols`	list	Das Protokoll, das zum Übertragen des OTA-Aktua lisierungsabbilds verwendet wird. Gültige Werte sind [HTTP], [MQTT], [HTTP, MQTT]. Wenn sowohl HTTP als auch MQTT angegeben sind, kann das Zielgerät das Protokoll auswählen.

Name	Typ	Description
`targetSelection`	Zeichenfolge	Gibt an, ob das Update weiterhin ausgeführt wird (CONTINUOUS), oder ob es abgeschlossen wird, nachdem alle als Ziele angegebenen Objekte das Update abgeschlossen haben (SNAPSHOT). Bei „continuous“ kann das Update auch für ein Objekt ausgeführt werden, wenn eine Änderung in einem Ziel erkannt wird. Beispielsweise wird ein Update für ein Objekt ausgeführt, wenn das Objekt einer Zielgruppe hinzugefügt wird, auch wenn das Update von allen ursprünglich in der Gruppe befindlichen Objekten abgeschlossen wurde. Gültige Werte: CONTINUOUS | SNAPSHOT. enum: CONTINUOUS | SNAPSHOT
`awsJobExecutionsRolloutConfig`		Konfiguration für den Rollout der OTA-Updates.
`maximumPerMinute`	Ganzzahl (max:1000 min:1)	Die maximale Anzahl der pro Minute gestarteten OTA-Update-Auftragsausführungen.

Name	Typ	Description
`exponentialRate`		Die Wachstumsrate für einen Auftrags-Rollout. Dieser Parameter ermöglicht das Definieren eines exponenti ellen Anstiegs der Rate für einen Aufgaben-Rollout.
`baseRatePerMinute`	Ganzzahl (max:1000 min:1)	Die Mindestanzahl der Objekte pro Minute, die zu Beginn des Aufgaben-Rollouts über eine anstehende Aufgabe benachrichtigt werden. Dies ist die anfängliche Rate des Rollouts.
`rateIncreaseCriteria`		Die Kriterien zur Initiierung der Erhöhung der Rollout-Rate für einen Auftrag. AWS IoT unterstützt bis zu einer Nachkommastelle (z. B. 1,5, aber nicht 1,55).
`numberOfNotifiedTh ings`	Ganzzahl (min:1)	Wenn diese Anzahl von Objekten benachrichtigt wurde, wird eine Erhöhung der Rollout-Rate initiiert.
`numberOfSucceededT hings`	Ganzzahl (min:1)	Wenn diese Anzahl von Objekten in ihrer Aufgabena usführung erfolgreich war, wird eine Erhöhung der Rollout-R ate initiiert.
`awsJobPresignedUrl Config`		Konfigurationsinformationen für vorsignierte URLs.

Name	Typ	Description
`expiresInSec`	long	Wie lange (in Sekunden) vorsignierte Zeichen gültig URLs sind. Gültige Werte sind 60 - 3600, der Standardwert ist 1800 Sekunden. Vorsignierte URLs werden generiert, wenn eine Anfrage für das Auftragsdokument eingeht.
`awsJobAbortConfig`		Die Kriterien, die bestimmen, wann und wie eine Auftragsunterbrechung stattfindet.
`abortCriteriaList`	list	Die Liste der Kriterien, die bestimmen, wann und wie der Job beendet werden soll.
`failureType`	Zeichenfolge	Die Art der Fehler bei der Auftragsausführung, die zu einer Unterbrechung des Jobs führen können. enum: FAILED | REJECTED | TIMED_OUT | ALL
`action`	Zeichenfolge	Die Art der Auftragsaktion, die ergriffen werden muss, um die Auftragsunterbrechung einzuleiten. enum: CANCEL

Name	Typ	Description
`minNumberOfExecutedThings`	Ganzzahl (min:1)	Die Mindestanzahl von Dingen, die Benachrichtigungen zur Auftragsausführung erhalten müssen, bevor der Job gestoppt werden kann.
`awsJobTimeoutConfig`		Gibt die Dauer an, die jedes Gerät für den Abschluss der Ausführung des Auftrags hat. Ein Timer wird gestartet, wenn der Status der Auftragsausführung auf `IN_PROGRESS` gesetzt wird. Wenn der Status der Auftragsausführung vor Ablauf des Timers auf keinen anderen Endstatus gesetzt wird, wird er automatisch auf `TIMED_OUT` gesetzt.

Name	Typ	Description
`inProgressTimeoutInMinutes`	long	Gibt die Dauer in Minuten an, die dieses Gerät für den Abschluss der Ausführung dieses Auftrags hat. Das Intervall für die Zeitüberschreitung kann zwischen 1 Minute und 7 Tage (1 bis 10 080 Minuten) betragen. Der Timer für „In Bearbeitung“ kann nicht aktualisiert werden und gilt für alle Auftragsausführungen für den Auftrag. Immer wenn eine Auftragsausführung länger als dieses Intervall im Status IN_PROGRESS bleibt, schlägt die Auftragsausführung fehl und wechselt in den Endstatus `TIMED_OUT` .
`files`	list	Die von dem OTA-Update zu streamenden Dateien.
`fileName`	Zeichenfolge	Der Name der Datei.
`fileType`	Ganzzahl range- max:255 min:0	Ein ganzzahliger Wert, den Sie in das Auftragsdokument aufnehmen können, damit Ihre Geräte den Typ der von der Cloud empfangenen Datei identifizieren können.
`fileVersion`	Zeichenfolge	Die Dateiversion.
`fileLocation`		Der Speicherort der aktualisierten Firmware.

Name	Typ	Description
`stream`		Der Stream, der das OTA-Update enthält.
`streamId`	Zeichenfolge (max:128 min:1)	Die Stream-ID.
`fileId`	Ganzzahl (max:255 min:0)	Die ID einer mit einem Stream verbundenen Datei.
`s3Location`		Der Speicherort der aktualisierten Firmware in S3.
`bucket`	Zeichenfolge (min:1)	Der S3-Bucket.
`key`	Zeichenfolge (min:1)	Der S3-Schlüssel
`version`	Zeichenfolge	Die S3-Bucket-Version.
`codeSigning`		Die Codesignaturmethode der Datei.
`awsSignerJobId`	Zeichenfolge	Die ID des AWSSigner Job, der zum Signieren der Datei erstellt wurde.
`startSigningJobParameter`		Beschreibt den Code-Signierungsauftrag.
`signingProfileParameter`		Beschreibt das Code-Signierungsprofil.
`certificateArn`	Zeichenfolge	Zertifikat-ARN.

Name	Typ	Description
`platform`	Zeichenfolge	Die Hardware-Plattform Ihres Geräts.
`certificatePathOnDevice`	Zeichenfolge	Der Speicherort des Zertifikats zur Codesignierung auf Ihrem Gerät.
`signingProfileName`	Zeichenfolge	Der Name des Code-Signierungsprofils.
`destination`		Der Speicherort für die mit Code signierte Datei.
`s3Destination`		Beschreibt den Speicherort der aktualisierten Firmware in S3.
`bucket`	Zeichenfolge (min:1)	Der S3-Bucket, der die aktualisierte Firmware enthält.
`prefix`	Zeichenfolge	Das S3-Präfix.
`customCodeSigning`		Eine benutzerdefinierte Methode zum Signieren einer Datei.
`signature`		Die Signatur für die Datei.
`inlineDocument`	blob	Eine base64-kodierte binäre Repräsentation der Codesignatur-Signatur.
`certificateChain`		Die Zertifikatskette.
`certificateName`	Zeichenfolge	Der Name des Zertifikats.

Name	Typ	Description
`inlineDocument`	Zeichenfolge	Eine base64-kodierte binäre Repräsentation der Codesignatur-Zertifikatskette.
`hashAlgorithm`	Zeichenfolge	Der für die Codesignatur der Datei verwendete Hash-Algorithmus
`signatureAlgorithm`	Zeichenfolge	Der für die Codesignatur der Datei verwendete Signatur-Algorithmus
`attributes`	map	Eine Liste von name/attribute Paaren.
`roleArn`	Zeichenfolge (max:2048 min:20)	Die IAM-Rolle, die AWS IoT Zugriff auf Amazon S3, AWS IoT Jobs und AWS Code Signing-Ressourcen gewährt, um einen OTA-Aktualisierungsjob zu erstellen.
`additionalParameters`	map	Eine Liste der zusätzlichen OTA-Update-Parameter, bei denen es sich um Name/Wert-Paare handelt.
`tags`	list	Metadaten, die verwendet werden können, um Updates zu verwalten.
`Key`	Zeichenfolge (max:128 min:1)	Der Tag-Schlüssel.

Name	Typ	Description
`Value`	Zeichenfolge (max:256 min:1)	Der Tag-Wert.

Ausgabe

```
{
  "otaUpdateId": "string",
  "awsIotJobId": "string",
  "otaUpdateArn": "string",
  "awsIotJobArn": "string",
  "otaUpdateStatus": "string"
}
```

AWS CLI Ausgabefelder

Name	Typ	Description
`otaUpdateId`	Zeichenfolge (max:128 min:1)	Die OTA-Update-ID.
`awsIotJobId`	Zeichenfolge	Die mit dem OTA-Update verknüpfte AWS IoT Job-ID.
`otaUpdateArn`	Zeichenfolge	Der ARN des OTA-Updates.
`awsIotJobArn`	Zeichenfolge	Der AWS IoT Job-ARN, der dem OTA-Update zugeordnet ist.
`otaUpdateStatus`	Zeichenfolge	Der Status des OTA-Updates. enum: CREATE_PENDING | CREATE_IN_PROGRESS | CREATE_COMPLETE | CREATE_FAILED

Im Folgenden finden Sie ein Beispiel für eine JSON-Datei, die an den create-ota-update Befehl übergeben wurde, der Code Signing for verwendet AWS IoT.

```
[
  {
    "fileName": "firmware.bin",
    "fileType": 1,
    "fileLocation": {
      "stream": {
        "streamId": "004",
        "fileId":123
      }
    },
    "codeSigning": {
      "awsSignerJobId": "48c67f3c-63bb-4f92-a98a-4ee0fbc2bef6"
    }
  }
]
```

Im Folgenden finden Sie ein Beispiel für eine JSON-Datei, die an den create-ota-update AWS CLI Befehl übergeben wurde und eine Inline-Datei verwendet, um benutzerdefiniertes Codesignaturmaterial bereitzustellen.

```
[
  {
    "fileName": "firmware.bin",
    "fileType": 1,
    "fileLocation": {
      "stream": {
        "streamId": "004",
        "fileId": 123
      }
    },
    "codeSigning": {
      "customCodeSigning":{
        "signature":{
          "inlineDocument":"your_signature"
        },
        "certificateChain": {
          "certificateName": "your_certificate_name",
          "inlineDocument":"your_certificate_chain"
        },
        "hashAlgorithm":"your_hash_algorithm",
```

```
        "signatureAlgorithm":"your_signature_algorithm"
      }
    }
  }
]
```

Das Folgende ist ein Beispiel für eine an den create-ota-update AWS CLI Befehl übergebene JSON-Datei, die es FreeRTOS OTA ermöglicht, einen Codesignaturauftrag zu starten und ein Codesignaturprofil und einen Stream zu erstellen.

```
[
  {
    "fileName": "your_firmware_path_on_device",
    "fileType": 1,
    "fileVersion": "1",
    "fileLocation": {
      "s3Location": {
        "bucket": "your_bucket_name",
        "key": "your_object_key",
        "version": "your_S3_object_version"
      }
    },
    "codeSigning":{
      "startSigningJobParameter":{
        "signingProfileName": "myTestProfile",
        "signingProfileParameter": {
          "certificateArn": "your_certificate_arn",
          "platform": "your_platform_id",
          "certificatePathOnDevice": "certificate_path"
        },
        "destination": {
          "s3Destination": {
            "bucket": "your_destination_bucket"
          }
        }
      }
    }
  }
]
```

Das Folgende ist ein Beispiel für eine JSON-Datei, die an den create-ota-update AWS CLI Befehl übergeben wird, der ein OTA-Update erstellt, das einen Codesignaturauftrag mit einem vorhandenen Profil startet und den angegebenen Stream verwendet.

```
[
  {
    "fileName": "your_firmware_path_on_device",
    "fileType": 1,
    "fileVersion": "1",
    "fileLocation": {
      "s3Location": {
        "bucket": "your_s3_bucket_name",
        "key": "your_object_key",
        "version": "your_S3_object_version"
      }
    },
    "codeSigning":{
      "startSigningJobParameter":{
        "signingProfileName": "your_unique_profile_name",
        "destination": {
          "s3Destination": {
            "bucket": "your_destination_bucket"
          }
        }
      }
    }
  }
]
```

Das Folgende ist ein Beispiel für eine an den create-ota-update AWS CLI Befehl übergebene JSON-Datei, die es FreeRTOS OTA ermöglicht, einen Stream mit einer vorhandenen Codesignatur-Job-ID zu erstellen.

```
[
  {
    "fileName": "your_firmware_path_on_device",
    "fileType": 1,
    "fileVersion": "1",
    "codeSigning":{
      "awsSignerJobId": "your_signer_job_id"
    }
  }
```

```
]
```

Das Folgende ist ein Beispiel für eine JSON-Datei, die an den create-ota-update AWS CLI Befehl übergeben wird, der ein OTA-Update erstellt. Das Update erstellt einen Stream aus dem angegebenen S3-Objekt und verwendet die benutzerdefinierte Codesignierung:

```
[
  {
    "fileName": "your_firmware_path_on_device",
    "fileType": 1,
    "fileVersion": "1",
    "fileLocation": {
      "s3Location": {
        "bucket": "your_bucket_name",
        "key": "your_object_key",
        "version": "your_S3_object_version"
      }
    },
    "codeSigning":{
      "customCodeSigning": {
        "signature":{
          "inlineDocument":"your_signature"
        },
        "certificateChain": {
          "inlineDocument":"your_certificate_chain",
          "certificateName": "your_certificate_path_on_device"
        },
        "hashAlgorithm":"your_hash_algorithm",
        "signatureAlgorithm":"your_sig_algorithm"
      }
    }
  }
]
```

Auflisten von OTA-Updates

Sie können den list-ota-updates AWS CLI Befehl verwenden, um eine Liste aller OTA-Updates abzurufen.

```
aws iot list-ota-updates
```

Die Ausgabe des Befehls list-ota-updates sieht wie folgt aus:

```
{
  "otaUpdates": [

    {
      "otaUpdateId": "my_ota_update2",
      "otaUpdateArn": "arn:aws:iot:us-west-2:123456789012:otaupdate/my_ota_update2",
      "creationDate": 1522778769.042
    },
    {
      "otaUpdateId": "my_ota_update1",
      "otaUpdateArn": "arn:aws:iot:us-west-2:123456789012:otaupdate/my_ota_update1",
      "creationDate": 1522775938.956
    },
    {
      "otaUpdateId": "my_ota_update",
      "otaUpdateArn": "arn:aws:iot:us-west-2:123456789012:otaupdate/my_ota_update",
      "creationDate": 1522775151.031
    }
  ]
}
```

Informationen über ein OTA-Update abrufen

Sie können den get-ota-update AWS CLI Befehl verwenden, um den Erstellungs- oder Löschstatus eines OTA-Updates abzurufen.

```
aws iot get-ota-update --ota-update-id your-ota-update-id
```

Die Ausgabe des Befehls get-ota-update sieht wie folgt aus.

```
{
    "otaUpdateInfo": {
        "otaUpdateId": "ota-update-001",
        "otaUpdateArn": "arn:aws:iot:region:123456789012:otaupdate/ota-update-001",
        "creationDate": 1575414146.286,
        "lastModifiedDate": 1575414149.091,
        "targets": [
            "arn:aws:iot:region:123456789012:thing/myDevice"
        ],
        "protocols": [ "HTTP" ],
        "awsJobExecutionsRolloutConfig": {
            "maximumPerMinute": 0
```

```
        },
        "awsJobPresignedUrlConfig": {
            "expiresInSec": 1800
        },
        "targetSelection": "SNAPSHOT",
        "otaUpdateFiles": [
            {
                "fileName": "my_firmware.bin",
                "fileType": 1,
                "fileLocation": {
                    "s3Location": {
                        "bucket": "my-bucket",
                        "key": "my_firmware.bin",
                        "version": "AvP3bfJC9gyqnwoxPHuTqM5GWENt4iii"
                    }
                },
                "codeSigning": {
                    "awsSignerJobId": "b7a55a54-fae5-4d3a-b589-97ed103737c2",
                    "startSigningJobParameter": {
                        "signingProfileParameter": {},
                        "signingProfileName": "my-profile-name",
                        "destination": {
                            "s3Destination": {
                                "bucket": "some-ota-bucket",
                                "prefix": "SignedImages/"
                            }
                        }
                    },
                    "customCodeSigning": {}
                }
            }
        ],
        "otaUpdateStatus": "CREATE_COMPLETE",
        "awsIotJobId": "AFR_OTA-ota-update-001",
        "awsIotJobArn": "arn:aws:iot:region:123456789012:job/AFR_OTA-ota-update-001"
    }
}
```

Für `otaUpdateStatus` können folgende Werte zurückgegeben werden:

`CREATE_PENDING`

Die Erstellung eines OTA-Updates steht noch aus.

`CREATE_IN_PROGRESS`

Ein OTA-Update wird erstellt.

`CREATE_COMPLETE`

Es wurde ein OTA-Update erstellt.

`CREATE_FAILED`

Die Erstellung eines OTA-Updates ist fehlgeschlagen.

`DELETE_IN_PROGRESS`

Ein OTA-Update wird gelöscht.

`DELETE_FAILED`

Das Löschen eines OTA-Updates ist fehlgeschlagen.

> **Note**
>
> Um den Ausführungsstatus eines OTA-Updates nach dessen Erstellung abzurufen, müssen Sie den Befehl describe-job-execution verwenden. Weitere Informationen finden Sie unter Beschreiben Sie die Jobausführung.

Löschen von OTA-bezogenen Daten

Derzeit können Sie die AWS IoT Konsole nicht zum Löschen von Streams oder OTA-Updates verwenden. Sie können den verwenden, AWS CLI um Streams, OTA-Updates und die während eines OTA-Updates erstellten AWS IoT Jobs zu löschen.

Löschen eines OTA-Streams

Wenn Sie ein OTA-Update erstellen, das MQTT verwendet, können Sie entweder die Befehlszeile oder die AWS IoT Konsole verwenden, um einen Stream zu erstellen, um die Firmware in Teile aufzuteilen, sodass sie über MQTT gesendet werden kann. Sie können diesen Stream mit dem delete-stream AWS CLI Befehl löschen, wie im folgenden Beispiel gezeigt.

```
aws iot delete-stream --stream-id your_stream_id
```

Löschen eines OTA-Updates

Beim Erstellen eines OTA-Updates werden folgende Elemente erstellt:

- Ein Eintrag in der OTA-Update-Job-Datenbank.
- Ein AWS IoT Job zur Durchführung des Updates.
- Eine AWS IoT Jobausführung für jedes Gerät, das aktualisiert wird.

Der Befehl delete-ota-update löscht nur den Eintrag in der OTA-Update-Job-Datenbank. Zum Löschen des AWS IoT -Jobs müssen Sie den Befehl delete-job verwenden.

Verwenden Sie den Befehl delete-ota-update, um ein OTA-Update zu löschen:

```
aws iot delete-ota-update --ota-update-id your_ota_update_id
```

`ota-update-id`

Die ID des zu löschenden OTA-Updates.

`delete-stream`

Löscht den Stream, der dem OTA-Update zugeordnet ist.

`force-delete-aws-job`

Löscht den AWS IoT Job, der dem OTA-Update zugeordnet ist. Wenn dieses Flag nicht gesetzt ist und der Job den Status `In_Progress` hat, wird der Job nicht gelöscht.

Löschen eines für ein OTA-Update erstellten IoT-Jobs

FreeRTOS erstellt einen AWS IoT Job, wenn Sie ein OTA-Update erstellen. Auch für jedes Gerät, das den Job verarbeitet, wird eine Job-Ausführung erstellt. Sie können den delete-job AWS CLI Befehl verwenden, um einen Job und die zugehörigen Jobausführungen zu löschen.

```
aws iot delete-job --job-id your-job-id --no-force
```

Der Parameter `no-force` legt fest, dass nur Jobs gelöscht werden können, die sich in einem Beendigungsstatus (COMPLETED oder CANCELLED) befinden. Indem Sie den Parameter `force` übergeben, können Sie einen Job löschen, der sich nicht in einem Beendigungsstatus befindet. Weitere Informationen finden Sie unter DeleteJob -API.

> **Note**
>
> Das Löschen eines Jobs mit dem Status IN_PROGRESS unterbricht alle Job-Ausführungen, die sich auf Ihren Geräten befinden. Dies kann dazu führen, dass ein Gerät in einem nicht deterministischen Status verbleibt. Stellen Sie sicher, dass jedes Gerät, das einen gelöschten Job ausführt, zu einem bekannten Status zurückkehren kann.

Abhängig von der Anzahl der für die Aufgabe erstellten Aufgabenausführungen und anderen Faktoren kann das Löschen einer Aufgabe einige Minuten dauern. Während die Aufgabe gelöscht wird, hat sie den Status DELETION_IN_PROGRESS. Der Versuch, einen Job zu löschen oder abzubrechen, dessen Status bereits DELETION_IN_PROGRESS ist, führt zu einem Fehler.

Mit der delete-job-execution können Sie eine Job-Ausführung löschen. Möglicherweise möchten Sie eine Job-Ausführung löschen, wenn eine kleine Anzahl von Geräten einen Job nicht verarbeiten kann. Dadurch wird die Job-Ausführung für ein einzelnes Gerät gelöscht, wie im folgenden Beispiel gezeigt.

```
aws iot delete-job-execution --job-id your-job-id --thing-name
                    your-thing-name --execution-number your-job-execution-number --no-
force
```

Wie beim delete-job AWS CLI Befehl können Sie den `--force` Parameter an die übergeben, delete-job-execution um das Löschen einer Jobausführung zu erzwingen. Weitere Informationen finden Sie unter DeleteJobExecutionAPI.

> **Note**
>
> Das Löschen einer Job-Ausführung mit dem Status IN_PROGRESS unterbricht alle Job-Ausführungen, die den Status IN_PROGRESS auf Ihren Geräten haben. Es kann dazu führen, dass ein Gerät in einem nicht deterministischen Status verbleibt. Stellen Sie sicher, dass jedes Gerät, das einen gelöschten Job ausführt, zu einem bekannten Status zurückkehren kann.

Weitere Informationen zur Verwendung der OTA-Update Demo-Anwendung finden Sie unter Over-the-air aktualisiert die Demo-Anwendung.

OTA Update Manager-Service

Der over-the-air (OTA) Update Manager-Dienst bietet folgende Möglichkeiten:

- Erstellen Sie ein OTA-Update und die Ressourcen, die es verwendet, einschließlich eines AWS IoT Jobs, eines AWS IoT Streams und der Codesignatur.
- Informationen über ein OTA-Update abrufen.
- Listet alle OTA-Updates auf, die mit Ihrem AWS Konto verknüpft sind.
- Löschen eines OTA-Updates.

Ein OTA-Update ist eine Datenstruktur, die vom OTA Update Manager-Service gepflegt wird. Sie enthält Folgendes:

- Eine OTA-Update-ID
- Eine optionale OTA-Update-Beschreibung
- Eine Liste der zu aktualisierenden Geräte (Ziele)
- Den Typ des OTA-Updates: CONTINUOUS oder SNAPSHOT Im Abschnitt Jobs des AWS IoT Entwicklerhandbuchs finden Sie eine Erläuterung der Art des Updates, das Sie benötigen.
- Das Protokoll, das zum Ausführen des OTA-Updates verwendet wird: [MQTT], [HTTP] oder [MQTT, HTTP]. Wenn Sie MQTT und HTTP angeben, bestimmt das Geräte-Setup das zu verwendende Protokoll.
- Eine Liste von Dateien, die an die Zielgeräte gesendet werden sollen
- Die IAM-Rolle, die AWS IoT Zugriff auf Amazon S3, AWS IoT Jobs und AWS Code Signing-Ressourcen gewährt, um einen OTA-Aktualisierungsjob zu erstellen.
- Eine optionale Liste von benutzerdefinierten Name-Wert-Paaren

OTA-Updates wurden entwickelt, um die Gerätefirmware zu aktualisieren. Sie können sie jedoch verwenden, um beliebige Dateien an ein oder mehrere Geräte zu senden, bei AWS IoT denen Sie registriert sind. Wenn Sie Firmware-Updates mit Over-the-Air senden, wird empfohlen, sie digital zu signieren, damit die Geräte, die die Dateien empfangen, überprüfen können, dass sie unterwegs nicht manipuliert wurden.

Sie können aktualisierte Firmware-Images mithilfe des HTTP- oder MQTT-Protokolls senden, abhängig von den von Ihnen ausgewählten Einstellungen. Sie können Ihre Firmware-Updates mit Code Signing for FreeRTOS signieren oder Sie können Ihre eigenen Codesignatur-Tools verwenden.

Für mehr Kontrolle über den Prozess können Sie die CreateStreamAPI verwenden, um beim Senden von Updates über MQTT einen Stream zu erstellen. In einigen Fällen können Sie den FreeRTOS-Agent-Code ändern, um die Größe der Blöcke, die Sie senden und empfangen, anzupassen.

Wenn Sie ein OTA-Update erstellen, erstellt der OTA-Manager-Service einen AWS IoT -Auftrag, um Ihre Geräte darüber zu informieren, dass ein Update verfügbar ist. Der FreeRTOS OTA Agent läuft auf Ihren Geräten und wartet auf Aktualisierungsnachrichten. Wenn ein Update verfügbar ist, fordert er das Firmware-Update-Image über HTTP oder MQTT an und speichert die Dateien lokal. Er überprüft die digitale Signatur der heruntergeladenen Dateien. Falls sie gültig ist, installiert er das Firmware-Update. Wenn Sie FreeRTOS nicht verwenden, müssen Sie Ihren eigenen OTA-Agenten implementieren, um auf Updates zu warten und diese herunterzuladen und alle Installationsvorgänge durchzuführen.

Integrieren des OTA-Agents in Ihre Anwendung

Der over-the-air (OTA) Agent wurde entwickelt, um die Menge an Code zu vereinfachen, die Sie schreiben müssen, um Ihrem Produkt OTA-Aktualisierungsfunktionen hinzuzufügen. Dieser Integrationsaufwand besteht hauptsächlich aus der Initialisierung des OTA-Agenten und der Erstellung einer benutzerdefinierten Rückruffunktion für die Reaktion auf die OTA-Agent-Ereignismeldungen. Während der Initialisierung werden OS-, MQTT-, HTTP- (wenn HTTP für das Herunterladen von Dateien verwendet wird) und plattformspezifische Implementierungsschnittstellen (PAL) an den OTA-Agenten übergeben. Puffer können auch initialisiert und an den OTA-Agenten übergeben werden.

> **Note**
>
> Obwohl die Integration der OTA-Update-Funktion in Ihre Anwendung recht einfach ist, müssen Sie sich für das OTA-Update-System nicht nur mit der Integration von Code auf dem Gerät auskennen. Weitere Informationen zur Konfiguration Ihres AWS Kontos mit AWS IoT Dingen, Anmeldeinformationen, Codesignaturzertifikaten, Bereitstellungsgeräten und OTA-Aktualisierungsaufträgen finden Sie unter FreeRTOS-Voraussetzungen.

Verbindungsverwaltung

Der OTA-Agent verwendet das MQTT-Protokoll für alle Kontrollkommunikationsvorgänge, die AWS IoT Dienste betreffen, verwaltet aber nicht die MQTT-Verbindung. Um sicherzustellen, dass der OTA-Agent nicht gegen die Verbindungsverwaltungsrichtlinie Ihrer Anwendung verstößt, muss die MQTT-Verbindung von der Haupt-Benutzeranwendung verwaltet werden (inkl. Funktionalität für

die Trennung und das erneute Verbinden). Die Datei kann über das MQTT- oder HTTP-Protokoll heruntergeladen werden. Sie können das gewünschter Protokoll auswählen, wenn Sie den OTA-Auftrag erstellen. Wenn Sie MQTT wählen, verwendet der OTA-Agent dieselbe Verbindung für Steuervorgänge und zum Herunterladen von Dateien.

Einfache OTA-Demo

Im Folgenden finden Sie einen Auszug aus einer einfachen OTA-Demo. Sie zeigt, wie sich der Agent mit dem MQTT-Broker verbindet und den OTA-Agent initialisiert. In diesem Beispiel konfigurieren wir die Demo so, dass sie den standardmäßigen OTA-Anwendungs-Callback verwendet und einige Statistiken einmal pro Sekunde zurückgibt. Zu Übersicht lassen wir hier einige Details aus dieser Demo weg.

Die OTA-Demo demonstriert auch die MQTT-Verbindungsverwaltung, indem sie den Disconnect-Callback überwacht und die Verbindung wieder herstellt. Wenn die Verbindung unterbrochen wird, unterbricht die Demo zunächst den Betrieb des OTA-Agenten und versucht dann, die MQTT-Verbindung wiederherzustellen. Die MQTT-Wiederverbindungsversuche werden um eine Zeit verzögert, die exponentiell bis zu einem Höchstwert erhöht wird, und ein Jitter wird ebenfalls hinzugefügt. Wenn die Verbindung wieder hergestellt wird, setzt der OTA-Agent seinen Betrieb fort.

Ein funktionierendes Beispiel, das den AWS IoT MQTT-Broker verwendet, finden Sie im OTA-Democode im `demos/ota` Verzeichnis.

Da der OTA-Agent ein eigener Task ist, betrifft die absichtliche Verzögerung von einer Sekunde in diesem Beispiel nur diese Anwendung. Sie hat keinen Einfluss auf die Leistung des Agents.

```
static BaseType_t prvRunOTADemo( void )
{
    /* Status indicating a successful demo or not. */
    BaseType_t xStatus = pdFAIL;

    /* OTA library return status. */
    OtaErr_t xOtaError = OtaErrUninitialized;

    /* OTA event message used for sending event to OTA Agent.*/
    OtaEventMsg_t xEventMsg = { 0 };

    /* OTA interface context required for library interface functions.*/
    OtaInterfaces_t xOtaInterfaces;

    /* OTA library packet statistics per job.*/
    OtaAgentStatistics_t xOtaStatistics = { 0 };
```

```
    /* OTA Agent state returned from calling OTA_GetState.*/
    OtaState_t xOtaState = OtaAgentStateStopped;

    /* Set OTA Library interfaces.*/
    prvSetOtaInterfaces( &xOtaInterfaces );

    /************************** Init OTA Library. **************************/

    if( ( xOtaError = OTA_Init( &xOtaBuffer,
                                &xOtaInterfaces,
                                ( const uint8_t * ) ( democonfigCLIENT_IDENTIFIER ),
                                prvOtaAppCallback ) ) != OtaErrNone )
    {
        LogError( ( "Failed to initialize OTA Agent, exiting = %u.",
                    xOtaError ) );
    }
    else
    {
        xStatus = pdPASS;
    }

    /************************ Create OTA Agent Task. ************************/

    if( xStatus == pdPASS )
    {
        xStatus = xTaskCreate( prvOTAAgentTask,
                               "OTA Agent Task",
                               otaexampleAGENT_TASK_STACK_SIZE,
                               NULL,
                               otaexampleAGENT_TASK_PRIORITY,
                               NULL );

        if( xStatus != pdPASS )
        {
            LogError( ( "Failed to create OTA agent task:" ) );
        }
    }

    /***************************** Start OTA *****************************/

    if( xStatus == pdPASS )
    {
        /* Send start event to OTA Agent.*/
```

```
        xEventMsg.eventId = OtaAgentEventStart;
        OTA_SignalEvent( &xEventMsg );
    }

    /******************* Loop and display OTA statistics ******************/

    if( xStatus == pdPASS )
    {
        while( ( xOtaState = OTA_GetState() ) != OtaAgentStateStopped )
        {
            /* Get OTA statistics for currently executing job. */
            if( xOtaState != OtaAgentStateSuspended )
            {
                OTA_GetStatistics( &xOtaStatistics );

                LogInfo( ( " Received: %u   Queued: %u   Processed: %u   Dropped: %u",
                           xOtaStatistics.otaPacketsReceived,
                           xOtaStatistics.otaPacketsQueued,
                           xOtaStatistics.otaPacketsProcessed,
                           xOtaStatistics.otaPacketsDropped ) );
            }

            vTaskDelay( pdMS_TO_TICKS( otaexampleEXAMPLE_TASK_DELAY_MS ) );
        }
    }

    return xStatus;
}
```

Hier ist der High-Level-Ablauf der Demo-Anwendung:

- Erstellen Sie einen MQTT-Agent-Kontext.
- Connect zu Ihrem AWS IoT Endpunkt her.
- Initialisieren Sie den OTA-Agent.
- Loop, der einen OTA-Aktualisierungsjob ermöglicht und einmal pro Sekunde Statistiken ausgibt.
- Wenn MQTT die Verbindung trennt, unterbrechen Sie den Betrieb des OTA-Agenten.
- Versuchen Sie erneut, eine Verbindung mit exponentieller Verzögerung und Jitter herzustellen.
- Wenn die Verbindung wieder hergestellt ist, setzen Sie den Betrieb des OTA-Agenten fort.
- Wenn der Agent stoppt, warten Sie eine Sekunde und versuchen Sie dann erneut, die Verbindung herzustellen.

Verwenden Sie den Anwendungsrückruf für OTA-Agent-Ereignisse

Das vorherige Beispiel wurde `prvOtaAppCallback` als Callback-Handler für OTA-Agent-Ereignisse verwendet. (Siehe den vierten Parameter des `OTA_Init` API-Aufrufs). Wenn Sie eine benutzerdefinierte Behandlung der Abschlussereignisse implementieren möchten, müssen Sie die Standardbehandlung in der OTA-Demo/Anwendung ändern. Während des OTA-Prozesses kann der OTA-Agent eine der folgenden Ereignisaufzählungen an den Callback-Handler senden. Der Anwendungsentwickler entscheidet, wie und wann er diese Ereignisse verarbeitet.

```
/**
 * @ingroup ota_enum_types
 * @brief OTA Job callback events.
 *
 * After an OTA update image is received and authenticated, the agent calls the user
 * callback (set with the @ref OTA_Init API) with the value OtaJobEventActivate to
 * signal that the device must be rebooted to activate the new image. When the device
 * boots, if the OTA job status is in self test mode, the agent calls the user callback
 * with the value OtaJobEventStartTest, signaling that any additional self tests
 * should be performed.
 *
 * If the OTA receive fails for any reason, the agent calls the user callback with
 * the value OtaJobEventFail instead to allow the user to log the failure and take
 * any action deemed appropriate by the user code.
 *
 * See the OtaImageState_t type for more information.
 */
typedef enum OtaJobEvent
{
    OtaJobEventActivate = 0,       /*!< @brief OTA receive is authenticated and ready
 to activate. */
    OtaJobEventFail = 1,           /*!< @brief OTA receive failed. Unable to use this
 update. */
    OtaJobEventStartTest = 2,      /*!< @brief OTA job is now in self test, perform
 user tests. */
    OtaJobEventProcessed = 3,      /*!< @brief OTA event queued by OTA_SignalEvent is
 processed. */
    OtaJobEventSelfTestFailed = 4, /*!< @brief OTA self-test failed for current job. */
    OtaJobEventParseCustomJob = 5, /*!< @brief OTA event for parsing custom job
 document. */
    OtaJobEventReceivedJob = 6,    /*!< @brief OTA event when a new valid AFT-OTA job
 is received. */
    OtaJobEventUpdateComplete = 7, /*!< @brief OTA event when the update is completed.
 */
```

```
    OtaLastJobEvent = OtaJobEventStartTest
} OtaJobEvent_t;
```

Der OTA-Agent kann während der aktiven Verarbeitung der Hauptanwendung im Hintergrund ein Update erhalten. Der Zweck der Bereitstellung dieser Ereignisse ist es, der Anwendung eine Entscheidung darüber zu ermöglichen, ob Maßnahmen sofort ergriffen werden können oder ob sie auf den Zeitpunkt nach Abschluss einer anderen anwendungsspezifischen Verarbeitung verschoben werden sollen. Dadurch wird eine unvorhergesehene Unterbrechung Ihres Gerätes während der aktiven Verarbeitung verhindert, die durch einen Reset nach einem Firmware-Update verursacht würde. Hier sind die Jobereignisse, die der Callback-Handler erhält:

`OtaJobEventActivate`

Wenn der Callback-Handler dieses Ereignis empfängt, können Sie das Gerät entweder sofort zurücksetzen oder einen Anruf planen, um das Gerät zu einem späteren Zeitpunkt zurückzusetzen. Hierdurch können Sie die Gerätezurücksetzung und die Selbsttestphase verschieben, wenn notwendig.

`OtaJobEventFail`

Wenn der Callback-Handler dieses Ereignis empfängt, ist die Aktualisierung fehlgeschlagen. In diesem Fall brauchen Sie nichts zu tun. Möglicherweise möchten Sie eine Protokollnachricht ausgeben oder etwas Anwendungsspezifisches tun.

`OtaJobEventStartTest`

Die Selbsttestphase soll es der neu aktualisierten Firmware ermöglichen, sich selbst auszuführen und zu testen, bevor sie feststellt, ob sie ordnungsgemäß funktioniert, und sich als aktuelles permanentes Anwendungs-Image festlegt. Wenn ein neues Update empfangen und authentifiziert wird und das Gerät zurückgesetzt wurde, sendet der OTA-Agent das Ereignis `OtaJobEventStartTest` an die Callback-Funktion, wenn das Update für den Test bereit ist. Der Entwickler kann alle erforderlichen Tests hinzufügen, um zu ermitteln, ob die Gerätefirmware nach der Aktualisierung korrekt funktioniert. Wenn die Gerätefirmware aufgrund der Selbsttests als zuverlässig identifiziert wird, muss der Code die Firmware durch Aufruf der Funktion `OTA_SetImageState( OtaImageStateAccepted )` als neues permanentes Image übergeben.

`OtaJobEventProcessed`

Das OTA-Ereignis in der Warteschlange `OTA_SignalEvent` wird verarbeitet, sodass Bereinigungsvorgänge wie das Freigeben der OTA-Puffer durchgeführt werden können.

`OtaJobEventSelfTestFailed`

Der OTA-Selbsttest ist für den aktuellen Job fehlgeschlagen. Die Standardbehandlung für dieses Ereignis besteht darin, den OTA-Agenten herunterzufahren und neu zu starten, sodass das Gerät zum vorherigen Image zurückkehrt.

`OtaJobEventUpdateComplete`

Das Benachrichtigungsereignis für den Abschluss der Aktualisierung des OTA-Jobs.

OTA-Sicherheit

Im Folgenden sind drei Aspekte der over-the-air (OTA-) Sicherheit aufgeführt:

Verbindungssicherheit

Der OTA Update Manager-Service stützt sich auf vorhandene Sicherheitsmechanismen, z. B. die gegenseitige Authentifizierung von Transport Layer Security (TLS), die von AWS IoT verwendet werden. Der OTA-Aktualisierungsverkehr durchläuft das AWS IoT Geräte-Gateway und verwendet AWS IoT Sicherheitsmechanismen. Jede über das Geräte-Gateway ein- und ausgehende HTTP- oder MQTT-Nachricht wird einer strengen Authentifizierung und Autorisierung unterzogen.

Authentizität und Integrität von OTA-Updates

Die Firmware kann vor einem OTA-Update digital signiert werden. Dies stellt sicher, dass sie aus einer zuverlässigen Quelle stammt und nicht manipuliert wurde.

Der FreeRTOS OTA Update Manager-Dienst verwendet Code Signing for AWS IoT , um Ihre Firmware automatisch zu signieren. Weitere Informationen finden Sie unter Code Signing for AWS IoT.

Der auf Ihrem Gerät ausgeführte OTA-Agent führt Integritätsprüfungen der Firmware durch, wenn sie auf dem Gerät ankommt.

Operator-Sicherheit

Jeder API-Aufruf, der über die API der Kontrollebene getätigt wird, wird der Standardauthentifizierung und -autorisierung mit IAM Signature Version 4 unterzogen. Um eine Bereitstellung zu erstellen, benötigen Sie Berechtigungen zum Aufrufen von`CreateDeployment`, und`CreateJob`. `CreateStream` APIs Darüber hinaus müssen Sie in Ihrer Amazon S3-Bucket-Richtlinie oder ACL dem AWS IoT Service Principal Leseberechtigungen erteilen, damit während des Streamings auf das in Amazon S3 gespeicherte Firmware-Update zugegriffen werden kann.

Codesignatur für AWS IoT

Die AWS IoT Konsole verwendet Code Signing for AWS IoT, um Ihr Firmware-Image für jedes Gerät, das von unterstützt wird, automatisch zu signieren AWS IoT.

Code Signing for AWS IoT verwendet ein Zertifikat und einen privaten Schlüssel, die Sie in ACM importieren. Sie können ein selbstsigniertes Zertifikat zum Testen verwenden, wir empfehlen Ihnen jedoch, ein Zertifikat von einer bekannten kommerziellen Zertifizierungsstelle (CA) zu erwerben.

Codesignaturzertifikate verwenden die X.509-Version 3 und deren Erweiterungen. `Key Usage` `Extended Key Usage` Die Erweiterung `Key Usage` ist auf `Digital Signature` festgelegt. Die Erweiterung `Extended Key Usage` ist auf `Code Signing` festgelegt. Weitere Informationen zum Signieren Ihres Code-Images finden Sie im Code Signing for AWS IoT Developer Guide und in der Code Signing for AWS IoT API-Referenz.

> **Note**
>
> Sie können das Code Signing for AWS IoT SDK von Tools for Amazon Web Services herunterladen.

OTA-Fehlerbehebung

Die folgenden Abschnitte enthalten Informationen, die Ihnen bei der Behebung von Problemen mit OTA-Updates helfen.

Themen

- Richten Sie CloudWatch Protokolle für OTA-Updates ein
- AWS IoT OTA-API-Aufrufe protokollieren mit AWS CloudTrail
- Rufen Sie die OTAUpdate Fehlerdetails von Create mithilfe von AWS CLI
- Holen Sie sich OTA-Fehlercodes mit dem AWS CLI
- Problembehandlung bei OTA-Updates für mehrere Geräte
- Beheben Sie Probleme mit OTA-Updates mit dem Texas Instruments SF Launchpad CC3220

Richten Sie CloudWatch Protokolle für OTA-Updates ein

Der OTA Update Service unterstützt die Protokollierung bei Amazon CloudWatch. Sie können die AWS IoT Konsole verwenden, um die CloudWatch Amazon-Protokollierung für OTA-Updates zu aktivieren und zu konfigurieren. Weitere Informationen finden Sie unter Cloudwatch Logs.

Um die Protokollierung zu aktivieren, müssen Sie eine IAM-Rolle erstellen und die OTA-Aktualisierungsprotokollierung konfigurieren.

> **Note**
>
> Bevor Sie die OTA-Aktualisierungsprotokollierung aktivieren, stellen Sie sicher, dass Sie die Zugriffsberechtigungen für CloudWatch Protokolle verstanden haben. Benutzer mit Zugriff auf CloudWatch Protokolle können Ihre Debugging-Informationen sehen. Weitere Informationen finden Sie unter Authentifizierung und Zugriffskontrolle für Amazon CloudWatch Logs.

Erstellen einer Protokollierungsrolle und Aktivieren der Protokollierung

Verwenden Sie die AWS IoT -Konsole, um eine Protokollierungsrolle zu erstellen und die Protokollierung zu aktivieren.

1. Wählen Sie im Navigationsbereich Settings (Einstellungen) aus.
2. Wählen Sie unter Logs (Protokolle) die Option Edit (Bearbeiten) aus.
3. Wählen Sie unter Level of verbosity (Umfang) die Option Debug (Debuggen) aus.
4. Wählen Sie unter Rolle festlegen die Option Neu erstellen aus, um eine IAM-Rolle für die Protokollierung zu erstellen.
5. Geben Sie unter Name einen eindeutigen Namen für Ihre Rolle ein. Ihre Rolle wird mit allen erforderlichen Berechtigungen erstellt.
6. Wählen Sie Aktualisieren aus.

OTA-Update-Protokolle

Der OTA-Update-Service veröffentlicht Protokolle für Ihr Konto, wenn einer der folgenden Fälle eintritt:

- Ein OTA-Update wird erstellt.
- Ein OTA-Update ist abgeschlossen.

- Ein Code-Signing-Job wird erstellt.
- Ein Code-Signing-Job ist abgeschlossen.
- Ein AWS IoT Job wird erstellt.
- Ein AWS IoT Job ist abgeschlossen.
- Ein Stream wird erstellt.

Sie können die Protokolle in der CloudWatch -Konsole anzeigen.

Um ein OTA-Update in CloudWatch Logs anzuzeigen

1. Wählen Sie im Navigationsbereich Logs (Protokolle) aus.
2. Wählen Sie unter Protokollgruppen die Option AWSIoTLogsV2.

OTA-Update-Protokolle können die folgenden Eigenschaften haben:

accountId

Die AWS Konto-ID, in der das Protokoll generiert wurde.

actionType

Die Aktion, die das Protokoll erzeugt hat. Diese Eigenschaft kann einen der folgenden Werte haben:

- `CreateOTAUpdate`: Es wurde ein OTA-Update erstellt.
- `DeleteOTAUpdate`: Es wurde ein OTA-Update gelöscht.
- `StartCodeSigning`: Es wurde eine Codesignierungsaufgabe gestartet.
- `CreateAWSJob`: Ein AWS IoT Job wurde erstellt.
- `CreateStream`: Es wurde ein Stream erstellt.
- `GetStream`: Eine Anfrage für einen Stream wurde an die AWS IoT MQTT-basierte Dateibereitstellungsfunktion gesendet.
- `DescribeStream`: Eine Anfrage nach Informationen über einen Stream wurde an die AWS IoT MQTT-basierte Dateilieferfunktion gesendet.

awsJobId

Die AWS IoT Job-ID, die das Protokoll generiert hat.

clientId

Die MQTT-Client-ID, die die Anfrage aus dem Protokolleintrag gestellt hat.

clientToken

Das Client-Token, das der Anforderung im Protokolleintrag zugeordnet ist.

Details

Zusätzliche Informationen über den Vorgang, der das Protokoll generiert hat.

logLevel

Die Protokollierungsstufe des Protokolls. Für OTA-Update-Protokolle ist diese Eigenschaft stets auf `DEBUG` festgelegt.

otaUpdateId

Die ID des OTA-Updates, das den Protokolleintrag generiert hat.

Protokoll

Das Protokoll, mit dem die Anforderung gestellt wurde, die den Protokolleintrag generiert hat.

Status

Der Status des Vorgangs, der den Protokolleintrag generiert hat. Folgende sind gültige Werte:

- Herzlichen Glückwunsch
- Fehler

streamId

Die AWS IoT Stream-ID, die das Protokoll generiert hat.

Zeitstempel

Der Zeitpunkt, zu dem der Protokolleintrag generiert wurde.

topicName

Ein MQTT-Thema, mit dem die Anforderung gestellt wurde, die den Protokolleintrag generiert hat.

Beispielprotokolle

Im Folgenden finden Sie ein Beispielprotokoll, das beim Starten eines Code-Signing-Jobs erzeugt wird:

```
{
    "timestamp": "2018-07-23 22:59:44.955",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "actionType": "StartCodeSigning",
    "otaUpdateId": "08957b03-eea3-448a-87fe-743e6891ca3a",
    "details": "Start code signing job. The request status is SUCCESS."
}
```

Im Folgenden finden Sie ein Beispielprotokoll, das bei der Erstellung eines AWS IoT Jobs generiert wird:

```
{
    "timestamp": "2018-07-23 22:59:45.363",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "actionType": "CreateAWSJob",
    "otaUpdateId": "08957b03-eea3-448a-87fe-743e6891ca3a",
    "awsJobId": "08957b03-eea3-448a-87fe-743e6891ca3a",
    "details": "Create AWS Job The request status is SUCCESS."
}
```

Im Folgenden finden Sie ein Beispielprotokoll, das beim Erstellen eines OTA-Updates erzeugt wird:

```
{
    "timestamp": "2018-07-23 22:59:45.413",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "actionType": "CreateOTAUpdate",
    "otaUpdateId": "08957b03-eea3-448a-87fe-743e6891ca3a",
    "details": "OTAUpdate creation complete. The request status is SUCCESS."
}
```

Im Folgenden finden Sie ein Beispielprotokoll, das beim Erstellen eines Streams erzeugt wird:

```
{
    "timestamp": "2018-07-23 23:00:26.391",
    "logLevel": "DEBUG",
```

```
    "accountId": "123456789012",
    "status": "Success",
    "actionType": "CreateStream",
    "otaUpdateId": "3d3dc5f7-3d6d-47ac-9252-45821ac7cfb0",
    "streamId": "6be2303d-3637-48f0-ace9-0b87b1b9a824",
    "details": "Create stream. The request status is SUCCESS."
}
```

Im Folgenden finden Sie ein Beispielprotokoll, das beim Löschen eines OTA-Updates erzeugt wird:

```
{
    "timestamp": "2018-07-23 23:03:09.505",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "actionType": "DeleteOTAUpdate",
    "otaUpdateId": "9bdd78fb-f113-4001-9675-1b595982292f",
    "details": "Delete OTA Update. The request status is SUCCESS."
}
```

Das Folgende ist ein Beispielprotokoll, das generiert wird, wenn ein Gerät einen Stream von der MQTT-basierten Dateizustellungsfunktion anfordert:

```
{
    "timestamp": "2018-07-25 22:09:02.678",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "actionType": "GetStream",
    "protocol": "MQTT",
    "clientId": "b9d2e49c-94fe-4ed1-9b07-286afed7e4c8",
    "topicName": "$aws/things/b9d2e49c-94fe-4ed1-9b07-286afed7e4c8/
streams/1e51e9a8-9a4c-4c50-b005-d38452a956af/get/json",
    "streamId": "1e51e9a8-9a4c-4c50-b005-d38452a956af",
    "details": "The request status is SUCCESS."
}
```

Im Folgenden finden Sie ein Beispielprotokoll, das erzeugt wird, wenn ein Gerät die `DescribeStream`-API aufruft:

```
{
```

```
    "timestamp": "2018-07-25 22:10:12.690",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "actionType": "DescribeStream",
    "protocol": "MQTT",
    "clientId": "581075e0-4639-48ee-8b94-2cf304168e43",
    "topicName": "$aws/things/581075e0-4639-48ee-8b94-2cf304168e43/streams/71c101a8-
bcc5-4929-9fe2-af563af0c139/describe/json",
    "streamId": "71c101a8-bcc5-4929-9fe2-af563af0c139",
    "clientToken": "clientToken",
    "details": "The request status is SUCCESS."
}
```

AWS IoT OTA-API-Aufrufe protokollieren mit AWS CloudTrail

FreeRTOS ist in einen Service integriert CloudTrail, der AWS IoT OTA-API-Aufrufe erfasst und die Protokolldateien an einen von Ihnen angegebenen Amazon S3 S3-Bucket übermittelt. CloudTrail erfasst API-Aufrufe von Ihrem Code an den AWS IoT OTA. APIs Anhand der von gesammelten Informationen können Sie die Anfrage CloudTrail, die an AWS IoT OTA gestellt wurde, die Quell-IP-Adresse, von der aus die Anfrage gestellt wurde, wer die Anfrage gestellt hat, wann sie gestellt wurde usw. ermitteln.

Weitere Informationen dazu CloudTrail, einschließlich der Konfiguration und Aktivierung, finden Sie im AWS CloudTrail Benutzerhandbuch.

FreeRTOS-Informationen in CloudTrail

Wenn die CloudTrail Protokollierung in Ihrem AWS Konto aktiviert ist, werden API-Aufrufe von AWS IoT OTA-Aktionen in CloudTrail Protokolldateien aufgezeichnet, wo sie zusammen mit anderen AWS Serviceaufzeichnungen geschrieben werden. CloudTrail bestimmt anhand eines Zeitraums und der Dateigröße, wann eine neue Datei erstellt und in sie geschrieben werden soll.

Die folgenden Aktionen auf der AWS IoT OTA-Steuerungsebene werden protokolliert von CloudTrail:

- CreateStream
- DescribeStream
- ListStreams
- UpdateStream
- DeleteStream

- CreateOTAUpdate
- Holen OTAUpdate
- ListeOTAUpdates
- LöschenOTAUpdate

> **Note**
>
> AWS IoT Aktionen auf der OTA-Datenebene (geräteseitig) werden nicht protokolliert CloudTrail. Wird verwendet CloudWatch , um diese zu überwachen.

Jeder Protokolleintrag enthält Informationen über den Ersteller der Anforderung. Der Benutzeridentität im Protokolleintrag können Sie folgende Informationen entnehmen:

- Gibt an, ob die Anforderung mit Root- oder IAM-Benutzer-Anmeldeinformationen ausgeführt wurde.
- Gibt an, ob die Anforderung mit temporären Sicherheitsanmeldeinformationen für eine Rolle oder einen Verbundbenutzer gesendet wurde.
- Ob die Anfrage von einem anderen AWS Dienst gestellt wurde.

Weitere Informationen finden Sie unter dem CloudTrail UserIdentity-Element. AWS IoT OTA-Aktionen sind in der AWS IoT OTA-API-Referenz dokumentiert.

Sie können Ihre Protokolldateien so lange in Ihrem Amazon S3 S3-Bucket speichern, wie Sie möchten, aber Sie können auch Amazon S3 S3-Lebenszyklusregeln definieren, um Protokolldateien automatisch zu archivieren oder zu löschen. Standardmäßig werden die Protokolldateien mit serverseitiger Amazon-S3-Verschlüsselung (SSE) verschlüsselt.

Wenn Sie benachrichtigt werden möchten, wenn Protokolldateien zugestellt werden, können Sie die Veröffentlichung von Amazon SNS SNS-Benachrichtigungen konfigurieren CloudTrail . Weitere Informationen finden Sie unter Konfiguration von Amazon SNS SNS-Benachrichtigungen für CloudTrail.

Sie können auch AWS IoT OTA-Protokolldateien aus mehreren AWS Regionen und mehreren AWS Konten in einem einzigen Amazon S3 S3-Bucket zusammenfassen.

Weitere Informationen finden Sie unter Empfangen von CloudTrail Protokolldateien aus mehreren Regionen und Empfangen von CloudTrail Protokolldateien von mehreren Konten.

FreeRTOS-Logdateieinträge verstehen

CloudTrail Protokolldateien können einen oder mehrere Protokolleinträge enthalten. In jedem Eintrag werden mehrere Ereignisse im JSON-Format aufgelistet. Ein Protokolleintrag stellt eine einzelne Anforderung aus einer beliebigen Quelle dar und enthält unter anderem Informationen über die angeforderte Aktion, das Datum und die Uhrzeit der Aktion sowie über die Anforderungsparameter. Protokolleinträge sind kein geordnetes Stacktrace der öffentlichen API-Aufrufe und erscheinen daher nicht in einer bestimmten Reihenfolge.

Das folgende Beispiel zeigt einen CloudTrail Protokolleintrag, der das Protokoll eines `CreateOTAUpdate` Aktionsaufrufs veranschaulicht.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EXAMPLE",
        "arn": "arn:aws:iam::your_aws_account:user/your_user_id",
        "accountId": "your_aws_account",
        "accessKeyId": "your_access_key_id",
        "userName": "your_username",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2018-08-23T17:27:08Z"
            }
        },
        "invokedBy": "apigateway.amazonaws.com"
    },
    "eventTime": "2018-08-23T17:27:19Z",
    "eventSource": "iot.amazonaws.com",
    "eventName": "CreateOTAUpdate",
    "awsRegion": "your_aws_region",
    "sourceIPAddress": "apigateway.amazonaws.com",
    "userAgent": "apigateway.amazonaws.com",
    "requestParameters": {
        "targets": [
            "arn:aws:iot:your_aws_region:your_aws_account:thing/Thing_CMH"
        ],
        "roleArn": "arn:aws:iam::your_aws_account:role/Role_FreeRTOSJob",
        "files": [
            {
                "fileName": "/sys/mcuflashimg.bin",
```

```
                "fileSource": {
                    "fileId": 0,
                    "streamId": "your_stream_id"
                },
                "codeSigning": {
                    "awsSignerJobId": "your_signer_job_id"
                }
            }
        ],
        "targetSelection": "SNAPSHOT",
        "otaUpdateId": "FreeRTOSJob_CMH-23-1535045232806-92"
    },
    "responseElements": {
        "otaUpdateArn": "arn:aws:iot:your_aws_region:your_aws_account:otaupdate/
FreeRTOSJob_CMH-23-1535045232806-92",
        "otaUpdateStatus": "CREATE_PENDING",
        "otaUpdateId": "FreeRTOSJob_CMH-23-1535045232806-92"
    },
    "requestID": "c9649630-a6f9-11e8-8f9c-e1cf2d0c9d8e",
    "eventID": "ce9bf4d9-5770-4cee-acf4-0e5649b845c0",
    "eventType": "AwsApiCall",
    "recipientAccountId": "recipient_aws_account"
}
```

Rufen Sie die OTAUpdate Fehlerdetails von Create mithilfe von AWS CLI

Wenn das Erstellen eines OTA-Aktualisierungsauftrags fehlschlägt, können Sie möglicherweise Maßnahmen ergreifen, um das Problem zu beheben. Wenn Sie einen OTA-Aktualisierungsauftrag erstellen, erstellt der OTA-Manager-Service einen IoT-Job und plant ihn für die Zielgeräte. Dieser Prozess erstellt oder verwendet auch andere Arten von AWS Ressourcen in Ihrem Konto (einen Codesignaturauftrag, einen AWS IoT Stream, ein Amazon S3 S3-Objekt). Jeder aufgetretene Fehler kann dazu führen, dass der Prozess fehlschlägt, ohne dass ein Job erstellt wird AWS IoT . In diesem Abschnitt zur Fehlerbehebung finden Sie Anweisungen zum Abrufen der Details des Fehlers.

1. Installieren und Konfigurieren der AWS CLI.
2. Führen Sie das `aws configure` Programm aus und geben Sie die folgenden Informationen ein.

```
$ aws configure
AWS Access Key ID [None]: AccessID
AWS Secret Access Key [None]: AccessKey
Default region name [None]: Region
```

```
Default output format [None]: json
```

Weitere Informationen finden Sie unter Schnellkonfiguration mit `aws configure`.

3. Führen Sie Folgendes aus:

```
aws iot get-ota-update --ota-update-id ota_update_job_001
```

Wo *ota_update_job_001* ist die ID, die Sie dem OTA-Update bei der Erstellung gegeben haben?

4. Die Ausgabe sieht etwa wie folgt aus:

```
{
    "otaUpdateInfo": {
        "otaUpdateId": "ota_update_job_001",
        "otaUpdateArn":
 "arn:aws:iot:region:account_id:otaupdate/ota_update_job_001",
        "creationDate": 1584646864.534,
        "lastModifiedDate": 1584646865.913,
        "targets": [
            "arn:aws:iot:region:account_id:thing/thing_001"
        ],
        "protocols": [
            "MQTT"
        ],
        "awsJobExecutionsRolloutConfig": {},
        "awsJobPresignedUrlConfig": {},
        "targetSelection": "SNAPSHOT",
        "otaUpdateFiles": [
            {
               "fileName": "/12ds",
                "fileLocation": {
                    "s3Location": {
                        "bucket": "bucket_name",
                        "key": "demo.bin",
                        "version": "Z7X.TWSAS7JSi4rybc02nMdcE41W1tV3"
                    }
                },
                "codeSigning": {
                    "startSigningJobParameter": {
                        "signingProfileParameter": {},
                        "signingProfileName": "signing_profile_name",
```

```
                        "destination": {
                            "s3Destination": {
                                "bucket": "bucket_name",
                                "prefix": "SignedImages/"
                            }
                        }
                    },
                    "customCodeSigning": {}
                }
            }
        ],
        "otaUpdateStatus": "CREATE_FAILED",
        "errorInfo": {
            "code": "AccessDeniedException",
            "message": "S3 object demo.bin not accessible. Please check
 your permissions (Service: AWSSigner; Status Code: 403; Error Code:
 AccessDeniedException; Request ID: 01d8e7a1-8c7c-4d85-9fd7-dcde975fdd2d)"
        }
    }
}
```

Wenn die Erstellung fehlgeschlagen ist, enthält das `otaUpdateStatus` Feld in der Befehlsausgabe die Details des Fehlers `CREATE_FAILED` und das `errorInfo` Feld enthält die Details des Fehlers.

Holen Sie sich OTA-Fehlercodes mit dem AWS CLI

1. Installieren und Konfigurieren der AWS CLI.
2. Führen Sie den `aws configure` Vorgang aus und geben Sie die folgenden Informationen ein.

```
$ aws configure
AWS Access Key ID [None]: AccessID
AWS Secret Access Key [None]: AccessKey
Default region name [None]: Region
Default output format [None]: json
```

Weitere Informationen finden Sie unter Schnellkonfiguration mit `aws configure`.

3. Führen Sie Folgendes aus:

```
aws iot describe-job-execution --job-id JobID --thing-name ThingName
```

Wo *JobID* ist die vollständige Job-ID-Zeichenfolge für den Job, dessen Status wir abrufen möchten (er wurde dem OTA-Aktualisierungsauftrag zugeordnet, als er erstellt wurde), und *ThingName* ist der AWS IoT Dingname, unter dem das Gerät registriert ist AWS IoT

4. Die Ausgabe sieht etwa wie folgt aus:

```
{
    "execution": {
        "jobId": "AFR_OTA-****************",
        "status": "FAILED",
        "statusDetails": {
            "detailsMap": {
                "reason": "0xEEEEEEEE: 0xffffffff"
            }
        },
        "thingArn": "arn:aws:iot:Region:AccountID:thing/ThingName",
        "queuedAt": 1569519049.9,
        "startedAt": 1569519052.226,
        "lastUpdatedAt": 1569519052.226,
        "executionNumber": 1,
        "versionNumber": 2
    }
}
```

In dieser Beispielausgabe hat „`reason`“ in der „`detailsmap`“ zwei Felder: Das Feld „0xEEEEEEEE“ enthält den generischen Fehlercode des OTA-Agenten; das Feld „0xffffffff“ enthält den Untercode. Die generischen Fehlercodes sind in https://docs.aws.amazon.com/freertos/latest/lib-ref/html1/aws__ota__agent_8h.html aufgeführt. Siehe Fehlercodes mit dem Präfix „`kOTA_Err_`“. Der Untercode kann ein plattformspezifischer Code sein oder weitere Details zum generischen Fehler enthalten.

Problembehandlung bei OTA-Updates für mehrere Geräte

Um OTAs auf mehreren Geräten (Dingen) mit demselben Firmware-Image zu arbeiten, implementieren Sie eine Funktion (z. B.`getThingName()`), die Daten `clientcredentialIOT_THING_NAME` aus einem nichtflüchtigen Speicher abruft. Stellen Sie sicher, dass diese Funktion den Namen des Objekts aus einem Teil des nicht flüchtigen Speichers ausliest, der beim OTA-Vorgang nicht überschrieben wird, und dass der Name des Objekts vor dem Ausführen des ersten Auftrags bereitgestellt wird. Bei Verwendung der Just-in-Time-Paketerstellung

(JITP) können Sie den Namen des Objekts aus dem allgemeinen Namen Ihres Gerätezertifikats auslesen.

Beheben Sie Probleme mit OTA-Updates mit dem Texas Instruments SF Launchpad CC3220

Die CC3220 SF Launchpad-Plattform bietet einen Mechanismus zur Erkennung von Softwaremanipulationen. Sie verwendet einen Sicherheitswarnungszähler, der bei einer Integritätsverletzung inkrementiert wird. Wenn der Sicherheitswarnungszähler einen vorab festgelegten Schwellenwert erreicht (der Standardwert ist 15) und der Host das asynchrone Ereignis `SL_ERROR_DEVICE_LOCKED_SECURITY_ALERT` empfängt, wird das Gerät gesperrt. Auf das gesperrte Gerät kann nur begrenzt zugegriffen werden. Um das Gerät wiederherzustellen, können Sie es neu programmieren oder den restore-to-factory Vorgang verwenden, um zum Werksabbild zurückzukehren. Sie sollten das gewünschte Verhalten programmieren, indem Sie den Handler für asynchrone Ereignisse in `network_if.c` aktualisieren.

FreeRTOS-Bibliotheken

FreeRTOS-Bibliotheken bieten zusätzliche Funktionen für den FreeRTOS-Kernel und seine internen Bibliotheken. Sie können FreeRTOS-Bibliotheken für Netzwerke und Sicherheit in eingebetteten Anwendungen verwenden. FreeRTOS-Bibliotheken ermöglichen es Ihren Anwendungen auch, mit AWS IoT Diensten zu interagieren. FreeRTOS enthält Bibliotheken, die Folgendes ermöglichen:

- Verbinden Sie Geräte mithilfe von MQTT und Device Shadows sicher mit der AWS IoT Cloud.
- Entdecken Sie AWS IoT Greengrass Kerne und stellen Sie eine Verbindung zu ihnen her.
- Verwalten von WLAN-Verbindungen
- Achten auf und Verarbeiten von FreeRTOS RTOS-Updates Over-the-Air.

Das `libraries` Verzeichnis enthält den Quellcode der FreeRTOS-Bibliotheken. Es gibt Helferfunktionen, die die Implementierung der Bibliotheksfunktionalität unterstützen. Wir empfehlen, dass Sie diese Helferfunktionen nicht ändern.

FreeRTOS-Portierungsbibliotheken

Die folgenden Portierungsbibliotheken sind in Konfigurationen von FreeRTOS enthalten, die auf der FreeRTOS-Konsole heruntergeladen werden können. Diese Bibliotheken sind plattformabhängig. Ihre Inhalte ändern sich entsprechend Ihrer Hardwareplattform. Informationen zur Portierung dieser Bibliotheken auf ein Gerät finden Sie im FreeRTOS Porting Guide.

FreeRTOS-Portierungsbibliotheken

Bibliothek	API-Referenz	Description
Bluetooth Low Energy	API-Referenz Bluetooth Low Energy (BLE)	Mithilfe der FreeRTOS Bluetooth Low Energy-Bibliothek kann Ihr Mikrocontroller über ein Gateway-Gerät mit dem AWS IoT MQTT-Broker kommunizieren. Weitere Informationen finden Sie unter Bluetooth Low Energy-Bibliothek.
Over-the-Air Aktualisierungen	AWS IoT Over-the-air API-Referenz aktualisieren	Mit der FreeRTOS AWS IoT Over-the-air (OTA) -Update-Bibliothek können Sie Update-Benachrichtigungen verwalten, Updates herunterladen und Firmware-Updates auf Ihrem FreeRTOS-Gerät kryptografisch verifizieren. Weitere Informationen finden Sie unter AWS IoT Over-the-Air-Bibliothek (OTA).
FreeRTOS+POSIX	API-Referenz für FreeRTOS+POSIX	Sie können die FreeRTOS+POSIX-Bibliothek verwenden, um POSIX-konforme Anwendungen in das FreeRTOS-Ökosystem zu portieren. Weitere Informationen finden Sie unter FreeRTOS+POSIX.
Secure Sockets	Secure Sockets API-Referenz	Weitere Informationen finden Sie unter Secure-Sockets-Bibliothek.
FreeRTOS+TCP	FreeRTOS+TCP API-Referenz	FreeRTOS+TCP ist ein skalierbarer, quelloffener und threadsicherer TCP/IP Stack für FreeRTOS.

Bibliothek	API-Referenz	Description
		Weitere Informationen finden Sie unter FreeRTOS+TCP.
WLAN	API-Referenz für WLAN	Mit der FreeRTOS Wi-Fi-Bibliothek können Sie eine Schnittstelle zum Wireless-Stack auf niedriger er Ebene Ihres Mikrocontrollers herstellen. Weitere Informationen hierzu finden Sie unter WLAN-Bibliothek.
Kern PKCS11		Die PKCS11 Kernbibliothek ist eine Referenzimplementierung des Public Key Cryptography Standard #11 zur Unterstützung von Bereitste llung und TLS-Client-Authent ifizierung. Weitere Informationen hierzu finden Sie unter PKCS11 Kernbibliothek.
TLS		Weitere Informationen finden Sie unter Transport Layer Security.
Gemeinsame E/A	Allgemeine API-Referenz I/O	Weitere Informationen finden Sie unter Gemeinsame E/A.
Mobilfunkschnittst elle	API-Referenz für die Mobilfunk schnittstelle	Die Cellular Interface-Biblioth ek stellt die Funktionen einiger gängiger Mobilfunkmodems über eine einheitliche API zur Verfügung. Weitere Informationen hierzu finden Sie unter Cellular Interface-Biblioth ek.

FreeRTOS-Anwendungsbibliotheken

Sie können optional die folgenden eigenständigen Anwendungsbibliotheken in Ihre FreeRTOS-Konfiguration aufnehmen, um mit AWS IoT Diensten in der Cloud zu interagieren.

> **Note**
>
> Einige der Anwendungsbibliotheken haben dieselben Funktionen APIs wie die Bibliotheken im AWS IoT Device SDK for Embedded C. Diese Bibliotheken finden Sie in der AWS IoT Device SDK C API-Referenz. Weitere Informationen zum AWS IoT Geräte-SDK für Embedded C finden Sie unterAWS IoT Geräte-SDK für Embedded C.

FreeRTOS-Anwendungsbibliotheken

Bibliothek	API-Referenz	Description
AWS IoT Device Defender	Device Defender C SDK API-Referenz	Die AWS IoT Device Defender FreeRTOS-Bibliothek verbindet Ihr FreeRTOS-Gerät mit. AWS IoT Device Defender Weitere Informationen finden Sie unter AWS IoT Device Defender Bibliothek.
AWS IoT Greengrass	Greengrass API-Referenz	Die AWS IoT Greengrass FreeRTOS-Bibliothek verbindet Ihr FreeRTOS-Gerät mit. AWS IoT Greengrass Weitere Informationen finden Sie unter AWS IoT Greengrass Discovery-Bibliothek.
MQTT	API-Referenz für die MQTT-Bibliothek (v1.x.x) MQTT (v1) Agent API-Referenz	Die CoreMQTT-Bibliothek bietet einen Client für Ihr FreeRTOS-Gerät zum Veröffentlichen und Abonnieren von MQTT-Themen. MQTT ist das

Bibliothek	API-Referenz	Description
	MQTT (v2.x.x) C SDK-API-Referenz	Protokoll, mit dem Geräte interagieren. AWS IoT Weitere Informationen zur Version 3.0.0 der CoreMQTT-Bibliothek finden Sie unter. CoreMQTT-Bibliothek
CoreMQTT-Agent	API-Referenz zur CoreMQTT-Agentenbibliothek	Die CoreMQTT Agent Library ist eine API auf hohem Niveau, die der CoreMQTT-Bibliothek Thread-Sicherheit verleiht. Damit können Sie eine spezielle MQTT-Agentenaufgabe erstellen, die eine MQTT-Verbindung im Hintergrund verwaltet und keine Intervention durch andere Aufgaben erfordert. Die Bibliothek bietet Thread-sichere Entsprechungen zu den CoreMQTTs APIs, sodass sie in Multithread-Umgebungen verwendet werden kann. Weitere Informationen zur CoreMQTT-Agentenbibliothek finden Sie unter. CoreMQTT-Agentenbibliothek
AWS IoT Device Shadow	Device Shadow C SDK API-Referenz	Die AWS IoT Device Shadow-Bibliothek ermöglicht es Ihrem FreeRTOS-Gerät, mit AWS IoT Geräteschatten zu interagieren. Weitere Informationen finden Sie unter AWS IoT Device Shadow-Bibliothek.

Konfiguration der FreeRTOS-Bibliotheken

Die Konfigurationseinstellungen für FreeRTOS und das AWS IoT Device SDK für Embedded C sind als C-Präprozessorkonstanten definiert. Sie können Konfigurationseinstellungen mit einer globalen Konfigurationsdatei oder mit einer Compileroption wie `-D` in `gcc` festlegen. Da Konfigurationseinstellungen als Kompilierungszeitkonstanten definiert sind, muss eine Bibliothek neu aufgebaut werden, wenn eine Konfigurationseinstellung geändert wird.

Wenn Sie eine globale Konfigurationsdatei verwenden möchten, um Konfigurationsoptionen festzulegen, erstellen und speichern Sie die Datei mit dem Namen `iot_config.h` und platzieren Sie sie in Ihrem Include-Pfad. Verwenden Sie in der Datei `#define` Direktiven, um die FreeRTOS-Bibliotheken, -Demos und -Tests zu konfigurieren.

Weitere Informationen zu den unterstützten globalen Konfigurationsoptionen finden Sie in der Global-Konfigurationsdatei-Referenz.

BackoffAlgorithmus-Bibliothek

> **Note**
>
> Der Inhalt dieser Seite ist möglicherweise nicht. up-to-date Das neueste Update finden Sie auf der FreeRTOS.org-Bibliotheksseite.

Einführung

Die BackoffAlgorithm Library ist eine Hilfsbibliothek, die verwendet wird, um wiederholte Übertragungen desselben Datenblocks zu vermeiden, um Netzwerküberlastungen zu vermeiden. Diese Bibliothek berechnet die Backoff-Periode für wiederholte Netzwerkoperationen (z. B. eine fehlgeschlagene Netzwerkverbindung mit dem Server) mithilfe eines exponentiellen Backoff-mit-Jitter-Algorithmus.

Exponentielles Backoff mit Jitter wird normalerweise verwendet, wenn eine fehlgeschlagene Verbindung oder Netzwerkanforderung an einen Server erneut versucht wird, die durch Netzwerküberlastung oder hohe Serverlast verursacht wurde. Es wird verwendet, um das Timing der Wiederholungsanforderungen zu verteilen, die von mehreren Geräten erzeugt werden, die gleichzeitig versuchen, Netzwerkverbindungen herzustellen. In einer Umgebung mit schlechter Konnektivität kann die Verbindung zu einem Client jederzeit unterbrochen werden. Eine Backoff-

Strategie hilft dem Client also auch, den Akku zu schonen, indem er nicht wiederholt versucht, Verbindungen wiederherzustellen, wenn es unwahrscheinlich ist, dass sie erfolgreich sind.

Die Bibliothek ist in C geschrieben und so konzipiert, dass sie ISO C90 und MISRA C:2012 entspricht. Die Bibliothek ist nicht von anderen Bibliotheken als der Standard-C-Bibliothek abhängig und hat keine Heap-Zuweisung, sodass sie für IoT-Mikrocontroller geeignet ist, aber auch vollständig auf andere Plattformen portierbar ist.

Diese Bibliothek kann frei verwendet werden und wird unter der MIT-Open-Source-Lizenz vertrieben.

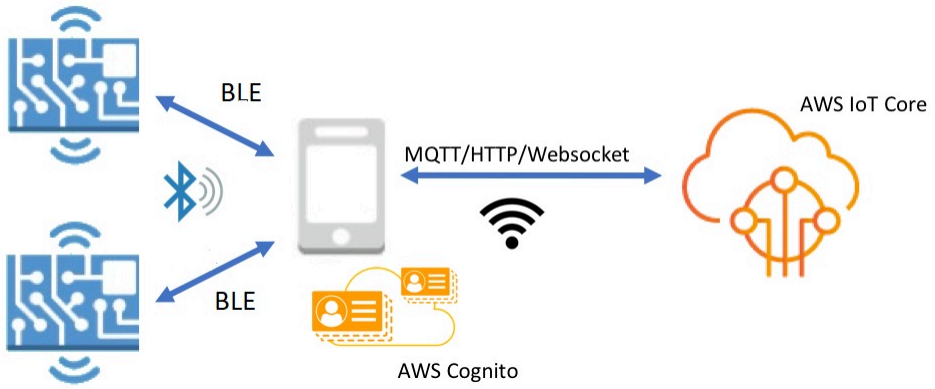
Codegröße des BackOffAlgorithm (Beispiel generiert mit GCC für ARM Cortex-M)		
Datei	Mit -O1-Optimierung	Mit -Os-Optimierung
backoff_algorithm.c	0,1 K	0,1 K
Schätzungen insgesamt	0,1 K	0,1 K

Bluetooth Low Energy-Bibliothek

> **Important**
>
> Diese Bibliothek wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

-Übersicht

FreeRTOS unterstützt das Veröffentlichen und Abonnieren von MQTT-Themen (Message Queuing Telemetry Transport) über Bluetooth Low Energy über ein Proxygerät, z. B. ein Mobiltelefon. Mit der FreeRTOS Bluetooth Low Energy (BLE) -Bibliothek kann Ihr Mikrocontroller sicher mit dem MQTT-Broker kommunizieren. AWS IoT

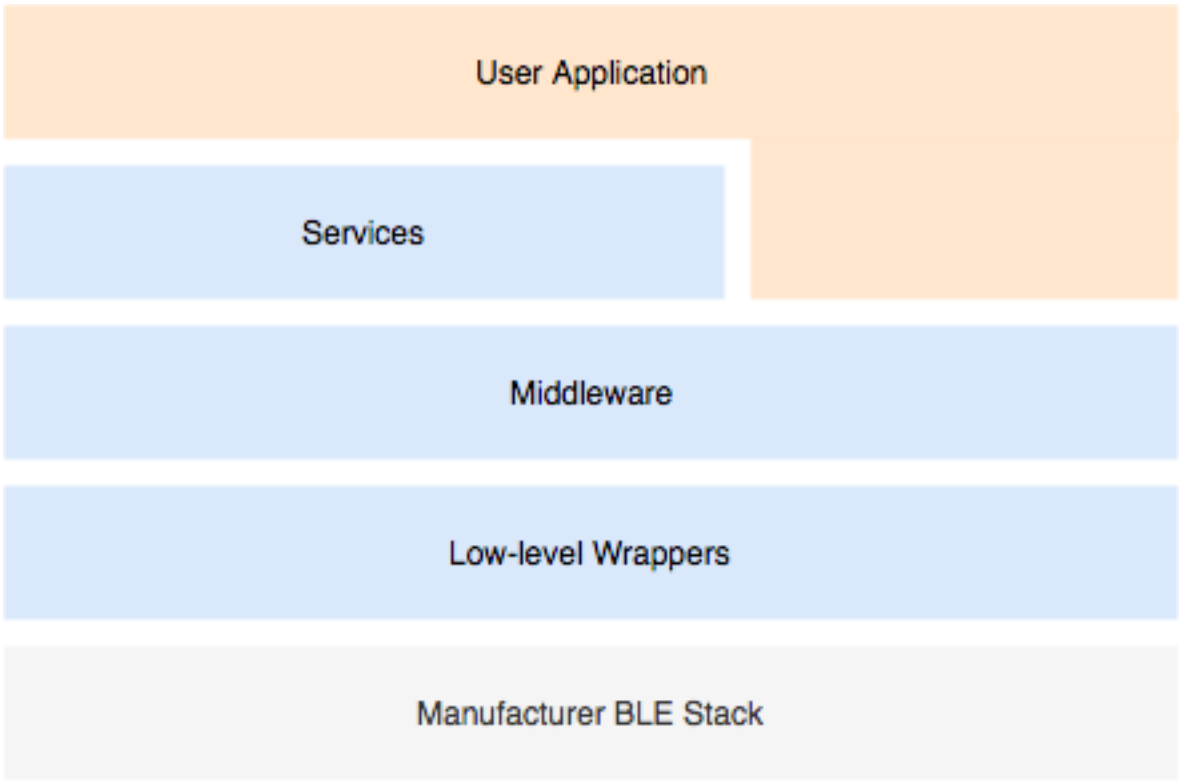


Mit Mobile SDKs for FreeRTOS Bluetooth-Geräten können Sie native mobile Anwendungen schreiben, die über BLE mit den eingebetteten Anwendungen auf Ihrem Mikrocontroller kommunizieren. Weitere Informationen über das Mobiltelefon SDKs finden Sie unter. Mobil SDKs für FreeRTOS-Bluetooth-Geräte

Die FreeRTOS BLE-Bibliothek umfasst Dienste für die Konfiguration von Wi-Fi-Netzwerken, die Übertragung großer Datenmengen und die Bereitstellung von Netzwerkabstraktionen über BLE. Die FreeRTOS BLE-Bibliothek umfasst auch Middleware und Lower-Level APIs für eine direktere Kontrolle über Ihren BLE-Stack.

Architektur

Die FreeRTOS BLE-Bibliothek besteht aus drei Ebenen: Dienste, Middleware und Low-Level-Wrapper.



Dienstleistungen

Die FreeRTOS BLE-Serviceschicht besteht aus vier GATT-Diensten (Generic Attribute), die die Middleware nutzen: APIs

- Geräteinformationen
- WLAN-Bereitstellung
- Network Abstraction
- Übertragung großer Objekte

Geräteinformationen

Der Geräteinformationsdienst sammelt Informationen über Ihren Mikrocontroller, darunter:

- Die Version von FreeRTOS, die Ihr Gerät verwendet.
- Der AWS IoT Endpunkt des Kontos, für das das Gerät registriert ist.
- Bluetooth Low Energy-MTU (Maximum Transmission Unit).

WLAN-Bereitstellung

Mit dem WLAN-Bereitstellungsservice können Mikrocontroller mit WLAN-Funktionen Folgendes durchführen:

- Auflisten von Netzwerken in Reichweite
- Speichern von Netzwerken und Netzwerk-Anmeldeinformationen auf Flash-Speicher
- Festlegen der Netzwerkpriorität
- Löschen von Netzwerken und Netzwerk-Anmeldeinformationen vom Flash-Speicher

Network Abstraction

Der Netzwerkabstraktionsdienst abstrahiert den Netzwerkverbindungstyp für Anwendungen. Eine gemeinsame API interagiert mit dem Wi-Fi-, Ethernet- und Bluetooth Low Energy-Hardware-Stack Ihres Geräts, sodass eine Anwendung mit mehreren Verbindungstypen kompatibel ist.

Large Object Transfer

Der Large Object Transfer-Dienst sendet Daten an einen Client und empfängt Daten von einem Client. Andere Dienste, wie Wi-Fi-Bereitstellung und Netzwerkabstraktion, verwenden den Large Object Transfer-Dienst zum Senden und Empfangen von Daten. Sie können auch die Large Object Transfer API verwenden, um direkt mit dem Dienst zu interagieren.

MQTT über BLE

MQTT over BLE enthält das GATT-Profil für die Erstellung eines MQTT-Proxydienstes über BLE. Der MQTT-Proxydienst ermöglicht es einem MQTT-Client, über ein Gateway-Gerät mit dem AWS MQTT-Broker zu kommunizieren. Sie können beispielsweise den Proxydienst verwenden, um ein Gerät, auf dem FreeRTOS ausgeführt wird, über eine Smartphone-App mit AWS MQTT zu verbinden. Das BLE-Gerät ist der GATT-Server und stellt Dienste und Eigenschaften für das Gateway-Gerät bereit. Der GATT-Server verwendet diese exponierten Dienste und Eigenschaften, um MQTT-Operationen mit der Cloud für dieses Gerät durchzuführen. Weitere Informationen finden Sie unter Anhang A: MQTT über BLE GATT-Profil.

Middleware

Die FreeRTOS Bluetooth Low Energy-Middleware ist eine Abstraktion von der unteren Ebene. APIs Die Middleware bietet eine benutzerfreundlichere Schnittstelle APIs zum Bluetooth Low Energy-Stack.

Mithilfe von Middleware APIs können Sie mehrere Callbacks auf mehreren Ebenen für ein einzelnes Ereignis registrieren. Die Initialisierung der Bluetooth Low Energy-Middleware initialisiert auch Services und beginnt mit dem Advertising.

Flexibles Callback-Abonnement

Nehmen wir an, Ihre Bluetooth Low Energy-Hardware wird getrennt, und der MQTT über den Bluetooth Low Energy-Service muss die Trennung erkennen. Eine Anwendung, die Sie geschrieben haben, muss diese Verbindungstrennung möglicherweise auch erkennen. Die Bluetooth Low Energy-Middleware kann das Ereignis an verschiedene Teile des Codes weiterleiten, in dem Callbacks registriert wurden, ohne dass die höheren Ebenen um Lower-Level-Ressourcen konkurrieren.

Low-Level-Wrapper

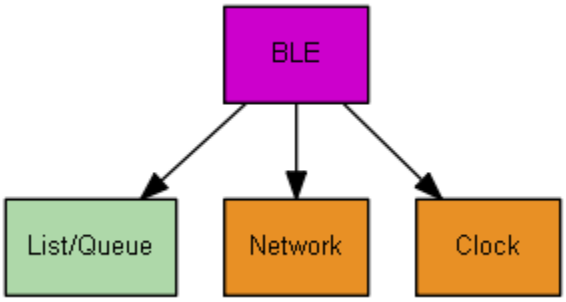
Die Low-Level-FreeRTOS Bluetooth Low Energy-Wrapper sind eine Abstraktion aus dem Bluetooth Low Energy-Stack des Herstellers. Low-Level-Wrapper bieten ein einheitliches Set zur direkten Steuerung der Hardware. APIs Die Low-Level-Module APIs optimieren die RAM-Nutzung, sind jedoch in ihrer Funktionalität eingeschränkt.

Verwenden Sie den Bluetooth Low Energy-Dienst APIs , um mit den Bluetooth Low Energy-Diensten zu interagieren. Der Dienst benötigt APIs mehr Ressourcen als der APIs Low-Level-Dienst.

Abhängigkeiten und Anforderungen

Die Bluetooth Low Energy-Bibliothek hat die folgenden direkten Abhängigkeiten:

- Bibliothek für lineare Container
- Eine Plattformschicht, die mit dem Betriebssystem für Thread-Management, Timer, Taktgeberfunktionen und Netzwerkzugriff verbunden ist.



Nur der Wi-Fi Provisioning Service hat Abhängigkeiten von der FreeRTOS-Bibliothek:

GATT-Service	-Abhängigkeit
WLAN-Bereitstellung	WLAN-Bibliothek

Um mit dem AWS IoT MQTT-Broker kommunizieren zu können, benötigen Sie ein AWS Konto und müssen Ihre Geräte als Dinge registrieren. AWS IoT Weitere Informationen zur Einrichtung finden Sie im AWS IoT Entwicklerhandbuch.

FreeRTOS Bluetooth Low Energy verwendet Amazon Cognito für die Benutzerauthentifizierung auf Ihrem Mobilgerät. Um MQTT-Proxydienste verwenden zu können, müssen Sie eine Amazon Cognito Cognito-Identität und Benutzerpools erstellen. Jeder Amazon Cognito Cognito-Identität muss die entsprechende Richtlinie beigefügt sein. Weitere Informationen finden Sie im Amazon Cognito Entwicklerhandbuch.

Bibliothekskonfigurationsdatei

Anwendungen, die den FreeRTOS MQTT over Bluetooth Low Energy-Dienst verwenden, müssen eine `iot_ble_config.h` Header-Datei bereitstellen, in der Konfigurationsparameter definiert sind. Nicht definierte Konfigurationsparameter nehmen die in `iot_ble_config_defaults.h` angegebenen Standardwerte an.

Einige wichtige Konfigurationsparameter sind:

`IOT_BLE_ADD_CUSTOM_SERVICES`

Ermöglicht es Benutzern, ihre eigenen Services zu erstellen.

`IOT_BLE_SET_CUSTOM_ADVERTISEMENT_MSG`

Ermöglicht Benutzern, das Advertising anzupassen und Antwortnachrichten zu scannen.

Weitere Informationen finden Sie unter API-Referenz Bluetooth Low Energy (BLE).

Optimierung

Wenn Sie die Leistung Ihres Boards optimieren, sollten Sie Folgendes berücksichtigen:

- Low-Level APIs verbrauchen weniger RAM, bieten aber eingeschränkte Funktionalität.
- Sie können den Parameter `bleconfigMAX_NETWORK` in der `iot_ble_config.h`-Header-Datei auf einen niedrigeren Wert setzen, um die Menge des verbrauchten Stacks zu verringern.

- Außerdem können Sie die MTU-Größe auf ihren maximalen Wert erhöhen, um das Puffern von Nachrichten zu begrenzen, den Code schneller ausführen zu lassen und so weniger RAM zu verbrauchen.

Nutzungsbeschränkungen

Standardmäßig setzt die FreeRTOS Bluetooth Low Energy-Bibliothek die `eBTpropertySecureConnectionOnly` Eigenschaft auf TRUE, wodurch das Gerät in den Modus Nur sichere Verbindungen versetzt wird. Wie unter Bluetooth-Kernspezifikation v5.0, Vol. 3, Teil C, 10.2.4 angegeben, ist für ein Gerät im Modus "Nur sichere Verbindungen" der höchste LE-Sicherheitsmodus "1 Level, Level 4" für den Zugriff auf alle Attribute erforderlich, die höhere Berechtigungen als den niedrigsten LE-Sicherheitsmodus "1 Level, Level 1" aufweisen. Im LE-Sicherheitsmodus 1, Level 4, muss ein Gerät über Ein- und Ausgabefunktionen für den numerischen Abgleich verfügen.

Hier sind die unterstützten Modi und die zugehörigen Eigenschaften:

Modus 1, Level 1 (Keine Sicherheit)

```
/* Disable numeric comparison */
#define IOT_BLE_ENABLE_NUMERIC_COMPARISON       ( 0 )
#define IOT_BLE_ENABLE_SECURE_CONNECTION        ( 0 )
#define IOT_BLE_INPUT_OUTPUT                    ( eBTIONone )
#define IOT_BLE_ENCRYPTION_REQUIRED              ( 0 )
```

Modus 1, Level 2 (Nicht authentifizierte Kopplung mit Verschlüsselung)

```
#define IOT_BLE_ENABLE_NUMERIC_COMPARISON       ( 0 )
#define IOT_BLE_ENABLE_SECURE_CONNECTION        ( 0 )
#define IOT_BLE_INPUT_OUTPUT                    ( eBTIONone )
```

Modus 1, Level 3 (Authentifizierte Kopplung mit Verschlüsselung)

Dieser Modus wird nicht unterstützt.

Modus 1, Level 4 (Authentifizierte LE Secure-Verbindungen mit Verschlüsselung)

Dieser Modus wird standardmäßig unterstützt.

Weitere Informationen über LE-Sicherheitsmodi finden Sie in der Bluetooth-Kernspezifikation v5.0, Vol. 3, Teil C, 10.2.1.

Initialisierung

Wenn Ihre Anwendung über die Middleware mit dem Bluetooth Low Energy-Stack interagiert, müssen Sie nur die Middleware initialisieren. Die Middleware übernimmt die Initialisierung der niedrigeren Ebenen des Stacks.

Middleware

So wird die Middleware initialisiert

1. Initialisieren Sie alle Bluetooth Low Energy-Hardwaretreiber, bevor Sie die Bluetooth Low Energy-Middleware-API aufrufen.
2. Aktivieren Sie Bluetooth Low Energy.
3. Initialisieren Sie die Middleware mit `IotBLE_Init()`.

> **Note**
>
> Dieser Initialisierungsschritt ist nicht erforderlich, wenn Sie die Demos ausführen. AWS Die Demo-Initialisierung wird vom Netzwerkmanager durchgeführt, der sich in *`freertos`*`/demos/network_manager` befindet.

Niedriges Niveau APIs

Wenn Sie die FreeRTOS Bluetooth Low Energy GATT-Dienste nicht nutzen möchten, können Sie die Middleware umgehen und direkt mit dem Low-Level interagieren, um Ressourcen zu sparen. APIs

Um das Low-Level zu initialisieren APIs

1. Initialisieren Sie alle Bluetooth Low Energy-Hardwaretreiber, bevor Sie den aufrufen. APIs Die Treiberinitialisierung ist nicht Teil des Bluetooth Low Energy-Low-Levels. APIs
2. Die Bluetooth Low Energy Low-Level-API enable/disable ruft den Bluetooth Low Energy-Stack auf, um Energie und Ressourcen zu optimieren. Bevor Sie den aufrufen APIs, müssen Sie Bluetooth Low Energy aktivieren.

```
const BTInterface_t * pxIface = BTGetBluetoothInterface();
xStatus = pxIface->pxEnable( 0 );
```

3. Der Bluetooth-Manager enthält APIs die Funktionen, die sowohl Bluetooth Low Energy als auch Bluetooth Classic gemeinsam haben. Die Callbacks für den gemeinsamen Manager müssen als Zweites initialisiert werden.

```
xStatus = xBTInterface.pxBTInterface->pxBtManagerInit( &xBTManagerCb );
```

4. Der Bluetooth Low Energy-Adapter befindet sich über der allgemeinern API. Sie müssen seine Callbacks auf dieselbe Art initialisieren, wie Sie die gemeinsame API initialisiert haben.

```
xBTInterface.pxBTLeAdapterInterface = ( BTBleAdapter_t * )
 xBTInterface.pxBTInterface->pxGetLeAdapter();
xStatus = xBTInterface.pxBTLeAdapterInterface-
>pxBleAdapterInit( &xBTBleAdapterCb );
```

5. Registrieren Sie Ihre neue Benutzeranwendung.

```
xBTInterface.pxBTLeAdapterInterface->pxRegisterBleApp( pxAppUuid );
```

6. Initialisieren Sie die Callbacks an die GATT-Server.

```
xBTInterface.pxGattServerInterface = ( BTGattServerInterface_t * )
 xBTInterface.pxBTLeAdapterInterface->ppvGetGattServerInterface();
xBTInterface.pxGattServerInterface->pxGattServerInit( &xBTGattServerCb );
```

   Nachdem Sie den Bluetooth Low Energy-Adapter initialisiert haben, können Sie einen GATT-Server hinzufügen. Es kann immer nur ein GATT-Server registriert werden.

```
xStatus = xBTInterface.pxGattServerInterface->pxRegisterServer( pxAppUuid );
```

7. Legen Sie Eigenschaften wie "Nur sichere Verbindungen" und MTU-Größe fest.

```
xStatus = xBTInterface.pxBTInterface-
>pxSetDeviceProperty( &pxProperty[ usIndex ] );
```

API-Referenz

Die vollständige API-Referenz finden Sie unter API-Referenz Bluetooth Low Energy (BLE).

Beispielverwendung

Die folgenden Beispiele zeigen, wie Sie die Bluetooth Low Energy-Bibliothek für Werbung und zur Erstellung neuer Services nutzen können. Vollständige FreeRTOS Bluetooth Low Energy-Demoanwendungen finden Sie unter Bluetooth Low Energy-Demo-Anwendungen.

Werbung

1. Legen Sie in Ihrer Anwendung die Advertising-UUID fest:

```
static const BTUuid_t _advUUID =
{
    .uu.uu128 = IOT_BLE_ADVERTISING_UUID,
    .ucType   = eBTuuidType128
};
```

2. Definieren Sie dann die Callback-Funktion `IotBle_SetCustomAdvCb`:

```
void IotBle_SetCustomAdvCb( IotBleAdvertisementParams_t * pAdvParams,
 IotBleAdvertisementParams_t * pScanParams)
{
    memset(pAdvParams, 0, sizeof(IotBleAdvertisementParams_t));
    memset(pScanParams, 0, sizeof(IotBleAdvertisementParams_t));

    /* Set advertisement message */
    pAdvParams->pUUID1 = &_advUUID;
    pAdvParams->nameType = BTGattAdvNameNone;

    /* This is the scan response, set it back to true. */
    pScanParams->setScanRsp = true;
    pScanParams->nameType = BTGattAdvNameComplete;
}
```

   Dieser Callback sendet die UUID in der Advertising-Nachricht und den vollständigen Namen in der Scan-Antwort.

3. Öffnen Sie `vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/iot_ble_config.h` und legen Sie `IOT_BLE_SET_CUSTOM_ADVERTISEMENT_MSG` auf 1 fest. Dadurch wird der `IotBle_SetCustomAdvCb`-Callback ausgelöst.

Hinzufügen eines neuen Services

Ausführliche Beispiele für Services finden Sie unter *freertos*`/.../ble/services`.

1. Erstellen Sie UUIDs für die Merkmale und Deskriptoren des Dienstes:

```
#define xServiceUUID_TYPE \
{\
    .uu.uu128 = gattDemoSVC_UUID, \
    .ucType   = eBTuuidType128 \
}
#define xCharCounterUUID_TYPE \
{\
    .uu.uu128 = gattDemoCHAR_COUNTER_UUID,\
    .ucType   = eBTuuidType128\
}
#define xCharControlUUID_TYPE \
{\
    .uu.uu128 = gattDemoCHAR_CONTROL_UUID,\
    .ucType   = eBTuuidType128\
}
#define xClientCharCfgUUID_TYPE \
{\
    .uu.uu16 = gattDemoCLIENT_CHAR_CFG_UUID,\
    .ucType  = eBTuuidType16\
}
```

2. Erstellen Sie einen Puffer, um die Handles der Charakteristika und Deskriptoren zu registrieren:

```
static uint16_t usHandlesBuffer[egattDemoNbAttributes];
```

3. Legen Sie die Attributstabelle an. Um etwas RAM zu sparen, definieren Sie die Tabelle als `const`.

> **Important**
>
> Legen Sie die Attribute immer einer Reihenfolge, in der der Service als erstes Attribut steht.

```
static const BTAttribute_t pxAttributeTable[] = {
     {
```

```
          .xServiceUUID =  xServiceUUID_TYPE
     },
    {
          .xAttributeType = eBTDbCharacteristic,
          .xCharacteristic =
          {
               .xUuid = xCharCounterUUID_TYPE,
               .xPermissions = ( IOT_BLE_CHAR_READ_PERM ),
               .xProperties = ( eBTPropRead | eBTPropNotify )
           }
     },
     {
          .xAttributeType = eBTDbDescriptor,
          .xCharacteristicDescr =
          {
              .xUuid = xClientCharCfgUUID_TYPE,
              .xPermissions = ( IOT_BLE_CHAR_READ_PERM | IOT_BLE_CHAR_WRITE_PERM )
           }
     },
    {
          .xAttributeType = eBTDbCharacteristic,
          .xCharacteristic =
          {
               .xUuid = xCharControlUUID_TYPE,
               .xPermissions = ( IOT_BLE_CHAR_READ_PERM | IOT_BLE_CHAR_WRITE_PERM
  ),
               .xProperties = ( eBTPropRead | eBTPropWrite )
           }
     }
};
```

4. Erstellen Sie ein Array von Callbacks. Dieses Array von Callbacks muss der gleichen Reihenfolge folgen wie das oben definierte Tabellenarray.

   Wenn beispielsweise `vReadCounter` beim Zugriff auf `xCharCounterUUID_TYPE` ausgelöst wird und `vWriteCommand` beim Zugriff auf `xCharControlUUID_TYPE` ausgelöst wird, definieren Sie das Array wie folgt:

```
static const IotBleAttributeEventCallback_t pxCallBackArray[egattDemoNbAttributes]
 =
    {
  NULL,
  vReadCounter,
```

```
  vEnableNotification,
  vWriteCommand
};
```

5. Erstellen Sie den Service.

```
static const BTService_t xGattDemoService =
{
  .xNumberOfAttributes = egattDemoNbAttributes,
  .ucInstId = 0,
  .xType = eBTServiceTypePrimary,
  .pusHandlesBuffer = usHandlesBuffer,
  .pxBLEAttributes = (BTAttribute_t *)pxAttributeTable
};
```

6. Rufen Sie die `IotBle_CreateService`-API mit der Struktur auf, die Sie im vorherigen Schritt erstellt haben. Die Middleware synchronisiert die Erstellung aller Services, sodass alle neuen Services bereits definiert sein müssen, wenn der `IotBle_AddCustomServicesCb`-Callback ausgelöst wird.

   a. Legen Sie in `vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/iot_ble_config.h` `IOT_BLE_ADD_CUSTOM_SERVICES` auf 1 fest.

   b. Erstellen Sie IotBle _ AddCustomServicesCb in Ihrer Anwendung:

```
void IotBle_AddCustomServicesCb(void)
{
    BTStatus_t xStatus;
    /* Select the handle buffer. */
    xStatus = IotBle_CreateService( (BTService_t *)&xGattDemoService,
 (IotBleAttributeEventCallback_t *)pxCallBackArray );
}
```

Portierung

Ein- und Ausgabeperipheriegeräte des Benutzers

Für den numerischen Abgleich benötigt eine sichere Verbindung sowohl Eingaben als auch Ausgaben. Das Ereignis `eBLENumericComparisonCallback` kann mit dem Ereignis-Manager registriert werden:

```
xEventCb.pxNumericComparisonCb = &prvNumericComparisonCb;
```

```
xStatus = BLE_RegisterEventCb( eBLENumericComparisonCallback, xEventCb );
```

Das Peripheriegerät muss den numerischen Hauptschlüssel anzeigen und das Ergebnis des Vergleichs als Eingabe verwenden.

Portieren von API-Implementierungen

Um FreeRTOS auf ein neues Ziel zu portieren, müssen Sie einige APIs für den Wi-Fi Provisioning Service und die Bluetooth Low Energy-Funktionalität implementieren.

Bluetooth Low Energy APIs

Um die FreeRTOS Bluetooth Low Energy-Middleware verwenden zu können, müssen Sie einige implementieren. APIs

APIs Gemeinsamkeiten zwischen GAP für Bluetooth Classic und GAP für Bluetooth Low Energy

- `pxBtManagerInit`
- `pxEnable`
- `pxDisable`
- `pxGetDeviceProperty`
- `pxSetDeviceProperty` (Alle Optionen außer `eBTpropertyRemoteRssi` und `eBTpropertyRemoteVersionInfo` sind obligatorisch.)
- `pxPair`
- `pxRemoveBond`
- `pxGetConnectionState`
- `pxPinReply`
- `pxSspReply`
- `pxGetTxpower`
- `pxGetLeAdapter`
- `pxDeviceStateChangedCb`
- `pxAdapterPropertiesCb`
- `pxSspRequestCb`
- `pxPairingStateChangedCb`
- `pxTxPowerCb`

APIs spezifisch für GAP für Bluetooth Low Energy

- `pxRegisterBleApp`
- `pxUnregisterBleApp`
- `pxBleAdapterInit`
- `pxStartAdv`
- `pxStopAdv`
- `pxSetAdvData`
- `pxConnParameterUpdateRequest`
- `pxRegisterBleAdapterCb`
- `pxAdvStartCb`
- `pxSetAdvDataCb`
- `pxConnParameterUpdateRequestCb`
- `pxCongestionCb`

GATT-Server

- `pxRegisterServer`
- `pxUnregisterServer`
- `pxGattServerInit`
- `pxAddService`
- `pxAddIncludedService`
- `pxAddCharacteristic`
- `pxSetVal`
- `pxAddDescriptor`
- `pxStartService`
- `pxStopService`
- `pxDeleteService`
- `pxSendIndication`
- `pxSendResponse`
- `pxMtuChangedCb`

- `pxCongestionCb`
- `pxIndicationSentCb`
- `pxRequestExecWriteCb`
- `pxRequestWriteCb`
- `pxRequestReadCb`
- `pxServiceDeletedCb`
- `pxServiceStoppedCb`
- `pxServiceStartedCb`
- `pxDescriptorAddedCb`
- `pxSetValCallbackCb`
- `pxCharacteristicAddedCb`
- `pxIncludedServiceAddedCb`
- `pxServiceAddedCb`
- `pxConnectionCb`
- `pxUnregisterServerCb`
- `pxRegisterServerCb`

Weitere Informationen zur Portierung der FreeRTOS Bluetooth Low Energy-Bibliothek auf Ihre Plattform finden Sie unter Portierung der Bluetooth Low Energy Library im FreeRTOS Porting Guide.

Mobil SDKs für FreeRTOS-Bluetooth-Geräte

> **Important**
>
> Diese Bibliothek wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Sie können Mobile SDKs for FreeRTOS Bluetooth-Geräte verwenden, um mobile Anwendungen zu erstellen, die über Bluetooth Low Energy mit Ihrem Mikrocontroller interagieren. Das Handy SDKs

kann auch mit AWS Diensten kommunizieren und Amazon Cognito für die Benutzerauthentifizierung verwenden.

Android-SDK für FreeRTOS-Bluetooth-Geräte

Verwenden Sie das Android SDK für FreeRTOS-Bluetooth-Geräte, um mobile Android-Anwendungen zu erstellen, die über Bluetooth Low Energy mit Ihrem Mikrocontroller interagieren. Das SDK ist verfügbar auf. GitHub

Um das Android-SDK für FreeRTOS-Bluetooth-Geräte zu installieren, folgen Sie den Anweisungen zum Einrichten des SDK in der README.md-Datei des Projekts.

Weitere Informationen zum Einrichten und Ausführen der Demoanwendung für Mobilgeräte, die in dem SDK enthalten ist, finden Sie unter Voraussetzungen und FreeRTOS Bluetooth Low Energy Mobile SDK-Demoanwendung.

iOS-SDK für FreeRTOS-Bluetooth-Geräte

Verwenden Sie das iOS-SDK für FreeRTOS-Bluetooth-Geräte, um mobile iOS-Anwendungen zu erstellen, die über Bluetooth Low Energy mit Ihrem Mikrocontroller interagieren. Das SDK ist verfügbar auf. GitHub

So installieren Sie das iOS-SDK

1. Installieren Sie CocoaPods:

   ```
   $ gem install cocoapods
   $ pod setup
   ```

   > **Note**
   >
   > Möglicherweise müssen Sie es für `sudo` die Installation verwenden CocoaPods.

2. Installieren Sie das SDK mit CocoaPods (fügen Sie dies zu Ihrer Poddatei hinzu):

   ```
   $ pod 'FreeRTOS', :git => 'https://github.com/aws/amazon-freertos-ble-ios-sdk.git'
   ```

Weitere Informationen zum Einrichten und Ausführen der Demoanwendung für Mobilgeräte, die in dem SDK enthalten ist, finden Sie unter Voraussetzungen und FreeRTOS Bluetooth Low Energy Mobile SDK-Demoanwendung.

Anhang A: MQTT über BLE GATT-Profil

Einzelheiten zum GATT-Dienst

MQTT over BLE verwendet eine Instanz des GATT-Datenübertragungsdienstes, um MQTT Concise Binary Object Representation (CBOR) -Nachrichten zwischen dem FreeRTOS-Gerät und dem Proxygerät zu senden. Der Datenübertragungsdienst bietet bestimmte Eigenschaften, die beim Senden und Empfangen von Rohdaten über das BLE-GATT-Protokoll helfen. Er kümmert sich auch um die Fragmentierung und Zusammenstellung von Nutzlasten, die über der MTU-Größe (Maximum Transfer Unit) von BLE liegen.

Dienst-UUID

`A9D7-166A-D72E-40A9-A002-4804-4CC3-FF00`

Dienstinstanzen

Für jede MQTT-Sitzung mit dem Broker wird eine Instanz des GATT-Dienstes erstellt. Jeder Dienst hat eine eindeutige UUID (zwei Byte), die seinen Typ identifiziert. Jede einzelne Instanz wird anhand der Instanz-ID unterschieden.

Jeder Dienst wird als primärer Dienst auf jedem BLE-Servergerät instanziiert. Sie können mehrere Instanzen des Dienstes auf einem bestimmten Gerät erstellen. Der MQTT-Proxy-Servicetyp hat eine eindeutige UUID.

Merkmale

Charakteristisches Inhaltsformat: CBOR

Max. Größe des Merkmalswerts: 512 Byte

Merkmal	Anforderung	Obligatorische Eigenschaften	Optionale Eigenschaften	Sicherheitsberechtigungen	Kurze Beschreibung	UUID
Steuerung	M	Schreiben	Keine	Schreiben braucht Verschlüsselung	Wird verwendet, um den MQTT-Proxy zu	`A9D7-166A-D72E-40A9-A002-48`

Merkmal	Anforderung	Obligatorische Eigenschaften	Optionale Eigenschaften	Sicherheitsberechtigungen	Kurze Beschreibung	UUID
					starten und zu stoppen.	`04-4CC3-FF01`
TXMessage	M	Lesen, Benachrichtigung	Keine	Lesen benötigt Verschlüsselung	Wird verwendet, um eine Benachrichtigung mit einer Nachricht über einen Proxy an einen Broker zu senden.	`A9D7-166A-D72E-40A9-A002-4804-4CC3-FF02`
RXMessage	M	Lesen, Schreiben ohne Antwort	Keine	Lesen, Schreiben benötigt Verschlüsselung	Wird verwendet, um eine Nachricht von einem Broker über einen Proxy zu empfangen.	`A9D7-166A-D72E-40A9-A002-4804-4CC3-FF03`

Merkmal	Anforderung	Obligatorische Eigenschaften	Optionale Eigenschaften	Sicherheitsberechtigungen	Kurze Beschreibung	UUID
TXLargeNachricht	M	Lesen, Benachrichtigung	Keine	Lesen benötigt Verschlüsselung	Wird verwendet, um eine große Nachricht (Nachricht > BLE-MTU-Größe) über einen Proxy an einen Broker zu senden.	A9D7-166A-D72E-40A9-A002-4804-4CC3-FF04
RXLargeNachricht	M	Lesen, Schreiben ohne Antwort	Keine	Lesen, Schreiben benötigt Verschlüsselung	Wird verwendet, um große Nachrichten (Nachricht > BLE-MTU-Größe) von einem Broker über einen Proxy zu empfangen.	A9D7-166A-D72E-40A9-A002-4804-4CC3-FF05

Anforderungen an das GATT-Verfahren

Lesen Sie Merkmalswerte	zwingend erforderlich
Lesen Sie lange Merkmalswerte	zwingend erforderlich
Merkmalswerte schreiben	zwingend erforderlich
Schreiben Sie lange Merkmalswerte	zwingend erforderlich
Lesen Sie Merkmalsdeskriptoren	zwingend erforderlich
Schreiben Sie charakteristische Deskriptoren	zwingend erforderlich
Benachrichtigungen	zwingend erforderlich
Indikationen	zwingend erforderlich

Arten von Nachrichten

Die folgenden Nachrichtentypen werden ausgetauscht.

Art der Nachricht	Fehlermeldung	Karte mit diesen Schlüssel/Wert-Paaren
0x01	CONNECT	• Schlüssel = „w“, Wert = Typ 0 Ganzzahl, Nachricht entyp (1) • Schlüssel = „d“, Wert = Typ 3, Textzeichenfolge, Client-ID für die Sitzung • Schlüssel = „a“, Wert = Typ 3, Textzeichenfolge, Broker-Endpunkt für die Sitzung

Art der Nachricht	Fehlermeldung	Karte mit diesen Schlüssel/Wert-Paaren
		• Schlüssel = „c“, Wert = Einfacher Werttyp Wahr/Falsch
0x02	CONNACK	• Schlüssel = „w“, Wert = Typ 0 Ganzzahl, Nachrichtentyp (2) • Schlüssel = „s“, Wert = Typ 0 Ganzzahl, Statuscode
0x03	PUBLISH	• Schlüssel = „w“, Wert = Typ 0 Ganzzahl, Nachrichtentyp (3) • Schlüssel = „u“, Wert = Typ 3, Textzeichenfolge, Thema für die Veröffentlichung • Schlüssel = „n“, Wert = Typ 0, Ganzzahl, QoS für die Veröffentlichung • Schlüssel = „i“, Wert = Typ 0, Ganzzahl, Nachrichten-ID, nur für QoS 1-Veröffentlichungen • Schlüssel = „k“, Wert = Typ 2, Byte-Zeichenfolge, Nutzlast für die Veröffentlichung

Art der Nachricht	Fehlermeldung	Karte mit diesen Schlüssel/Wert-Paaren
0x04	PUBACK	• Wird nur für QoS 1-Nachrichten gesendet. • Schlüssel = „w“, Wert = Typ 0 Ganzzahl, Nachrichtentyp (4) • Schlüssel = „i“, Wert = Typ 0, Ganzzahl, Nachrichten-ID
0x08	SUBSCRIBE	• Schlüssel = „w“, Wert = Typ 0 Ganzzahl, Nachrichtentyp (8) • Schlüssel = „v“, Wert = Typ 4, Reihe von Textzeichenfolgen, Themen für das Abonnement • Schlüssel = „o“, Wert = Typ 4, Array von Ganzzahlen, QoS für das Abonnement • Schlüssel = „i“, Wert = Typ 0, Ganzzahl, Nachrichten-ID
0x09	U-BACK	• Schlüssel = „w“, Wert = Typ 0 Ganzzahl, Nachrichtentyp (9) • Schlüssel = „i“, Wert = Typ 0, Ganzzahl, Nachrichten-ID • Schlüssel = „s“, Wert = Typ 0, Ganzzahl, Statuscode für das Abonnement

Art der Nachricht	Fehlermeldung	Karte mit diesen Schlüssel/Wert-Paaren
0X0A	UNSUBSCRIBE	• Schlüssel = „w“, Wert = Typ 0 Ganzzahl, Nachricht entyp (10) • Schlüssel = „v“, Wert = Typ 4, Reihe von Textzeich enfolgen, Themen für die Abmeldung • Schlüssel = „i“, Wert = Typ 0, Ganzzahl, Nachrichten-ID
0x0B	ABMELDEN	• Schlüssel = „w“, Wert = Typ 0 Ganzzahl, Nachricht entyp (11) • Schlüssel = „i“, Wert = Typ 0, Ganzzahl, Nachrichten-ID • Schlüssel = „s“, Wert = Typ 0, Ganzzahl, Statuscode für UnSubscription
0X0C	PINGREQ	• Schlüssel = „w“, Wert = Typ 0 Ganzzahl, Nachricht entyp (12)
0x0D	PINGRESP	• Schlüssel = „w“, Wert = Typ 0 Ganzzahl, Nachricht entyp (13)
0x0E	VERBINDUNG TRENNEN	• Schlüssel = „w“, Wert = Typ 0 Ganzzahl, Nachricht entyp (14)

Eigenschaften der Übertragung großer Nutzlasten

TXLargeNachricht

TXLargeDie Nachricht wird vom Gerät verwendet, um eine große Nutzlast zu senden, die größer ist als die für die BLE-Verbindung ausgehandelte MTU-Größe.

- Das Gerät sendet die ersten MTU-Bytes der Nutzlast als Benachrichtigung über das Merkmal.
- Der Proxy sendet eine Leseanforderung für dieses Merkmal für die verbleibenden Byte.
- Das Gerät sendet maximal die MTU-Größe oder die verbleibenden Byte der Nutzlast, je nachdem, welcher Wert niedriger ist. Jedes Mal wird der gelesene Offset um die Größe der gesendeten Nutzlast erhöht.
- Der Proxy liest das Merkmal weiter, bis er eine Nutzlast mit der Länge Null oder eine Nutzlast erhält, die kleiner als die MTU-Größe ist.
- Wenn das Gerät innerhalb eines bestimmten Timeouts keine Leseanforderung erhält, schlägt die Übertragung fehl und der Proxy und das Gateway geben den Puffer frei.
- Wenn der Proxy innerhalb eines bestimmten Timeouts keine Leseantwort erhält, schlägt die Übertragung fehl und der Proxy gibt den Puffer frei.

RXLargeNachricht

RXLargeDie Nachricht wird vom Gerät verwendet, um eine große Nutzlast zu empfangen, die größer ist als die für die BLE-Verbindung ausgehandelte MTU-Größe.

- Der Proxy schreibt Nachrichten bis zur MTU-Größe nacheinander und verwendet dabei Write with Response für dieses Merkmal.
- Das Gerät puffert die Nachricht, bis es eine Schreibanforderung mit einer Länge von Null oder einer Länge erhält, die kleiner als die MTU-Größe ist.
- Wenn das Gerät innerhalb eines bestimmten Timeouts keine Schreibanforderung erhält, schlägt die Übertragung fehl und das Gerät gibt den Puffer frei.
- Wenn der Proxy innerhalb eines bestimmten Timeouts keine Schreibantwort erhält, schlägt die Übertragung fehl und der Proxy gibt den Puffer frei.

Cellular Interface-Bibliothek

> **Note**
>
> Der Inhalt dieser Seite ist möglicherweise nicht. up-to-date Das neueste Update finden Sie auf der FreeRTOS.org-Bibliotheksseite.

Einführung

Die Cellular Interface-Bibliothek implementiert eine einfache, einheitliche API, die die Komplexität der für Mobilfunkmodems spezifischen AT-Befehle verbirgt und C-Programmierern eine socket-ähnliche Schnittstelle zur Verfügung stellt.

Die meisten Mobilfunkmodems implementieren mehr oder weniger der AT-Befehle, die im 3GPP TS v27.007-Standard definiert sind. Dieses Projekt bietet eine Implementierung solcher Standard-AT-Befehle in einer wiederverwendbaren gemeinsamen Komponente. Die drei Cellular Interface-Bibliotheken in diesem Projekt nutzen alle diesen gemeinsamen Code. Die Bibliothek für jedes Modem implementiert nur die herstellerspezifischen AT-Befehle und stellt dann die vollständige API der Cellular Interface-Bibliothek zur Verfügung.

Die gemeinsame Komponente, die den 3GPP TS v27.007-Standard implementiert, wurde in Übereinstimmung mit den folgenden Codequalitätskriterien geschrieben:

- Die GNU-Komplexitätswerte liegen nicht über 8
- MISRA C:2012 Codierungsstandard. Alle Abweichungen vom Standard werden in Quellcode-Kommentaren dokumentiert, die mit „Coverity“ gekennzeichnet sind.

Abhängigkeiten und Anforderungen

Es besteht keine direkte Abhängigkeit von der Cellular Interface-Bibliothek. Ethernet, WLAN und Mobilfunk können jedoch im FreeRTOS-Netzwerkstapel nicht koexistieren. Entwickler müssen eine der Netzwerkschnittstellen auswählen, um sie in die Secure Sockets-Bibliothek zu integrieren.

Portierung

Informationen zur Portierung der Cellular Interface-Bibliothek auf Ihre Plattform finden Sie unter Portierung der Cellular Interface-Bibliothek im FreeRTOS Porting Guide.

Speichernutzung

Codegröße der Mobilfunkschnittstellenbibliothek (mit GCC für ARM Cortex-M generiertes Beispiel)		
Datei	Mit -O1-Optimierung	Mit -Os-Optimierung
cellular_3gpp_api.c	6,3 K	5,7 K
cellular_3gpp_urc_handler.c	0,9 K	0,8 K
cellular_at_core.c	1,4 K	1,2 K
cellular_common_api.c	0,5 K	0,5 K
cellular_common.c	1,6 K	1,4 K
cellular_pkthandler.c	1,4 K	1,2 K
cellular_pktio.c	1,8 K	1,6 K
Schätzungen insgesamt	13,9 K	12,4 K

Erste Schritte

Laden Sie den Quellcode herunter

Der Quellcode kann als Teil der FreeRTOS-Bibliotheken oder eigenständig heruntergeladen werden.

Um die Bibliothek mit HTTPS von Github zu klonen:

```
git clone https://github.com/FreeRTOS/FreeRTOS-Cellular-Interface.git
```

SSH verwenden:

```
git clone git@github.com:FreeRTOS/FreeRTOS-Cellular-Interface.git
```

Orderstruktur

Im Stammverzeichnis dieses Repositorys werden Sie diese Ordner sehen:

- `source`: wiederverwendbarer allgemeiner Code, der die in 3GPP TS v27.007 definierten AT-Standardbefehle implementiert
- `doc`: Dokumentation
- `test`: Unit-Test und CBMC
- `tools`: Tools für die statische Analyse von Coverity und CMock

Konfiguration und Erstellung der Bibliothek

Die Cellular Interface-Bibliothek sollte als Teil einer Anwendung erstellt werden.
Um dies zu tun, müssen Sie bestimmte Konfigurationen angeben. Das Projekt FreeRTOS_Cellular_Interface_Windows_Simulator bietet ein Beispiel für die Konfiguration des Builds.
Weitere Informationen finden Sie in den Cellular API-Referenzen.

Weitere Informationen finden Sie auf der Seite Cellular Interface.

Integrieren Sie die Cellular Interface-Bibliothek in MCU-Plattformen

Die Cellular Interface-Bibliothek MCUs verwendet eine abstrakte Schnittstelle, das Comm Interface, um mit Mobilfunkmodems zu kommunizieren. Eine Kommunikationsschnittstelle muss ebenfalls auf der MCU-Plattform implementiert werden. Die gängigsten Implementierungen der Kommunikationsschnittstelle kommunizieren über UART-Hardware, können aber auch über andere physische Schnittstellen wie SPI implementiert werden. Die Dokumentation für das Comm Interface finden Sie in den API-Referenzen der Cellular Library. Die folgenden Beispielimplementierungen des Comm Interface sind verfügbar:

- FreeRTOS Windows-Simulator-Kommunikationsschnittstelle
- FreeRTOS Common IO UART Kommunikationsschnittstelle
- STM32 Kommunikationsschnittstelle für das L475 Discovery Board
- Kommunikationsschnittstelle für die Platine von Sierra Sensor Hub

Gemeinsame E/A

> **Important**
>
> Diese Bibliothek wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-

FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Übersicht

Im Allgemeinen sind Gerätetreiber unabhängig vom zugrunde liegenden Betriebssystem und spezifisch für eine bestimmte Hardwarekonfiguration. Eine Hardware-Abstraktionsschicht (HAL) bietet eine gemeinsame Schnittstelle zwischen Treibern und übergeordnetem Anwendungscode. Die HAL abstrahiert die Details, wie ein bestimmter Treiber funktioniert, und stellt eine einheitliche API zur Steuerung solcher Geräte bereit. Sie können dasselbe verwenden, APIs um auf verschiedene Gerätetreiber auf verschiedenen Mikrocontrollern (MCU) -basierten Referenzplatinen zuzugreifen.

FreeRTOS Common I/O fungiert als diese Hardware-Abstraktionsschicht. Es bietet eine Reihe von Standards APIs für den Zugriff auf gängige serielle Geräte auf unterstützten Referenzplatinen. Diese Geräte APIs kommunizieren und interagieren mit diesen Peripheriegeräten und ermöglichen es Ihrem Code, plattformübergreifend zu funktionieren. Ohne gemeinsame E/A ist das Schreiben von Code für die Arbeit mit Low-Level-Geräten Siliziumanbieter-spezifisch.

Unterstützte Peripheriegeräte

- UART
- SPI
- I2C

Unterstützte Features

- Synchrones Lesen/Schreiben — Die Funktion kehrt erst zurück, wenn die angeforderte Datenmenge übertragen wurde.
- Asynchrones Lesen/Schreiben — Die Funktion kehrt sofort zurück und die Datenübertragung erfolgt asynchron. Wenn die Aktion abgeschlossen ist, wird ein registriertes Benutzer-Callback aufgerufen.

Peripheriegeräte-spezifisch

- I2C — Kombiniert mehrere Operationen zu einer Transaktion. Wird verwendet, um Schreib- und Leseaktionen in einer Transaktion auszuführen.

- SPI — Übertragung von Daten zwischen primär und sekundär, was bedeutet, dass das Schreiben und Lesen gleichzeitig erfolgt.

Portierung

Weitere Informationen finden Sie im FreeRTOS Porting Guide.

AWS IoT Device Defender Bibliothek

> **Note**
>
> Der Inhalt dieser Seite ist möglicherweise nicht. up-to-date Das neueste Update finden Sie auf der FreeRTOS.org-Bibliotheksseite.

Einführung

Sie können die AWS IoT Device Defender Bibliothek verwenden, um Sicherheitsmetriken von Ihren IoT-Geräten an zu senden AWS IoT Device Defender. Sie können AWS IoT Device Defender sie verwenden, um diese Sicherheitsmetriken von Geräten kontinuierlich auf Abweichungen von dem zu überprüfen, was Sie als angemessenes Verhalten für jedes Gerät definiert haben. Wenn etwas nicht richtig aussieht, AWS IoT Device Defender sendet es eine Warnung, sodass Sie Maßnahmen ergreifen können, um das Problem zu beheben. Interaktionen AWS IoT Device Defender verwenden MQTT, ein einfaches Publish-Subscribe-Protokoll. Diese Bibliothek bietet eine API zum Verfassen und Erkennen der von verwendeten MQTT-Themenstrings. AWS IoT Device Defender

Weitere Informationen finden Sie unter AWS IoT Device Defender im AWS IoT -Entwicklerhandbuch.

Die Bibliothek ist in C geschrieben und so konzipiert, dass sie ISO C90 und MISRA C:2012 entspricht. Die Bibliothek ist nicht von anderen Bibliotheken als der Standard-C-Bibliothek abhängig. Sie hat auch keine Plattformabhängigkeiten wie Threading oder Synchronisation. Es kann mit jeder MQTT-Bibliothek und jeder JSON - oder CBOR-Bibliothek verwendet werden. Die Bibliothek verfügt über Beweise für eine sichere Speichernutzung und keine Heap-Zuweisung, sodass sie für IoT-Mikrocontroller geeignet ist, aber auch vollständig auf andere Plattformen portierbar ist.

Die AWS IoT Device Defender Bibliothek kann frei verwendet werden und wird unter der MIT-Open-Source-Lizenz vertrieben.

Codegröße von AWS IoT Device Defender (Beispiel generiert mit GCC für ARM Cortex-M)		
Datei	Mit -O1-Optimierung	Mit -Os-Optimierung
defender.c	1,1 K	0,6 K
Schätzungen insgesamt	1,1 K	0,6 K

AWS IoT Greengrass Discovery-Bibliothek

> **Note**
>
> Der Inhalt dieser Seite ist möglicherweise nicht. up-to-date Das neueste Update finden Sie auf der FreeRTOS.org-Bibliotheksseite.

Übersicht

Die AWS IoT Greengrass Discovery-Bibliothek wird von Ihren Mikrocontroller-Geräten verwendet, um einen Greengrass-Kern in Ihrem Netzwerk zu erkennen. Mithilfe von AWS IoT Greengrass Discovery APIs kann Ihr Gerät Nachrichten an einen Greengrass-Core senden, nachdem es den Endpunkt des Cores gefunden hat.

Abhängigkeiten und Anforderungen

Um die Greengrass Discovery-Bibliothek verwenden zu können, müssen Sie ein Objekt erstellen AWS IoT, einschließlich eines Zertifikats und einer Richtlinie. Weitere Informationen finden Sie unter Erste Schritte mit AWS IoT.

Sie müssen in der Datei „*freertos*/demos/include/aws_clientcredential.h“ Werte für die folgenden Konstanten festlegen:

clientcredentialMQTT_BROKER_ENDPOINT

Ihr AWS IoT Endpunkt.

clientcredentialIOT_THING_NAME

Der Name Ihres IoT-Objekts.

`clientcredentialWIFI_SSID`

Die SSID für Ihr WLAN-Netzwerk.

`clientcredentialWIFI_PASSWORD`

Ihr WLAN-Passwort.

`clientcredentialWIFI_SECURITY`

Der von Ihrem WLAN-Netzwerk verwendete Sicherheitstyp.

Sie müssen darüber hinaus in der Datei „*`freertos`*`/demos/include/aws_clientcredential_keys.h`“ Werte für die folgenden Konstanten festlegen:

`keyCLIENT_CERTIFICATE_PEM`

Die Zertifikat-PEM, die Ihrem Objekt zugeordnet ist.

`keyCLIENT_PRIVATE_KEY_PEM`

Die PEM des privaten Schlüssels, der Ihrem Objekt zugeordnet ist.

Sie müssen eine Greengrass-Gruppe und ein Kerngerät über die Konsole eingerichtet haben. Weitere Informationen finden Sie unter Erste Schritte mit AWS IoT Greengrass.

Obwohl die CoreMQTT-Bibliothek für die Greengrass-Konnektivität nicht erforderlich ist, empfehlen wir dringend, sie zu installieren. Die Bibliothek kann nach der Erkennung genutzt werden, um mit dem Greengrass-Kern zu kommunizieren.

API-Referenz

Die vollständige API-Referenz finden Sie unter Greengrass API-Referenz.

Beispielverwendung

Greengrass-Workflow

Das MCU-Gerät initiiert den Erkennungsprozess, indem es AWS IoT eine JSON-Datei anfordert, die die Greengrass-Kernkonnektivitätsparameter enthält. Es gibt zwei Methoden, um die Greengrass-Kern-Verbindungsparameter aus der JSON-Datei abzurufen:

- Die automatische Auswahl durchläuft alle in der JSON-Datei aufgeführten Greengrass-Kerne und verbindet sich mit dem ersten verfügbaren Kern.
- Die manuelle Auswahl verwendet die Informationen in `aws_ggd_config.h`, um sich mit dem angegebenen Greengrass-Kern zu verbinden.

So verwenden Sie die Greengrass-API:

Alle Standardkonfigurationsoptionen für die Greengrass-API sind in `aws_ggd_config_defaults.h` definiert.

Wenn nur ein Greengrass-Kern vorhanden ist, rufen Sie `GGD_GetGGCIPandCertificate` auf, um die JSON-Datei mit den Greengrass-Kern-Verbindungsinformationen anzufordern. Wenn `GGD_GetGGCIPandCertificate` zurückgegeben wird, enthält der Parameter `pcBuffer` den Text der JSON-Datei. Der Parameter `pxHostAddressData` enthält die IP-Adresse und den Port des Greengrass-Kerns, mit dem Sie sich verbinden können.

Für weitere Anpassungsoptionen, wie z. B. die dynamische Zuweisung von Zertifikaten, müssen Sie Folgendes aufrufen: APIs

`GGD_JSONRequestStart`

Sendet eine HTTP-GET-Anfrage an, AWS IoT um die Discovery-Anfrage zur Erkennung eines Greengrass-Kerns zu initiieren. `GD_SecureConnect_Send`wird verwendet, um die Anfrage an zu AWS IoT senden.

`GGD_JSONRequestGetSize`

Ruft die Größe der JSON-Datei aus der HTTP-Antwort ab.

`GGD_JSONRequestGetFile`

Ruft die JSON-Objektzeichenfolge ab. `GGD_JSONRequestGetSize` und `GGD_JSONRequestGetFile` verwenden `GGD_SecureConnect_Read`, um die JSON-Daten aus dem Socket abzurufen. `GGD_JSONRequestStart`, `GGD_SecureConnect_Send` und `GGD_JSONRequestGetSize` müssen aufgerufen werden, um die JSON-Daten von AWS IoT zu empfangen.

`GGD_GetIPandCertificateFromJSON`

Extrahiert die IP-Adresse und das Greengrass-Kern-Zertifikat aus den JSON-Daten. Sie können die automatische Auswahl aktivieren, indem Sie die `xAutoSelectFlag` auf `True` festlegen.

Die automatische Auswahl findet das erste Kerngerät, mit dem sich Ihr FreeRTOS-Gerät verbinden kann. Um sich mit einem Greengrass-Kern zu verbinden, rufen Sie die Funktion `GGD_SecureConnect_Connect` auf und übergeben die IP-Adresse, den Port und das Zertifikat des Kerngeräts. Um die manuelle Auswahl zu nutzen, legen Sie die folgenden Felder des Parameters `HostParameters_t` fest:

`pcGroupName`

Die ID der Greengrass-Gruppe, zu der der Kern gehört. Mit dem CLI-Befehl `aws greengrass list-groups` können Sie die ID Ihrer Greengrass-Gruppen ermitteln.

`pcCoreAddress`

Der ARN des Greengrass-Kerns, mit dem Sie sich verbinden.

CoreHTTP-Bibliothek

> **Note**
>
> Der Inhalt dieser Seite ist möglicherweise nicht. up-to-date Das neueste Update finden Sie auf der FreeRTOS.org-Bibliotheksseite.

HTTP-C-Clientbibliothek für kleine IoT-Geräte (MCU oder kleine MPU)

Einführung

Die CoreHTTP-Bibliothek ist eine Client-Implementierung einer Teilmenge des HTTP/1.1-Standards. Der HTTP-Standard bietet ein zustandsloses Protokoll, das auf TCP/IP aufbaut und häufig in verteilten, kollaborativen Hypertext-Informationssystemen verwendet wird.

Die CoreHTTP-Bibliothek implementiert eine Teilmenge des HTTP/1.1-Protokollstandards. Diese Bibliothek wurde für einen geringen Speicherbedarf optimiert. Die Bibliothek bietet eine vollständig synchrone API, sodass Anwendungen ihre Parallelität vollständig verwalten können. Sie verwendet nur feste Puffer, sodass Anwendungen die vollständige Kontrolle über ihre Speicherzuweisungsstrategie haben.

Die Bibliothek ist in C geschrieben und so konzipiert, dass sie ISO C90 und MISRA C:2012 entspricht. Die einzigen Abhängigkeiten der Bibliothek sind die Standard-C-Bibliothek und die LTS-Version (v12.19.1) des HTTP-Parsers von Node.js. Die Bibliothek verfügt über Beweise für eine

sichere Speichernutzung und keine Heap-Zuweisung, sodass sie für IoT-Mikrocontroller geeignet ist, aber auch vollständig auf andere Plattformen portierbar ist.

Bei der Verwendung von HTTP-Verbindungen in IoT-Anwendungen empfehlen wir, eine sichere Transportschnittstelle zu verwenden, z. B. eine, die das TLS-Protokoll verwendet, wie in der gezeigtDemo zur gegenseitigen CoreHTTP-Authentifizierung.

Diese Bibliothek kann frei verwendet werden und wird unter der MIT-Open-Source-Lizenz vertrieben.

Codegröße von CoreHTTP (mit GCC für ARM Cortex-M generiertes Beispiel)		
Datei	Mit -O1-Optimierung	Mit -Os-Optimierung
core_http_client.c	3,2 K	2,6 K
api.c (llhttp)	2,6 K	2,0 K
http.c (llhttp)	0,3 K	0,3 K
llhttp.c (llhttp)	17,9	15,9
Schätzungen insgesamt	23,9 K	20,7 K

CoreJSON-Bibliothek

> **Note**
>
> Der Inhalt dieser Seite ist möglicherweise nicht. up-to-date Das neueste Update finden Sie auf der FreeRTOS.org-Bibliotheksseite.

Einführung

JSON (JavaScript Object Notation) ist ein menschenlesbares Datenserialisierungsformat. Es wird häufig für den Datenaustausch verwendet, z. B. mit dem AWS IoT Device Shadow-Dienst, und ist Teil vieler Dienste APIs, beispielsweise der GitHub REST-API. JSON wird als Standard von Ecma International verwaltet.

Die CoreJSON-Bibliothek bietet einen Parser, der Schlüsselsuchvorgänge unterstützt und gleichzeitig die ECMA-404-Standardsyntax für den JSON-Datenaustausch strikt durchsetzt. Die Bibliothek ist

in C geschrieben und so konzipiert, dass sie ISO C90 und MISRA C:2012 entspricht. Es verfügt über Nachweise für eine sichere Speichernutzung und keine Heap-Zuweisung, sodass es für IoT-Mikrocontroller geeignet ist, aber auch vollständig auf andere Plattformen portierbar ist.

Speichernutzung

Die CoreJSON-Bibliothek verwendet einen internen Stack, um verschachtelte Strukturen in einem JSON-Dokument zu verfolgen. Der Stapel existiert für die Dauer eines einzelnen Funktionsaufrufs; er wird nicht beibehalten. Die Stackgröße kann durch die Definition des Makros angegeben werden`JSON_MAX_DEPTH`, das standardmäßig 32 Stufen hat. Jede Ebene verbraucht ein einzelnes Byte.

Codegröße von CoreJSON (mit GCC für ARM Cortex-M generiertes Beispiel)		
Datei	Mit -O1-Optimierung	Mit -Os-Optimierung
core_json.c	2,9 K	2,4 K
Schätzungen insgesamt	2,9 K	2,4 K

CoreMQTT-Bibliothek

> **Note**
>
> Der Inhalt dieser Seite ist möglicherweise nicht. up-to-date Das neueste Update finden Sie auf der FreeRTOS.org-Bibliotheksseite.

Einführung

Die CoreMQTT-Bibliothek ist eine Client-Implementierung des MQTT-Standards (Message Queue Telemetry Transport). Der MQTT-Standard bietet ein einfaches Publish/Subscribe- (oder PubSub) Messaging-Protokoll, das auf TCP/IP läuft und häufig in Anwendungsfällen von Machine to Machine (M2M) und Internet der Dinge (IoT) verwendet wird.

Die CoreMQTT-Bibliothek entspricht dem MQTT 3.1.1-Protokollstandard. Diese Bibliothek wurde für einen geringen Speicherbedarf optimiert. Das Design dieser Bibliothek umfasst verschiedene Anwendungsfälle, von Plattformen mit eingeschränkten Ressourcen, die nur QoS 0 MQTT PUBLISH-

Nachrichten verwenden, bis hin zu ressourcenreichen Plattformen, die QoS 2 MQTT PUBLISH über TLS (Transport Layer Security) -Verbindungen verwenden. Die Bibliothek bietet ein Menü mit zusammensetzbaren Funktionen, die ausgewählt und kombiniert werden können, um genau den Anforderungen eines bestimmten Anwendungsfalls zu entsprechen.

Die Bibliothek ist in C geschrieben und so konzipiert, dass sie ISO C90 und MISRA C:2012 entspricht. Diese MQTT-Bibliothek hat keine Abhängigkeiten von zusätzlichen Bibliotheken außer den folgenden:

- Die Standard-C-Bibliothek
- Eine vom Kunden implementierte Netzwerktransportschnittstelle
- (Optional) Eine vom Benutzer implementierte Plattformzeitfunktion

Die Bibliothek ist durch die Bereitstellung einer einfachen Spezifikation für die Sende- und Empfangsschnittstelle von den zugrunde liegenden Netzwerktreibern entkoppelt. Der Anwendungsautor kann je nach Anwendung eine vorhandene Transportschnittstelle auswählen oder eine eigene Schnittstelle implementieren.

Die Bibliothek bietet eine High-Level-API, um eine Verbindung zu einem MQTT-Broker herzustellen, ein Thema zu abonnieren/abzubestellen, eine Nachricht zu einem Thema zu veröffentlichen und eingehende Nachrichten zu empfangen. Diese API verwendet die oben beschriebene Transportschnittstelle als Parameter und verwendet sie zum Senden und Empfangen von Nachrichten an und vom MQTT-Broker.

Die Bibliothek stellt auch eine serializer/deserializer API. This API can be used to build a simple IoT application consisting of only the required a subset of MQTT functionality, without any other overhead. The serializer/deserializer Low-Level-API zur Verfügung, die in Verbindung mit jeder verfügbaren Transportschicht-API, wie Sockets, verwendet werden kann, um Nachrichten an und vom Broker zu senden und zu empfangen.

Bei der Verwendung von MQTT-Verbindungen in IoT-Anwendungen empfehlen wir, eine sichere Transportschnittstelle zu verwenden, z. B. eine, die das TLS-Protokoll verwendet.

Diese MQTT-Bibliothek hat keine Plattformabhängigkeiten wie Threading oder Synchronisation. Diese Bibliothek verfügt über Beweise, die eine sichere Speichernutzung und keine Heap-Zuweisung belegen. Dadurch ist sie für IoT-Mikrocontroller geeignet, aber auch vollständig auf andere Plattformen portierbar. Sie kann frei verwendet werden und wird unter der MIT-Open-Source-Lizenz vertrieben.

Codegröße von CoreMQTT (Beispiel generiert mit GCC für ARM Cortex-M)		
Datei	Mit -O1-Optimierung	Mit -Os-Optimierung
core_mqtt.c	4,0 K	3,4 K
core_mqtt_state.c	1,7 K	1,3 K
core_mqtt_serializer.c	2,8K	2,2 K
Schätzungen insgesamt	8,5 K	6,9 K

CoreMQTT-Agentenbibliothek

> **Note**
>
> Der Inhalt dieser Seite ist möglicherweise nicht. up-to-date Das neueste Update finden Sie auf der FreeRTOS.org-Bibliotheksseite.

Einführung

Die CoreMQTT Agentenbibliothek ist eine API auf hohem Niveau, die Thread-Sicherheit erhöht. CoreMQTT-Bibliothek Damit können Sie eine spezielle MQTT-Agentenaufgabe erstellen, die eine MQTT-Verbindung im Hintergrund verwaltet und keine Intervention durch andere Aufgaben erfordert. Die Bibliothek bietet Thread-sichere Entsprechungen zu den CoreMQTTs APIs, sodass sie in Multithread-Umgebungen verwendet werden kann.

Der MQTT-Agent ist eine unabhängige Aufgabe (oder ein Ausführungsthread). Er gewährleistet Thread-Sicherheit, da er die einzige Aufgabe ist, die auf die API der MQTT-Bibliothek zugreifen darf. Sie serialisiert den Zugriff, indem sie alle MQTT-API-Aufrufe für eine einzelne Aufgabe isoliert, und macht Semaphore oder andere Synchronisationsprimitive überflüssig.

Die Bibliothek verwendet eine Thread-sichere Messaging-Warteschlange (oder einen anderen Kommunikationsmechanismus zwischen Prozessen), um alle Anfragen zum Aufrufen von MQTT zu serialisieren. APIs Die Messaging-Implementierung ist über eine Messaging-Schnittstelle von der Bibliothek entkoppelt, sodass die Bibliothek auf andere Betriebssysteme portiert werden kann. Die Messaging-Schnittstelle besteht aus Funktionen zum Senden und Empfangen von Zeigern auf die

Befehlsstrukturen des Agenten und Funktionen zur Zuweisung dieser Befehlsobjekte, sodass der Anwendungsautor die für seine Anwendung geeignete Speicherzuweisungsstrategie festlegen kann.

Die Bibliothek ist in C geschrieben und so konzipiert, dass sie den Normen ISO C90 und MISRA C:2012 entspricht. Die Bibliothek ist nicht von anderen Bibliotheken als der Standard-C-Bibliothek CoreMQTT-Bibliothek abhängig. Die Bibliothek verfügt über Beweise, die eine sichere Speichernutzung und keine Heap-Zuweisung belegen. Sie kann also für IoT-Mikrocontroller verwendet werden, ist aber auch vollständig auf andere Plattformen portierbar.

Diese Bibliothek kann frei verwendet werden und wird unter der MIT-Open-Source-Lizenz vertrieben.

Codegröße des CoreMQTT-Agenten (Beispiel generiert mit GCC für ARM Cortex-M)		
Datei	Mit -O1-Optimierung	Mit -Os-Optimierung
core_mqtt_agent.c	1,7 K	1,5 K
core_mqtt_agent_command_functions.c	0,3 K	0,2 K
core_mqtt.c (CoreMQTT)	4,0 K	3,4 K
core_mqtt_state.c (CoreMQTT)	1,7 K	1,3 K
core_mqtt_serializer.c (CoreMQTT)	2,8K	2,2 K
Schätzungen insgesamt	10,5 K	8,6 K

AWS IoT Over-the-Air-Bibliothek (OTA)

> **Note**
>
> Der Inhalt dieser Seite ist möglicherweise nicht. up-to-date Das neueste Update finden Sie auf der FreeRTOS.org-Bibliotheksseite.

Einführung

Mit der AWS IoT Over-the-air (OTA-) Update-Bibliothek können Sie die Benachrichtigung, den Download und die Überprüfung von Firmware-Updates für FreeRTOS-Geräte verwalten, die HTTP oder MQTT als Protokoll verwenden. Mit der OTA-Agent-Bibliothek können Sie Firmware-Updates und die auf Ihren Geräten laufende Anwendung logisch trennen. Der OTA-Agent kann eine Netzwerkverbindung mit der Anwendung teilen. Durch die gemeinsame Nutzung einer Netzwerkverbindung können Sie möglicherweise eine beträchtliche Menge an RAM einsparen. Darüber hinaus können Sie mit der OTA-Agent-Bibliothek eine anwendungsspezifische Logik zum Testen, Bestätigen oder Zurücksetzen eines Firmware-Updates definieren.

Das Internet der Dinge (IoT) erweitert die Internetkonnektivität auf eingebettete Geräte, die traditionell nicht verbunden waren. Diese Geräte können so programmiert werden, dass sie nutzbare Daten über das Internet übertragen, und sie können fernüberwacht und gesteuert werden. Dank technologischer Fortschritte verfügen diese herkömmlichen eingebetteten Geräte in rasantem Tempo über Internetfunktionen für Verbraucher, Industrie und Unternehmen.

IoT-Geräte werden in der Regel in großen Mengen und oft an Orten eingesetzt, die für einen menschlichen Bediener schwer oder unpraktisch zugänglich sind. Stellen Sie sich ein Szenario vor, in dem eine Sicherheitslücke entdeckt wird, durch die Daten offengelegt werden können. In solchen Szenarien ist es wichtig, die betroffenen Geräte schnell und zuverlässig mit Sicherheitsupdates zu aktualisieren. Ohne die Möglichkeit, OTA-Updates durchzuführen, kann es auch schwierig sein, Geräte zu aktualisieren, die geografisch verstreut sind. Die Aktualisierung dieser Geräte durch einen Techniker ist kostspielig, zeitaufwändig und oft unpraktisch. Aufgrund der Zeit, die für die Aktualisierung dieser Geräte benötigt wird, sind sie für einen längeren Zeitraum Sicherheitslücken ausgesetzt. Der Rückruf dieser Geräte zur Aktualisierung ist ebenfalls kostspielig und kann aufgrund von Ausfallzeiten zu erheblichen Störungen für die Verbraucher führen.

Over-the-Air-Updates (OTA) ermöglichen es, die Gerätefirmware ohne teuren Rückruf oder Technikerbesuch zu aktualisieren. Diese Methode bietet die folgenden Vorteile:

- Sicherheit — Die Fähigkeit, schnell auf Sicherheitslücken und Softwarefehler zu reagieren, die nach dem Einsatz der Geräte vor Ort entdeckt werden.
- Innovation — Produkte können häufig aktualisiert werden, wenn neue Funktionen entwickelt werden, was den Innovationszyklus vorantreibt. Die Updates können schnell und mit minimalen Ausfallzeiten im Vergleich zu herkömmlichen Aktualisierungsmethoden wirksam werden.
- Kosten — OTA-Updates können die Wartungskosten im Vergleich zu Methoden, die traditionell zur Aktualisierung dieser Geräte verwendet werden, erheblich senken.

Die Bereitstellung der OTA-Funktionalität erfordert die folgenden Designüberlegungen:

- Sichere Kommunikation — Updates müssen verschlüsselte Kommunikationskanäle verwenden, um zu verhindern, dass die Downloads während der Übertragung manipuliert werden.
- Wiederherstellung — Updates können z. B. aufgrund einer unterbrochenen Netzwerkverbindung oder aufgrund des Empfangs eines ungültigen Updates fehlschlagen. In diesen Szenarien muss das Gerät in der Lage sein, in einen stabilen Zustand zurückzukehren und zu verhindern, dass es beschädigt wird.
- Autorenverifizierung — Es muss verifiziert werden, dass Updates von einer vertrauenswürdigen Quelle stammen, zusammen mit anderen Validierungen wie der Überprüfung von Version und Kompatibilität.

Weitere Informationen zum Einrichten von OTA-Updates mit FreeRTOS finden Sie unter. FreeRTOS RTOS-Updates Over-the-Air

AWS IoT Over-the-Air-Bibliothek (OTA)

Mit AWS IoT der OTA-Bibliothek können Sie Benachrichtigungen über neu verfügbare Updates verwalten, diese herunterladen und eine kryptografische Überprüfung von Firmware-Updates durchführen. Mithilfe der over-the-air (OTA-) Clientbibliothek können Sie die Firmware-Aktualisierungsmechanismen logisch von der Anwendung trennen, die auf Ihrem Gerät ausgeführt wird. Die over-the-air (OTA-) Clientbibliothek kann eine Netzwerkverbindung mit der Anwendung gemeinsam nutzen, wodurch Speicherplatz auf Geräten mit begrenzten Ressourcen gespart wird. Darüber hinaus können Sie mit der over-the-air (OTA-) Clientbibliothek anwendungsspezifische Logik für das Testen, Festschreiben oder Zurücksetzen eines Firmware-Updates definieren. Die Bibliothek unterstützt verschiedene Anwendungsprotokolle wie Message Queuing Telemetry Transport (MQTT) und Hypertext Transfer Protocol (HTTP) und bietet verschiedene Konfigurationsoptionen, die Sie an Ihren Netzwerktyp und Ihre Netzwerkbedingungen anpassen können.

Diese Bibliothek APIs bietet die folgenden Hauptfunktionen:

- Registrieren Sie sich für Benachrichtigungen oder fragen Sie nach neuen Aktualisierungsanfragen ab, die verfügbar sind.
- Empfangen, analysieren und validieren Sie die Aktualisierungsanfrage.
- Laden Sie die Datei herunter und überprüfen Sie sie anhand der Informationen in der Aktualisierungsanfrage.

- Führen Sie vor der Aktivierung des empfangenen Updates einen Selbsttest durch, um die Funktionsfähigkeit des Updates sicherzustellen.
- Aktualisieren Sie den Status des Geräts.

Diese Bibliothek verwendet AWS Dienste zur Verwaltung verschiedener Cloud-bezogener Funktionen wie dem Senden von Firmware-Updates, der Überwachung einer großen Anzahl von Geräten in mehreren Regionen, der Reduzierung des Explosionsradius fehlerhafter Bereitstellungen und der Überprüfung der Sicherheit von Updates. Diese Bibliothek kann mit jeder MQTT- oder HTTP-Bibliothek verwendet werden.

Die Demos für diese Bibliothek zeigen vollständige over-the-air Updates mit der CoreMQTT-Bibliothek und den AWS Diensten auf einem FreeRTOS-Gerät.

Features

Hier ist die komplette OTA-Agent-Schnittstelle:

`OTA_Init`

Initialisiert die OTA-Engine, indem der OTA-Agent („OTA-Task“) im System gestartet wird. Es darf nur ein OTA-Agent existieren.

`OTA_Shutdown`

Signalisieren Sie dem OTA-Agenten, dass er herunterfahren soll. Der OTA-Agent meldet sich optional von allen Themen zur Benachrichtigung über MQTT-Jobs ab, stoppt laufende OTA-Jobs, falls vorhanden, und löscht alle Ressourcen.

`OTA_GetState`

Ruft den aktuellen Status des OTA-Agents ab.

`OTA_ActivateNewImage`

Aktiviert das neueste Mikrocontroller-Firmware-Image, das über OTA empfangen wird. (Der detaillierte Jobstatus sollte dann "Selbsttest" sein.)

`OTA_SetImageState`

Legt den Validierungsstatus des aktuell laufenden Mikrocontroller-Firmware-Images fest (Test, Akzeptiert oder Abgelehnt).

OTA_GetImageState

Ruft den Status des aktuell laufenden Mikrocontroller-Firmware-Images ab (Test, Akzeptiert oder Abgelehnt).

OTA_CheckForUpdate

Fordert das nächste verfügbare OTA-Update vom OTA-Update-Service an.

OTA_Suspend

Unterbrechen Sie alle OTA-Agent-Operationen.

OTA_Resume

Setzen Sie den Betrieb des OTA-Agenten fort.

OTA_SignalEvent

Signalisieren Sie der OTA-Agent-Aufgabe ein Ereignis.

OTA_EventProcessingTask

Ereignisverarbeitungsschleife für OTA-Agenten.

OTA_GetStatistics

Rufen Sie die Statistiken der OTA-Nachrichtenpakete ab, einschließlich der Anzahl der empfangenen, in die Warteschlange gestellten, verarbeiteten und verworfenen Pakete.

OTA_Err_strerror

Konvertierung von Fehlercode in Zeichenfolge bei OTA-Fehlern.

OTA_JobParse_strerror

Konvertiert einen OTA-Job-Parsing-Fehlercode in eine Zeichenfolge.

OTA_PalStatus_strerror

Konvertierung von Statuscode in Zeichenfolge für den OTA-PAL-Status.

OTA_OsStatus_strerror

Konvertierung von Statuscode in Zeichenfolge für den OTA-Betriebssystemstatus.

API-Referenz

Weitere Informationen finden Sie im AWS IoT Over-the-air Update: Funktionen.

Beispielverwendung

Eine typische OTA-fähige Geräteanwendung, die das MQTT-Protokoll verwendet, steuert den OTA-Agent über die folgende Abfolge von API-Aufrufen.

1. Connect zum AWS IoT CoreMQTT Agent her. Weitere Informationen finden Sie unter CoreMQTT-Agentenbibliothek.
2. Initialisieren Sie den OTA-Agenten, indem Sie ihn aufrufen`OTA_Init`, einschließlich der Puffer, der erforderlichen OTA-Schnittstellen, des Dingnamens und des Anwendungs-Callbacks. Der Callback implementiert eine anwendungsspezifische Logik, die nach Abschluss eines OTA-Update-Jobs ausgeführt wird.
3. Wenn das OTA-Update abgeschlossen ist, ruft FreeRTOS den Job-Completion-Callback mit einem der folgenden Ereignisse auf:`accepted`,, `rejected` oder. `self test`
4. Wenn das neue Firmware-Image abgelehnt wurde (z. B. aufgrund eines Validierungsfehlers), kann die Anwendung die Benachrichtigung in der Regel ignorieren und auf das nächste Update warten.
5. Wenn das Update gültig ist und als akzeptiert markiert wurde, rufen Sie `OTA_ActivateNewImage` auf, um das Gerät zurückzusetzen und das neue Firmware-Image zu starten.

Portierung

Informationen zur Portierung der OTA-Funktionalität auf Ihre Plattform finden Sie unter Portierung der OTA-Bibliothek im FreeRTOS Porting Guide.

Speichernutzung

Codegröße von AWS IoT OTA (mit GCC für ARM Cortex-M generiertes Beispiel)		
Datei	Mit -O1-Optimierung	Mit -Os-Optimierung
ota.c	8,3 K	7,5 K
ota_interface.c	0,1 K	0,1 K
ota_base64.c	0,6 K	0,6 K
ota_mqtt.c	2,4 K	2,2 K

Codegröße von AWS IoT OTA (mit GCC für ARM Cortex-M generiertes Beispiel)		
Datei	Mit -O1-Optimierung	Mit -Os-Optimierung
ota_cbor.c	0,8 K	0,6 K
ota_http.c	0,3 K	0,3 K
Schätzungen insgesamt	12,5 K	11,3 K

PKCS11 Kernbibliothek

> **Note**
>
> Der Inhalt dieser Seite ist möglicherweise nicht. up-to-date Das neueste Update finden Sie auf der FreeRTOS.org-Bibliotheksseite.

-Übersicht

Der Public Key Cryptography Standard #11 definiert eine plattformunabhängige API zur Verwaltung und Verwendung kryptografischer Token. PKCS #11 bezieht sich auf die durch den Standard definierte API und auf den Standard selbst. Die kryptografische PKCS #11 -API abstrahiert Schlüsselspeicher, get/set Eigenschaften für kryptografische Objekte und Sitzungssemantik. Sie wird häufig für die Manipulation gängiger kryptografischer Objekte verwendet und ist wichtig, weil die darin spezifizierten Funktionen es der Anwendungssoftware ermöglichen, kryptografische Objekte zu verwenden, zu erstellen, zu ändern und zu löschen, ohne dass diese Objekte jemals dem Speicher der Anwendung zugänglich gemacht werden. Zum Beispiel verwenden AWS FreeRTOS-Referenzintegrationen eine kleine Teilmenge der PKCS #11 -API, um auf den geheimen (privaten) Schlüssel zuzugreifen, der erforderlich ist, um eine Netzwerkverbindung herzustellen, die durch das Transport Layer Security (TLS) -Protokoll authentifiziert und gesichert wird, ohne dass die Anwendung den Schlüssel jemals „sieht“.

Die PKCS11 Kernbibliothek enthält eine softwarebasierte Scheinimplementierung der PKCS #11 -Schnittstelle (API), die die von Mbed TLS bereitgestellten kryptografischen Funktionen verwendet. Die Verwendung eines Softwaremocks ermöglicht eine schnelle Entwicklung und Flexibilität. Es wird jedoch erwartet, dass Sie das Modell durch eine Implementierung ersetzen, die speziell auf den sicheren Schlüsselspeicher Ihrer Produktionsgeräte zugeschnitten ist. Im Allgemeinen verteilen

Anbieter von sicheren Kryptoprozessoren, wie Trusted Platform Module (TPM), Hardware Security Module (HSM), Secure Element oder jede andere Art von sicherer Hardware-Enklave, zusammen mit der Hardware eine PKCS #11 -Implementierung. Der Zweck der rein PKCS11 softwarebasierten Modellbibliothek besteht daher darin, eine nicht hardwarespezifische PKCS #11 -Implementierung bereitzustellen, die ein schnelles Prototyping und eine schnelle Entwicklung ermöglicht, bevor auf eine kryptoprozessorspezifische PKCS #11 -Implementierung in Produktionsgeräten umgestellt wird.

Nur ein Teil des PKCS #11 -Standards ist implementiert, wobei der Schwerpunkt auf Operationen mit asymmetrischen Schlüsseln, Generierung von Zufallszahlen und Hashing liegt. Zu den gezielten Anwendungsfällen gehören die Zertifikats- und Schlüsselverwaltung für die TLS-Authentifizierung sowie die Überprüfung von Codesignaturen auf kleinen eingebetteten Geräten. Sehen Sie sich die Datei `pkcs11.h` (bezogen von OASIS, dem Standardtext) im FreeRTOS-Quellcode-Repository an. In der FreeRTOS-Referenzimplementierung werden PKCS #11 -API-Aufrufe von der TLS-Hilfsschnittstelle ausgeführt, um währenddessen die TLS-Client-Authentifizierung durchzuführen. `SOCKETS_Connect` PKCS #11 -API-Aufrufe werden auch von unserem einmaligen Entwickler-Bereitstellungs-Workflow ausgeführt, um ein TLS-Client-Zertifikat und einen privaten Schlüssel für die Authentifizierung beim MQTT-Broker zu importieren. AWS IoT Für diese beiden Anwendungsfälle, Bereitstellung und TLS-Client-Authentifizierung, muss nur ein kleiner Teil des PKCS #11 -Schnittstellenstandards implementiert werden.

Features

Die folgende Teilmenge von PKCS #11 wird verwendet. Diese Liste entspricht in etwa der Reihenfolge, in der die Routinen für die Bereitstellung, TLS-Client-Authentifizierung und Bereinigung aufgerufen werden. Eine ausführliche Beschreibung der Funktionen finden Sie in der vom Standardausschuss bereitgestellten PKCS #11 -Dokumentation.

Allgemeine Einrichtung und Teardown-API

- `C_Initialize`
- `C_Finalize`
- `C_GetFunctionList`
- `C_GetSlotList`
- `C_GetTokenInfo`
- `C_OpenSession`
- `C_CloseSession`
- `C_Login`

Bereitstellungs-API

- `C_CreateObject CKO_PRIVATE_KEY` (für privaten Schlüssel des Geräts)
- `C_CreateObject CKO_CERTIFICATE` (für Gerätezertifikat und Codeverifizierungszertifikat)
- `C_GenerateKeyPair`
- `C_DestroyObject`

Client-Authentifizierung

- `C_GetAttributeValue`
- `C_FindObjectsInit`
- `C_FindObjects`
- `C_FindObjectsFinal`
- `C_GenerateRandom`
- `C_SignInit`
- `C_Sign`
- `C_VerifyInit`
- `C_Verify`
- `C_DigestInit`
- `C_DigestUpdate`
- `C_DigestFinal`

Unterstützung asymmetrischer Kryptosysteme

Die FreeRTOS-Referenzimplementierung verwendet PKCS #11 2048-Bit-RSA (nur Signierung) und ECDSA mit der NIST-P-256-Kurve. Die folgenden Anweisungen beschreiben, wie Sie etwas erstellen, das auf einem P-256-Client-Zertifikat basiert. AWS IoT

Stellen Sie sicher, dass Sie die folgenden (oder neueren) Versionen von AWS CLI und OpenSSL verwenden:

```
aws --version
aws-cli/1.11.176 Python/2.7.9 Windows/8 botocore/1.7.34

openssl version
```

```
OpenSSL 1.0.2g  1 Mar 2016
```

Beim folgenden Verfahren wird davon ausgegangen, dass Sie den `aws configure` Befehl zur Konfiguration von verwendet haben. AWS CLI Weitere Informationen finden Sie unter Schnellkonfiguration mit `aws configure` im AWS Command Line Interface Benutzerhandbuch.

Um AWS IoT etwas zu erstellen, das auf einem P-256-Client-Zertifikat basiert

1. Erstelle ein AWS IoT Ding.

   ```
   aws iot create-thing --thing-name thing-name
   ```

2. Verwenden Sie OpenSSL, um einen P-256-Schlüssel zu erstellen.

   ```
   openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -pkeyopt
    ec_param_enc:named_curve -outform PEM -out thing-name.key
   ```

3. Erstellen Sie eine Zertifikatregistrierungsanfrage, die mit dem in Schritt 2 erstellten Schlüssel signiert ist.

   ```
   openssl req -new -nodes -days 365 -key thing-name.key -out thing-name.req
   ```

4. Senden Sie die Anfrage zur Registrierung des Zertifikats an. AWS IoT

   ```
   aws iot create-certificate-from-csr  \
     --certificate-signing-request file://thing-name.req --set-as-active  \
     --certificate-pem-outfile thing-name.crt
   ```

5. Fügen Sie das Zertifikat (auf das die ARN-Ausgabe des vorherigen Befehls verweist) an das Objekt an.

   ```
   aws iot attach-thing-principal --thing-name thing-name \
     --principal "arn:aws:iot:us-
   east-1:123456789012:cert/
   86e41339a6d1bbc67abf31faf455092cdebf8f21ffbc67c4d238d1326c7de729"
   ```

6. Erstellen Sie eine Richtlinie. (Diese Richtlinie ist zu freizügig. Sie sollte nur für Entwicklungszwecke verwendet werden.)

   ```
   aws iot create-policy --policy-name FullControl --policy-document file://
   policy.json
   ```

Im Folgenden finden Sie die Datei policy.json, die im Befehl `create-policy` angegeben ist. Sie können die `greengrass:*` Aktion auslassen, wenn Sie die FreeRTOS-Demo für Greengrass-Konnektivität und -Erkennung nicht ausführen möchten.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:*",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "greengrass:*",
      "Resource": "*"
    }
  ]
}
```

7. Fügen Sie den Prinzipal (Zertifikat) und die Richtlinie an das Objekt an.

```
aws iot attach-principal-policy --policy-name FullControl \
  --principal "arn:aws:iot:us-
east-1:123456789012:cert/
86e41339a6d1bbc67abf31faf455092cdebf8f21ffbc67c4d238d1326c7de729"
```

Führen Sie nun die Schritte im Abschnitt Erste Schritte mit AWS IoT in diesem Handbuch aus. Vergessen Sie nicht, das von Ihnen erstellte Zertifikat und den privaten Schlüssel in die `aws_clientcredential_keys.h`-Datei zu kopieren. Kopieren Sie den Objektnamen in `aws_clientcredential.h`.

> **Note**
>
> Das Zertifikat und der private Schlüssel sind nur für Demonstrationszwecke hardcodiert. Anwendungen auf Produktionsebene sollten diese Dateien an einem sicheren Ort speichern.

Portierung

Informationen zur Portierung der PKCS11 Core-Bibliothek auf Ihre Plattform finden Sie unter Porting the core PKCS11 Library im FreeRTOS Porting Guide.

Speichernutzung

Codegröße des Kerns PKCS11 (mit GCC für ARM Cortex-M generiertes Beispiel)		
Datei	Mit -O1-Optimierung	Mit -Os-Optimierung
core_pkcs11.c	0,8 K	0,8 K
core_pki_utils.c	0,5 K	0,3 K
core_pkcs11_mbedtls.c	8,9 K	7,5 K
Schätzungen insgesamt	10,2 K	8,6 K

Secure-Sockets-Bibliothek

> **Important**
>
> Diese Bibliothek wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

-Übersicht

Sie können die FreeRTOS Secure Sockets-Bibliothek verwenden, um eingebettete Anwendungen zu erstellen, die sicher kommunizieren. Die Bibliothek dient dem einfachen Einstieg von Softwareentwicklern mit unterschiedlichem Hintergrund im Bereich Netzwerkprogrammierung.

Die FreeRTOS Secure Sockets-Bibliothek basiert auf der Berkeley-Sockets-Schnittstelle und bietet eine zusätzliche sichere Kommunikationsoption per TLS-Protokoll. Informationen zu den

Unterschieden zwischen der FreeRTOS Secure Sockets-Bibliothek und der Berkeley-Sockets-Schnittstelle finden Sie `SOCKETS_SetSockOpt` in der Secure Sockets API-Referenz.

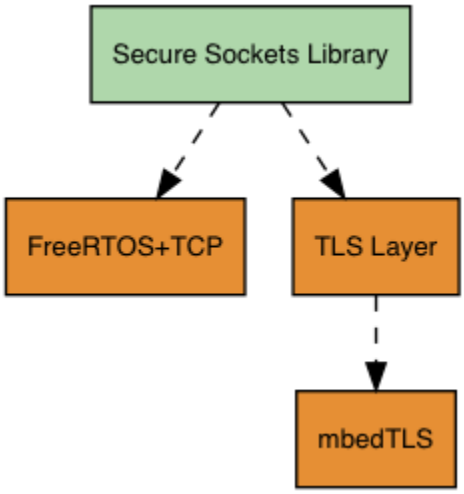
> **Note**
>
> Derzeit werden für APIs FreeRTOS Secure Sockets nur Clients und eine Lightweight IP-Implementierung (LwIP) der serverseitigen `Bind` API unterstützt.

Abhängigkeiten und Anforderungen

Die FreeRTOS Secure Sockets-Bibliothek hängt von einem TCP/IP Stack und einer TLS-Implementierung ab. Ports für FreeRTOS erfüllen diese Abhängigkeiten auf eine von drei Arten:

- Eine benutzerdefinierte Implementierung von beiden TCP/IP und TLS
- Eine benutzerdefinierte Implementierung von TCP/IP und der FreeRTOS-TLS-Schicht mit mbedTLS
- FreeRTOS+TCP und die FreeRTOS-TLS-Schicht mit mbedTLS

Das folgende Abhängigkeitsdiagramm zeigt die Referenzimplementierung, die in der FreeRTOS Secure Sockets-Bibliothek enthalten ist. Diese Referenzimplementierung unterstützt TLS und TCP/IP über Ethernet und Wi-Fi mit FreeRTOS+TCP und mbedTLS als Abhängigkeiten. Weitere Hinweise zur FreeRTOS-TLS-Schicht finden Sie unter. Transport Layer Security



Features

Zu den Funktionen der FreeRTOS Secure Sockets-Bibliothek gehören:

- Eine Standard-, Berkeley-Sockets-basierte Schnittstelle
- Thread-sicher APIs für das Senden und Empfangen von Daten
- Easy-to-enable TLS

Fehlerbehebung

Fehlercodes

Die Fehlercodes, die die FreeRTOS Secure Sockets-Bibliothek zurückgibt, sind negative Werte. Weitere Informationen zu den einzelnen Fehlercodes finden Sie unter „Secure Sockets-Fehlercodes“ in der Secure Sockets API-Referenz.

> **Note**
>
> Wenn die FreeRTOS Secure Sockets API einen Fehlercode zurückgibt, gibt derCoreMQTT-Bibliothek, der von der FreeRTOS Secure Sockets-Bibliothek abhängt, den Fehlercode zurück. `AWS_IOT_MQTT_SEND_ERROR`

Developer Support

Die FreeRTOS Secure Sockets-Bibliothek enthält zwei Hilfsmakros für den Umgang mit IP-Adressen:

`SOCKETS_inet_addr_quick`

Dieses Makro wandelt eine IP-Adresse, die als vier separate numerische Oktette ausgedrückt wird, in eine IP-Adresse um, die als 32-Bit-Zahl in Netzwerk-Byte-Reihenfolge ausgedrückt wird.

`SOCKETS_inet_ntoa`

Dieses Makro wandelt eine IP-Adresse, die als 32-Bit-Zahl in Netzwerk-Byte-Reihenfolge ausgedrückt wird, in eine Zeichenkette in Dezimalpunkt-Notation um.

Nutzungsbeschränkungen

Nur TCP-Sockets werden von der FreeRTOS Secure Sockets-Bibliothek unterstützt. UDP-Sockets werden nicht unterstützt.

Server APIs werden von der FreeRTOS Secure Sockets-Bibliothek nicht unterstützt, mit Ausnahme einer Lightweight IP (lwIP) -Implementierung der serverseitigen API. `Bind` Clients werden unterstützt. APIs

Initialisierung

Um die FreeRTOS Secure Sockets-Bibliothek zu verwenden, müssen Sie die Bibliothek und ihre Abhängigkeiten initialisieren. Um die Secure-Sockets-Bibliothek zu initialisieren, verwenden Sie den folgenden Code in Ihrer Anwendung:

```
BaseType_t xResult = pdPASS;
xResult = SOCKETS_Init();
```

Abhängige Bibliotheken müssen separat initialisiert werden. Beispiel: Wenn FreeRTOS+TCP eine Abhängigkeit ist, müssen Sie auch `FreeRTOS_IPInit` in Ihrer Anwendung aufrufen.

API-Referenz

Eine vollständige API-Referenz finden Sie unter Secure Sockets API-Referenz.

Beispielverwendung

Der folgende Code verbindet einen Client mit einem Server.

```
#include "aws_secure_sockets.h"

#define configSERVER_ADDR0                     127
#define configSERVER_ADDR1                     0
#define configSERVER_ADDR2                     0
#define configSERVER_ADDR3                     1
#define configCLIENT_PORT                      443

/* Rx and Tx timeouts are used to ensure the sockets do not wait too long for
 * missing data. */
static const TickType_t xReceiveTimeOut = pdMS_TO_TICKS( 2000 );
static const TickType_t xSendTimeOut = pdMS_TO_TICKS( 2000 );

/* PEM-encoded server certificate */
/* The certificate used below is one of the Amazon Root CAs.\
Change this to the certificate of your choice. */
static const char cTlsECHO_SERVER_CERTIFICATE_PEM[] =
"-----BEGIN CERTIFICATE-----\n"
"MIIBtjCCAVugAwIBAgITBmyf1XSXNmY/Owua2eiedgPySjAKBggqhkjOPQQDAjA5\n"
```

```
"MQswCQYDVQQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwFwYDVQQDExBBbWF6b24g\n"
"Um9vdCBDQSAzMB4XDTE1MDUyNjAwMDAwMFoXDTQwMDUyNjAwMDAwMFowOTELMAkG\n"
"A1UEBhMCVVMxDzANBgNVBAoTBkFtYXpvbjEZMBcGA1UEAxMQQW1hem9uIFJvb3Qg\n"
"Q0EgMzBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABCmXp8ZBf8ANm+gBG1bG8lKl\n"
"ui2yEujSLtf6ycXYqm0fc4E7O5hrOXwzpcVOho6AF2hiRVd9RFgdszflZwjrZt6j\n"
"QjBAMA8GA1UdEwEB/wQFMAMBAf8wDgYDVR0PAQH/BAQDAgGGMB0GA1UdDgQWBBSr\n"
"ttvXBp43rDCGB5Fwx5zEGbF4wDAKBggqhkjOPQQDAgNJADBGAiEA4IWSoxe3jfkr\n"
"BqWTrBqYaGFy+uGh0PsceGCmQ5nFuMQCIQCcAu/xlJyzlvnrxir4tiz+OpAUFteM\n"
"YyRIHN8wfdVoOw==\n"
"-----END CERTIFICATE-----\n";

static const uint32_t ulTlsECHO_SERVER_CERTIFICATE_LENGTH =
 sizeof( cTlsECHO_SERVER_CERTIFICATE_PEM );

void vConnectToServerWithSecureSocket( void )
{
    Socket_t xSocket;
    SocketsSockaddr_t xEchoServerAddress;
    BaseType_t xTransmitted, lStringLength;

    xEchoServerAddress.usPort = SOCKETS_htons( configCLIENT_PORT );
    xEchoServerAddress.ulAddress = SOCKETS_inet_addr_quick( configSERVER_ADDR0,
                                                            configSERVER_ADDR1,
                                                            configSERVER_ADDR2,
                                                            configSERVER_ADDR3 );

    /* Create a TCP socket. */
    xSocket = SOCKETS_Socket( SOCKETS_AF_INET, SOCKETS_SOCK_STREAM,
 SOCKETS_IPPROTO_TCP );
    configASSERT( xSocket != SOCKETS_INVALID_SOCKET );

    /* Set a timeout so a missing reply does not cause the task to block indefinitely.
 */
    SOCKETS_SetSockOpt( xSocket, 0, SOCKETS_SO_RCVTIMEO, &xReceiveTimeOut,
 sizeof( xReceiveTimeOut ) );
    SOCKETS_SetSockOpt( xSocket, 0, SOCKETS_SO_SNDTIMEO, &xSendTimeOut,
 sizeof( xSendTimeOut ) );

    /* Set the socket to use TLS. */
    SOCKETS_SetSockOpt( xSocket, 0, SOCKETS_SO_REQUIRE_TLS, NULL, ( size_t ) 0 );
    SOCKETS_SetSockOpt( xSocket, 0, SOCKETS_SO_TRUSTED_SERVER_CERTIFICATE,
 cTlsECHO_SERVER_CERTIFICATE_PEM, ulTlsECHO_SERVER_CERTIFICATE_LENGTH );
```

```
    if( SOCKETS_Connect( xSocket, &xEchoServerAddress, sizeof( xEchoServerAddress ) )
 == 0 )
    {
        /* Send the string to the socket. */
        xTransmitted = SOCKETS_Send( xSocket,                          /* The socket
 receiving. */
                                     ( void * )"some message",         /* The data being
 sent. */
                                     12,                               /* The length of
 the data being sent. */
                                     0 );                              /* No flags. */

        if( xTransmitted < 0 )
        {
            /* Error while sending data */
            return;
        }

        SOCKETS_Shutdown( xSocket, SOCKETS_SHUT_RDWR );
    }
    else
    {
        //failed to connect to server
    }

    SOCKETS_Close( xSocket );
}
```

Ein vollständiges Beispiel finden Sie unter Secure Sockets Echo-Client-Demo.

Portierung

FreeRTOS Secure Sockets hängt von einem TCP/IP Stack und einer TLS-Implementierung ab. Abhängig von Ihrem Stack müssen Sie für die Portierung der Secure-Sockets-Bibliothek möglicherweise einige der folgenden Portierungen vornehmen:

- Der TCP/IP FreeRTOS+TCP-Stack
- Die PKCS11 Kernbibliothek
- Die Transport Layer Security

Weitere Informationen zur Portierung finden Sie unter Portierung der Secure Sockets Library im FreeRTOS Porting Guide.

AWS IoT Device Shadow-Bibliothek

> **Note**
>
> Der Inhalt dieser Seite ist möglicherweise nicht. up-to-date Das neueste Update finden Sie auf der FreeRTOS.org-Bibliotheksseite.

Einführung

Sie können die AWS IoT Device Shadow-Bibliothek verwenden, um den aktuellen Status (den Shadow) jedes registrierten Geräts zu speichern und abzurufen. Der Schatten des Geräts ist eine persistente, virtuelle Darstellung Ihres Geräts, mit der Sie in Ihren Webanwendungen interagieren können, auch wenn das Gerät offline ist. Der Gerätestatus wird als sein Schatten in einem JSON-Dokument erfasst. Sie können Befehle über MQTT oder HTTP an den AWS IoT Device Shadow-Dienst senden, um den letzten bekannten Gerätestatus abzufragen oder den Status zu ändern. Der Shadow jedes Geräts wird eindeutig durch den Namen der entsprechenden Sache identifiziert, eine Repräsentation eines bestimmten Geräts oder einer logischen Entität in der AWS Cloud. Weitere Informationen finden Sie unter Geräte verwalten mit AWS IoT. Weitere Informationen zu Schatten finden Sie in der AWS IoT Dokumentation.

Die AWS IoT Device Shadow-Bibliothek ist nicht von anderen Bibliotheken als der Standard-C-Bibliothek abhängig. Sie hat auch keine Plattformabhängigkeiten wie Threading oder Synchronisation. Es kann mit jeder MQTT-Bibliothek und jeder JSON-Bibliothek verwendet werden.

Diese Bibliothek kann frei verwendet werden und wird unter der MIT-Open-Source-Lizenz vertrieben.

Codegröße von AWS IoT Device Shadow (mit GCC für ARM Cortex-M generiertes Beispiel)		
Datei	Mit -O1-Optimierung	Mit -Os-Optimierung
shadow.c	1,2 K	0,9 K
Schätzungen insgesamt	1,2 K	0,9 K

AWS IoT Bibliothek mit Stellenangeboten

> **Note**
>
> Der Inhalt dieser Seite ist möglicherweise nicht. up-to-date Das neueste Update finden Sie auf der FreeRTOS.org-Bibliotheksseite.

Einführung

AWS IoT Jobs ist ein Dienst, der ein oder mehrere verbundene Geräte über einen ausstehenden Job informiert. Sie können einen Job verwenden, um Ihre Geräteflotte zu verwalten, Firmware und Sicherheitszertifikate auf Ihren Geräten zu aktualisieren oder administrative Aufgaben wie den Neustart von Geräten und die Durchführung von Diagnosen durchzuführen. Weitere Informationen finden Sie unter Jobs im AWS IoT Developer Guide. Interaktionen mit dem AWS IoT Jobs-Service verwenden MQTT, ein einfaches Publish-Subscribe-Protokoll. Diese Bibliothek bietet eine API zum Verfassen und Erkennen der vom Jobs-Service verwendeten MQTT-Themenzeichenfolgen. AWS IoT

Die AWS IoT Jobs-Bibliothek ist in C geschrieben und so konzipiert, dass sie ISO C90 und MISRA C:2012 entspricht. Die Bibliothek ist nicht von anderen Bibliotheken als der Standard-C-Bibliothek abhängig. Sie kann mit jeder MQTT-Bibliothek und jeder JSON-Bibliothek verwendet werden. Die Bibliothek verfügt über Beweise für eine sichere Speichernutzung und keine Heap-Zuweisung, sodass sie für IoT-Mikrocontroller geeignet ist, aber auch vollständig auf andere Plattformen portierbar ist.

Diese Bibliothek kann frei verwendet werden und wird unter der MIT-Open-Source-Lizenz vertrieben.

Codegröße der AWS IoT Jobs (mit GCC für ARM Cortex-M generiertes Beispiel)		
Datei	Mit -O1-Optimierung	Mit -Os-Optimierung
jobs.c	1,9 TSD.	1,6 K
Schätzungen insgesamt	1,9 K	1,6 K

Transport Layer Security

> **Important**
>
> Diese Bibliothek wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Die FreeRTOS Transport Layer Security (TLS) -Schnittstelle ist ein dünner, optionaler Wrapper, der verwendet wird, um kryptografische Implementierungsdetails von der Secure Sockets Layer (SSL) -Schnittstelle zu abstrahieren, die sich darüber im Protokollstapel befindet. Der Zweck der TLS-Schnittstelle ist es, die aktuelle Software-Krypto-Bibliothek (mbed TLS) durch eine einfachere alternative Implementierung für die Aushandlung des TLS-Protokolls und Kryptografie-Primitiven zu ersetzen. Die TLS-Schnittstelle kann ohne Änderungen an der SSL-Schnittstelle ausgetauscht werden. Siehe `iot_tls.h` im FreeRTOS-Quellcode-Repository.

Die TLS-Schnittstelle ist optional. Sie können eine Schnittstelle auch direkt von SSL in eine Krypto-Bibliothek erstellen. Die -Schnittstelle wird nicht für MCU-Lösungen verwendet, die eine Full-Stack-Offload-Implementierung von TLS und Netzwerktransport beinhalten.

Weitere Informationen zur Portierung der TLS-Schnittstelle finden Sie unter Porting the TLS Library im FreeRTOS Porting Guide.

WLAN-Bibliothek

> **Important**
>
> Diese Bibliothek wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

-Übersicht

Die FreeRTOS Wi-Fi-Bibliothek abstrahiert portspezifische Wi-Fi-Implementierungen in einer gemeinsamen API, die die Anwendungsentwicklung und Portierung für alle FreeRTOS-qualifizierten Boards mit Wi-Fi-Funktionen vereinfacht. Mithilfe dieser gemeinsamen API können Anwendungen mit ihrem untergeordneten Wireless-Stack über eine gemeinsame Schnittstelle kommunizieren.

Abhängigkeiten und Anforderungen

Die FreeRTOS Wi-Fi-Bibliothek benötigt den FreeRTOS+TCP-Kern.

Features

Die WLAN-Bibliothek enthält die folgenden Funktionen:

- Support für WEP, WPA und WPA2 Authentifizierung WPA3
- Zugriffspunkt-Scanning
- Energiemanagement
- Netzwerk-Profiling

Weitere Informationen zu den Funktionen der WLAN-Bibliothek finden Sie unten.

WLAN-Modi

WLAN-Geräte können sich in einem von drei Modi befinden: Station, Zugriffspunkt oder P2P. Sie können den aktuellen Modus eines WLAN-Geräts abrufen, indem Sie `WIFI_GetMode` aufrufen. Sie können den WLAN-Modus eines Geräts festlegen, indem Sie `WIFI_SetMode` aufrufen. Das Umschalten des Modus durch Aufruf von `WIFI_SetMode` trennt das Gerät (wenn es bereits mit einem Netzwerk verbunden ist).

Stationsmodus

Stellen Sie Ihr Gerät auf Stationsmodus ein, um die Karte mit einem vorhandenen Zugriffspunkt zu verbinden.

Access Point (AP)-Modus

Stellen Sie Ihr Gerät auf AP-Modus ein, um das Gerät zu einem Zugriffspunkt zu machen, über das andere Geräte eine Verbindung herstellen können. Wenn sich Ihr Gerät im AP-Modus

befindet, können Sie ein anderes Gerät mit Ihrem FreeRTOS-Gerät verbinden und die neuen WLAN-Anmeldeinformationen konfigurieren. Rufen Sie `WIFI_ConfigureAP` auf, um den AP-Modus zu konfigurieren. Rufen Sie `WIFI_StartAP` auf, um Ihr Gerät in den AP-Modus zu versetzen. Rufen Sie `WIFI_StopAP` auf, um den AP-Modus zu deaktivieren.

> **Note**
>
> FreeRTOS-Bibliotheken bieten keine Wi-Fi-Bereitstellung im AP-Modus. Sie müssen die zusätzliche Funktionalität, einschließlich der DHCP- und HTTP-Serverfunktionen, bereitstellen, um eine vollständige Unterstützung des AP-Modus zu erreichen.

P2P-Modus

Stellen Sie Ihr Gerät auf P2P-Modus ein, um mehreren Geräten zu erlauben, sich miteinander ohne Zugriffspunkt zu verbinden.

Sicherheit

Die Wi-Fi-API unterstützt WEP-, WPA- und Sicherheitstypen. WPA2 WPA3 Wenn sich ein Gerät im Stationsmodus befindet, müssen Sie beim Aufruf der Funktion `WIFI_ConnectAP` den Netzwerksicherheitstyp angeben. Wenn sich ein Gerät im AP-Modus befindet, kann das Gerät so konfiguriert werden, dass es einen der unterstützten Sicherheitstypen verwendet:

- `eWiFiSecurityOpen`
- `eWiFiSecurityWEP`
- `eWiFiSecurityWPA`
- `eWiFiSecurityWPA2`
- `eWiFiSecurityWPA3`

Scannen und Verbinden

Um nach nahegelegenen Zugriffspunkten zu suchen, stellen Sie Ihr Gerät auf Stationsmodus und rufen Sie die Funktion `WIFI_Scan` auf. Wenn Sie über den Scan ein gewünschtes Netzwerk finden, können Sie sich mit dem Netzwerk verbinden, indem Sie `WIFI_ConnectAP` aufrufen und die Anmeldeinformationen für das Netzwerk angeben. Sie können ein WLAN-Gerät vom

Netzwerk trennen, indem Sie `WIFI_Disconnect` aufrufen. Weitere Informationen zum Scannen und Verbinden finden Sie unter Beispielverwendung und API-Referenz.

Energiemanagement

Verschiedene WLAN-Geräte haben unterschiedliche Energieanforderungen – je nach Anwendung und Stromquellen. Ein Gerät kann immer eingeschaltet sein (um die Latenz zu reduzieren) oder es kann intermittierend verbunden sein und in einen Energiesparmodus wechseln, wenn WLAN nicht erforderlich ist. Die Schnittstellen-API unterstützt verschiedene Energiesparmodi wie Always On, Low Power und Normal Mode. Sie legen den Energiesparmodus für ein Gerät mit der Funktion `WIFI_SetPMMode` fest. Den aktuellen Energiesparmodus eines Gerätes erhalten Sie durch Aufruf der Funktion `WIFI_GetPMMode`.

Netzwerkprofile

Mit der WLAN-Bibliothek können Sie Netzwerkprofile im nichtflüchtigen Speicher Ihrer Geräte speichern. Auf diese Weise können Sie Netzwerkeinstellungen speichern, damit sie abgerufen werden können, wenn sich ein Gerät wieder mit einem WLAN-Netzwerk verbindet. So ist es nicht mehr erforderlich, Geräte erneut bereitzustellen, nachdem sie mit einem Netzwerk verbunden wurden. `WIFI_NetworkAdd` fügt ein Netzwerkprofil hinzu. `WIFI_NetworkGet` ruft ein Netzwerkprofil ab. `WIFI_NetworkDel` löscht ein Netzwerkprofil. Die Anzahl der speicherbaren Profile hängt von der Plattform ab.

Konfiguration

Um die WLAN-Bibliothek nutzen zu können, müssen Sie mehrere IDs in einer Konfigurationsdatei definieren. Weitere Informationen zu diesen IDs finden Sie unter API-Referenz.

> **Note**
>
> Die Bibliothek enthält nicht die erforderliche Konfigurationsdatei. Sie müssen eine erstellen. Achten Sie bei der Erstellung Ihrer Konfigurationsdatei darauf, dass Sie alle Board-spezifischen Konfigurationskennungen angeben, die Ihr Board benötigt.

Initialisierung

Bevor Sie die WLAN-Bibliothek nutzen können, müssen Sie neben den FreeRTOS-Komponenten auch einige Board-spezifische Komponenten initialisieren. Gehen Sie unter Verwendung der Datei

`vendors/vendor/boards/board/aws_demos/application_code/main.c` als Vorlage für die Initialisierung wie folgt vor:

1. Entfernen Sie die Beispiel-WLAN-Verbindungslogik in `main.c`, wenn Ihre Anwendung WLAN-Verbindungen verarbeitet. Ersetzen Sie den folgenden `DEMO_RUNNER_RunDemos()`-Funktionsaufruf:

   ```
   if( SYSTEM_Init() == pdPASS )
       {
       ...
           DEMO_RUNNER_RunDemos();
       ...
           }
   ```

   Mit einem Aufruf Ihrer eigenen Anwendung:

   ```
   if( SYSTEM_Init() == pdPASS )
       {
       ...
           // This function should create any tasks
           // that your application requires to run.
           YOUR_APP_FUNCTION();
       ...
           }
   ```

2. Rufen Sie `WIFI_On()` auf, um Ihren WLAN-Chip zu initialisieren und einzuschalten.

   > **Note**
   >
   > Einige Boards erfordern möglicherweise zusätzliche Hardware-Initialisierung.

3. Übergeben Sie eine konfigurierte `WIFINetworkParams_t`-Struktur an `WIFI_ConnectAP()`, um Ihr Board mit einem verfügbaren WLAN-Netzwerk zu verbinden. Weitere Informationen zu der `WIFINetworkParams_t`-Struktur finden Sie unter Beispielverwendung und API-Referenz.

API-Referenz

Die vollständige API-Referenz finden Sie in der WLAN-API-Referenz.

Beispielverwendung

Herstellen einer Verbindung zu einem bekannten AP

```
#define clientcredentialWIFI_SSID    "MyNetwork"
#define clientcredentialWIFI_PASSWORD   "hunter2"

WIFINetworkParams_t xNetworkParams;
WIFIReturnCode_t xWifiStatus;

xWifiStatus = WIFI_On(); // Turn on Wi-Fi module

// Check that Wi-Fi initialization was successful
if( xWifiStatus == eWiFiSuccess )
{
    configPRINT( ( "WiFi library initialized.\n") );
}
else
{
    configPRINT( ( "WiFi library failed to initialize.\n" ) );
    // Handle module init failure
}

/* Setup parameters. */
xNetworkParams.pcSSID = clientcredentialWIFI_SSID;
xNetworkParams.ucSSIDLength = sizeof( clientcredentialWIFI_SSID );
xNetworkParams.pcPassword = clientcredentialWIFI_PASSWORD;
xNetworkParams.ucPasswordLength = sizeof( clientcredentialWIFI_PASSWORD );
xNetworkParams.xSecurity = eWiFiSecurityWPA2;

// Connect!
xWifiStatus = WIFI_ConnectAP( &( xNetworkParams ) );

if( xWifiStatus == eWiFiSuccess )
{
    configPRINT( ( "WiFi Connected to AP.\n" ) );
    // IP Stack will receive a network-up event on success
}
else
{
    configPRINT( ( "WiFi failed to connect to AP.\n" ) );
    // Handle connection failure
}
```

Es wird nach nahegelegenen Objekten gesucht APs

```
WIFINetworkParams_t xNetworkParams;
WIFIReturnCode_t xWifiStatus;

configPRINT( ("Turning on wifi...\n") );
xWifiStatus = WIFI_On();

configPRINT( ("Checking status...\n") );
if( xWifiStatus == eWiFiSuccess )
{
    configPRINT( ("WiFi module initialized.\n") );
}
else
{
    configPRINTF( ("WiFi module failed to initialize.\n" ) );
    // Handle module init failure
}

WIFI_SetMode(eWiFiModeStation);

/* Some boards might require additional initialization steps to use the Wi-Fi library.
 */

while (1)
{
    configPRINT( ("Starting scan\n") );
    const uint8_t ucNumNetworks = 12; //Get 12 scan results
    WIFIScanResult_t xScanResults[ ucNumNetworks ];
    xWifiStatus = WIFI_Scan( xScanResults, ucNumNetworks ); // Initiate scan

    configPRINT( ("Scan started\n") );

    // For each scan result, print out the SSID and RSSI
    if ( xWifiStatus == eWiFiSuccess )
    {
        configPRINT( ("Scan success\n") );
        for ( uint8_t i=0; i<ucNumNetworks; i++ )
        {
            configPRINTF( ("%s : %d \n", xScanResults[i].cSSID,
 xScanResults[i].cRSSI) );
        }
    } else {
        configPRINTF( ("Scan failed, status code: %d\n", (int)xWifiStatus) );
```

```
    }

    vTaskDelay(200);
}
```

Portierung

Die Implementierung von `iot_wifi.c` muss die in `iot_wifi.h` definierten Funktionen implementieren. Zumindest muss die Implementierung `eWiFiNotSupported` für alle nicht wesentlichen oder nicht unterstützten Funktionen zurückgeben.

Weitere Informationen zur Portierung der Wi-Fi-Bibliothek finden Sie unter Portierung der Wi-Fi-Bibliothek im FreeRTOS Porting Guide.

FreeRTOS RTOS-Demos

FreeRTOS enthält einige Demo-Anwendungen im `demos` Ordner unter dem FreeRTOS-Hauptverzeichnis. Alle Beispiele, die von FreeRTOS ausgeführt werden können, `common` befinden sich im Ordner unter. `demos` Unter dem Ordner befindet sich auch ein Ordner für jede FreeRTOS-qualifizierte Plattform. `demos`

Bevor Sie die Demo-Anwendungen ausprobieren, empfehlen wir Ihnen, das Tutorial unter Erste Schritte mit FreeRTOS abzuschließen. Es zeigt Ihnen, wie Sie die CoreMQTT Agent-Demo einrichten und ausführen.

Die FreeRTOS-Demos ausführen

Die folgenden Themen zeigen Ihnen, wie Sie die FreeRTOS-Demos einrichten und ausführen:

- Bluetooth Low Energy-Demoanwendungen
- Demo-Bootloader für den Microchip Curiosity MZEF PIC32
- AWS IoT Device Defender Demo
- AWS IoT Greengrass V1 Discovery-Demoanwendung
- AWS IoT Greengrass V2
- CoreHTTP-Demos
- AWS IoT Demo der Jobbibliothek
- CoreMQTT-Demos
- Over-the-air aktualisiert die Demo-Anwendung

- Secure Sockets Echo-Client-Demo
- AWS IoT Device Shadow-Demoanwendung

Die `DEMO_RUNNER_RunDemos` Funktion, die sich in der *`freertos`*`/demos/demo_runner/iot_demo_runner.c` Datei befindet, initialisiert einen getrennten Thread, auf dem eine einzelne Demo-Anwendung ausgeführt wird. Standardmäßig ruft und startet `DEMO_RUNNER_RunDemos` nur die CoreMQTT Agent-Demo. Abhängig von der Konfiguration, die Sie beim Herunterladen von FreeRTOS ausgewählt haben, und davon, wo Sie FreeRTOS heruntergeladen haben, werden die anderen Runner-Beispielfunktionen möglicherweise standardmäßig gestartet. Um eine Demo-Anwendung zu aktivieren, öffnen Sie die *`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/aws_demo_config.h` Datei und definieren Sie die Demo, die Sie ausführen möchten.

> **Note**
>
> Nicht alle Kombinationen von Beispielen funktionieren zusammen. Je nach Kombination kann die Software aufgrund von Speicherbeschränkungen möglicherweise nicht auf dem ausgewählten Ziel ausgeführt werden. Wir empfehlen Ihnen, immer nur jeweils eine Demo auszuführen.

Konfigurieren der -Demos

Die Demos wurden so konfiguriert, dass Sie schnell loslegen können. Möglicherweise möchten Sie einige Konfigurationen für Ihr Projekt ändern, um eine Version zu erstellen, die auf Ihrer Plattform ausgeführt wird. Die Konfigurationsdateien finden Sie unter `vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files`.

Bluetooth Low Energy-Demoanwendungen

> **Important**
>
> Diese Demo wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

-Übersicht

FreeRTOS Bluetooth Low Energy umfasst drei Demo-Anwendungen:

- MQTT über Bluetooth Low Energy-Demo

  Diese Anwendung zeigt, wie Sie den MQTT über den Bluetooth Low Energy-Service nutzen können.
- WLAN-Bereitstellung-Demo

  Diese Anwendung zeigt, wie Sie den Bluetooth Low Energy Wi-Fi Provisioning Service nutzen können.
- Generic Attributes Server-Demo

  Diese Anwendung demonstriert, wie die FreeRTOS Bluetooth Low Energy-Middleware verwendet wird, um einen einfachen APIs GATT-Server zu erstellen.

> **Note**
>
> Folgen Sie den Schritten unter, um die FreeRTOS-Demos einzurichten und auszuführen. Erste Schritte mit FreeRTOS

Voraussetzungen

Um diese Demos auszuführen, benötigen Sie einen Mikrocontroller mit Bluetooth Low Energy-Funktion. Sie brauchen außerdem iOS-SDK für FreeRTOS-Bluetooth-Geräte oder Android-SDK für FreeRTOS-Bluetooth-Geräte.

Einrichtung AWS IoT und Amazon Cognito für FreeRTOS Bluetooth Low Energy

Um Ihre Geräte mit AWS IoT Across MQTT zu verbinden, müssen Sie Amazon Cognito einrichten AWS IoT .

Zum Einrichten AWS IoT

1. Richten Sie ein AWS Konto auf https://aws.amazon.com/ein.
2. Öffnen Sie die AWS IoT -Konsole und wählen Sie im Navigationsbereich Manage (Verwalten) und dann Things (Objekte).

3. Wählen Sie Create (Erstellen) und dann Create a single thing (Einzelnes Objekt erstellen).
4. Geben Sie einen Namen für Ihr Gerät ein und wählen Sie dann Next (Weiter).
5. Wenn Sie Ihren Mikrocontroller über ein mobiles Gerät mit der Cloud verbinden, wählen Sie Create thing without certificate (Erstellen Sie das Objekt ohne Zertifikat). Da Mobile Amazon Cognito für die Geräteauthentifizierung SDKs verwendet, müssen Sie kein Gerätezertifikat für Demos erstellen, die Bluetooth Low Energy verwenden.

   Wenn Sie Ihren Mikrocontroller direkt über WLAN mit der Cloud verbinden, wählen Sie Create certificate (Zertifikat erstellen), wählen Sie Activate (Aktivieren) und laden Sie dann das Zertifikat, den öffentlichen Schlüssel und den privaten Schlüssel des Objekts herunter.
6. Wählen Sie das Objekt, das Sie gerade erstellt haben, aus der Liste der registrierten Objekte aus und wählen Sie dann Interact (Interagieren) auf der Seite Ihres Objekts. Notieren Sie sich den AWS IoT REST-API-Endpunkt.

Weitere Informationen zur Einrichtung finden Sie unter Erste Schritte mit AWS IoT.

So erstellen Sie einen Amazon Cognito Cognito-Benutzerpool

1. Öffnen Sie die Amazon Cognito Cognito-Konsole und wählen Sie Benutzerpools verwalten.
2. Wählen Sie Create a user pool.
3. Geben Sie dem Benutzerpool einen Namen und wählen Sie dann Review defaults (Standardwerte prüfen).
4. Wählen Sie im Navigationsbereich App clients (App-Clients) und wählen Sie dann Add an app client (App-Client hinzufügen).
5. Geben Sie einen Namen für den App-Client ein und wählen Sie dann Create app client (App-Client erstellen).
6. Wählen Sie im Navigationsbereich Review (Prüfen) und dann Create pool (Pool erstellen).

   Notieren Sie sich die Pool-ID, die auf der Seite General Settings (Allgemeine Einstellungen) Ihres eigenen Benutzerpools angezeigt wird.
7. Wählen Sie im Navigationsbereich App clients (App-Clients) und wählen Sie dann Show details (Details anzeigen). Notieren Sie sich die App-Client-ID und den App-Clientschlüssel.

So erstellen Sie einen Amazon Cognito Cognito-Identitätspool

1. Öffnen Sie die Amazon Cognito Cognito-Konsole und wählen Sie Manage Identity Pools aus.

2. Geben Sie einen Namen für den Identitäten-Pool ein.
3. Erweitern Sie Authentication providers (Authentifizierungsanbieter), wählen Sie die Registerkarte Cognito und geben Sie dann Ihre Benutzerpool-ID und App-Client-ID ein.
4. Wählen Sie Pool erstellen.
5. Erweitern Sie View Details (Details anzeigen) und notieren Sie sich die beiden IAM-Rollennamen. Wählen Sie Allow, um die IAM-Rollen für authentifizierte und nicht authentifizierte Identitäten für den Zugriff auf Amazon Cognito zu erstellen.
6. Wählen Sie Edit identity pool (Identitäten-Pool bearbeiten). Notieren Sie sich die Identitätspool-ID. Sie sollte ein Format wie `us-west-2:12345678-1234-1234-1234-123456789012` haben.

Weitere Informationen zur Einrichtung von Amazon Cognito finden Sie unter Erste Schritte mit Amazon Cognito.

Um eine IAM-Richtlinie zu erstellen und an die authentifizierte Identität anzuhängen

1. Öffnen Sie die IAM-Konsole und wählen Sie im Navigationsbereich Rollen aus.
2. Finden Sie die Rolle Ihrer authentifizierten Identität und wählen Sie sie aus, wählen Sie Attach policies (Richtlinien anfügen) und wählen Sie dann Add inline policy (Eingebundene Richtlinie hinzufügen).
3. Wählen Sie die Registerkarte JSON und fügen Sie die folgende JSON ein:

   JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "iot:AttachPolicy",
            "iot:AttachPrincipalPolicy",
            "iot:Connect",
            "iot:Publish",
            "iot:Subscribe",
            "iot:Receive",
            "iot:GetThingShadow",
            "iot:UpdateThingShadow",
```

```
            "iot:DeleteThingShadow"
         ],
         "Resource":[
            "*"
         ]
      }
   ]
}
```

4. Wählen Sie Review policy (Richtlinie prüfen), geben Sie einen Namen für die Richtlinie ein und wählen Sie dann Create policy (Richtlinie erstellen).

Halten Sie Ihre Daten AWS IoT und Amazon Cognito stets griffbereit. Sie benötigen den Endpunkt und IDs müssen Ihre mobile Anwendung mit der AWS Cloud authentifizieren.

Richten Sie Ihre FreeRTOS-Umgebung für Bluetooth Low Energy ein

Um Ihre Umgebung einzurichten, müssen Sie FreeRTOS mit dem Bluetooth Low Energy-Bibliothek auf Ihrem Mikrocontroller herunterladen und das Mobile SDK für FreeRTOS Bluetooth-Geräte auf Ihr Mobilgerät herunterladen und konfigurieren.

Um die Umgebung Ihres Mikrocontrollers mit FreeRTOS Bluetooth Low Energy einzurichten

1. Laden Sie FreeRTOS von herunter oder klonen Sie es. GitHub Anweisungen finden Sie in der Datei README.md.
2. Richten Sie FreeRTOS auf Ihrem Mikrocontroller ein.

   Informationen zu den ersten Schritten mit FreeRTOS auf einem FreeRTOS-qualifizierten Mikrocontroller finden Sie in der Anleitung für Ihr Board unter Getting Started with FreeRTOS.

   > **Note**
   >
   > Sie können die Demos auf jedem Bluetooth Low Energy-fähigen Mikrocontroller mit FreeRTOS und portierten FreeRTOS Bluetooth Low Energy-Bibliotheken ausführen. Derzeit ist das MQTT über Bluetooth Low Energy FreeRTOS-Demoprojekt vollständig auf die folgenden Bluetooth Low Energy-fähigen Geräte portiert:
   >
   > - Espressif - C ESP32 und der DevKit ESP-WROVER-KIT
   > - Nordisch n -DK RF52840

Gemeinsame Komponenten

Die FreeRTOS-Demoanwendungen haben zwei gemeinsame Komponenten:

- Network Manager
- Bluetooth Low Energy Mobile SDK Demo-Anwendung für mobile Endgeräte

Network Manager

Der Network Manager verwaltet die Netzwerkverbindung Ihres Mikrocontrollers. Es befindet sich in Ihrem FreeRTOS-Verzeichnis unter. `demos/network_manager/aws_iot_network_manager.c` Wenn der Netzwerkmanager sowohl für Wi-Fi als auch für Bluetooth Low Energy aktiviert ist, beginnen die Demos standardmäßig mit Bluetooth Low Energy. Wenn die Bluetooth Low Energy-Verbindung unterbrochen ist, Ihr Motherboard aber, wechselt der Network Manager zu einer verfügbaren Wi-Fi-Verbindung Wi-Fi-enabled, um zu verhindern, dass Sie die Verbindung zum Netzwerk trennen.

Um einen Netzwerkverbindungstyp mit dem Network Manager zu aktivieren, fügen Sie den Netzwerkverbindungstyp zum `configENABLED_NETWORKS` Parameter in hinzu `vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/aws_iot_network_config.h` (wobei der *`vendor`* Name des Anbieters und der Name der Karte *`board`* ist, mit der Sie die Demos ausführen).

Wenn Sie beispielsweise sowohl Bluetooth Low Energy als auch Wi-Fi aktiviert haben, lautet die Zeile, die mit `#define configENABLED_NETWORKS` in `aws_iot_network_config.h` beginnt, wie folgt:

```
#define  configENABLED_NETWORKS  ( AWSIOT_NETWORK_TYPE_BLE | AWSIOT_NETWORK_TYPE_WIFI )
```

Um eine Liste der aktuell unterstützten Netzwerkverbindungstypen zu erhalten, sehen Sie sich die Zeilen, die mit `#define AWSIOT_NETWORK_TYPE` beginnen, in `aws_iot_network.h` an.

FreeRTOS Bluetooth Low Energy Mobile SDK-Demoanwendung

Die FreeRTOS Bluetooth Low Energy Mobile SDK-Demoanwendung befindet sich GitHub unter Android SDK für FreeRTOS Bluetooth-Geräte unter `amazon-freertos-ble-android-sdk/app` und im iOS SDK für FreeRTOS Bluetooth-Geräte unter. `amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo` In diesem Beispiel verwenden wir Screenshots der iOS-Version der Demoanwendung für mobile Geräte.

> **Note**
>
> Wenn Sie ein iOS-Gerät verwenden, benötigen Sie Xcode, um die mobile Demo-Anwendung zu erstellen. Wenn Sie ein Android-Gerät verwenden, können Sie Android Studio verwenden, um die mobile Demo-Anwendung zu erstellen.

So konfigurieren Sie die iOS-SDK-Demoanwendung

Wenn Sie Konfigurationsvariablen definieren, verwenden Sie das Format der Platzhalterwerte aus den Konfigurationsdateien.

1. Vergewissern Sie sich, dass iOS-SDK für FreeRTOS-Bluetooth-Geräte installiert ist.
2. Führen Sie den folgenden Befehl über die `amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/` aus:

   ```
   $ pod install
   ```

3. Öffnen Sie das Projekt `amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/AmazonFreeRTOSDemo.xcworkspace` mit Xcode und wechseln Sie vom Konto des signierenden Entwicklers zu Ihrem Konto.
4. Erstellen Sie eine AWS IoT Richtlinie in Ihrer Region (falls Sie dies noch nicht getan haben).

   > **Note**
   >
   > Diese Richtlinie unterscheidet sich von der IAM-Richtlinie, die für die authentifizierte Amazon Cognito Cognito-Identität erstellt wurde.

   a. Öffnen Sie die AWS IoT -Konsole.

   b. Wählen Sie im Navigationsbereich erst Sicher, dann Richtlinien und anschließend Erstellen aus. Geben Sie einen Namen zur Identifizierung Ihrer Richtlinie ein. Wählen Sie im Abschnitt Anweisungen hinzufügen die Option Erweiterter Modus aus. Kopieren Sie die folgende JSON und fügen Sie sie in das Fenster des Richtlinien-Editors ein. Ersetzen Sie *`aws-region`* und *`aws-account`* durch Ihre AWS Region und Konto-ID.

   c. Wählen Sie Erstellen aus.

5. Öffnen Sie `amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/AmazonFreeRTOSDemo/Amazon/AmazonConstants.swift` und definieren Sie die folgenden Variablen neu:

   - `region`: Deine AWS Region.
   - `iotPolicyName`: Name Ihrer AWS IoT Richtlinie.
   - `mqttCustomTopic`: Das MQTT-Thema, zu dem Sie veröffentlichen möchten
6. Öffnen Sie `amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/AmazonFreeRTOSDemo/Support/awsconfiguration.json`.

   Definieren Sie die folgenden Variablen unter `CognitoIdentity` neu:

   - `PoolId`: Ihre Amazon Cognito Cognito-Identitätspool-ID.
   - `Region`: Ihre AWS Region.

   Definieren Sie die folgenden Variablen unter `CognitoUserPool` neu:

   - `PoolId`: Ihre Amazon Cognito Cognito-Benutzerpool-ID.
   - `AppClientId`: Ihre App-Client-ID
   - `AppClientSecret`: Ihren App-Clientschlüssel
   - `Region`: Ihre AWS Region.

So konfigurieren Sie die Android-SDK-Demoanwendung

Wenn Sie Konfigurationsvariablen definieren, verwenden Sie das Format der Platzhalterwerte aus den Konfigurationsdateien.

1. Vergewissern Sie sich, dass Android-SDK für FreeRTOS-Bluetooth-Geräte installiert ist.
2. Erstellen Sie eine AWS IoT Richtlinie in Ihrer Region (falls Sie dies noch nicht getan haben).

   > **Note**
   >
   > Diese Richtlinie unterscheidet sich von der IAM-Richtlinie, die für die authentifizierte Amazon Cognito Cognito-Identität erstellt wurde.

   a. Öffnen Sie die AWS IoT -Konsole.

b. Wählen Sie im Navigationsbereich erst Sicher, dann Richtlinien und anschließend Erstellen aus. Geben Sie einen Namen zur Identifizierung Ihrer Richtlinie ein. Wählen Sie im Abschnitt Anweisungen hinzufügen die Option Erweiterter Modus aus. Kopieren Sie die folgende JSON und fügen Sie sie in das Fenster des Richtlinien-Editors ein. Ersetzen Sie *aws-region* und *aws-account* durch Ihre AWS Region und Konto-ID.

c. Wählen Sie Erstellen aus.

3. Öffnen Sie https://github.com/aws/amazon-freertos-ble-androidsdk/blob/master/app/src/main/java/software/amazon/freertos/demo/DemoConstants-.java und definieren Sie die folgenden Variablen neu:

   - `AWS_IOT_POLICY_NAME`: Ihr AWS IoT Richtlinienname.
   - `AWS_IOT_REGION`: Ihre AWS Region.

4. Öffnen Sie https://github.com/aws/amazon-freertos-ble-android- sdk/blob/master/app/src/main/res/raw/awsconfiguration .json.

   Definieren Sie die folgenden Variablen unter `CognitoIdentity` neu:

   - `PoolId`: Ihre Amazon Cognito Cognito-Identitätspool-ID.
   - `Region`: Ihre AWS Region.

   Definieren Sie die folgenden Variablen unter `CognitoUserPool` neu:

   - `PoolId`: Ihre Amazon Cognito Cognito-Benutzerpool-ID.
   - `AppClientId`: Ihre App-Client-ID
   - `AppClientSecret`: Ihren App-Clientschlüssel
   - `Region`: Ihre AWS Region.

So erkennen und stellen Sie sichere Verbindungen mit Ihrem Mikrocontroller über Bluetooth Low Energy her

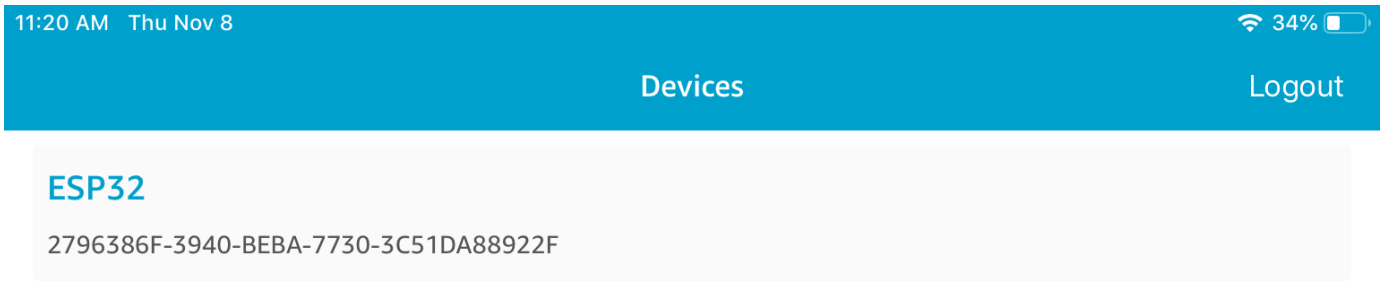
1. Um Ihren Mikrocontroller und Ihr Mobilgerät sicher zu koppeln (Schritt 6), benötigen Sie einen seriellen Terminal-Emulator mit Eingabe- und Ausgabefunktionen (z. B. TeraTerm). Konfigurieren Sie das Terminal für eine serielle Verbindung zu Ihrem Board gemäß den Anweisungen in Installieren eines Terminal-Emulators.
2. Starten Sie das Bluetooth Low Energy Demo-Projekt auf Ihrem Mikrocontroller.

3. Starten Sie die Bluetooth Low Energy Mobile SDK-Demo-Anwendung auf Ihrem mobilen Gerät.

   Um die Demoanwendung im Android-SDK von der Befehlszeile aus zu starten, führen Sie den folgenden Befehl aus:

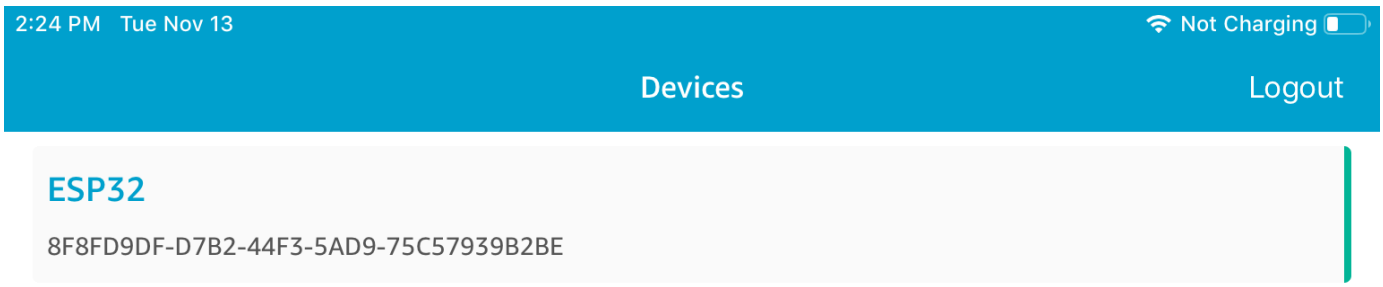
   ```
   $ ./gradlew installDebug
   ```

4. Vergewissern Sie sich, dass Ihr Mikrocontroller unter Devides (Geräte) in der Bluetooth Low Energy Mobile SDK Demo-Anwendung erscheint.

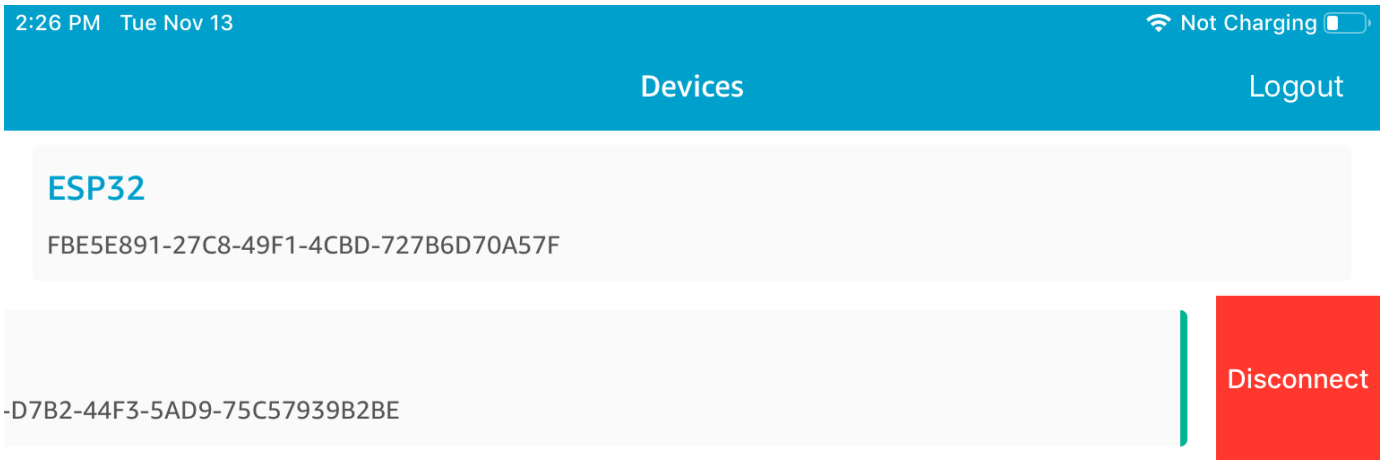


   > **Note**
   >
   > Alle Geräte mit FreeRTOS und dem Geräteinformationsdienst (*freertos*`/.../device_information`), die sich in Reichweite befinden, werden in der Liste angezeigt.

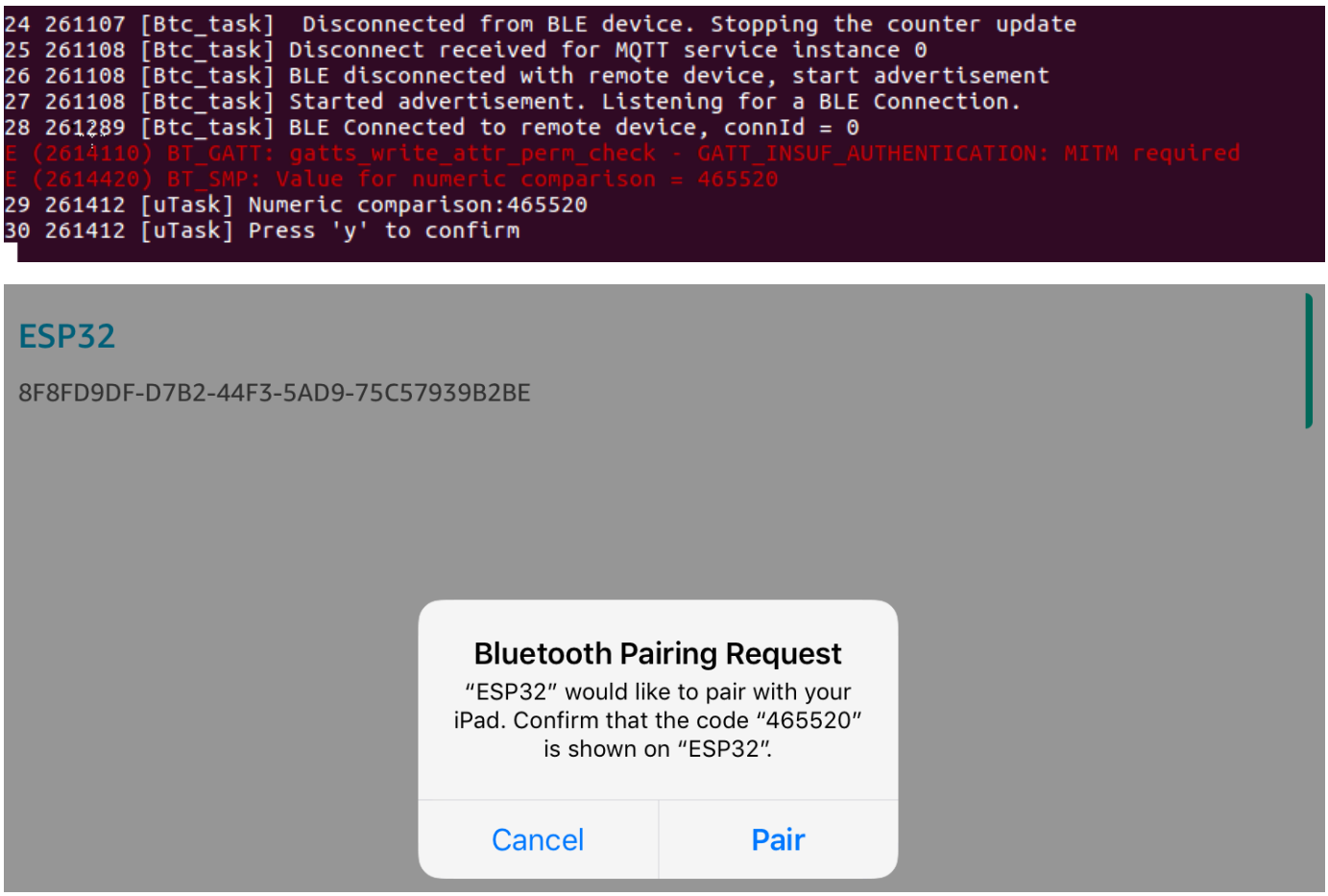
5. Wählen Sie Ihren Mikrocontroller aus der Liste der Geräte aus. Die Anwendung stellt eine Verbindung mit dem Board her und eine grüne Linie wird neben dem verbundenen Gerät angezeigt.



Sie können die Verbindung zu Ihrem Mikrocontroller trennen, indem Sie die Linie nach links ziehen.



6. Wenn Sie dazu aufgefordert werden, koppeln Sie Ihren Mikrocontroller und Ihr mobiles Gerät.



Wenn der Code für den numerischen Abgleich auf beiden Geräten übereinstimmt, verbinden Sie die Geräte.

> **Note**
>
> Die Bluetooth Low Energy Mobile SDK-Demoanwendung verwendet Amazon Cognito für die Benutzerauthentifizierung. Stellen Sie sicher, dass Sie einen Amazon Cognito Cognito-Benutzer und Identitätspools eingerichtet haben und dass Sie authentifizierten Identitäten IAM-Richtlinien angehängt haben.

MQTT über Bluetooth Low Energy

In der Demo MQTT over Bluetooth Low Energy veröffentlicht Ihr Mikrocontroller Nachrichten über einen MQTT-Proxy in der AWS Cloud.

So abonnieren Sie ein Demo-MQTT-Thema

1. Melden Sie sich bei der Konsole an. AWS IoT
2. Wählen Sie im Navigationsbereich Test und dann MQTT-Testclient aus, um den MQTT-Client zu öffnen.
3. Geben Sie im Feld Subscription topic (Abonnementthema) die Option ***thing-name*/example/topic1** ein und wählen Sie dann Subscribe to topic (Thema abonnieren).

Wenn Sie den Mikrocontroller mit Bluetooth Low Energy mit Ihrem mobilen Gerät koppeln, werden die MQTT-Nachrichten über die Bluetooth Low Energy Mobile SDK-Demoanwendung auf Ihrem mobilen Gerät weitergeleitet.

Um die Demo über Bluetooth Low Energy zu aktivieren

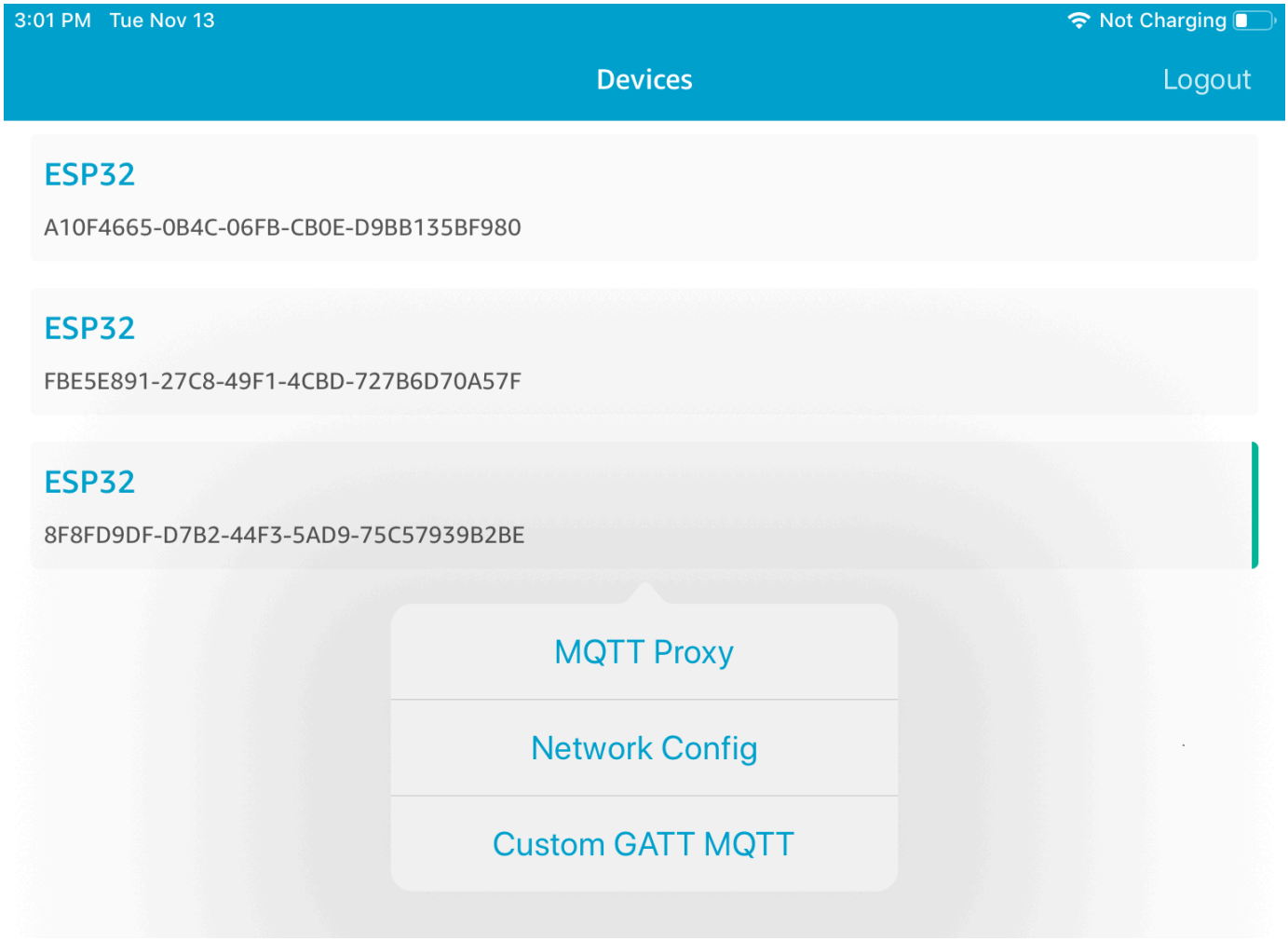
1. Öffnen Sie `vendors/vendor/boards/board/aws_demos/config_files/aws_demo_config.h`, und definieren Sie `CONFIG_MQTT_BLE_TRANSPORT_DEMO_ENABLED`.
2. Öffnen `demos/include/aws_clientcredential.h` und konfigurieren Sie `clientcredentialMQTT_BROKER_ENDPOINT` mit dem AWS IoT Broker-Endpunkt. Konfigurieren Sie `clientcredentialIOT_THING_NAME` mit dem Ding-Namen für das BLE-Mikrocontroller-Gerät. Der AWS IoT Broker-Endpunkt kann über die AWS IoT Konsole abgerufen

werden, indem Sie im linken Navigationsbereich Einstellungen auswählen, oder über die CLI, indem Sie den folgenden Befehl ausführen:`aws iot describe-endpoint --endpoint-type=iot:Data-ATS`.

> **Note**
>
> Der AWS IoT Broker-Endpunkt und der Ding-Name müssen sich beide in derselben Region befinden, in der die Cognito-Identität und der Benutzerpool konfiguriert sind.

So führen Sie die Demo aus

1. Erstellen Sie das Demoprojekt und führen Sie es auf dem Mikrocontroller aus.
2. Stellen Sie sicher, dass Sie Ihr Board und Ihr mobiles Gerät mithilfe der FreeRTOS Bluetooth Low Energy Mobile SDK-Demoanwendung verbunden haben.
3. Wählen Sie aus der Liste Devices (Geräte) in der Demoanwendung für mobile Geräte Ihren Mikrocontroller aus und wählen Sie dann MQTT Proxy (MQTT-Proxy), um die MQTT-Proxy-Einstellungen zu öffnen.



4. Nachdem Sie den MQTT-Proxy aktiviert haben, werden MQTT-Nachrichten im *thing-name*`/example/topic1`-Thema angezeigt und die Daten werden im UART-Terminal ausgegeben.

WLAN-Bereitstellung

Wi-Fi Provisioning ist ein FreeRTOS Bluetooth Low Energy-Dienst, mit dem Sie Wi-Fi-Netzwerkanmeldedaten sicher über Bluetooth Low Energy von einem Mobilgerät an einen Mikrocontroller senden können. Den Quellcode für den WLAN-Bereitstellungsservice finden Sie unter *freertos*`/.../wifi_provisioning`.

> **Note**
>
> Die Wi-Fi Provisioning-Demo wird derzeit auf dem Espressif - C unterstützt. ESP32 DevKit

So aktivieren Sie die Demo:

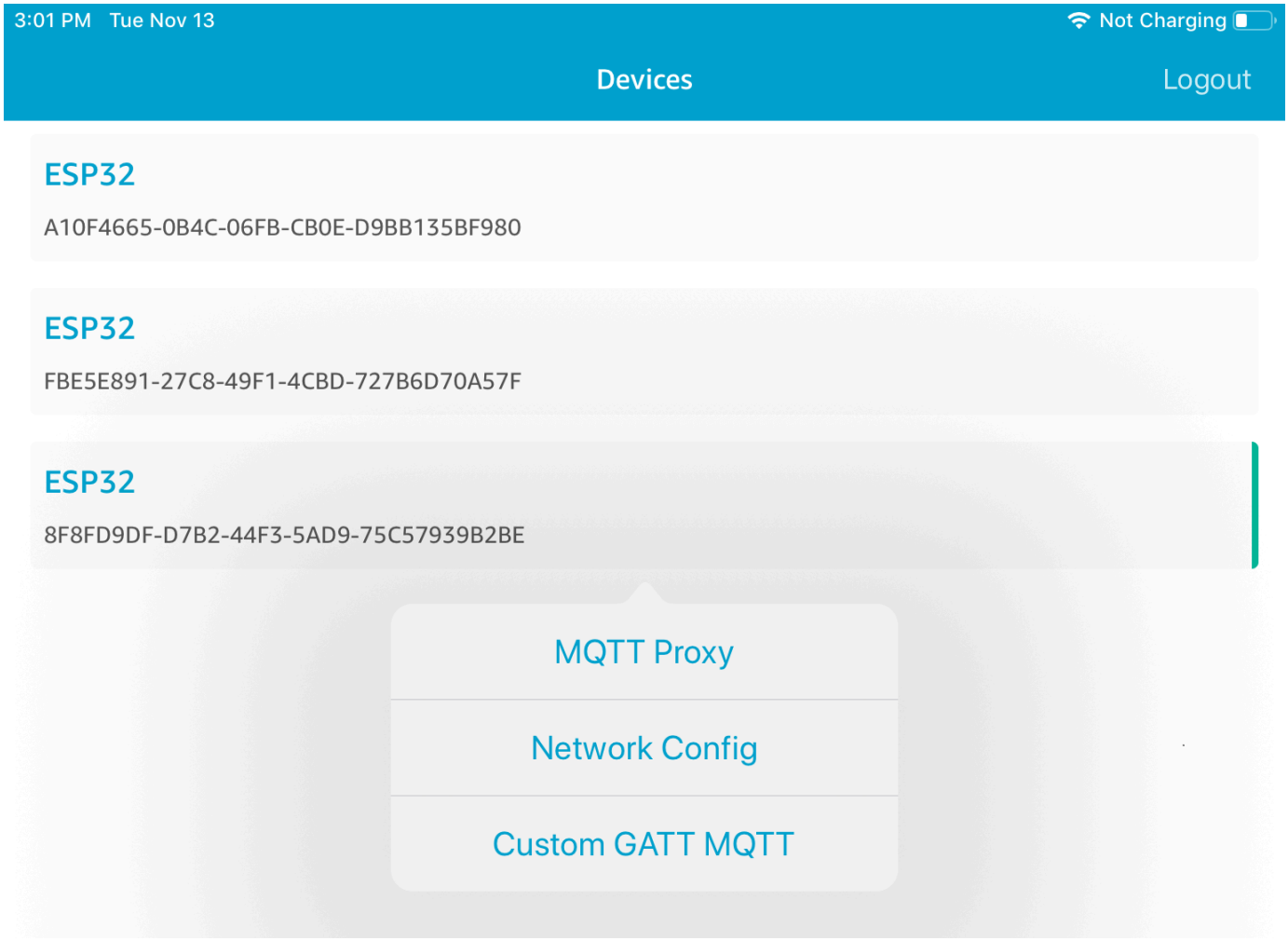
1. Aktivieren Sie den WLAN-Bereitstellungsservice. Öffnen `vendors/vendor/boards/board/aws_demos/config_files/iot_ble_config.h` Sie und stellen Sie `#define IOT_BLE_ENABLE_WIFI_PROVISIONING` auf *vendor* ein `1` (wobei das der Name des Anbieters und der Name des Boards *board* ist, auf dem Sie die Demos ausführen).

   > **Note**
   >
   > Der WLAN-Bereitstellungsservice ist standardmäßig deaktiviert.

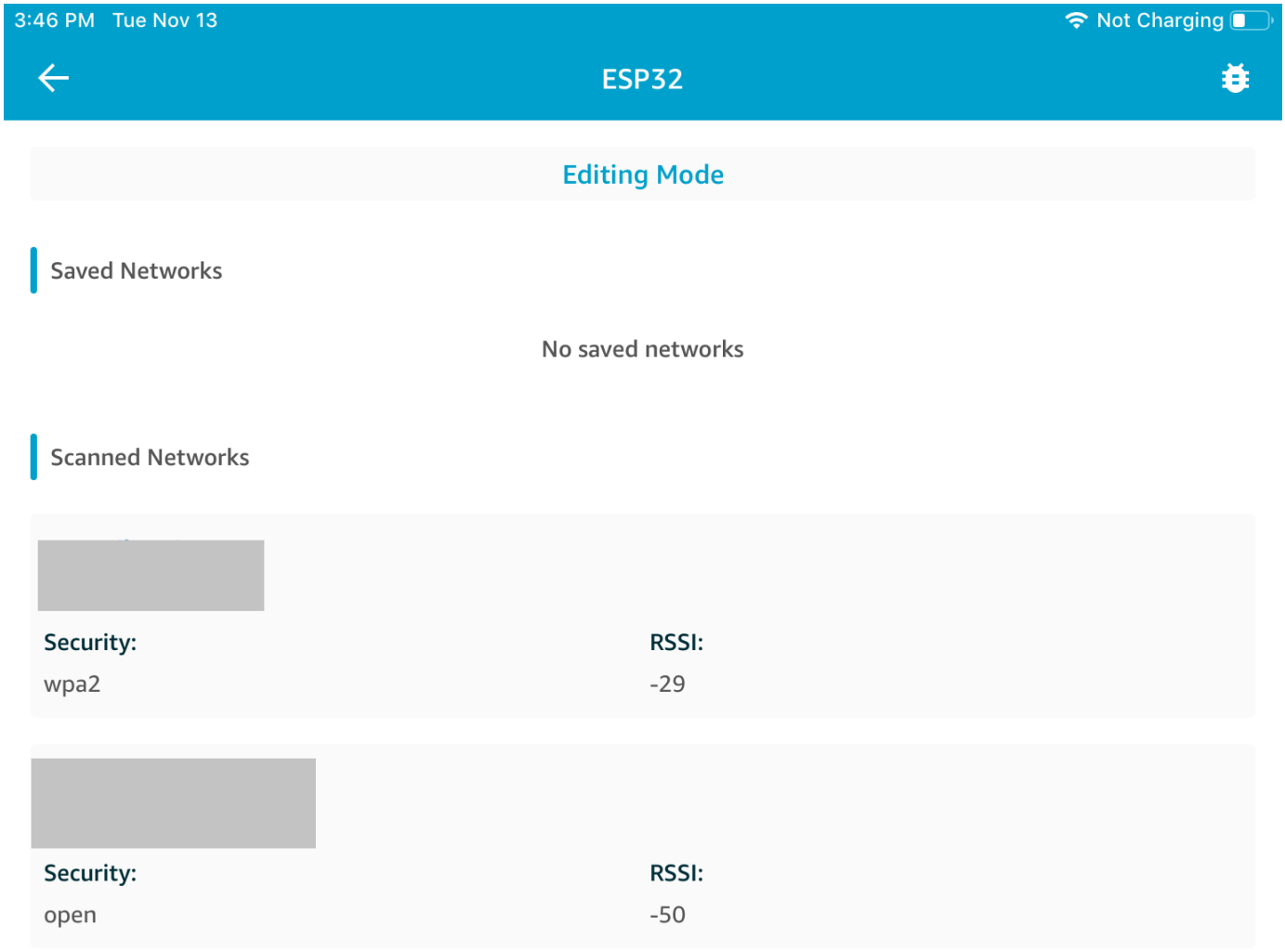
2. Konfigurieren Sie Network Manager, um sowohl Bluetooth Low Energy als auch Wi-Fi zu aktivieren.

So führen Sie die Demo aus

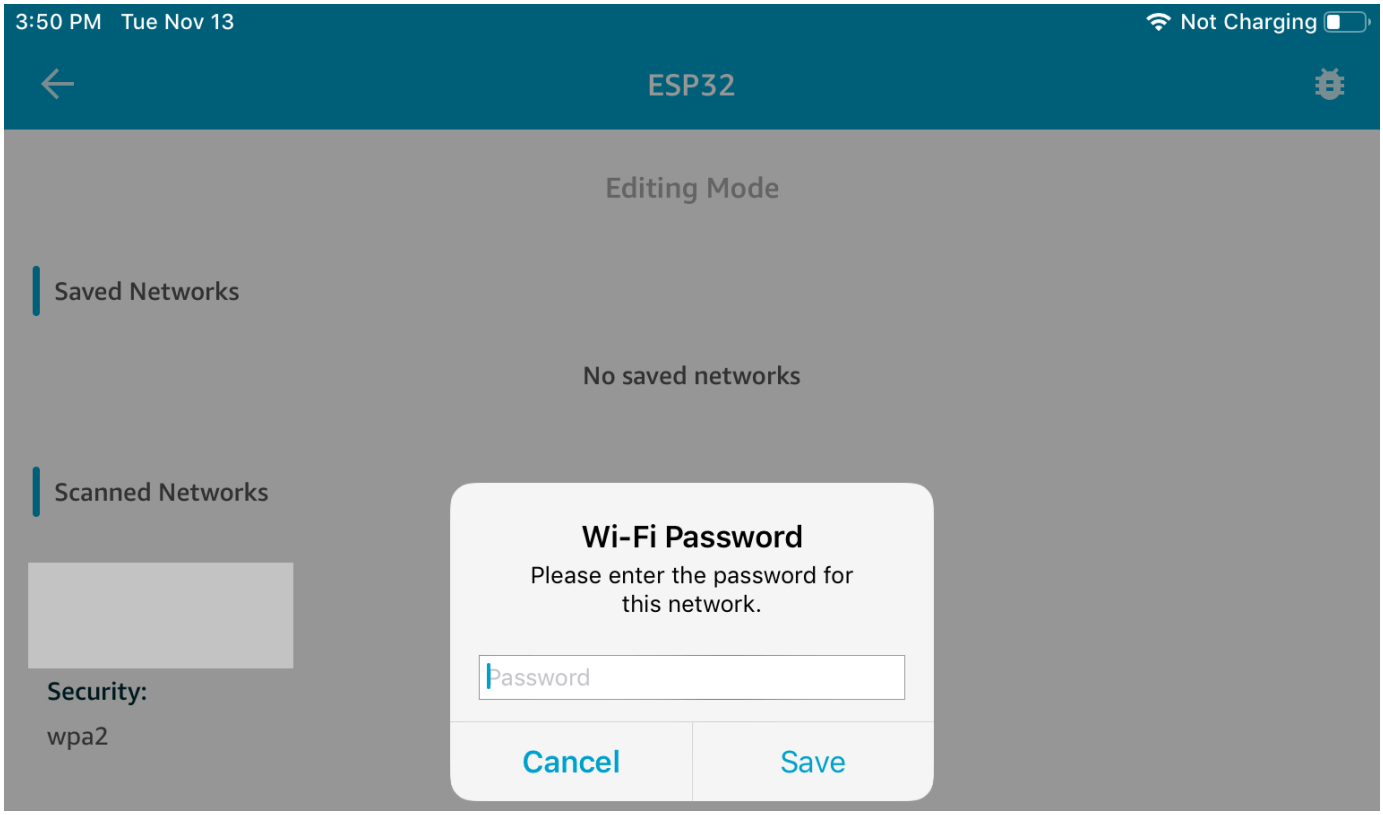
1. Erstellen Sie das Demoprojekt und führen Sie es auf dem Mikrocontroller aus.
2. Stellen Sie sicher, dass Sie Ihren Mikrocontroller und Ihr Mobilgerät mit dem gekoppelt haben. FreeRTOS Bluetooth Low Energy Mobile SDK-Demoanwendung
3. Wählen Sie erst aus der Liste Devices (Geräte) in der Demoanwendung für mobile Geräte Ihren Mikrocontroller und dann (Nework Config) Netzwerkkonfiguration aus, um die Netzwerkkonfiguration zu öffnen.



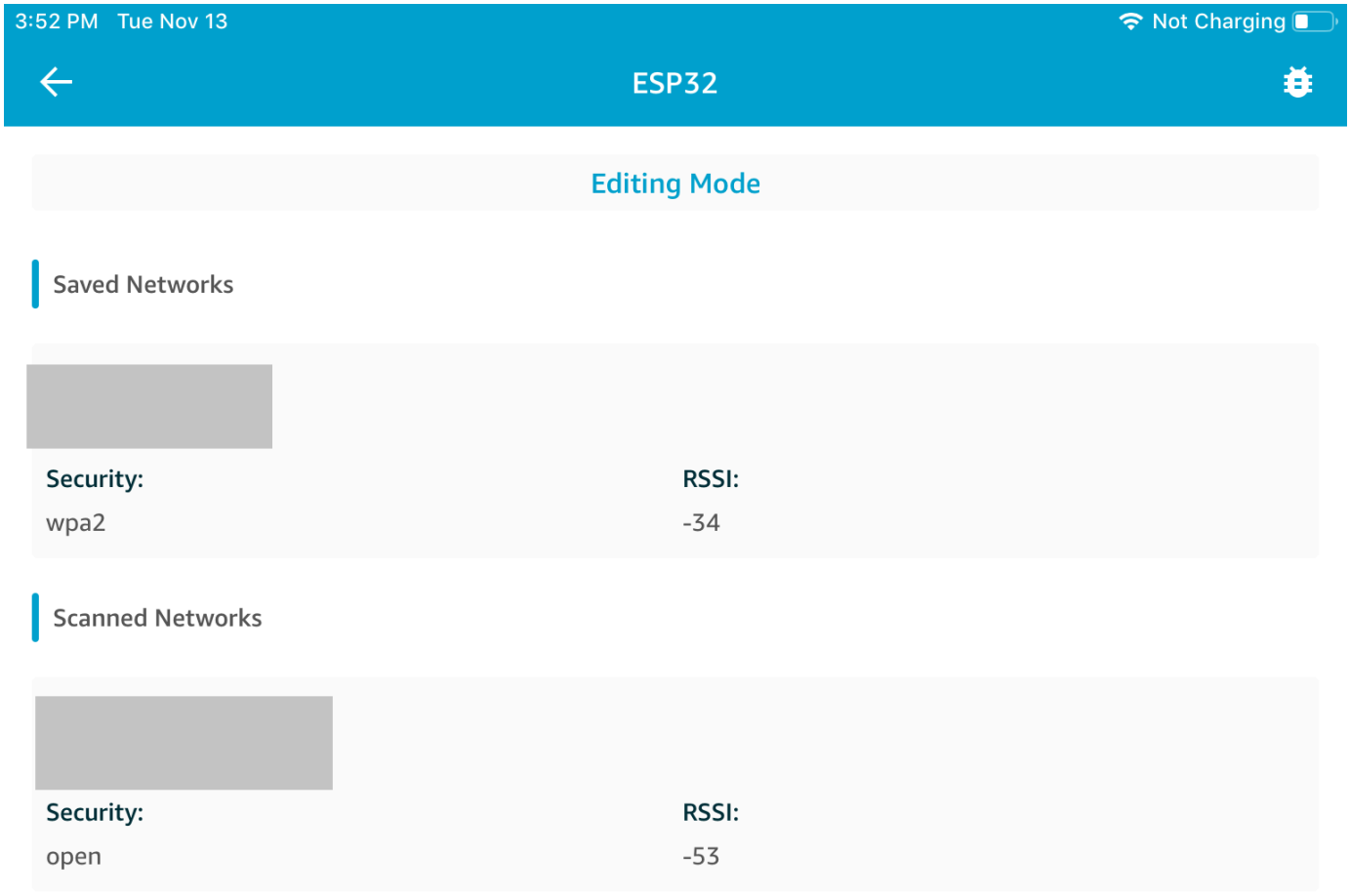
4. Nachdem Sie Netzwerkkonfiguration für Ihr Board ausgewählt haben, sendet der Mikrocontroller eine Liste der Netzwerke in der Umgebung an das mobile Gerät. Die verfügbaren WLAN-Netzwerke werden in einer Liste unter Gescannte Netzwerke angezeigt.



Wählen Sie aus der Liste Gescannte Netzwerke Ihr Netzwerk aus und geben Sie dann die SSID und Ihr Passwort ein, falls erforderlich.

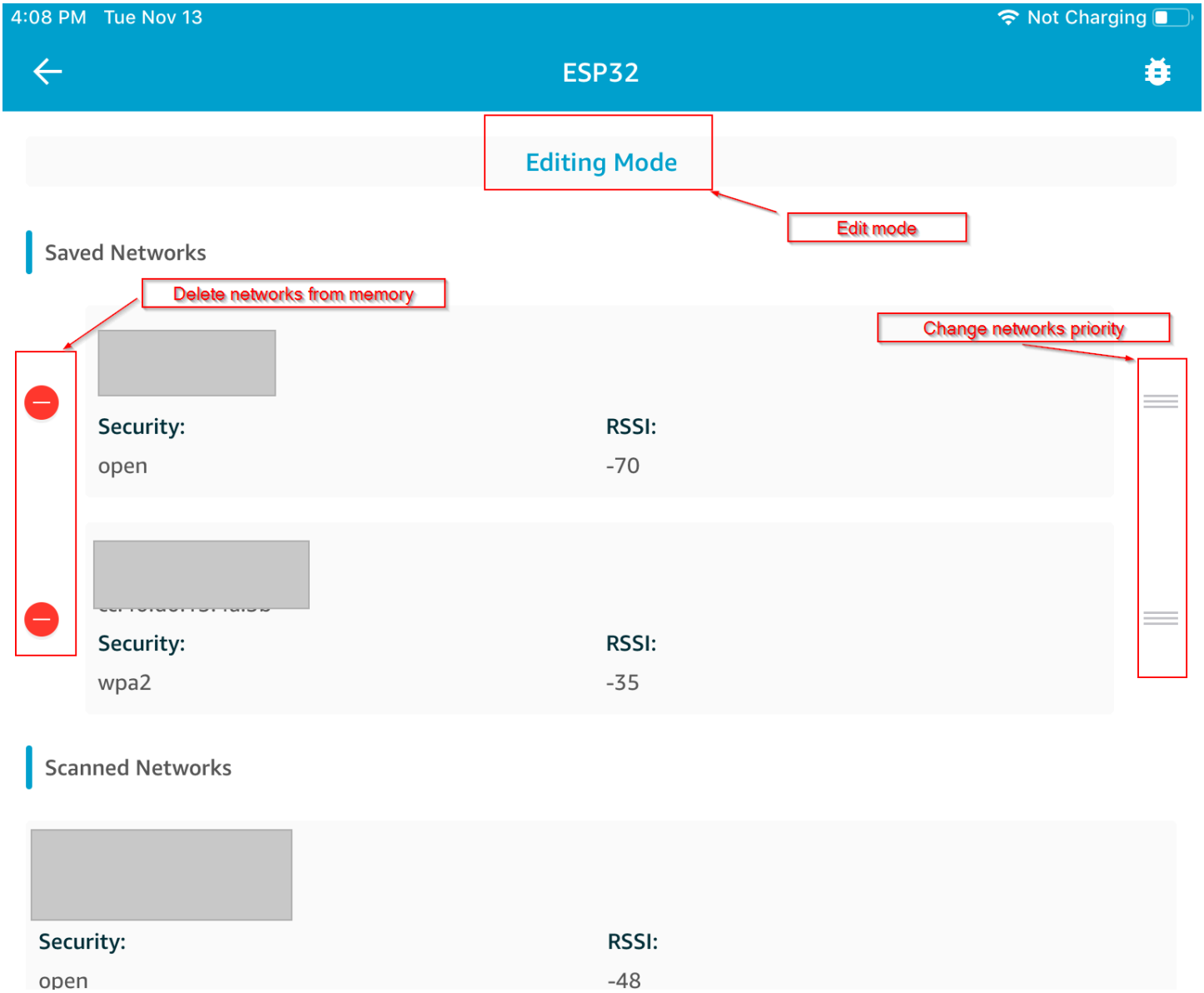


Der Mikrocontroller stellt eine Verbindung zum Netzwerk her und speichert es. Das Netzwerk wird unter Saved Networks (Gespeicherte Netzwerke) angezeigt.



Sie können mehrere Netzwerke in der Demoanwendung für mobile Geräte speichern. Wenn Sie die Anwendung und Demo neu starten, verbindet sich der Mikrocontroller mit dem ersten verfügbaren gespeicherten Netzwerk, das oben in der Liste Gespeicherte Netzwerke angezeigt wird.

Um die Netzwerk-Prioritätsreihenfolge zu ändern oder Netzwerke zu löschen, wählen Sie auf der Seite Netzwerkkonfiguration die Option Bearbeitungsmodus aus. Um die Netzwerk-Prioritätsreihenfolge zu ändern, wählen Sie die rechte Seite des Netzwerks, das Sie anders anordnen möchten, aus und ziehen Sie das Netzwerk nach oben oder unten. Um ein Netzwerk zu löschen, wählen Sie die rote Schaltfläche auf der linken Seite des Netzwerks, das Sie löschen möchten.



Generic Attributes Server

In diesem Beispiel sendet eine Generic Attribute(GATT)-Demo-Serveranwendung auf Ihrem Mikrocontroller einen einfachen Zählerwert an FreeRTOS Bluetooth Low Energy Mobile SDK-Demoanwendung.

Mit dem Bluetooth Low Energy Mobile SDKs können Sie Ihren eigenen GATT-Client für ein Mobilgerät erstellen, das eine Verbindung zum GATT-Server auf Ihrem Mikrocontroller herstellt und parallel zur mobilen Demo-Anwendung läuft.

So aktivieren Sie die Demo:

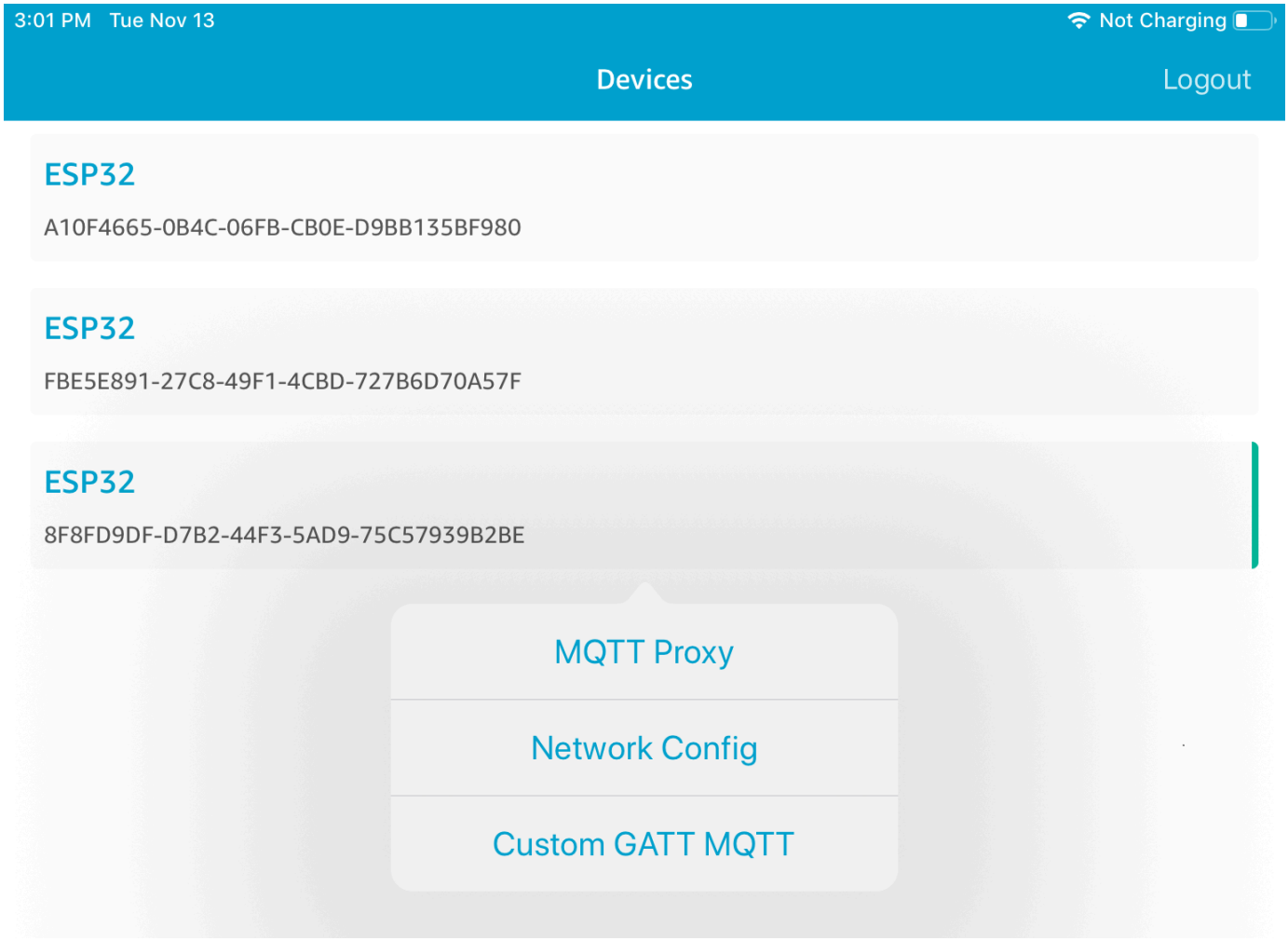
1. Aktivieren Sie die Bluetooth Low Energy GATT-Demo. Fügen Sie in `vendors/vendor/boards/board/aws_demos/config_files/iot_ble_config.h` (wobei das *vendor* der Name des Anbieters und der Name des Boards *board* ist, mit dem Sie die Demos ausführen) der Liste der `#define IOT_BLE_ADD_CUSTOM_SERVICES ( 1 )` define-Anweisungen hinzu.

   > **Note**
   >
   > Die Bluetooth Low Energy GATT-Demo ist standardmäßig deaktiviert.

2. Öffnen Sie *freertos*`/vendors/`*vendor*`/boards/`*board*`/aws_demos/config_files/aws_demo_config.h`, kommentieren Sie `#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED` aus und definieren Sie `CONFIG_BLE_GATT_SERVER_DEMO_ENABLED`.

So führen Sie die Demo aus

1. Erstellen Sie das Demoprojekt und führen Sie es auf dem Mikrocontroller aus.
2. Stellen Sie sicher, dass Sie Ihr Board und Ihr mobiles Gerät mithilfe der FreeRTOS Bluetooth Low Energy Mobile SDK-Demoanwendung verbunden haben.
3. Wählen Sie aus der Liste Devices (Geräte) in der App Ihr Board und dann MQTT Proxy (MQTT-Proxy), um die MQTT-Proxy-Einstellungen zu öffnen.



4. Kehren Sie zur Liste Geräte zurück, wählen Sie erst Ihr Board und dann Benutzerdefiniertes GATT MQTT zum Öffnen der benutzerdefinierten GATT-Serviceoptionen aus.
5. Wählen Sie Start Counter (Zähler starten) aus, um die Daten im ***your-thing-name*/example/topic**-MQTT-Thema zu veröffentlichen.

   Nachdem Sie den MQTT-Proxy aktiviert haben, werden Nachrichten von Hello World und vom inkrementellen Zähler im *your-thing-name*/example/topic-Thema angezeigt.

Demo-Bootloader für den Microchip Curiosity MZEF PIC32

> **Important**
>
> Diese Demo wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-

FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Note

In Absprache mit Microchip entfernen wir den Curiosity PIC32 MZEF (DM320104) aus dem Hauptzweig des FreeRTOS Reference Integration-Repositorys und werden ihn nicht mehr in neuen Releases anbieten. Microchip hat eine offizielle Mitteilung veröffentlicht, dass der PIC32 MZEF (DM320104) nicht mehr für neue Designs empfohlen wird. Auf die PIC32 MZEF-Projekte und den Quellcode kann weiterhin über die Tags der vorherigen Version zugegriffen werden. Microchip empfiehlt Kunden, das Curiosity PIC32MZ-EF-2.0-Entwicklungsboard (0209) für neue Designs zu verwenden. DM32 Die PIC32 MZv1 Plattform befindet sich immer noch in Version 202012.00 des FreeRTOS Reference Integration-Repositorys. Die Plattform wird jedoch von v202107.00 der FreeRTOS Reference nicht mehr unterstützt.

Dieser Demo-Bootloader implementiert die Überprüfung der Firmware-Version, die Verifizierung der kryptografischenSignatur und einen Selbsttest der Anwendung. Diese Funktionen unterstützen Firmware-Updates over-the-air (OTA) für FreeRTOS.

Die Firmware-Verifizierung beinhaltet die Verifizierung der Authentizität und die Integrität der neuen Firmware, die Over The Air empfangen wurde. Vor einem Neustart verifiziert der Bootloader die kryptografische Signatur der Anwendung. Die Demo verwendet einen Elliptic Curve Digital Signature Algorithm (ECDSA) über SHA256. Die bereitgestellten Dienstprogrammen können zur Erstellung einer signierten Anwendung verwendet werden, die auf dem Gerät geflasht werden kann.

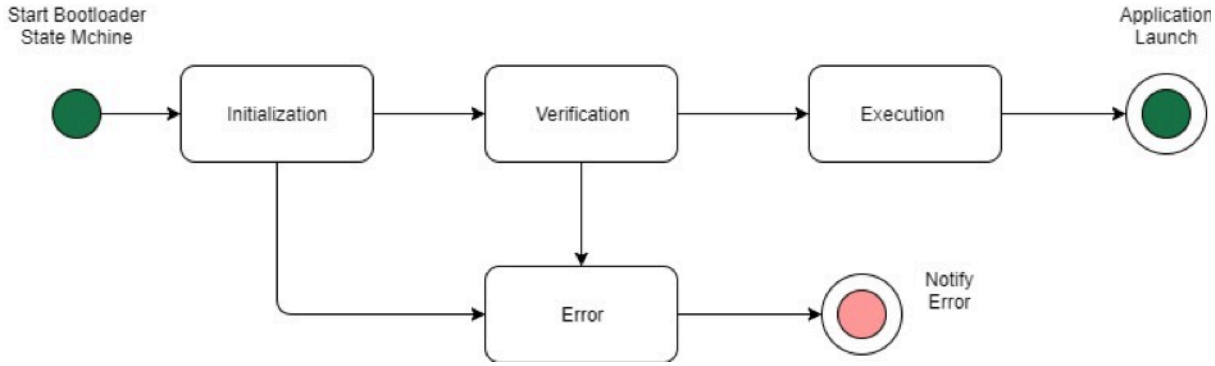
Der Bootloader unterstützt die folgenden für OTA erforderlichen Funktionen:

- Behält Anwendungsabbilder auf dem Gerät und wechselt zwischen ihnen.
- Ermöglicht die Ausführung eines Selbsttests eines empfangenen OTA-Images und ein Rollback bei Fehlern.
- Prüft die Signatur und die Version des OTA-Aktualisierungsabbilds.

> **Note**
>
> Folgen Sie den Schritten unter, um die FreeRTOS-Demos einzurichten und auszuführen. Erste Schritte mit FreeRTOS

Bootloader-Zustände

Der Bootloader-Prozess wird in der folgenden Zustands-Engine gezeigt.



Die folgende Tabelle beschreibt die Bootloader-Zustände.

Bootloader-Zustand	Beschreibung
Initialisierung	Der Bootloader befindet sich im Initialis ierungszustand.
Verifizierung	Der Bootloader verifiziert die auf dem Gerät vorhandenen Abbilder.
Abbildung ausführen	Der Bootloader startet das ausgewählte Abbild.
Standard ausführen	Der Bootloader startet das Standard-Abbild.
Fehler	Der Bootloader befindet sich im Fehlerzustand.

In der obigen Abbildung werden sowohl `Execute Image` als auch `Execute Default` als `Execution`-Zustand angezeigt.

Bootloader-Ausführungszustand

Der Bootloader befindet sich im `Execution`-Zustand und ist zum Starten des ausgewählten verifizierten Abbilds bereit. Wenn sich das zu startende Abbild in der oberen Bank befindet, werden die Banken ausgetauscht, bevor das Abbild ausgeführt wird, da die Anwendung immer für die untere Bank erstellt wird.

Bootloader-Standard-Ausführungszustand

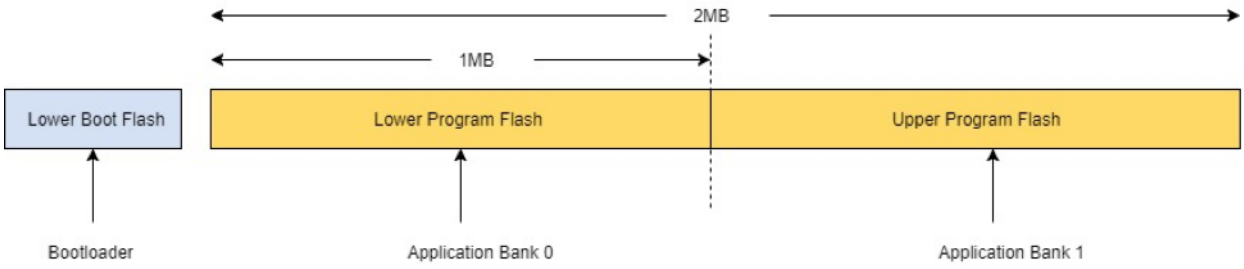
Wenn die Konfigurationsoption zum Starten des Standard-Abbilds aktiviert ist, startet der Bootloader die Anwendung von einer Standard-Ausführungsadresse. Diese Option muss außer beim Debuggen deaktiviert sein.

Bootloader-Fehlerzustand

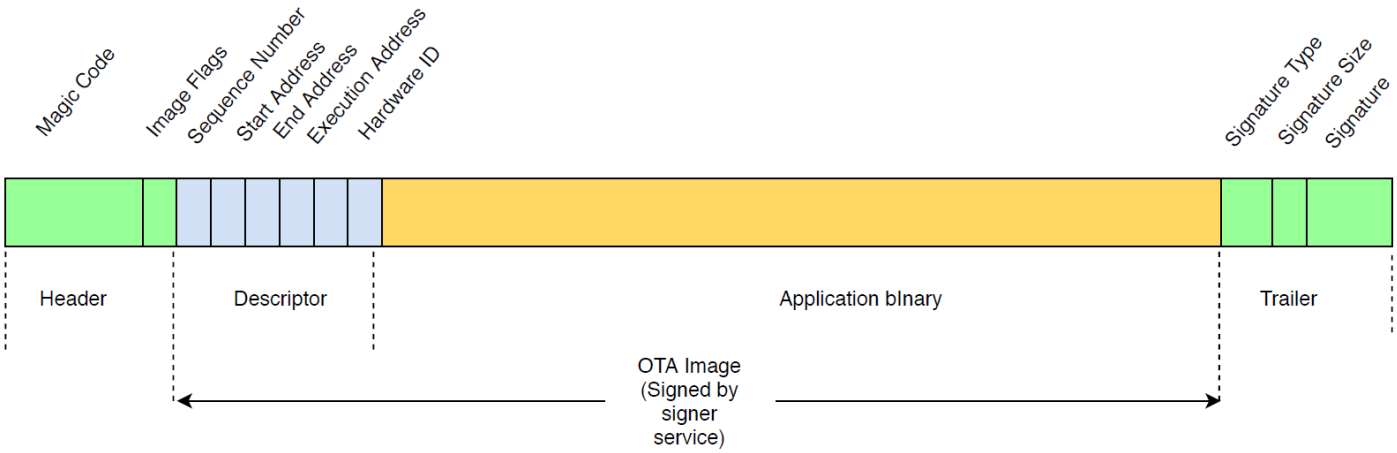
Der Bootloader befindet sich in einem Fehlerzustand und es sind keine gültigen Abbilder auf dem Gerät vorhanden. Der Bootloader muss den Benutzer benachrichtigen. Die Standardimplementierung sendet eine Protokollnachricht an die Konsole und löst für eine unbestimmte Zeit ein schnelles Blinken der LED auf der Tafel aus.

Flash-Gerät

Die Microchip Curiosity PIC32 MZEF-Plattform enthält einen internen Programmspeicher mit einer Größe von zwei Megabyte (MB), der in zwei Bänke aufgeteilt ist. Sie unterstützt den Austausch von Memory Maps zwischen diesen beiden Banken und Live-Updates. Der Demo-Bootloader wird in einer separaten niedrigeren Boot-Flash-Region programmiert.



Struktur des Anwendungsabbilds



Das Diagramm zeigt die Hauptkomponenten des Anwendungsabbilds, die in jeder Bank des Geräts gespeichert sind.

Komponente	Größe (in Bytes)
Abbild-Header	8 Bytes
Abbild-Deskriptor	24 Bytes
Binärcode der Anwendung	< 1 MB - (324)
Trailer	292 Bytes

Abbild-Header

Die Anwendungsabbilder auf dem Gerät müssen mit einem Header beginnen, der aus einem magischen Code und Abbild-Flags besteht.

Header-Feld	Größe (in Bytes)
Magischer Code	7 Bytes
Abbild-Flags	1 Byte

Magischer Code

Das Abbild auf dem Flash-Gerät muss mit einem magischen Code beginnen. Der standardmäßige magische Code lautet `@AFRTOS`. Der Bootloader prüft, ob ein gültiger magischer Code vorhanden ist, bevor das Abbild gestartet wird. Dies ist der erste Schritt der Verifizierung.

Abbild-Flags

Die Abbild-Flags werden verwendet, um den Status der Abbilder der Anwendung zu speichern. Die Flags werden im OTA-Prozess verwendet. Die Abbild-Flags beider Banken bestimmen den Zustand des Geräts. Wenn das ausführende Abbild als schwebender Commit markiert ist, bedeutet dies, dass sich das Gerät in der OTA-Selbsttest-Phase befindet. Selbst wenn Abbilder auf den Geräten als gültig markiert sind, durchlaufen sie bei jedem Start dieselben Verifizierungsschritte. Wenn ein Abbild als neu markiert ist, wird es vom Bootloader als schwebender Commit markiert und nach der Überprüfung zum Selbsttest gestartet. Der Bootloader initialisiert und startet außerdem den Watchdog-Timer, sodass das Gerät einen Neustart durchführt, wenn das neue OTA-Abbild den Selbsttest nicht besteht. Der Bootloader weist dann die Abbildung durch Löschen dieser zurück und führt das vorherige gültige Abbild aus.

Das Gerät kann nur ein gültiges Abbild haben. Das andere Abbild kann ein neues OTA-Abbild oder ein schwebender Commit (Selbsttest) sein. Nach einem erfolgreichen OTA-Update wird das alte Abbild vom Gerät gelöscht.

Status	Wert	Beschreibung
Neues Abbild	0xFF	Das Abbild der Anwendung ist neu und wurde noch nie ausgeführt.
Schwebender Commit	0xFE	Das Abbild der Anwendung ist für die Testausführung gekennzeichnet.
Valid	0xFC	Das Abbild der Anwendung ist als gültig und übernommen markiert.
Ungültig	0xF8	Das Abbild der Anwendung ist als ungültig markiert.

Abbild-Deskriptor

Das Abbild der Anwendung auf dem Flash-Gerät muss den Abbild-Deskriptor nach dem Abbild-Header enthalten. Der Abbild-Deskriptor wird von einem Post-Build-Dienstprogramm erstellt, das die Konfigurationsdateien (`ota-descriptor.config`) verwendet, um den entsprechenden Deskriptor zu generieren, und stellt diesen dem Binärcode der Anwendung voran. Die Ausgabe dieses Post-Build-Schrittes ist das Binary-Abbild, das für OTA verwendet werden kann.

Feld-Deskriptor	Größe (in Bytes)
Sequenznummer	4 Bytes
Startadresse	4 Bytes
Endadresse	4 Bytes
Ausführungsadresse	4 Bytes
Hardware-ID	4 Bytes
Reserved Instances	4 Bytes

Sequenznummer

Die Sequenznummer muss erhöht werden, bevor ein neues OTA-Abbild erstellt wird. Siehe Datei `ota-descriptor.config`. Der Bootloader ermittelt anhand dieser Nummer das zu startende Abbild. Gültige Werte liegen zwischen 1 und 4294967295.

Startadresse

Die Startadresse des Anwendungsabbilds auf dem Gerät. Da der Abbild-Deskriptor der Binärdatei der Anwendung vorangestellt ist, ist diese Adresse der Start des Abbild-Deskriptors.

Endadresse

Die Endadresse des Anwendungsabbilds auf dem Gerät, ohne Abbild-Trailer.

Ausführungsadresse

Die Ausführungsadresse des Abbilds.

Hardware-ID

Eine vom Bootloader verwendete eindeutige Hardware-ID zur Überprüfung, ob das OTA-Abbild für die korrekte Plattform erstellt wurde.

Reserved Instances

Dies ist für die zukünftige Verwendung reserviert.

Abbild-Trailer

Der Abbild-Trailer wird an den Binärcode der Anwendung angehängt. Er enthält die Signaturtyp-Zeichenfolge, die Signaturgröße und die Signatur des Abbilds.

Trailer-Feld	Größe (in Bytes)
Signaturtyp	32 Bytes
Signaturgröße	4 Bytes
Signature	256 Byte

Signaturtyp

Der Signaturtyp ist eine Zeichenfolge, die den verwendeten kryptografischen Algorithmus darstellt und als Markierung für den Trailer dient. Der Bootloader unterstützt den ECDSA-Signaturalgorithmus (Elliptic Curve Digital Signature Algorithm). Der Standardwert ist sig-sha256-ecdsa.

Signaturgröße

Die Größe der kryptografischen Signatur in Bytes.

Signatur

Die kryptografische Signatur des Binärcodes der Anwendung mit dem Abbild-Deskriptor.

Bootloader-Konfiguration

Die grundlegenden Bootloader-Konfigurationsoptionen finden Sie unter *freertos*`/vendors/microchip/boards/curiosity_pic32mzef/bootloader/config_files/aws_boot_config.h`. Einige Optionen werden nur für Debuggingzwecke angeboten.

Aktivieren des Standardstarts

Aktiviert die Ausführung der Anwendung von der Standardadresse und darf nur für Debugging-Zwecke aktiviert werden. Das Abbild wird ohne Verifizierung von der Standardadresse aus ausgeführt.

Aktivieren der kryptischen Signaturverifizierung

Aktiviert die kryptografische Signaturverifizierung beim Booten. Die fehlgeschlagene Abbilder werden vom Gerät gelöscht. Diese Option steht nur für Debugging-Zwecke zur Verfügung und muss in der Produktivumgebung aktiviert sein.

Löschen eines ungültigen Abbilds

Ermöglicht das vollständige Löschen der Bank, falls die Abbild-Verifizierung für diese Bank fehlschlägt. Diese Option steht nur für Debugging-Zwecke zur Verfügung und muss in der Produktivumgebung aktiviert sein.

Aktivieren der Hardware-ID-Verifizierung

Ermöglicht die Verifizierung der Hardware-ID im Deskriptor des OTA-Abbilds und der Hardware-ID, die im Bootloader programmiert ist. Dies ist optional und kann deaktiviert werden, wenn die Verifizierung der Hardware-ID nicht erforderlich ist.

Aktivieren der Adressenverifikation

Ermöglicht die Überprüfung der Start-, End- und Ausführungsadresse im Deskriptor des OTA-Abbilds. Wir empfehlen, dass Sie diese Option aktiviert lassen.

Entwickeln des Bootloaders

Der Demo-Bootloader ist als ladbares Projekt in dem `aws_demos` Projekt enthalten, das sich im FreeRTOS-Quellcode-Repository befindet. *`freertos`*`/vendors/microchip/boards/curiosity_pic32mzef/aws_demos/mplab/` Wenn das `aws_demos`-Projekt erstellt wurde, wird zuerst der Bootloader, gefolgt von der Anwendung, erstellt. Die endgültige Ausgabe ist ein einheitliches Hex-Abbild für den Bootloader und die Anwendung. Das `factory_image_generator.py`-Dienstprogramm wird zum Generieren eines einheitlichen Hex-Abbilds mit kryptografischer Signatur bereitgestellt. Die Skripts des Bootloader-Dienstprogramms befinden sich in *`freertos`*`/demos/ota/bootloader/utility/`.

Schritt vor dem Bootloader-Build

Dieser Schritt vor dem Build führt ein Dienstprogramm-Skript mit dem Namen `codesigner_cert_utility.py` aus, das den öffentlichen Schlüssel aus dem Codesignierungszertifikat extrahiert und eine C-Header-Datei generiert, die den öffentlichen Schlüssel im Abstract Syntax Notation One (ASN.1)-codierten Format enthält. Dieser Header ist in das Bootloader-Projekt kompiliert. Der generierte Header enthält zwei Konstanten: ein Array des öffentlichen Schlüssels und die Länge des Schlüssels. Das Bootloader-Projekt kann auch ohne `aws_demos` erstellt und als normale Anwendung debuggt werden.

AWS IoT Device Defender Demo

> **Important**
>
> Diese Demo wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Einführung

Diese Demo zeigt Ihnen, wie Sie die Device Defender-Bibliothek verwenden, um eine Verbindung herzustellen. AWS IoT AWS IoT Device Defender Die Demo verwendet die CoreMQTT-Bibliothek, um eine MQTT-Verbindung über TLS (gegenseitige Authentifizierung) zum AWS IoT MQTT-Broker herzustellen, und die CoreJSON-Bibliothek, um die vom Dienst empfangenen Antworten zu validieren und zu analysieren. AWS IoT Device Defender Die Demo zeigt Ihnen, wie Sie einen Bericht im JSON-Format mithilfe der vom Gerät gesammelten Metriken erstellen und wie Sie den erstellten Bericht an den Service senden. AWS IoT Device Defender Die Demo zeigt auch, wie Sie eine Callback-Funktion in der CoreMQTT-Bibliothek registrieren, um die Antworten des AWS IoT Device Defender Dienstes zu verarbeiten und zu bestätigen, ob ein gesendeter Bericht akzeptiert oder abgelehnt wurde.

> **Note**
>
> Folgen Sie den Schritten unter, um die FreeRTOS-Demos einzurichten und auszuführen. Erste Schritte mit FreeRTOS

Funktionalität

Diese Demo erstellt eine einzelne Anwendungsaufgabe, die demonstriert, wie Sie Metriken sammeln, einen Device Defender-Bericht im JSON-Format erstellen und ihn über eine sichere MQTT-Verbindung zum MQTT-Broker an den AWS IoT Device Defender Service senden. AWS IoT Die Demo umfasst sowohl die standardmäßigen Netzwerkmetriken als auch benutzerdefinierte Metriken. Für benutzerdefinierte Metriken umfasst die Demo:

- Eine Metrik namens "`task_numbers`", bei der es sich um eine Liste von FreeRTOS-Aufgaben handelt. IDs Der Typ dieser Metrik ist „Zahlenliste“.
- Eine Metrik mit dem Namen "`stack_high_water_mark`", bei der es sich um das Stack-High-Wasserzeichen für die Demo-Anwendungsaufgabe handelt. Der Typ dieser Metrik ist „Zahl“.

Wie wir Netzwerkmetriken sammeln, hängt vom verwendeten TCP/IP-Stack ab. Für FreeRTOS+TCP und unterstützte LwIP-Konfigurationen bieten wir Implementierungen zur Erfassung von Metriken an, die echte Metriken vom Gerät sammeln und sie im Bericht einreichen. AWS IoT Device Defender Sie finden die Implementierungen für FreeRTOS+TCP und lwIP auf. GitHub

Für Boards, die einen anderen TCP/IP-Stack verwenden, bieten wir Stub-Definitionen der Funktionen zur Erfassung von Metriken an, die für alle Netzwerkmetriken Nullen zurückgeben. Implementieren Sie die Funktionen *`freertos`*`/demos/device_defender_for_aws/metrics_collector/stub/metrics_collector.c` für Ihren Netzwerk-Stack, um echte Metriken zu senden. Die Datei ist auch auf der GitHubWebsite verfügbar.

Denn ESP32 die Standard-LwIP-Konfiguration verwendet kein Core-Locking und daher verwendet die Demo Stubbed-Metriken. Wenn Sie die Referenzimplementierung zur Erfassung von LWIP-Metriken verwenden möchten, definieren Sie die folgenden Makros in: `lwiopts.h`

```
#define LINK_SPEED_OF_YOUR_NETIF_IN_BPS 0
#define LWIP_TCPIP_CORE_LOCKING         1
#define LWIP_STATS                      1
#define MIB2_STATS                      1
```

Im Folgenden finden Sie ein Beispiel für die Ausgabe, wenn Sie die Demo ausführen.

```
24 1682 [iot_thread] [INFO] MQTT connection successfully established with broker.


25 1682 [iot_thread] [INFO] A clean MQTT connection is established. Cleaning up all the stored outgoing publishes.


26 1682 [iot_thread] [INFO] Attempt to subscribe to the MQTT topic $aws/things/DemoThing/defender/metrics/json/accepted.27 1682 [
iot_thread] [INFO] SUBSCRIBE sent for topic $aws/things/DemoThing/defender/metrics/json/accepted to broker.
28 1722 [iot_thread] [INFO] Packet received. ReceivedBytes=3.
29 1722 [iot_thread] [INFO] MQTT_PACKET_TYPE_SUBACK.


30 1722 [iot_thread] [INFO] Subscribed to the topic $aws/things/DemoThing/defender/metrics/json/accepted with maximum QoS 1.
31 2322 [iot_thread] [INFO] Attempt to subscribe to the MQTT topic $aws/things/DemoThing/defender/metrics/json/rejected.32 2322 [
iot_thread] [INFO] SUBSCRIBE sent for topic $aws/things/DemoThing/defender/metrics/json/rejected to broker.
33 2382 [iot_thread] [INFO] Packet received. ReceivedBytes=3.
34 2382 [iot_thread] [INFO] MQTT_PACKET_TYPE_SUBACK.


35 2382 [iot_thread] [INFO] Subscribed to the topic $aws/things/DemoThing/defender/metrics/json/rejected with maximum QoS 1.
36 2982 [iot_thread] [INFO] the published payload:{"hed": {"rid": 1109,"v": "1.0"},"met": {"tp": {"pts": [{"pt": 33251}],"t": 1},
"up": {"pts": [],"t": 0},"ns": {"bi": 0,"bo": 0,"pi": 0,"po": 0},"tc": {"ec": {"cs": [{"lp": 33251,"rad": "44.236.152.27:888337 2
982 [iot_thread] [INFO] PUBLISH sent for topic $aws/things/DemoThing/defender/metrics/json to broker with packet ID 3.


38 3102 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
39 3102 [iot_thread] [INFO] Ack packet deserialized with result: MQTTSuccess.
40 3102 [iot_thread] [INFO] State record updated. New state=MQTTPublishDone.
41 3102 [iot_thread] [INFO] PUBACK received for packet id 3.


42 3102 [iot_thread] [INFO] Cleaned up outgoing publish packet with packet id 3.


43 3102 [iot_thread] [INFO] Packet received. ReceivedBytes=141.
44 3102 [iot_thread] [INFO] De-serialized incoming PUBLISH packet: DeserializerResult=MQTTSuccess.
45 3102 [iot_thread] [INFO] State record updated. New state=MQTTPublishDone.
46 3502 [iot_thread] [INFO] UNSUBSCRIBE sent topic $aws/things/DemoThing/defender/metrics/json/accepted to broker.


47 3542 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
48 3542 [iot_thread] [INFO] MQTT_PACKET_TYPE_UNSUBACK.


49 4142 [iot_thread] [INFO] UNSUBSCRIBE sent topic $aws/things/DemoThing/defender/metrics/json/rejected to broker.


50 4202 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
51 4202 [iot_thread] [INFO] MQTT_PACKET_TYPE_UNSUBACK.


52 4802 [iot_thread] [INFO] Disconnected from the broker.
53 4802 [iot_thread] [INFO ][DEMO][4802] memory_metrics::freertos_heap::before::bytes::2088152

54 4802 [iot_thread] [INFO ][DEMO][4802] memory_metrics::freertos_heap::after::bytes::1985556

55 4802 [iot_thread] [INFO ][DEMO][4802] memory_metrics::demo_task_stack::before::bytes::1908

56 4802 [iot_thread] [INFO ][DEMO][4802] memory_metrics::demo_task_stack::after::bytes::1908

57 5802 [iot_thread] [INFO ][DEMO][5802] Demo completed successfully.

58 5804 [iot_thread] [INFO ][INIT][5804] SDK cleanup done.

59 5804 [iot_thread] [INFO ][DEMO][5804] -------DEMO FINISHED-------
```

Wenn dein Board nicht FreeRTOS+TCP oder eine unterstützte LwIP-Konfiguration verwendet, sieht die Ausgabe wie folgt aus.

```
24 1716 [iot_thread] [INFO] MQTT connection successfully established with broker.


25 1716 [iot_thread] [INFO] A clean MQTT connection is established. Cleaning up all the stored outgoing publishes.


26 1716 [iot_thread] [INFO] Attempt to subscribe to the MQTT topic $aws/things/DemoThing/defender/metrics/json/accepted.27 1716
 [iot_thread] [INFO] SUBSCRIBE sent for topic $aws/things/DemoThing/defender/metrics/json/accepted to broker.
28 1756 [iot_thread] [INFO] Packet received. ReceivedBytes=3.
29 1756 [iot_thread] [INFO] MQTT_PACKET_TYPE_SUBACK.


30 1756 [iot_thread] [INFO] Subscribed to the topic $aws/things/DemoThing/defender/metrics/json/accepted with maximum QoS 1.
31 2356 [iot_thread] [INFO] Attempt to subscribe to the MQTT topic $aws/things/DemoThing/defender/metrics/json/rejected.32 2356
 [iot_thread] [INFO] SUBSCRIBE sent for topic $aws/things/DemoThing/defender/metrics/json/rejected to broker.
33 2436 [iot_thread] [INFO] Packet received. ReceivedBytes=3.
34 2436 [iot_thread] [INFO] MQTT_PACKET_TYPE_SUBACK.


35 2436 [iot_thread] [INFO] Subscribed to the topic $aws/things/DemoThing/defender/metrics/json/rejected with maximum QoS 1.
36 3036 [iot_thread] [ERROR] Using stub definition of GetNetworkStats! Please implement for your network stack to get correct m
etrics.
37 3036 [iot_thread] [ERROR] Using stub definition of GetOpenTcpPorts! Please implement for your network stack to get correct m
etrics.
38 3036 [iot_thread] [ERROR] Using stub definition of GetOpenUdpPorts! Please implement for your network stack to get correct m
etrics.
39 3036 [iot_thread] [ERROR] Using stub definition of GetEstablishedConnections! Please implement for your network stack to get
 correct metrics.
40 3036 [iot_thread] [INFO] the published payload:{"hed": {"rid": 1079,"v": "1.0"},"met": {"tp": {"pts": [],"t": 0},"up": {"pts
": [],"t": 0},"ns": {"bi": 0,"bo": 0,"pi": 0,"po": 0},"tc": {"ec": {"cs": [],"t": 0}}},"cmet": {"stack_high_water_mark": [{"num
41 3036 [iot_thread] [INFO] PUBLISH sent for topic $aws/things/DemoThing/defender/metrics/json to broker with packet ID 3.


42 3196 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
43 3196 [iot_thread] [INFO] Ack packet deserialized with result: MQTTSuccess.
44 3196 [iot_thread] [INFO] State record updated. New state=MQTTPublishDone.
45 3196 [iot_thread] [INFO] PUBACK received for packet id 3.


46 3196 [iot_thread] [INFO] Cleaned up outgoing publish packet with packet id 3.


47 3196 [iot_thread] [INFO] Packet received. ReceivedBytes=141.
48 3196 [iot_thread] [INFO] De-serialized incoming PUBLISH packet: DeserializerResult=MQTTSuccess.
49 3196 [iot_thread] [INFO] State record updated. New state=MQTTPublishDone.
50 3596 [iot_thread] [INFO] UNSUBSCRIBE sent topic $aws/things/DemoThing/defender/metrics/json/accepted to broker.


51 3656 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
52 3656 [iot_thread] [INFO] MQTT_PACKET_TYPE_UNSUBACK.


53 4256 [iot_thread] [INFO] UNSUBSCRIBE sent topic $aws/things/DemoThing/defender/metrics/json/rejected to broker.


54 4336 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
55 4336 [iot_thread] [INFO] MQTT_PACKET_TYPE_UNSUBACK.


56 4936 [iot_thread] [INFO] Disconnected from the broker.
57 4936 [iot_thread] [INFO ][DEMO][4936] memory_metrics::freertos_heap::before::bytes::2088152

58 4936 [iot_thread] [INFO ][DEMO][4936] memory_metrics::freertos_heap::after::bytes::1985556

59 4936 [iot_thread] [INFO ][DEMO][4936] memory_metrics::demo_task_stack::before::bytes::1908

60 4936 [iot_thread] [INFO ][DEMO][4936] memory_metrics::demo_task_stack::after::bytes::1908

61 5936 [iot_thread] [INFO ][DEMO][5936] Demo completed successfully.

62 5938 [iot_thread] [INFO ][INIT][5938] SDK cleanup done.

63 5938 [iot_thread] [INFO ][DEMO][5938] -------DEMO FINISHED-------
```

Der Quellcode der Demo befindet sich in Ihrem *freertos*/demos/device_defender_for_aws/ Download-Verzeichnis oder auf der GitHubWebsite.

Themen abonnieren AWS IoT Device Defender

Mit der subscribeToDefenderThemenfunktion abonnieren Sie die MQTT-Themen, zu denen Antworten auf veröffentlichte Device Defender-Berichte eingehen werden. Sie verwendet das Makro`DEFENDER_API_JSON_ACCEPTED`, um die Themenzeichenfolge zu erstellen, zu der Antworten auf akzeptierte Device Defender-Berichte eingehen. Es verwendet das Makro`DEFENDER_API_JSON_REJECTED`, um die Themenzeichenfolge zu erstellen, zu der Antworten auf abgelehnte Device Defender-Berichte eingehen.

Erfassung von Gerätemetriken

Die collectDeviceMetricsFunktion sammelt Netzwerkmetriken mithilfe der unter definierten Funktionen. `metrics_collector.h` Bei den gesammelten Metriken handelt es sich um die Anzahl der gesendeten und empfangenen Bytes und Pakete, die offenen TCP-Ports, die offenen UDP-Ports und die hergestellten TCP-Verbindungen.

Der AWS IoT Device Defender Bericht wird generiert

Die generateDeviceMetricsBerichtsfunktion generiert mithilfe der unter definierten Funktion einen Device Defender-Bericht`report_builder.h`. Diese Funktion verwendet die Netzwerkmetriken und einen Puffer, erstellt ein JSON-Dokument in dem von erwarteten Format AWS IoT Device Defender und schreibt es in den bereitgestellten Puffer. Das von erwartete Format des JSON-Dokuments AWS IoT Device Defender ist unter Geräteseitige Metriken im AWS IoT Entwicklerhandbuch angegeben.

Den Bericht veröffentlichen AWS IoT Device Defender

Der AWS IoT Device Defender Bericht wird zum MQTT-Thema für die Veröffentlichung von AWS IoT Device Defender JSON-Berichten veröffentlicht. Der Bericht wird mithilfe des Makros erstellt`DEFENDER_API_JSON_PUBLISH`, wie in diesem Codeausschnitt auf der Website gezeigt. GitHub

Rückruf zur Bearbeitung von Antworten

Die Funktion publishCallback verarbeitet eingehende MQTT-Veröffentlichungsnachrichten. Sie verwendet die `Defender_MatchTopic` API aus der AWS IoT Device Defender Bibliothek, um zu überprüfen, ob die eingehende MQTT-Nachricht vom Dienst stammt. AWS IoT Device Defender Wenn die Nachricht vom AWS IoT Device Defender Dienst stammt, analysiert er die empfangene JSON-Antwort und extrahiert die Berichts-ID aus der Antwort. Anschließend wird überprüft, ob die Berichts-ID mit der im Bericht gesendeten identisch ist.

AWS IoT Greengrass V1 Discovery-Demoanwendung

> **Important**
>
> Diese Demo wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Bevor Sie die AWS IoT Greengrass Discovery-Demo für FreeRTOS ausführen, müssen Sie AWS AWS IoT Greengrass, und einrichten. AWS IoT Folgen Sie zur Einrichtung AWS den Anweisungen unter. Richten Sie Ihr AWS Konto und Ihre Berechtigungen ein Zur Einrichtung AWS IoT Greengrass müssen Sie eine Greengrass-Gruppe erstellen und dann einen Greengrass-Kern hinzufügen. Weitere Informationen zur Einrichtung finden Sie AWS IoT Greengrass unter Erste Schritte mit. AWS IoT Greengrass

Nachdem Sie AWS und eingerichtet haben AWS IoT Greengrass, müssen Sie einige zusätzliche Berechtigungen für konfigurieren AWS IoT Greengrass.

Um AWS IoT Greengrass Berechtigungen einzurichten

1. Rufen Sie die IAM-Konsole auf.
2. Wählen Sie im Navigationsbereich Roles aus, suchen Sie dann ServiceRoleGreengrass_ und wählen Sie es aus.
3. Wählen Sie Richtlinien anhängen, wählen Sie AmazonS3 FullAccess und AWSIoTFullAccess aus und wählen Sie dann Richtlinie anhängen aus.
4. Navigieren Sie zur AWS IoT -Konsole.
5. Wählen Sie im Navigationsbereich Greengrass, Groups (Gruppen) und dann die zuvor erstellte Greengrass-Gruppe aus.
6. Wählen Sie Settings (Einstellungen) und anschließend Add Role (Rolle hinzufügen) aus.
7. Wählen Sie Greengrass_ und ServiceRole dann Speichern.

Connect dein Board mit deiner FreeRTOS-Demo AWS IoT und konfiguriere sie.

1. Registrieren Sie Ihr MCU-Board bei AWS IoT

Nach dem Registrieren Ihres Boards müssen Sie eine neue Greengrass-Richtlinie erstellen und dem Zertifikat des Geräts anfügen.

Um eine neue Richtlinie zu erstellen AWS IoT Greengrass

1. Navigieren Sie zur AWS IoT -Konsole.
2. Wählen Sie im Navigationsbereich erst Sicher, dann Richtlinien und anschließend Erstellen aus.
3. Geben Sie einen Namen zur Identifizierung Ihrer Richtlinie ein.
4. Wählen Sie im Abschnitt Anweisungen hinzufügen die Option Erweiterter Modus aus. Kopieren Sie die folgende JSON und fügen Sie sie in das Fenster des Richtlinien-Editors ein:

   ```
   {
         "Effect": "Allow",
         "Action": [
           "greengrass:*"
         ],
         "Resource": "*"
   }
   ```

   Diese Richtlinie gewährt allen Ressourcen AWS IoT Greengrass Berechtigungen.
5. Wählen Sie Create (Erstellen) aus.

Um die AWS IoT Greengrass Richtlinie an das Zertifikat Ihres Geräts anzuhängen

1. Navigieren Sie zur AWS IoT -Konsole.
2. Wählen Sie im Navigationsbereich Manage (Verwalten), Things (Objekte) und anschließend das zuvor erstellte Objekt aus.
3. Wählen Sie Security (Sicherheit) und dann das Ihrem Gerät angefügte Zertifikat aus.
4. Wählen Sie Policies (Richtlinien), Actions (Aktionen) und anschließend Attach Policy (Richtlinie anfügen) aus.
5. Suchen Sie die zuvor von Ihnen erstellte Greengrass-Richtlinie, wählen Sie diese aus und klicken Sie dann auf Attach (Anfügen).

2. FreeRTOS wird heruntergeladen

> **Note**
>
> Wenn Sie FreeRTOS von der FreeRTOS-Konsole herunterladen, wählen Sie Connect to AWS IoT Greengrass- ***Platform*** statt Connect to -. AWS IoT***Platform***

3. Konfiguration der FreeRTOS-Demos.

   Öffnen Sie *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h, kommentieren Sie #define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED aus und definieren Sie CONFIG_GREENGRASS_DISCOVERY_DEMO_ENABLED.

Nachdem Sie FreeRTOS eingerichtet AWS IoT und AWS IoT Greengrass heruntergeladen und konfiguriert haben, können Sie die Greengrass-Demo auf Ihrem Gerät erstellen, flashen und ausführen. Befolgen Sie für die Einrichtung der Hardware und Softwareentwicklungsumgebung Ihres Boards die Anweisungen im Abschnitt Board-spezifische Handbücher "Erste Schritte".

Die Greengrass-Demo veröffentlicht eine Reihe von Nachrichten an den Greengrass-Kern und an den AWS IoT MQTT-Client. Um die Nachrichten im AWS IoT MQTT-Client anzuzeigen, öffnen Sie die AWS IoT Konsole, wählen Sie Test, wählen Sie MQTT-Testclient und fügen Sie dann ein Abonnement für hinzu. `freertos/demos/ggd`

Die folgenden Zeichenfolgen müssten im MQTT-Client angezeigt werden:

```
Message from Thing to Greengrass Core: Hello world msg #1!
Message from Thing to Greengrass Core: Hello world msg #0!
Message from Thing to Greengrass Core: Address of Greengrass Core
 found! 123456789012.us-west-2.compute.amazonaws.com
```

Verwenden einer EC2 Amazon-Instance

Wenn Sie mit einer EC2 Amazon-Instance arbeiten

1. Suchen Sie das öffentliche DNS (IPv4), das mit Ihrer EC2 Amazon-Instance verknüpft ist. Gehen Sie zur EC2 Amazon-Konsole und wählen Sie im linken Navigationsbereich Instances aus. Wählen Sie Ihre EC2 Amazon-Instance und dann das Beschreibungsfeld aus. Suchen Sie nach dem Eintrag für den öffentlichen DNS (IPv4) und notieren Sie ihn.

2. Suchen Sie den Eintrag für Sicherheitsgruppen und wählen Sie die Sicherheitsgruppe aus, die Ihrer EC2 Amazon-Instance zugeordnet ist.
3. Wählen Sie die Registerkarte Eingehende Regeln und dann Eingehende Regeln bearbeiten und fügen Sie die folgenden Regeln hinzu.

   Regeln für eingehenden Datenverkehr

Typ	Protocol (Protokoll)	Port-Bereich	Quelle	Beschreibung - optional
HTTP	TCP	80	0.0.0.0/0	-
HTTP	TCP	80	::/0	-
SSH	TCP	22	0.0.0.0/0	-
Custom TCP	TCP	8883	0.0.0.0/0	MQTT-Kommunikation
Custom TCP	TCP	8883	::/0	MQTT-Kommunikation
HTTPS	TCP	443	0.0.0.0/0	-
HTTPS	TCP	443	::0/0	-
Alles ICMP - IPv4	ICMP	Alle	0.0.0.0/0	-
Alles ICMP - IPv4	ICMP	Alle	::0/0	-

4. Wählen Sie in der AWS IoT Konsole Greengrass, dann Groups und wählen Sie die Greengrass-Gruppe aus, die Sie zuvor erstellt haben. Wählen Sie Einstellungen aus. Ändern Sie Erkennung lokaler Verbindungen in Verbindungsinformationen manuell verwalten.
5. Wählen Sie im Navigationsbereich die Option Kerne und dann Ihren Gruppenkern aus.
6. Wählen Sie Konnektivität und stellen Sie sicher, dass Sie nur einen Kernendpunkt haben (alle übrigen löschen) und dass es sich nicht um eine IP-Adresse handelt (da dieser sich ändern kann). Am besten verwenden Sie das öffentliche DNS (IPv4), das Sie im ersten Schritt notiert haben.

7. Fügen Sie das FreeRTOS-IoT-Objekt, das von Ihnen erstellt wurde, der GG-Gruppe hinzu.

   a. Wählen Sie den Zurück-Pfeil, um zur AWS IoT Greengrass Gruppenseite zurückzukehren. Wählen Sie im Navigationsbereich Geräte und anschließend Gerät hinzufügen aus.

   b. Wählen Sie Ein IoT-Objekt auswählen aus. Wählen Sie Ihr Gerät und dann Fertig stellen aus.

8. Fügen Sie die erforderlichen Abonnements hinzu — Wählen Sie auf der Greengrass-Gruppenseite Abonnements und dann Abonnement hinzufügen aus und geben Sie die Informationen wie hier gezeigt ein.

   Subscriptions (Abonnements)

Quelle	Ziel	Thema
TIGG1	IoT Cloud	freertos/demos/ggd

   Wobei „Source“ der Name ist, der dem AWS IoT Ding gegeben wurde, das in der AWS IoT Konsole erstellt wurde, als Sie Ihr Board registriert haben - TIGG1 "" in dem hier angegebenen Beispiel.

9. Starte ein Deployment deiner AWS IoT Greengrass Gruppe und stelle sicher, dass das Deployment erfolgreich ist. Sie sollten jetzt in der Lage sein, die AWS IoT Greengrass Discovery-Demo erfolgreich auszuführen.

AWS IoT Greengrass V2

Kompatibilität mit Geräten AWS IoT Greengrass V2

AWS IoT Greengrass V2 Die Unterstützung für Client-Geräte ist abwärtskompatibel mit. AWS IoT Greengrass V1 Sie können FreeRTOS-Client-Geräte mit V2-Core-Geräten verbinden, ohne den Anwendungscode zu ändern. Gehen Sie wie folgt vor, damit Client-Geräte eine Verbindung zu einem V2-Core-Gerät herstellen können.

- Stellen Sie die Greengrass-Software auf dem Greengrass-Core-Gerät bereit. Weitere Informationen finden Connect unter Client-Geräte mit Core-Geräten verbinden, mit denen Sie ein Gerät verbinden können AWS IoT Greengrass V2.

- Um Nachrichten (einschließlich Lambda-Funktionen) zwischen Client-Geräten, AWS IoT Core Cloud-Diensten und Greengrass-Komponenten weiterzuleiten, müssen Sie die MQTT-Bridge-Komponente bereitstellen und konfigurieren.
- Stellen Sie die IP-Detektorkomponente bereit, um Verbindungsinformationen automatisch zu erkennen, oder verwalten Sie die Endpunkte manuell.
- Weitere Informationen finden Sie unter Interagieren mit lokalen AWS IoT Geräten.

Weitere Informationen finden Sie in der AWS Dokumentation zum Ausführen von AWS IoT Greengrass V1 Anwendungen auf AWS IoT Greengrass V2.

CoreHTTP-Demos

> **Important**
>
> Diese Demo wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Diese Demos können Ihnen helfen, den Umgang mit der CoreHTTP-Bibliothek zu erlernen.

Themen

- Demo zur gegenseitigen CoreHTTP-Authentifizierung
- CoreHTTP Basic Amazon S3 S3-Upload-Demo
- Laden Sie die CoreHttp Basic S3-Demo herunter
- CoreHTTP-Basis-Multithread-Demo

Demo zur gegenseitigen CoreHTTP-Authentifizierung

> **Important**
>
> Diese Demo wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-

FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Einführung

Das CoreHTTP-Demoprojekt (Mutual Authentication) zeigt Ihnen, wie Sie mithilfe von TLS eine Verbindung zu einem HTTP-Server mit gegenseitiger Authentifizierung zwischen dem Client und dem Server herstellen. Diese Demo verwendet eine MbedTLS-basierte Transportschnittstellenimplementierung, um eine server- und clientauthentifizierte TLS-Verbindung herzustellen, und demonstriert einen Anforderungsantwort-Workflow in HTTP.

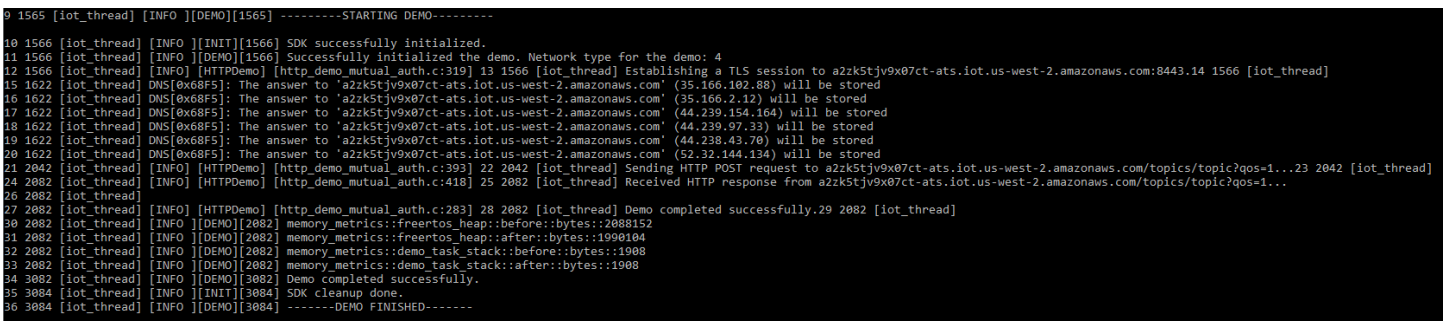
> **Note**
>
> Folgen Sie den Schritten unter, um die FreeRTOS-Demos einzurichten und auszuführen. Erste Schritte mit FreeRTOS

Funktionalität

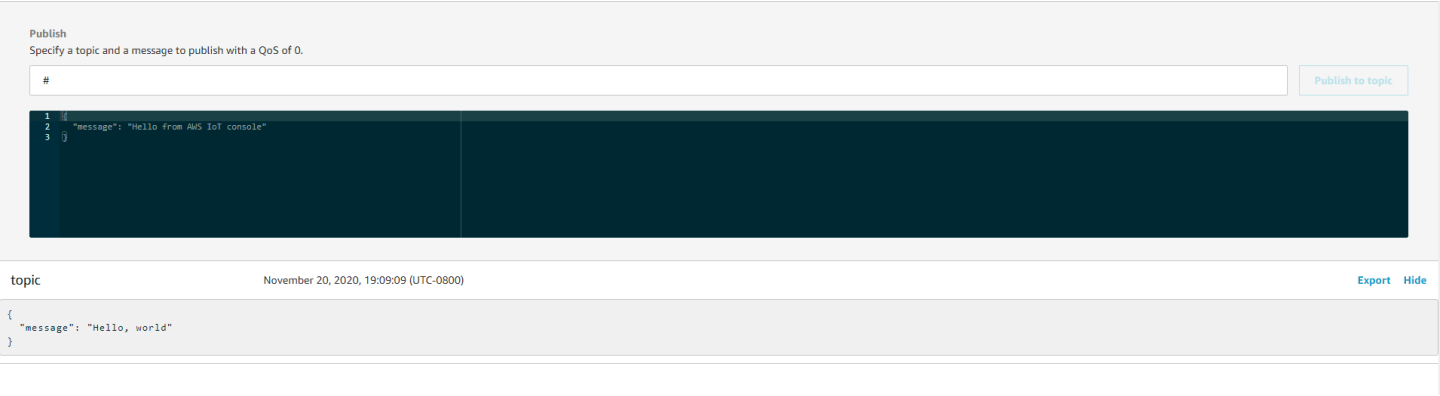
Diese Demo erstellt eine einzelne Anwendungsaufgabe mit Beispielen, die zeigen, wie Sie Folgendes ausführen können:

- Connect zum HTTP-Server auf dem AWS IoT Endpunkt her.
- Senden Sie eine POST-Anfrage.
- Empfangen Sie die Antwort.
- Trennen Sie die Verbindung zum Server.

Nachdem Sie diese Schritte abgeschlossen haben, generiert die Demo eine Ausgabe, die der folgenden Abbildung ähnelt.



Die AWS IoT Konsole generiert eine Ausgabe, die der folgenden Abbildung ähnelt.



Organisation des Quellcodes

Die Demo-Quelldatei ist benannt `http_demo_mutual_auth.c` und befindet sich im *`freertos`*`/demos/coreHTTP/` Verzeichnis und auf der GitHubWebsite.

Verbindung zum AWS IoT HTTP-Server herstellen

Die connectToServerWithBackoffRetriesFunktion versucht, eine gegenseitig authentifizierte TLS-Verbindung zum AWS IoT HTTP-Server herzustellen. Wenn die Verbindung fehlschlägt, versucht sie es nach einem Timeout erneut. Der Timeout-Wert steigt exponentiell an, bis die maximale Anzahl von Versuchen oder der maximale Timeout-Wert erreicht ist. Die `RetryUtils_BackoffAndSleep` Funktion liefert exponentiell steigende Timeout-Werte und kehrt zurück, `RetryUtilsRetriesExhausted` wenn die maximale Anzahl von Versuchen erreicht wurde. Die `connectToServerWithBackoffRetries` Funktion gibt einen Fehlerstatus zurück, wenn die TLS-Verbindung zum Broker nach der konfigurierten Anzahl von Versuchen nicht hergestellt werden kann.

Senden einer HTTP-Anfrage und Empfangen der Antwort

Die prvSendHttpRequest-Funktion demonstriert, wie eine POST-Anfrage an den AWS IoT HTTP-Server gesendet wird. Weitere Informationen zum Stellen einer Anfrage an die REST-API finden Sie unter Gerätekommunikationsprotokolle — HTTPS. AWS IoT Die Antwort wird mit demselben CoreHTTP-API-Aufruf empfangen,`HTTPClient_Send`.

CoreHTTP Basic Amazon S3 S3-Upload-Demo

> **Important**
>
> Diese Demo wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits

ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Einführung

Dieses Beispiel zeigt, wie Sie eine PUT-Anfrage an den Amazon Simple Storage Service (Amazon S3) HTTP-Server senden und eine kleine Datei hochladen. Außerdem wird eine GET-Anfrage ausgeführt, um die Größe der Datei nach dem Upload zu überprüfen. In diesem Beispiel wird eine Netzwerktransportschnittstelle verwendet, die MbedTLS verwendet, um eine gegenseitig authentifizierte Verbindung zwischen einem IoT-Geräteclient, auf dem CoreHTTP ausgeführt wird, und dem Amazon S3 S3-HTTP-Server herzustellen.

> **Note**
>
> Folgen Sie den Schritten unter, um die FreeRTOS-Demos einzurichten und auszuführen. Erste Schritte mit FreeRTOS

Single-Thread versus Multi-Threading

Es gibt zwei CoreHTTP-Nutzungsmodelle: Singlethread und Multithread (Multitasking). Die Demo in diesem Abschnitt führt zwar die HTTP-Bibliothek in einem Thread aus, zeigt aber tatsächlich, wie CoreHTTP in einer Single-Thread-Umgebung verwendet wird. Nur eine Aufgabe in dieser Demo verwendet die HTTP-API. Obwohl Single-Thread-Anwendungen die HTTP-Bibliothek wiederholt aufrufen müssen, können Multithread-Anwendungen stattdessen HTTP-Anfragen im Hintergrund innerhalb einer Agenten- (oder Daemon-) Aufgabe senden.

Organisation des Quellcodes

Die Demo-Quelldatei ist benannt `http_demo_s3_upload.c` und befindet sich im *`freertos`*`/demos/coreHTTP/` Verzeichnis und auf der GitHubWebsite.

Konfiguration der Amazon S3 S3-HTTP-Serververbindung

Diese Demo verwendet eine vorsignierte URL, um eine Verbindung zum Amazon S3 S3-HTTP-Server herzustellen und den Zugriff auf das herunterzuladende Objekt zu autorisieren. Die TLS-Verbindung des Amazon S3 S3-HTTP-Servers verwendet nur die Serverauthentifizierung. Auf

Anwendungsebene wird der Zugriff auf das Objekt mit Parametern in der vorsignierten URL-Abfrage authentifiziert. Gehen Sie wie folgt vor, um Ihre Verbindung zu zu konfigurieren. AWS

1. Richten Sie ein AWS Konto ein:

   a. Falls Sie es noch nicht getan haben, erstellen Sie ein AWS Konto.

   b. Konten und Berechtigungen werden mithilfe von AWS Identity and Access Management (IAM) festgelegt. Sie verwenden IAM, um die Berechtigungen für jeden Benutzer in Ihrem Konto zu verwalten. Standardmäßig verfügt ein Benutzer erst dann über Berechtigungen, wenn sie vom Root-Besitzer erteilt wurden.

      i. Informationen zum Hinzufügen eines Benutzers zu Ihrem AWS Konto finden Sie im IAM-Benutzerhandbuch.

      ii. Erteilen Sie Ihrem AWS Konto die Erlaubnis, auf FreeRTOS zuzugreifen, und AWS IoT fügen Sie diese Richtlinie hinzu:

         • Amazon S3 FullAccess

2. Erstellen Sie einen Bucket in Amazon S3, indem Sie den Schritten unter Wie erstelle ich einen S3-Bucket folgen? im Amazon Simple Storage Service-Benutzerhandbuch.
3. Laden Sie eine Datei auf Amazon S3 hoch, indem Sie den Schritten unter Wie lade ich Dateien und Ordner in einen S3-Bucket hoch? folgen .
4. Generieren Sie mithilfe des Skripts, das sich in der `FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/presigned_urls_gen.py` Datei befindet, eine vorsignierte URL.

   Anweisungen zur Verwendung finden Sie in der `FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/README.md` Datei.

Funktionalität

Die Demo stellt zunächst eine Verbindung zum Amazon S3 S3-HTTP-Server mit TLS-Serverauthentifizierung her. Anschließend wird eine HTTP-Anfrage zum Hochladen der in angegebenen Daten erstelltdemo`configDEMO_HTTP_UPLOAD_DATA`. Nach dem Hochladen der Datei wird überprüft, ob die Datei erfolgreich hochgeladen wurde, indem die Größe der Datei abgefragt wird. Der Quellcode für die Demo ist auf der GitHubWebsite zu finden.

Verbindung zum Amazon S3 S3-HTTP-Server herstellen

Die connectToServerWithBackoffRetriesFunktion versucht, eine TCP-Verbindung zum HTTP-Server herzustellen. Wenn die Verbindung fehlschlägt, versucht sie es nach einem Timeout erneut. Der Timeout-Wert wird exponentiell erhöht, bis die maximale Anzahl von Versuchen oder der maximale Timeout-Wert erreicht ist. Die `connectToServerWithBackoffRetries` Funktion gibt einen Fehlerstatus zurück, wenn die TCP-Verbindung zum Server nach der konfigurierten Anzahl von Versuchen nicht hergestellt werden kann.

Die `prvConnectToServer` Funktion zeigt, wie eine Verbindung zum Amazon S3 S3-HTTP-Server ausschließlich mithilfe der Serverauthentifizierung hergestellt wird. Es verwendet die MbedTLS-basierte Transportschnittstelle, die in der Datei implementiert ist. `FreeRTOS-Plus/Source/Application-Protocols/network_transport/freertos_plus_tcp/using_mbedtls/using_mbedtls.c` Die Definition von `prvConnectToServer` finden Sie auf der Website. GitHub

Daten hochladen

Die `prvUploadS3ObjectFile` Funktion zeigt, wie Sie eine PUT-Anfrage erstellen und die hochzuladende Datei angeben. Der Amazon S3 S3-Bucket, in den die Datei hochgeladen wird, und der Name der hochzuladenden Datei sind in der vorsignierten URL angegeben. Um Speicherplatz zu sparen, wird derselbe Puffer sowohl für die Anforderungsheader als auch für den Empfang der Antwort verwendet. Die Antwort wird synchron mithilfe der `HTTPClient_Send` API-Funktion empfangen. Ein `200 OK` Antwortstatuscode wird vom Amazon S3 S3-HTTP-Server erwartet. Jeder andere Statuscode ist ein Fehler.

Der Quellcode für `prvUploadS3ObjectFile()` kann auf der GitHubWebsite gefunden werden.

Überprüfung des Uploads

Die `prvVerifyS3ObjectFileSize` Funktion ruft `prvGetS3ObjectFileSize` auf, um die Größe des Objekts im S3-Bucket abzurufen. Der Amazon S3 S3-HTTP-Server unterstützt derzeit keine HEAD-Anfragen, die eine vorsignierte URL verwenden, sodass das 0-te Byte angefordert wird. Die Größe der Datei ist im `Content-Range` Header-Feld der Antwort enthalten. Eine `206 Partial Content` Antwort wird vom Server erwartet. Jeder andere Antwortstatuscode ist ein Fehler.

Der Quellcode für `prvGetS3ObjectFileSize()` kann auf der GitHubWebsite gefunden werden.

Laden Sie die CoreHttp Basic S3-Demo herunter

> **Important**
>
> Diese Demo wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Einführung

Diese Demo zeigt, wie Sie Bereichsanforderungen verwenden, um Dateien vom Amazon S3 S3-HTTP-Server herunterzuladen. Bereichsanforderungen werden in der CoreHTTP-API nativ unterstützt, wenn Sie sie `HTTPClient_AddRangeHeader` zur Erstellung der HTTP-Anfrage verwenden. In einer Mikrocontroller-Umgebung werden Bereichsanfragen dringend empfohlen. Durch das Herunterladen einer großen Datei in separaten Bereichen statt in einer einzigen Anfrage kann jeder Abschnitt der Datei verarbeitet werden, ohne den Netzwerk-Socket zu blockieren. Bereichsanfragen verringern das Risiko, dass Pakete verworfen werden, was erneute Übertragungen über die TCP-Verbindung erfordert, und verbessern so den Stromverbrauch des Geräts.

In diesem Beispiel wird eine Netzwerktransportschnittstelle verwendet, die MbedTLS verwendet, um eine gegenseitig authentifizierte Verbindung zwischen einem IoT-Geräteclient, auf dem CoreHTTP ausgeführt wird, und dem Amazon S3 S3-HTTP-Server herzustellen.

> **Note**
>
> Folgen Sie den Schritten unter, um die FreeRTOS-Demos einzurichten und auszuführen. Erste Schritte mit FreeRTOS

Single-Thread versus Multi-Threading

Es gibt zwei CoreHTTP-Nutzungsmodelle: Singlethread und Multithread (Multitasking). Die Demo in diesem Abschnitt führt zwar die HTTP-Bibliothek in einem Thread aus, zeigt aber tatsächlich, wie CoreHTTP in einer Single-Thread-Umgebung verwendet wird (nur eine Aufgabe verwendet die HTTP-API in der Demo). Obwohl Singlethread-Anwendungen die HTTP-Bibliothek wiederholt

aufrufen müssen, können Multithread-Anwendungen stattdessen HTTP-Anfragen im Hintergrund innerhalb einer Agenten- (oder Daemon-) Aufgabe senden.

Organisation des Quellcodes

Das Demo-Projekt hat einen Namen `http_demo_s3_download.c` und ist im *`freertos`*`/demos/coreHTTP/` Verzeichnis und auf der GitHubWebsite zu finden.

Konfiguration der Amazon S3 S3-HTTP-Serververbindung

Diese Demo verwendet eine vorsignierte URL, um eine Verbindung zum Amazon S3 S3-HTTP-Server herzustellen und den Zugriff auf das herunterzuladende Objekt zu autorisieren. Die TLS-Verbindung des Amazon S3 S3-HTTP-Servers verwendet nur die Serverauthentifizierung. Auf Anwendungsebene wird der Zugriff auf das Objekt mit Parametern in der vorsignierten URL-Abfrage authentifiziert. Gehen Sie wie folgt vor, um Ihre Verbindung zu zu konfigurieren. AWS

1. Richten Sie ein AWS Konto ein:
    a. Falls Sie es noch nicht getan haben, erstellen und aktivieren Sie ein AWS Konto.
    b. Konten und Berechtigungen werden mithilfe von AWS Identity and Access Management (IAM) festgelegt. Mit IAM können Sie die Berechtigungen für jeden Benutzer in Ihrem Konto verwalten. Standardmäßig verfügt ein Benutzer erst dann über Berechtigungen, wenn sie vom Root-Besitzer erteilt wurden.
        i. Informationen zum Hinzufügen eines Benutzers zu Ihrem AWS Konto finden Sie im IAM-Benutzerhandbuch.
        ii. Erteilen Sie Ihrem AWS Konto die Erlaubnis, auf FreeRTOS zuzugreifen, und AWS IoT fügen Sie diese Richtlinien hinzu:
            • Amazon S3 FullAccess
2. Erstellen Sie einen Bucket in S3, indem Sie den Schritten unter Wie erstelle ich einen S3-Bucket? folgen im Amazon Simple Storage Service Console-Benutzerhandbuch.
3. Laden Sie eine Datei auf S3 hoch, indem Sie den Schritten unter Wie lade ich Dateien und Ordner in einen S3-Bucket hoch? folgen .
4. Generieren Sie eine vorsignierte URL mithilfe des Skripts unter`FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/presigned_urls_gen.py`. Anweisungen zur Verwendung finden Sie unter`FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/README.md`.

Funktionalität

Die Demo ruft zuerst die Größe der Datei ab. Dann fordert sie jeden Bytebereich sequentiell, in einer Schleife, mit einer Bereichsgröße von an. `democonfigRANGE_REQUEST_LENGTH`

Der Quellcode für die Demo ist auf der GitHubWebsite zu finden.

Verbindung zum Amazon S3 S3-HTTP-Server herstellen

Die Funktion connectToServerWithBackoffRetries() versucht, eine TCP-Verbindung zum HTTP-Server herzustellen. Wenn die Verbindung fehlschlägt, versucht sie es nach einem Timeout erneut. Der Timeout-Wert wird exponentiell erhöht, bis die maximale Anzahl von Versuchen oder der maximale Timeout-Wert erreicht ist. `connectToServerWithBackoffRetries()`gibt einen Fehlerstatus zurück, wenn die TCP-Verbindung zum Server nach der konfigurierten Anzahl von Versuchen nicht hergestellt werden kann.

Die Funktion `prvConnectToServer()` zeigt, wie eine Verbindung zum Amazon S3 S3-HTTP-Server ausschließlich mithilfe der Serverauthentifizierung hergestellt wird. Es verwendet die MbedTLS-basierte Transportschnittstelle, die in der Datei FreeRTOS- _mbedtls.c implementiert ist. Plus/Source/Application-Protocols/network_transport/freertos_plus_tcp/using_mbedtls/using

Den Quellcode für finden Sie auf`prvConnectToServer()`. GitHub

Eine Bereichsanfrage erstellen

Die API-Funktion `HTTPClient_AddRangeHeader()` unterstützt die Serialisierung eines Bytebereichs in die HTTP-Anforderungsheader, um eine Bereichsanforderung zu bilden. Bereichsanfragen werden in dieser Demo verwendet, um die Dateigröße abzurufen und jeden Abschnitt der Datei anzufordern.

Die Funktion `prvGetS3ObjectFileSize()` ruft die Größe der Datei im S3-Bucket ab. Der `Connection: keep-alive` Header wird in dieser ersten Anfrage zu Amazon S3 hinzugefügt, um die Verbindung nach dem Senden der Antwort aufrechtzuerhalten. Der S3-HTTP-Server unterstützt derzeit keine HEAD-Anfragen, die eine vorsignierte URL verwenden, daher wird das 0-te Byte angefordert. Die Größe der Datei ist im `Content-Range` Header-Feld der Antwort enthalten. Eine `206 Partial Content` Antwort wird vom Server erwartet; jeder andere empfangene Antwortstatuscode ist ein Fehler.

Den Quellcode für `prvGetS3ObjectFileSize()` finden Sie unter. GitHub

Nach dem Abrufen der Dateigröße erstellt diese Demo eine neue Bereichsanforderung für jeden Bytebereich der herunterzuladenden Datei. Es wird `HTTPClient_AddRangeHeader()` für jeden Abschnitt der Datei verwendet.

Senden von Bereichsanfragen und Empfangen von Antworten

Die Funktion `prvDownloadS3ObjectFile()` sendet die Bereichsanfragen in einer Schleife, bis die gesamte Datei heruntergeladen ist. Die API-Funktion `HTTPClient_Send()` sendet eine Anfrage und empfängt die Antwort synchron. Wenn die Funktion zurückkehrt, wird die Antwort in einer `xResponse` empfangen. Anschließend wird überprüft, ob der Statuscode korrekt ist, `206 Partial Content` und die Anzahl der bisher heruntergeladenen Byte wird um den `Content-Length` Header-Wert erhöht.

Den Quellcode für `prvDownloadS3ObjectFile()` finden Sie unter. GitHub

CoreHTTP-Basis-Multithread-Demo

> **Important**
>
> Diese Demo wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Einführung

Diese Demo verwendet die threadsicheren Warteschlangen von FreeRTOS, um Anfragen und Antworten zu speichern, die auf ihre Bearbeitung warten. In dieser Demo gibt es drei Aufgaben, die Sie beachten sollten.

- Die Hauptaufgabe wartet darauf, dass Anfragen in der Anforderungswarteschlange erscheinen. Sie sendet diese Anfragen über das Netzwerk und platziert die Antwort dann in die Antwortwarteschlange.
- Eine Anforderungsaufgabe erstellt Anforderungsobjekte der HTTP-Bibliothek, die an den Server gesendet werden sollen, und platziert sie in der Anforderungswarteschlange. Jedes Anforderungsobjekt gibt einen Bytebereich der S3-Datei an, die die Anwendung für den Download konfiguriert hat.

- Eine Antwortaufgabe wartet darauf, dass Antworten in der Antwortwarteschlange erscheinen. Sie protokolliert jede Antwort, die sie erhält.

Diese grundlegende Multithread-Demo ist so konfiguriert, dass sie nur eine TLS-Verbindung mit Serverauthentifizierung verwendet. Dies ist für den Amazon S3 S3-HTTP-Server erforderlich. Die Authentifizierung auf Anwendungsebene erfolgt mithilfe der Signature Version 4-Parameter in der vorsignierten URL-Abfrage.

Organisation des Quellcodes

Das Demo-Projekt hat einen Namen `http_demo_s3_download_multithreaded.c` und ist im *`freertos`*`/demos/coreHTTP/` Verzeichnis und GitHubauf der Website zu finden.

Das Demo-Projekt erstellen

Das Demo-Projekt verwendet die kostenlose Community-Edition von Visual Studio. Um die Demo zu erstellen:

1. Öffnen Sie die `mqtt_multitask_demo.sln` Visual Studio-Lösungsdatei in der Visual Studio-IDE.
2. Wählen Sie im Build-Menü der IDE die Option Build Solution aus.

> **Note**
>
> Wenn Sie Microsoft Visual Studio 2017 oder früher verwenden, müssen Sie ein Platform Toolset auswählen, das mit Ihrer Version kompatibel ist: Projekt -> RTOSDemos Eigenschaften -> Platform Toolset.

Konfiguration des Demo-Projekts

Die Demo verwendet den TCP/IP FreeRTOS+TCP-Stack. Folgen Sie daher den Anweisungen für das TCP/IP-Starterprojekt, um:

1. Installieren Sie die erforderlichen Komponenten (z. B. Win). PCap
2. Legen Sie optional eine statische oder dynamische IP-Adresse, Gateway-Adresse und Netzmaske fest.
3. Legen Sie optional eine MAC-Adresse fest.

4. Wählen Sie eine Ethernet-Netzwerkschnittstelle auf Ihrem Host-Computer aus.
5. Testen Sie unbedingt Ihre Netzwerkverbindung, bevor Sie versuchen, die HTTP-Demo auszuführen.

Konfiguration der Amazon S3 S3-HTTP-Serververbindung

Folgen Sie den Anweisungen Konfiguration der Amazon S3 S3-HTTP-Serververbindung in der CoreHTTP Basic Download-Demo.

Funktionalität

Die Demo erstellt insgesamt drei Aufgaben:

- Eine, die Anfragen sendet und Antworten über das Netzwerk empfängt.
- Eine, die Sendeanfragen erstellt.
- Eine, die die empfangenen Antworten verarbeitet.

In dieser Demo ist die Hauptaufgabe:

1. Erstellt die Anforderungs- und Antwortwarteschlangen.
2. Stellt die Verbindung zum Server her.
3. Erstellt die Anforderungs- und Antwortaufgaben.
4. Wartet darauf, dass die Anforderungswarteschlange Anfragen über das Netzwerk sendet.
5. Platziert Antworten, die über das Netzwerk empfangen wurden, in die Antwortwarteschlange.

Die Anforderungsaufgabe:

1. Erstellt jede der Bereichsanforderungen.

Die Antwortaufgabe:

1. Verarbeitet jede der eingegangenen Antworten.

Typdefinitionen

Die Demo definiert die folgenden Strukturen zur Unterstützung von Multithreading.

Artikel anfragen

Die folgenden Strukturen definieren ein Anforderungselement, das in die Anforderungswarteschlange gestellt werden soll. Das Anforderungselement wird in die Warteschlange kopiert, nachdem die Anforderungsaufgabe eine HTTP-Anfrage erstellt hat.

```
/**
 * @brief Data type for the request queue.
 *
 * Contains the request header struct and its corresponding buffer, to be
 * populated and enqueued by the request task, and read by the main task. The
 * buffer is included to avoid pointer inaccuracy during queue copy operations.
 */
typedef struct RequestItem
{
    HTTPRequestHeaders_t xRequestHeaders;
    uint8_t ucHeaderBuffer[ democonfigUSER_BUFFER_LENGTH ];
} RequestItem_t;
```

Antwortelement

Die folgenden Strukturen definieren ein Antwortelement, das in die Antwortwarteschlange aufgenommen werden soll. Das Antwortelement wird in die Warteschlange kopiert, nachdem die HTTP-Hauptaufgabe eine Antwort über das Netzwerk erhalten hat.

```
/**
 * @brief Data type for the response queue.
 *
 * Contains the response data type and its corresponding buffer, to be enqueued
 * by the main task, and interpreted by the response task. The buffer is
 * included to avoid pointer inaccuracy during queue copy operations.
 */
typedef struct ResponseItem
{
    HTTPResponse_t xResponse;
    uint8_t ucResponseBuffer[ democonfigUSER_BUFFER_LENGTH ];
} ResponseItem_t;
```

Hauptaufgabe zum Senden von HTTP

Die Hauptaufgabe der Anwendung:

1. Analysiert die vorsignierte URL für die Hostadresse, um eine Verbindung mit dem Amazon S3 S3-HTTP-Server herzustellen.
2. Analysiert die vorsignierte URL nach dem Pfad zu den Objekten im S3-Bucket.
3. Stellt über TLS mit Serverauthentifizierung eine Verbindung zum Amazon S3 S3-HTTP-Server her.
4. Erstellt die Anforderungs- und Antwortwarteschlangen.
5. Erstellt die Anfrage- und Antwortaufgaben.

Die Funktion `prvHTTPDemoTask()` führt diese Einrichtung durch und gibt den Demo-Status an. Der Quellcode für diese Funktion ist auf Github zu finden.

In der Funktion `prvDownloadLoop()` blockiert und wartet die Hauptaufgabe auf Anfragen aus der Anforderungswarteschlange. Wenn sie eine Anfrage erhält, sendet sie sie mithilfe der API-Funktion`HTTPClient_Send()`. Wenn die API-Funktion erfolgreich war, wird die Antwort in die Antwortwarteschlange gestellt.

Der Quellcode für `prvDownloadLoop()` ist auf Github zu finden.

HTTP-Anforderungsaufgabe

Die Anforderungsaufgabe ist in der Funktion angegeben`prvRequestTask`. Der Quellcode für diese Funktion ist auf Github zu finden.

Die Anforderungsaufgabe ruft die Größe der Datei im Amazon S3 S3-Bucket ab. Dies erfolgt in der Funktion`prvGetS3ObjectFileSize`. Der Header „Connection: keep-alive“ wird zu dieser Anfrage an Amazon S3 hinzugefügt, damit die Verbindung auch nach dem Senden der Antwort bestehen bleibt. Der Amazon S3 S3-HTTP-Server unterstützt derzeit keine HEAD-Anfragen, die eine vorsignierte URL verwenden, daher wird das 0-te Byte angefordert. Die Größe der Datei ist im `Content-Range` Header-Feld der Antwort enthalten. Eine `206 Partial Content` Antwort wird vom Server erwartet; jeder andere empfangene Antwortstatuscode ist ein Fehler.

Der Quellcode für `prvGetS3ObjectFileSize` kann auf Github gefunden werden.

Nach dem Abrufen der Dateigröße fordert die Anforderungsaufgabe weiterhin jeden Bereich der Datei an. Jede Bereichsanforderung wird in die Anforderungswarteschlange gestellt, damit

die Hauptaufgabe gesendet werden kann. Die Dateibereiche werden vom Demo-Benutzer im Makro konfiguriertdemoconfigRANGE_REQUEST_LENGTH. Bereichsanforderungen werden in der HTTP-Clientbibliothek-API mithilfe der Funktion HTTPClient_AddRangeHeader nativ unterstützt. Die Funktion prvRequestS3ObjectRange demonstriert, wie man sie benutztHTTPClient_AddRangeHeader().

Der Quellcode für die Funktion prvRequestS3ObjectRange ist auf Github zu finden.

HTTP-Antwort-Aufgabe

Die Antwortaufgaben warten in der Antwortwarteschlange auf Antworten, die über das Netzwerk empfangen wurden. Die Hauptaufgabe füllt die Antwortwarteschlange, wenn sie erfolgreich eine HTTP-Antwort empfängt. Diese Aufgabe verarbeitet die Antworten, indem sie den Statuscode, die Header und den Hauptteil protokolliert. Eine reale Anwendung kann die Antwort verarbeiten, indem sie beispielsweise den Antworttext in den Flash-Speicher schreibt. Wenn der Antwortstatuscode nicht lautet206 partial content, teilt die Aufgabe der Hauptaufgabe mit, dass die Demo fehlschlagen sollte. Die Antwortaufgabe ist in der Funktion angegeben. prvResponseTask Der Quellcode für diese Funktion ist auf Github zu finden.

AWS IoT Demo der Jobbibliothek

> **Important**
>
> Diese Demo wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Einführung

Die Demo der AWS IoT Jobs-Bibliothek zeigt Ihnen, wie Sie über eine MQTT-Verbindung eine Verbindung zum AWS IoT Jobs-Service herstellen, einen Job von einem Gerät abrufen und ihn auf einem Gerät verarbeiten. AWS IoT Das AWS IoT Jobs-Demoprojekt verwendet den FreeRTOS-Windows-Port, sodass es mit der Visual Studio Community-Version unter Windows erstellt und evaluiert werden kann. Es wird keine Mikrocontroller-Hardware benötigt. Die Demo stellt mithilfe von TLS auf dieselbe Weise eine sichere Verbindung zum AWS IoT MQTT-Broker her wie die. Demo zur gegenseitigen CoreMQTT-Authentifizierung

> **Note**
>
> Folgen Sie den Schritten unter, um die FreeRTOS-Demos einzurichten und auszuführen. Erste Schritte mit FreeRTOS

Organisation des Quellcodes

Der Demo-Code befindet sich in der `jobs_demo.c` Datei und kann auf der GitHubWebsite oder im *`freertos`*`/demos/jobs_for_aws/` Verzeichnis gefunden werden.

Konfigurieren Sie die AWS IoT MQTT-Broker-Verbindung

In dieser Demo verwenden Sie eine MQTT-Verbindung zum AWS IoT MQTT-Broker. Diese Verbindung ist auf die gleiche Weise konfiguriert wie die. Demo zur gegenseitigen CoreMQTT-Authentifizierung

Funktionalität

Die Demo zeigt den Workflow, der verwendet wird, um Jobs von einem Gerät zu empfangen AWS IoT und sie auf einem Gerät zu verarbeiten. Die Demo ist interaktiv und erfordert, dass Sie Jobs entweder mit der AWS IoT Konsole oder mit AWS Command Line Interface (AWS CLI) erstellen. Weitere Informationen zum Erstellen eines Jobs finden Sie unter create-job in der AWS CLI Befehlsreferenz. Für die Demo muss im Jobdokument ein `action` Schlüssel gesetzt sein, mit dem `print` eine Nachricht an die Konsole gedruckt werden kann.

Sehen Sie sich das folgende Format für dieses Jobdokument an.

```
{
    "action": "print",
    "message": "ADD_MESSAGE_HERE"
}
```

Sie können den verwenden AWS CLI , um einen Job wie im folgenden Beispielbefehl zu erstellen.

```
aws iot create-job \
    --job-id t12 \
    --targets arn:aws:iot:region:123456789012:thing/device1 \
    --document '{"action":"print","message":"hello world!"}'
```

In der Demo wird auch ein Jobdokument verwendet, für das der `action` Schlüssel auf `publish` zum erneuten Veröffentlichen der Nachricht zu einem Thema gesetzt ist. Sehen Sie sich das folgende Format für dieses Jobdokument an.

```
{
    "action": "publish",
    "message": "ADD_MESSAGE_HERE",
    "topic": "topic/name/here"
}
```

Die Demo wird solange wiederholt, bis sie ein Jobdokument erhält, bei dem der `action` Schlüssel auf `exit` zum Beenden der Demo gesetzt ist. Das Format für das Jobdokument lautet wie folgt.

```
{
    "action: "exit"
}
```

Einstiegspunkt der Jobs-Demo

Den Quellcode für die Jobs-Demo-Einstiegsfunktion finden Sie unter GitHub. Diese Funktion führt die folgenden Operationen aus:

1. Stellen Sie mithilfe der Hilfsfunktionen in `mqtt_demo_helpers.c` eine MQTT-Verbindung her.
2. Abonnieren Sie das MQTT-Thema für die `NextJobExecutionChanged` API unter Verwendung von Hilfsfunktionen in. `mqtt_demo_helpers.c` Die Themenzeichenfolge wurde zuvor mithilfe von Makros zusammengestellt, die von der AWS IoT Jobs-Bibliothek definiert wurden.
3. Veröffentlichen Sie im MQTT-Thema für die `StartNextPendingJobExecution` API, indem Sie Hilfsfunktionen in verwenden. `mqtt_demo_helpers.c` Die Themenzeichenfolge wurde zuvor mithilfe von Makros zusammengestellt, die in der AWS IoT Jobs-Bibliothek definiert wurden.
4. Rufen Sie wiederholt `MQTT_ProcessLoop` auf, um eingehende Nachrichten zu empfangen, die `prvEventCallback` zur Verarbeitung weitergeleitet werden.
5. Nachdem die Demoversion die Exit-Aktion erhalten hat, melden Sie sich vom MQTT-Thema ab und trennen Sie die Verbindung mithilfe der Hilfsfunktionen in der `mqtt_demo_helpers.c` Datei.

Callback für empfangene MQTT-Nachrichten

Die prvEventCallbackFunktion ruft `Jobs_MatchTopic` aus der AWS IoT Jobs-Bibliothek auf, um die eingehende MQTT-Nachricht zu klassifizieren. Wenn der Nachrichtentyp einem neuen Job entspricht, `prvNextJobHandler()` wird aufgerufen.

Die prvNextJobHandler-Funktion und die von ihr aufgerufenen Funktionen analysieren das Jobdokument anhand der Nachricht im JSON-Format und führen die im Job angegebene Aktion aus. Von besonderem Interesse ist die Funktion. `prvSendUpdateForJob`

Senden Sie ein Update für einen laufenden Job

Die Funktion prvSendUpdateForJob() ruft `Jobs_Update()` aus der Jobs-Bibliothek auf, um die Themenzeichenfolge aufzufüllen, die in der unmittelbar darauf folgenden MQTT-Veröffentlichungsoperation verwendet wird.

CoreMQTT-Demos

> **Important**
>
> Diese Demo wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Diese Demos können Ihnen helfen, den Umgang mit der CoreMQTT-Bibliothek zu erlernen.

Themen

- Demo zur gegenseitigen CoreMQTT-Authentifizierung
- Demo zur gemeinsamen Nutzung von CoreMQTT-Agenten

Demo zur gegenseitigen CoreMQTT-Authentifizierung

> **Important**
>
> Diese Demo wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits

ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Einführung

Das Demo-Projekt zur gegenseitigen Authentifizierung von CoreMQTT zeigt Ihnen, wie Sie mithilfe von TLS eine Verbindung zu einem MQTT-Broker mit gegenseitiger Authentifizierung zwischen dem Client und dem Server herstellen. Diese Demo verwendet eine MbedTLS-basierte Transportschnittstellenimplementierung, um eine server- und clientauthentifizierte TLS-Verbindung herzustellen, und demonstriert den Subscribe-Publish-Workflow von MQTT auf QoS-1-Ebene. Es abonniert einen Themenfilter, veröffentlicht dann Themen, die dem Filter entsprechen, und wartet auf den Empfang dieser Nachrichten vom Server auf QoS-1-Ebene. Dieser Zyklus der Veröffentlichung auf dem Broker und dem Empfang derselben Nachricht vom Broker wird auf unbestimmte Zeit wiederholt. Nachrichten in dieser Demo werden mit QoS 1 gesendet, was mindestens eine Lieferung gemäß der MQTT-Spezifikation garantiert.

> **Note**
>
> Folgen Sie den Schritten unter, um die FreeRTOS-Demos einzurichten und auszuführen. Erste Schritte mit FreeRTOS

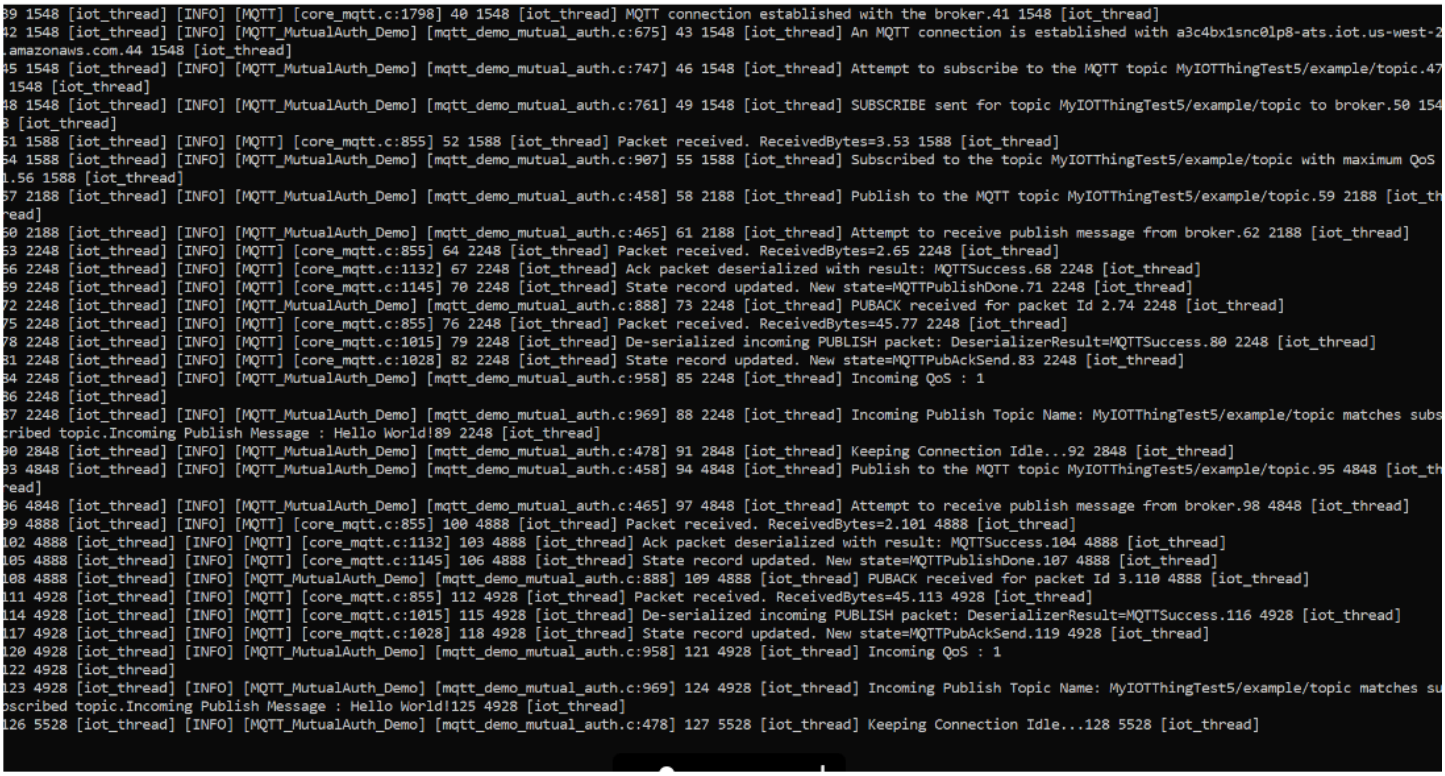
Quellcode

Die Demo-Quelldatei ist benannt `mqtt_demo_mutual_auth.c` und befindet sich im *`freertos`*`/demos/coreMQTT/` Verzeichnis und auf der Website. GitHub

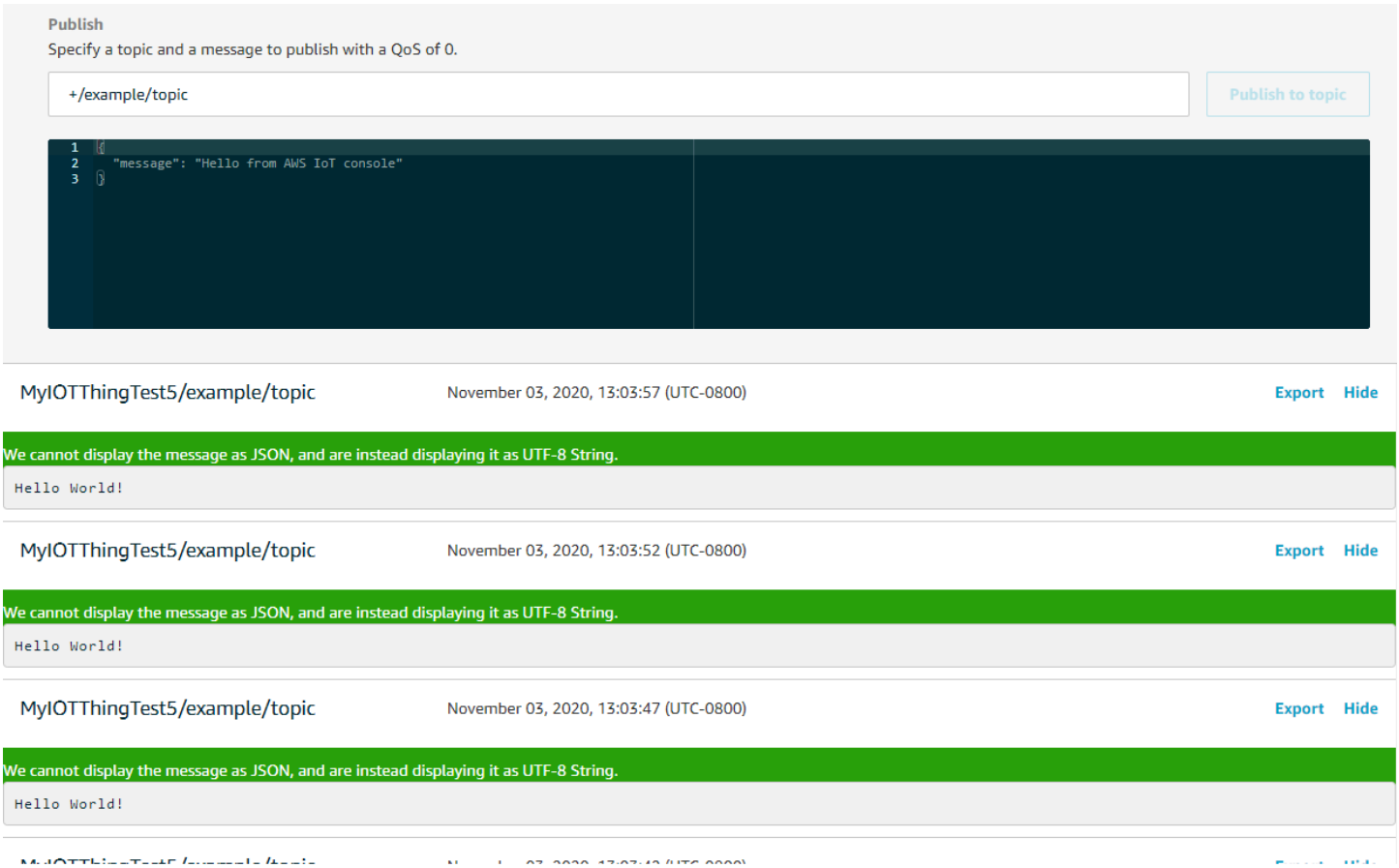
Funktionalität

Die Demo erstellt eine einzelne Anwendungsaufgabe, die eine Reihe von Beispielen durchläuft, die zeigen, wie eine Verbindung zum Broker hergestellt, ein Thema auf dem Broker abonniert, ein Thema auf dem Broker veröffentlicht und schließlich die Verbindung zum Broker getrennt wird. Die Demo-Anwendung abonniert und veröffentlicht dasselbe Thema. Jedes Mal, wenn die Demo eine Nachricht an den MQTT-Broker veröffentlicht, sendet der Broker dieselbe Nachricht zurück an die Demo-Anwendung.

Bei erfolgreichem Abschluss der Demo wird eine Ausgabe generiert, die der folgenden Abbildung ähnelt.



Die AWS IoT Konsole generiert eine Ausgabe, die der folgenden Abbildung ähnelt.



Wiederholen Sie die Logik mit exponentiellem Backoff und Jitter

Die prvBackoffForWiederholungsfunktion zeigt, wie fehlgeschlagene Netzwerkoperationen mit dem Server, z. B. TLS-Verbindungen oder MQTT-Abonnementanfragen, mit exponentiellem Backoff und Jitter wiederholt werden können. Die Funktion berechnet die Backoff-Periode für den nächsten Wiederholungsversuch und führt die Backoff-Verzögerung durch, wenn die Wiederholungsversuche noch nicht ausgeschöpft sind. Da die Berechnung der Backoff-Periode die Generierung einer Zufallszahl erfordert, verwendet die Funktion das Modul, um die Zufallszahl zu generieren. PKCS11 Die Verwendung des PKCS11 Moduls ermöglicht den Zugriff auf einen True Random Number Generator (TRNG), sofern die Herstellerplattform dies unterstützt. Wir empfehlen, den Zufallszahlengenerator mit einer gerätespezifischen Entropiequelle auszustatten, um die Wahrscheinlichkeit von Kollisionen durch Geräte bei erneuten Verbindungsversuchen zu verringern.

Verbindung zum MQTT-Broker herstellen

Die prvConnectToServerWithBackoffRetriesFunktion versucht, eine gegenseitig authentifizierte TLS-Verbindung zum MQTT-Broker herzustellen. Wenn die Verbindung fehlschlägt, versucht sie es nach einer Backoff-Periode erneut. Die Backoff-Periode wird exponentiell erhöht, bis

die maximale Anzahl von Versuchen oder die maximale Backoff-Periode erreicht ist. Die `BackoffAlgorithm_GetNextBackoff` Funktion liefert einen exponentiell steigenden Backoff-Wert und kehrt zurück, `RetryUtilsRetriesExhausted` wenn die maximale Anzahl von Versuchen erreicht wurde. Die `prvConnectToServerWithBackoffRetries` Funktion gibt einen Fehlerstatus zurück, wenn die TLS-Verbindung zum Broker nach der konfigurierten Anzahl von Versuchen nicht hergestellt werden kann.

Die MQTTConnection WithBroker Funktion PrvCreate demonstriert, wie eine MQTT-Verbindung zu einem MQTT-Broker mit einer sauberen Sitzung hergestellt wird. Sie verwendet die TLS-Transportschnittstelle, die in der Datei implementiert ist. `FreeRTOS-Plus/Source/Application-Protocols/platform/freertos/transport/src/tls_freertos.c` Denken Sie daran, dass wir die Keep-Alive-Sekunden für den Broker festlegen. `xConnectInfo`

Die nächste Funktion zeigt, wie die TLS-Transportschnittstelle und die Zeitfunktion mithilfe der Funktion in einem MQTT-Kontext eingestellt werden. `MQTT_Init` Sie zeigt auch, wie eine Event-Callback-Funktion pointer (`prvEventCallback`) gesetzt wird. Dieser Callback wird für die Meldung eingehender Nachrichten verwendet.

Ein MQTT-Thema abonnieren

Die MQTTSubscribe WithBackoffRetries Funktion prv demonstriert, wie man einen Themenfilter auf dem MQTT-Broker abonniert. Das Beispiel zeigt, wie ein Themenfilter abonniert wird, aber es ist möglich, eine Liste von Themenfiltern in demselben Abonnement-API-Aufruf zu übergeben, um mehr als einen Themenfilter zu abonnieren. Falls der MQTT-Broker die Abonnementanfrage ablehnt, versucht das Abonnement außerdem erneut, mit exponentiellem Backoff, für. `RETRY_MAX_ATTEMPTS`

So veröffentlichen Sie in einem Thema

Die MQTTPublishToTopicprv-Funktion demonstriert, wie Sie zu einem Thema auf dem MQTT-Broker veröffentlichen.

Empfangen eingehender Nachrichten

Die Anwendung registriert eine Rückruffunktion für Ereignisse, bevor sie eine Verbindung zum Broker herstellt, wie zuvor beschrieben. Die `prvMQTTDemoTask` Funktion ruft die `MQTT_ProcessLoop` Funktion auf, um eingehende Nachrichten zu empfangen. Wenn eine eingehende MQTT-Nachricht empfangen wird, ruft sie die von der Anwendung registrierte Event-Callback-Funktion auf. Die prvEventCallbackFunktion ist ein Beispiel für eine solche Event-

Callback-Funktion. `prvEventCallback`untersucht den Typ des eingehenden Pakets und ruft den entsprechenden Handler auf. Im folgenden Beispiel ruft die Funktion entweder die Verarbeitung `prvMQTTProcessIncomingPublish()` eingehender Veröffentlichungsnachrichten oder die Verarbeitung von Bestätigungen (ACK) `prvMQTTProcessResponse()` auf.

Verarbeitung eingehender MQTT-Veröffentlichungspakete

Die MQTTProcess IncomingPublish Funktion prv demonstriert, wie ein vom MQTT-Broker veröffentlichtes Paket verarbeitet wird.

Abmeldung von einem Thema

Der letzte Schritt im Workflow besteht darin, das Abonnement des Themas zu kündigen, sodass der Broker keine veröffentlichten Nachrichten von sendet. `mqttexampleTOPIC` Hier ist die Definition der Funktion prv MQTTUnsubscribe FromTopic.

Änderung der in der Demo verwendeten Root-CA

Standardmäßig verwenden die FreeRTOS-Demos das Amazon Root CA 1-Zertifikat (RSA 2048-Bit-Schlüssel), um sich beim Server zu authentifizieren. AWS IoT Core Es ist möglich, andere CA-Zertifikate für die Serverauthentifizierung zu verwenden, einschließlich des Amazon Root CA 3-Zertifikats (ECC 256-Bit-Schlüssel). Um die Root-CA für die Demo zur gegenseitigen CoreMQTT-Authentifizierung zu ändern:

1. Öffnen Sie die Datei *`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/mqtt_demo_mutual_auth_config.h` in einem Texteditor.
2. Suchen Sie in der Datei nach der folgenden Zeile.

   ```
   * #define democonfigROOT_CA_PEM    "...insert here..."
   ```

   Entfernen Sie den Kommentar zu dieser Zeile und verschieben Sie sie gegebenenfalls über das Ende `*/` des Kommentarblocks hinaus.
3. Kopieren Sie das CA-Zertifikat, das Sie verwenden möchten, und fügen Sie es dann in den `"...insert here..."` Text ein. Das Ergebnis sollte wie folgt aussehen:

   ```
   #define democonfigROOT_CA_PEM   "-----BEGIN CERTIFICATE-----\n"\
   "MIIBtjCCAVugAwIBAgITBmyf1XSXNmY/Owua2eiedgPySjAKBggqhkjOPQQDAjA5\n"\
   "MQswCQYDVQQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwFwYDVQQDExBBbWF6b24g\n"\
   "Um9vdCBDQSAzMB4XDTE1MDUyNjAwMDAwMFoXDTQwMDUyNjAwMDAwMFowOTELMAkG\n"\
   ```

```
"A1UEBhMCVVMxDzANBgNVBAoTBkFtYXpvbjEZMBcGA1UEAxMQQW1hem9uIFJvb3Qg\n"\
"Q0EgMzBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABCmXp8ZBf8ANm+gBG1bG8lKl\n"\
"ui2yEujSLtf6ycXYqm0fc4E7O5hrOXwzpcVOho6AF2hiRVd9RFgdszflZwjrZt6j\n"\
"QjBAMA8GA1UdEwEB/wQFMAMBAf8wDgYDVR0PAQH/BAQDAgGGMB0GA1UdDgQWBBSr\n"\
"ttvXBp43rDCGB5Fwx5zEGbF4wDAKBggqhkjOPQQDAgNJADBGAiEA4IWSoxe3jfkr\n"\
"BqWTrBqYaGFy+uGh0PsceGCmQ5nFuMQCIQCcAu/xlJyzlvnrxir4tiz+OpAUFteM\n"\
"YyRIHN8wfdVoOw==\n"\
"-----END CERTIFICATE-----\n"
```

4. (Optional) Sie können die Stammzertifizierungsstelle für andere Demos ändern. Wiederholen Sie die Schritte 1 bis 3 für jede *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/*demo-name*_config.h Datei.

Demo zur gemeinsamen Nutzung von CoreMQTT-Agenten

> **Important**
>
> Diese Demo wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Einführung

Das Demo-Projekt CoreMQTT Connection Sharing zeigt Ihnen, wie Sie mit einer Multithread-Anwendung eine Verbindung zum AWS MQTT-Broker mithilfe von TLS mit gegenseitiger Authentifizierung zwischen dem Client und dem Server herstellen können. Diese Demo verwendet eine MbedTLS-basierte Transportschnittstellenimplementierung, um eine server- und clientauthentifizierte TLS-Verbindung herzustellen, und demonstriert den Subscribe-Publish-Workflow von MQTT auf QoS 1-Ebene. Die Demo abonniert einen Themenfilter, veröffentlicht Themen, die dem Filter entsprechen, und wartet dann auf den Empfang dieser Nachrichten vom Server auf QoS 1-Ebene. Dieser Zyklus, bei dem die Veröffentlichung auf dem Broker und die gleiche Nachricht vom Broker zurückgesendet wird, wird für jede erstellte Aufgabe mehrmals wiederholt. Nachrichten in dieser Demo werden mit QoS 1 gesendet, was mindestens eine Lieferung gemäß der MQTT-Spezifikation garantiert.

> **Note**
>
> Folgen Sie den Schritten unter, um die FreeRTOS-Demos einzurichten und auszuführen. Erste Schritte mit FreeRTOS

Diese Demo verwendet eine Thread-sichere Warteschlange, in der Befehle für die Interaktion mit der MQTT-API gespeichert werden. In dieser Demo gibt es zwei Aufgaben, die Sie beachten sollten.

- Eine (Haupt-) Aufgabe des MQTT-Agenten verarbeitet die Befehle aus der Befehlswarteschlange, während andere Aufgaben sie in die Warteschlange einreihen. Diese Aufgabe tritt in eine Schleife ein, in der sie Befehle aus der Befehlswarteschlange verarbeitet. Wenn ein Terminierungsbefehl eingeht, wird diese Aufgabe aus der Schleife ausbrechen.
- Eine Demo-Subpub-Aufgabe erstellt ein Abonnement für ein MQTT-Thema, erstellt dann Veröffentlichungsvorgänge und schiebt sie in die Befehlswarteschlange. Diese Veröffentlichungsvorgänge werden dann von der MQTT-Agent-Aufgabe ausgeführt. Die Demo-Subpub-Aufgabe wartet, bis die Veröffentlichung abgeschlossen ist, was durch die Ausführung des Callbacks zur Befehlsvervollständigung angezeigt wird, und gibt dann eine kurze Verzögerung ein, bevor sie mit der nächsten Veröffentlichung beginnt. Diese Aufgabe zeigt Beispiele dafür, wie Anwendungsaufgaben die CoreMQTT-Agenten-API verwenden würden.

Für eingehende Veröffentlichungsnachrichten ruft der CoreMQTT-Agent eine einzige Callback-Funktion auf. Diese Demo beinhaltet auch einen Abonnement-Manager, der es Aufgaben ermöglicht, einen Callback anzugeben, der für eingehende Veröffentlichungsnachrichten zu Themen aufgerufen werden soll, die sie abonniert haben. Der eingehende Veröffentlichungsrückruf des Agenten in dieser Demo ruft den Abonnementmanager auf, um Veröffentlichungen für alle Aufgaben, für die ein Abonnement registriert wurde, aufzufächern.

In dieser Demo wird eine TLS-Verbindung mit gegenseitiger Authentifizierung für die Verbindung verwendet. AWS Wenn das Netzwerk während der Demo unerwartet unterbrochen wird, versucht der Client, mithilfe der exponentiellen Backoff-Logik erneut eine Verbindung herzustellen. Wenn der Client die Verbindung erfolgreich wiederherstellt, der Broker die vorherige Sitzung jedoch nicht fortsetzen kann, abonniert der Client dieselben Themen wie in der vorherigen Sitzung erneut.

Single-Threading oder Multithreading-fähig

Es gibt zwei CoreMQTT-Nutzungsmodelle: Singlethread und Multithread (Multitasking). Das Single-Thread-Modell verwendet die CoreMQTT-Bibliothek ausschließlich von einem Thread aus

und erfordert, dass Sie wiederholte explizite Aufrufe in der MQTT-Bibliothek tätigen. Multithread-Anwendungsfälle können stattdessen das MQTT-Protokoll im Hintergrund innerhalb einer Agenten- (oder Daemon-) Aufgabe ausführen, wie in der hier dokumentierten Demo gezeigt. Wenn Sie das MQTT-Protokoll in einer Agententask ausführen, müssen Sie keinen MQTT-Status explizit verwalten oder die API-Funktion aufrufen. `MQTT_ProcessLoop` Wenn Sie eine Agententask verwenden, können sich außerdem mehrere Anwendungsaufgaben eine einzige MQTT-Verbindung teilen, ohne dass Synchronisationsprimitive wie Mutexe erforderlich sind.

Quellcode

Die Demo-Quelldateien sind benannt `mqtt_agent_task.c` `simple_sub_pub_demo.c` und befinden sich im *`freertos`*`/demos/coreMQTT_Agent/` Verzeichnis und auf der Website. GitHub

Funktionalität

Diese Demo erstellt mindestens zwei Aufgaben: eine primäre, die Anfragen für MQTT-API-Aufrufe verarbeitet, und eine konfigurierbare Anzahl von Unteraufgaben, die diese Anfragen erstellen. In dieser Demo erstellt die Hauptaufgabe die Unteraufgaben, ruft die Verarbeitungsschleife auf und bereinigt sie anschließend. Die Hauptaufgabe erstellt eine einzelne MQTT-Verbindung zum Broker, die von den Unteraufgaben gemeinsam genutzt wird. Die Unteraufgaben erstellen ein MQTT-Abonnement beim Broker und veröffentlichen dann Nachrichten an diesen. Jede Unteraufgabe verwendet ein eigenes Thema für ihre Veröffentlichungen.

Hauptaufgabe

Die Hauptaufgabe der Anwendung, RunCoreMQTTAgentDemo, richtet eine MQTT-Sitzung ein, erstellt die Unteraufgaben und führt die Verarbeitungsschleife MQTTAgent_ aus, CommandLoop bis ein Abschlussbefehl empfangen wird. Wenn das Netzwerk unerwartet unterbrochen wird, stellt die Demo im Hintergrund wieder eine Verbindung zum Broker her und stellt die Abonnements beim Broker wieder her. Nachdem die Verarbeitungsschleife beendet wurde, wird die Verbindung zum Broker getrennt.

Befehle

Wenn Sie eine CoreMQTT-Agenten-API aufrufen, erstellt sie einen Befehl, der an die Warteschlange der Agentenaufgabe gesendet wird, in der sie verarbeitet wird. `MQTTAgent_CommandLoop()` Zum Zeitpunkt der Erstellung des Befehls können optionale Callback- und Kontextparameter für die Fertigstellung übergeben werden. Sobald der entsprechende Befehl abgeschlossen ist, wird der Abschluss-Callback mit dem übergebenen Kontext und allen Rückgabewerten, die als Ergebnis des Befehls erstellt wurden, aufgerufen. Die Signatur für den Abschluss-Callback lautet wie folgt:

```
typedef void (* MQTTAgentCommandCallback_t )( void * pCmdCallbackContext,
                                              MQTTAgentReturnInfo_t * pReturnInfo );
```

Der Kontext der Befehlsvervollständigung ist benutzerdefiniert; für diese Demo lautet er: struct. MQTTAgent CommandContext

Befehle gelten als abgeschlossen, wenn:

- Abonniert, kündigt und veröffentlicht mit QoS > 0: Sobald das entsprechende Bestätigungspaket empfangen wurde.
- Alle anderen Operationen: Sobald die entsprechende CoreMQTT-API aufgerufen wurde.

Alle vom Befehl verwendeten Strukturen, einschließlich Veröffentlichungsinformationen, Abonnementinformationen und Abschlusskontexten, müssen im Gültigkeitsbereich bleiben, bis der Befehl abgeschlossen ist. Eine aufrufende Aufgabe darf keine der Strukturen eines Befehls wiederverwenden, bevor der Abschluss-Callback aufgerufen wurde. Beachten Sie, dass der Completion-Callback, da er vom MQTT-Agenten aufgerufen wird, im Thread-Kontext der Agentenaufgabe ausgeführt wird, nicht mit der Aufgabe, die den Befehl erstellt hat. Prozessübergreifende Kommunikationsmechanismen, wie Aufgabenbenachrichtigungen oder Warteschlangen, können verwendet werden, um der aufrufenden Aufgabe zu signalisieren, dass der Befehl abgeschlossen ist.

Die Befehlsschleife ausführen

Befehle werden kontinuierlich in verarbeitet`MQTTAgent_CommandLoop()`. Wenn es keine Befehle gibt, die verarbeitet werden müssen, wartet die Schleife, bis maximal einer `MQTT_AGENT_MAX_EVENT_QUEUE_WAIT_TIME` zur Warteschlange hinzugefügt wird, und wenn kein Befehl hinzugefügt wird, wird eine einzelne Iteration von `MQTT_ProcessLoop()` ausgeführt. Dadurch wird sichergestellt, dass MQTT Keep-Alive verwaltet wird und dass alle eingehenden Veröffentlichungen auch dann empfangen werden, wenn sich keine Befehle in der Warteschlange befinden.

Die Befehlsschleifenfunktion wird aus den folgenden Gründen zurückkehren:

- Ein Befehl gibt außerdem einen beliebigen Statuscode zurück`MQTTSuccess`. Der Fehlerstatus wird von der Befehlsschleife zurückgegeben, sodass Sie entscheiden können, wie Sie damit umgehen möchten. In dieser Demo wird die TCP-Verbindung wieder hergestellt und es wird versucht,

die Verbindung wiederherzustellen. Wenn ein Fehler auftritt, kann eine Wiederverbindung im Hintergrund erfolgen, ohne dass andere Aufgaben, die MQTT verwenden, eingreifen müssen.

- Ein Befehl zum Trennen der Verbindung (von`MQTTAgent_Disconnect`) wird verarbeitet. Die Befehlsschleife wird beendet, sodass die TCP-Verbindung getrennt werden kann.
- Ein Terminierungsbefehl (von`MQTTAgent_Terminate`) wird verarbeitet. Dieser Befehl markiert außerdem jeden Befehl, der sich noch in der Warteschlange befindet oder auf ein Bestätigungspaket wartet, als Fehler mit dem Rückgabecode. `MQTTRecvFailed`

Abonnement-Manager

Da die Demo mehrere Themen verwendet, ist ein Abonnement-Manager eine bequeme Möglichkeit, abonnierte Themen mit eindeutigen Rückrufen oder Aufgaben zu verknüpfen. Der Abonnementmanager in dieser Demo ist Single-Thread-fähig und sollte daher nicht von mehreren Aufgaben gleichzeitig verwendet werden. In dieser Demo werden Abonnement-Manager-Funktionen nur von Callback-Funktionen aufgerufen, die an den MQTT-Agenten übergeben werden, und nur im Thread-Kontext der Agentenaufgabe ausgeführt.

Einfache Aufgabe zum Abonnieren und Veröffentlichen

Jede Instanz von `prvSimpleSubscribePublishTask`erstellt ein Abonnement für ein MQTT-Thema und erstellt Veröffentlichungsvorgänge für dieses Thema. Um mehrere Veröffentlichungstypen zu demonstrieren, verwenden gerade nummerierte Aufgaben QoS 0 (die abgeschlossen sind, sobald das Veröffentlichungspaket gesendet wurde) und ungerade Aufgaben verwenden QoS 1 (die nach Erhalt eines PUBACK-Pakets abgeschlossen sind).

Over-the-air aktualisiert die Demo-Anwendung

FreeRTOS enthält eine Demo-Anwendung, die die Funktionalität der over-the-air (OTA) - Bibliothek demonstriert. Die OTA-Demoanwendung befindet sich in der *`freertos`*`/demos/ota/ota_demo_core_mqtt/ota_demo_core_mqtt.c` oder der *`freertos`*`/demos/ota/ota_demo_core_http/ota_demo_core_http.c` Datei.

Die OTA-Demo-Anwendung führt folgende Aktionen aus:

1. Initialisiert den FreeRTOS-Netzwerk-Stacks und den MQTT-Buffer-Pool.
2. Erstellt eine Aufgabe, mit der die OTA-Bibliothek ausgeführt werden kann`vRunOTAUpdateDemo()`.
3. Erstellt einen MQTT-Client unter Verwendung von `_establishMqttConnection()`.

4. Stellt eine Verbindung zum AWS IoT MQTT-Broker her `IotMqtt_Connect()` und registriert einen MQTT-Disconnect-Callback:. `prvNetworkDisconnectCallback`
5. Ruft `OTA_AgentInit()` zum Erstellen der OTA-Task auf und registriert einen Rückruf, der nach Abschluss der OTA-Task verwendet werden soll.
6. Verwendet die MQTT-Verbindung erneut mit `xOTAConnectionCtx.pvControlClient = _mqttConnection;`
7. Wenn MQTT die Verbindung trennt, unterbricht die Anwendung den OTA-Agenten, versucht, die Verbindung mithilfe einer exponentiellen Verzögerung mit Jitter wiederherzustellen, und nimmt dann den OTA-Agenten wieder auf.

Bevor Sie OTA-Updates verwenden können, müssen Sie alle Voraussetzungen in erfüllen FreeRTOS RTOS-Updates Over-the-Air

Nachdem Sie die Einrichtung für OTA-Updates abgeschlossen haben, laden Sie die FreeRTOS OTA-Demo herunter, erstellen, flashen und führen Sie sie auf einer Plattform aus, die OTA-Funktionen unterstützt. Gerätespezifische Demo-Anweisungen sind für die folgenden FreeRTOS-qualifizierten Geräte verfügbar:

- Texas Instruments SF-LAUNCHXL CC3220
- Microchip Curiosity MZEF PIC32
- Espressif ESP32
- Laden Sie die FreeRTOS OTA-Demo auf dem Renesas N herunter, erstellen Sie sie, flashen Sie sie und führen Sie sie aus RX65

Nachdem Sie die OTA-Demoanwendung auf Ihrem Gerät erstellt, flasht und ausgeführt haben, können Sie die AWS IoT Konsole oder die verwenden, um einen OTA-Aktualisierungsauftrag AWS CLI zu erstellen. Nachdem Sie einen OTA-Aktualisierungsauftrag erstellt haben, verbinden Sie einen Terminal-Emulator, um den Fortschritt des OTA-Updates zu sehen. Notieren Sie alle Fehler, die während des Prozesses auftreten.

Ein erfolgreicher OTA-Aktualisierungsauftrag zeigt eine Ausgabe ähnlich der Folgenden an. Einige Zeilen in diesem Beispiel wurden aus Gründen der Übersichtlichkeit aus der Liste entfernt.

```
   249 21207 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT]  Received: 0
Queued: 0   Processed: 0   Dropped: 0
   250 21247 [MQTT Agent Task] [core_mqtt.c:886] [INFO] [MQTT] Packet received.
ReceivedBytes=601.
```

```
    251 21247 [MQTT Agent Task] [core_mqtt.c:1045] [INFO] [MQTT] De-serialized incoming
 PUBLISH packet: DeserializerResult=MQTTSuccess.
    252 21248 [MQTT Agent Task] [core_mqtt.c:1058] [INFO] [MQTT] State record updated.
 New state=MQTTPubAckSend.
    253 21249 [MQTT Agent Task] [ota_demo_core_mqtt.c:976] [INFO] [MQTT] Received job
 message callback, size 548.
    254 21252 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[execution.jobId: AFR_OTA-9702f1a3-b747-4c3e-a0eb-a3b0cf83ddbb]
    255 21253 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[execution.jobDocument.afr_ota.streamname: AFR_OTA-945d320b-a18b-441b-
b435-4a18d4e7671f]
    256 21255 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[execution.jobDocument.afr_ota.protocols: ["MQTT"]]
    257 21256 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[filepath: aws_demos.bin]
    258 21257 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
 value]=[filesize: 1164016]
    259 21258 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
 value]=[fileid: 0]
    260 21259 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[certfile: ecdsa-sha256-signer.crt.pem]
    261 21260 [OTA Agent Task] [ota.c:1575] [INFO] [OTA] Extracted parameter [ sig-
sha256-ecdsa: MEQCIE1SFkIHHiZAvkPpu6McJtx7SYoD... ]
    262 21261 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
 value]=[fileType: 0]
    263 21262 [OTA Agent Task] [ota.c:2199] [INFO] [OTA] Job document was accepted.
 Attempting to begin the update.
    264 21263 [OTA Agent Task] [ota.c:2323] [INFO] [OTA] Job parsing success:
 OtaJobParseErr_t=OtaJobParseErrNone, Job name=AFR_OTA-9702f1a3-b747-4c3e-a0eb-
a3b0cf83ddbb
    265 21318 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT]  Received: 0
 Queued: 0   Processed: 0   Dropped: 0
    266 21418 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT]  Received: 0
 Queued: 0   Processed: 0   Dropped: 0
    267 21469 [OTA Agent Task] [ota.c:938] [INFO] [OTA] Setting OTA data interface.
    268 21470 [OTA Agent Task] [ota.c:2839] [INFO] [OTA] Current State=[CreatingFile],
 Event=[ReceivedJobDocument], New state=[CreatingFile]
    269 21482 [MQTT Agent Task] [core_mqtt.c:886] [INFO] [MQTT] Packet received.
 ReceivedBytes=3.
    270 21483 [OTA Agent Task] [ota_demo_core_mqtt.c:1503] [INFO] [MQTT] SUBSCRIBED
 to topic $aws/things/__test_infra_thing71/streams/AFR_OTA-945d320b-a18b-441b-
b435-4a18d4e7671f/data/cbor to bro
    271 21484 [OTA Agent Task] [ota.c:2839] [INFO] [OTA] Current
 State=[RequestingFileBlock], Event=[CreateFile], New state=[RequestingFileBlock]
```

```
    272 21518 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT]  Received: 0
 Queued: 0   Processed: 0   Dropped: 0
    273 21532 [MQTT Agent Task] [core_mqtt_agent_command_functions.c:76] [INFO] [MQTT]
 Publishing message to $aws/things/__test_infra_thing71/streams/AFR_OTA-945d320b-
a18b-441b-b435-4a18d4e7671f/
    274 21534 [OTA Agent Task] [ota_demo_core_mqtt.c:1553] [INFO] [MQTT] Sent PUBLISH
 packet to broker $aws/things/__test_infra_thing71/streams/AFR_OTA-945d320b-a18b-441b-
b435-4a18d4e7671f/get/cbor
    275 21534 [OTA Agent Task] [ota_mqtt.c:1112] [INFO] [OTA] Published to MQTT
 topic to request the next block: topic=$aws/things/__test_infra_thing71/streams/
AFR_OTA-945d320b-a18b-441b-b435-4a1
    276 21537 [OTA Agent Task] [ota.c:2839] [INFO] [OTA] Current
 State=[WaitingForFileBlock], Event=[RequestFileBlock], New state=[WaitingForFileBlock]
    277 21558 [MQTT Agent Task] [core_mqtt.c:886] [INFO] [MQTT] Packet received.
 ReceivedBytes=4217.
    278 21559 [MQTT Agent Task] [core_mqtt.c:1045] [INFO] [MQTT] De-serialized incoming
 PUBLISH packet: DeserializerResult=MQTTSuccess.
    279 21560 [MQTT Agent Task] [core_mqtt.c:1058] [INFO] [MQTT] State record updated.
 New state=MQTTPublishDone.
    280 21561 [MQTT Agent Task] [ota_demo_core_mqtt.c:1026] [INFO] [MQTT] Received data
 message callback, size 4120.
    281 21563 [OTA Agent Task] [ota.c:2464] [INFO] [OTA] Received valid file block:
 Block index=0, Size=4096
    282 21566 [OTA Agent Task] [ota.c:2683] [INFO] [OTA] Number of blocks remaining:
 284

    ... // Output removed for brevity

    3672 42745 [OTA Agent Task] [ota.c:2464] [INFO] [OTA] Received valid file block:
 Block index=284, Size=752
    3673 42747 [OTA Agent Task] [ota.c:2633] [INFO] [OTA] Received final block of the
 update.
    (428298) ota_pal: No such certificate file: ecdsa-sha256-signer.crt.pem. Using
 certificate in ota_demo_config.h.
    3674 42818 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT]  Received: 285
 Queued: 285   Processed: 284   Dropped: 0
    3675 42918 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT]  Received: 285
 Queued: 285   Processed: 284   Dropped: 0

    ... // Output removed for brevity

    3678 43197 [OTA Agent Task] [ota.c:2654] [INFO] [OTA] Received entire update and
 validated the signature.
```

```
    3685 43215 [OTA Agent Task] [ota_demo_core_mqtt.c:862] [INFO] [MQTT] Received
 OtaJobEventActivate callback from OTA Agent.

    ... // Output removed for brevity

    2 39 [iot_thread] [INFO ][DEMO][390] ---------STARTING DEMO---------

    [0;32mI (3633) WIFI: WIFI_EVENT_STA_CONNECTED
    [0;32mI (4373) WIFI: SYSTEM_EVENT_STA_GOT_IP

    ... // Output removed for brevity

    4 351 [sys_evt] [INFO ][DEMO][3510] Connected to WiFi access point, ip address:
 255.255.255.0.
    5 351 [iot_thread] [INFO ][DEMO][3510] Successfully initialized the demo. Network
 type for the demo: 1
    6 351 [iot_thread] [ota_demo_core_mqtt.c:1902] [INFO] [MQTT] OTA over MQTT demo,
 Application version 0.9.1
    7 351 [iot_thread] [ota_demo_core_mqtt.c:1323] [INFO] [MQTT] Creating a TLS
 connection to <endpoint>-ats.iot.us-west-2.amazonaws.com:8883.
    9 718 [iot_thread] [core_mqtt.c:886] [INFO] [MQTT] Packet received.
 ReceivedBytes=2.
    10 718 [iot_thread] [core_mqtt_serializer.c:970] [INFO] [MQTT] CONNACK session
 present bit not set.
    11 718 [iot_thread] [core_mqtt_serializer.c:912] [INFO] [MQTT] Connection accepted.

    ... // Output removed for brevity

    17 736 [OTA Agent Task] [ota_demo_core_mqtt.c:1503] [INFO] [MQTT] SUBSCRIBED to
 topic $aws/things/__test_infra_thing71/jobs/notify-next to broker.
    18 737 [OTA Agent Task] [ota_mqtt.c:381] [INFO] [OTA] Subscribed to MQTT topic:
 $aws/things/__test_infra_thing71/jobs/notify-next
    30 818 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT]  Received: 0
 Queued: 0   Processed: 0   Dropped: 0
    31 819 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[execution.jobId: AFR_OTA-9702f1a3-b747-4c3e-a0eb-a3b0cf83ddbb]
    32 820 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
 value]=[execution.statusDetails.updatedBy: 589824]
    33 822 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[execution.jobDocument.afr_ota.streamname: AFR_OTA-945d320b-a18b-441b-
b435-4a18d4e7671f]
    34 823 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[execution.jobDocument.afr_ota.protocols: ["MQTT"]]
```

```
    35 824 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[filepath: aws_demos.bin]
    36 825 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
 value]=[filesize: 1164016]
    37 826 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
 value]=[fileid: 0]
    38 827 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[certfile: ecdsa-sha256-signer.crt.pem]
    39 828 [OTA Agent Task] [ota.c:1575] [INFO] [OTA] Extracted parameter [ sig-sha256-
ecdsa: MEQCIE1SFkIHHiZAvkPpu6McJtx7SYoD... ]
    40 829 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
 value]=[fileType: 0]
    41 830 [OTA Agent Task] [ota.c:2102] [INFO] [OTA] In self test mode.
    42 830 [OTA Agent Task] [ota.c:1936] [INFO] [OTA] New image has a higher version
 number than the current image: New image version=0.9.1, Previous image version=0.9.0
    43 832 [OTA Agent Task] [ota.c:2120] [INFO] [OTA] Image version is valid: Begin
 testing file: File ID=0
    53 896 [OTA Agent Task] [ota.c:794] [INFO] [OTA] Beginning self-test.
    62 971 [OTA Agent Task] [ota_demo_core_mqtt.c:1553] [INFO] [MQTT] Sent PUBLISH
 packet to broker $aws/things/__test_infra_thing71/jobs/AFR_OTA-9702f1a3-b747-4c3e-
a0eb-a3b0cf83ddbb/update to br63 971 [MQTT Agent Task] [core_mqtt.c:1045] [INFO] [MQTT]
 De-serialized incoming PUBLISH packet: DeserializerResult=MQTTSuccess.
    65 973 [MQTT Agent Task] [core_mqtt.c:1058] [INFO] [MQTT] State record updated. New
 state=MQTTPublishDone.
    64 973 [OTA Agent Task] [ota_demo_core_mqtt.c:902] [INFO] [MQTT] Successfully
 updated with the new image.
```

Over-the-air Demo-Konfigurationen

Bei den OTA-Demokonfigurationen handelt es sich um demospezifische Konfigurationsoptionen, die unter bereitgestellt werden. `aws_iot_ota_update_demo.c` Diese Konfigurationen unterscheiden sich von den OTA-Bibliothekskonfigurationen, die in der Konfigurationsdatei der OTA-Bibliothek bereitgestellt werden.

OTA_DEMO_KEEP_ALIVE_SECONDS

> Für den MQTT-Client ist diese Konfiguration das maximale Zeitintervall, das zwischen dem Abschluss der Übertragung eines Steuerpakets und dem Beginn des Sendens des nächsten vergehen kann. In Ermangelung eines Kontrollpakets wird ein PINGREQ gesendet. Der Broker muss die Verbindung zu einem Client, der keine Nachricht oder kein PINGREQ-Paket sendet, innerhalb von eineinhalb Mal dieses Keep-Alive-Intervalls trennen. Diese Konfiguration sollte an die Anforderungen der Anwendung angepasst werden.

OTA_DEMO_CONN_RETRY_BASE_INTERVAL_SECONDS

Das Basisintervall in Sekunden, bevor erneut versucht wird, die Netzwerkverbindung herzustellen. Die OTA-Demo versucht, nach diesem Basiszeitintervall erneut eine Verbindung herzustellen. Das Intervall wird nach jedem fehlgeschlagenen Versuch verdoppelt. Eine zufällige Verzögerung, bis zu einem Maximum dieser Basisverzögerung, wird ebenfalls zum Intervall hinzugefügt.

OTA_DEMO_CONN_RETRY_MAX_INTERVAL_SECONDS

Das maximale Intervall in Sekunden, bevor erneut versucht wird, die Netzwerkverbindung herzustellen. Die Wiederverbindungsverzögerung wird bei jedem fehlgeschlagenen Versuch verdoppelt, sie kann jedoch nur bis zu diesem Höchstwert plus einem Jitter im gleichen Intervall ansteigen.

Laden Sie die FreeRTOS OTA-Demo auf dem Texas Instruments SF-LAUNCHXL herunter, erstellen Sie sie, flashen Sie sie und führen Sie sie aus CC3220

> **Important**
>
> Diese Referenzintegration wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Um FreeRTOS und den OTA-Democode herunterzuladen

- Sie können den Quellcode auf der GitHub Website unter https://github.com/FreeRTOS/FreeRTOS herunterladen.

So erstellen Sie die Demo-Anwendung:

1. Folgen Sie den Anweisungen unterErste Schritte mit FreeRTOS, um das `aws_demos` Projekt in Code Composer Studio zu importieren, Ihren AWS IoT Endpunkt, Ihre Wi-Fi-SSID und Ihr Passwort sowie einen privaten Schlüssel und ein Zertifikat für Ihr Board zu konfigurieren.
2. Öffne*`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/aws_demo_config.h`, kommentiere und definiere `CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED`

   oder`CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`. `#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED`

3. Erstellen Sie die Lösung. Prüfen Sie, ob die Erstellung fehlerfrei ausgeführt wurde.
4. Starten Sie einen Terminalemulator und verwenden Sie die folgenden Einstellungen, um sich mit Ihrem Board zu verbinden:

   - Baudrate: 115200
   - Datenbits: 8
   - Parität: Keine
   - Stop-Bits: 1
5. Führen Sie das Projekt auf Ihrem Board aus, um zu überprüfen, ob es eine Verbindung mit dem WLAN und dem AWS IoT -MQTT-Nachrichten-Broker herstellen kann.

Bei der Ausführung sollte der Terminalemulator den folgenden Text anzeigen:

```
    0 1000 [Tmr Svc] Simple Link task created
    Device came up in Station mode
    1 2534 [Tmr Svc] Write certificate...
    2 5486 [Tmr Svc] [ERROR] Failed to destroy object. PKCS11_PAL_DestroyObject failed.
    3 5486 [Tmr Svc] Write certificate...
    4 5776 [Tmr Svc] Security alert threshold = 15
    5 5776 [Tmr Svc] Current number of alerts = 1
    6 5778 [Tmr Svc] Running Demos.
    7 5779 [iot_thread] [INFO ][DEMO][5779] ---------STARTING DEMO---------
    8 5779 [iot_thread] [INFO ][INIT][5779] SDK successfully initialized.
    Device came up in Station mode
    [WLAN EVENT] STA Connected to the AP: afrlab-pepper , BSSID: 74:83:c2:b4:46:27
    [NETAPP EVENT] IP acquired by the device
    Device has connected to afrlab-pepper
    Device IP Address is 192.168.36.176
    9 8283 [iot_thread] [INFO ][DEMO][8282] Successfully initialized the demo. Network
 type for the demo: 1
    10 8283 [iot_thread] [INFO] OTA over MQTT demo, Application version 0.9.0
    11 8283 [iot_thread] [INFO] Creating a TLS connection to <endpoint>-ats.iot.us-
west-2.amazonaws.com:8883.
    12 8852 [iot_thread] [INFO] Creating an MQTT connection to <endpoint>-ats.iot.us-
west-2.amazonaws.com.
    13 8914 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
    14 8914 [iot_thread] [INFO] CONNACK session present bit not set.
    15 8914 [iot_thread] [INFO] Connection accepted.
```

```
    16 8914 [iot_thread] [INFO] Received MQTT CONNACK successfully from broker.
    17 8914 [iot_thread] [INFO] MQTT connection established with the broker.
    18 8915 [iot_thread] [INFO]  Received: 0   Queued: 0   Processed: 0   Dropped: 0
    19 8953 [OTA Agent T] [INFO] Current State=[RequestingJob], Event=[Start], New
 state=[RequestingJob]
    20 9008 [MQTT Agent ] [INFO] Packet received. ReceivedBytes=3.
    21 9015 [OTA Agent T] [INFO] SUBSCRIBED to topic $aws/things/__test_infra_thing73/
jobs/notify-next to broker.
    22 9015 [OTA Agent T] [INFO] Subscribed to MQTT topic: $aws/things/
__test_infra_thing73/jobs/notify-next
    23 9504 [MQTT Agent ] [INFO] Publishing message to $aws/things/
__test_infra_thing73/jobs/$next/get.
    24 9535 [MQTT Agent ] [INFO] Packet received. ReceivedBytes=2.
    25 9535 [MQTT Agent ] [INFO] Ack packet deserialized with result: MQTTSuccess.
    26 9536 [MQTT Agent ] [INFO] State record updated. New state=MQTTPublishDone.
    27 9537 [OTA Agent T] [INFO] Sent PUBLISH packet to broker $aws/things/
__test_infra_thing73/jobs/$next/get to broker.
    28 9537 [OTA Agent T] [WARN] OTA Timer handle NULL for Timerid=0, can't stop.
    29 9537 [OTA Agent T] [INFO] Current State=[WaitingForJob],
 Event=[RequestJobDocument], New state=[WaitingForJob]
    30 9539 [MQTT Agent ] [INFO] Packet received. ReceivedBytes=120.
    31 9539 [MQTT Agent ] [INFO] De-serialized incoming PUBLISH packet:
 DeserializerResult=MQTTSuccess.
    32 9540 [MQTT Agent ] [INFO] State record updated. New state=MQTTPublishDone.
    33 9540 [MQTT Agent ] [INFO] Received job message callback, size 62.
    34 9616 [OTA Agent T] [INFO] Failed job document content check: Required job
 document parameter was not extracted: parameter=execution
    35 9616 [OTA Agent T] [INFO] Failed job document content check: Required job
 document parameter was not extracted: parameter=execution.jobId
    36 9617 [OTA Agent T] [INFO] Failed job document content check: Required job
 document parameter was not extracted: parameter=execution.jobDocument
    37 9617 [OTA Agent T] [INFO] Failed job document content check: Required job
 document parameter was not extracted: parameter=execution.jobDocument.afr_ota
    38 9617 [OTA Agent T] [INFO] Failed job document content
 check: Required job document parameter was not extracted:
 parameter=execution.jobDocument.afr_ota.protocols
    39 9618 [OTA Agent T] [INFO] Failed job document content check: Required job
 document parameter was not extracted: parameter=execution.jobDocument.afr_ota.files
    40 9618 [OTA Agent T] [INFO] Failed job document content check: Required job
 document parameter was not extracted: parameter=filesize
    41 9618 [OTA Agent T] [INFO] Failed job document content check: Required job
 document parameter was not extracted: parameter=fileid
    42 9619 [OTA Agent T] [INFO] Failed to parse JSON document as AFR_OTA job:
 DocParseErr_t=7
```

```
   43 9619 [OTA Agent T] [INFO] No active job available in received job document:
OtaJobParseErr_t=OtaJobParseErrNoActiveJobs
   44 9619 [OTA Agent T] [ERROR] Failed to execute state transition handler: Handler
returned error: OtaErr_t=OtaErrJobParserError
   45 9620 [OTA Agent T] [INFO] Current State=[WaitingForJob],
Event=[ReceivedJobDocument], New state=[CreatingFile]
   46 9915 [iot_thread] [INFO]  Received: 0   Queued: 0   Processed: 0   Dropped: 0
   47 10915 [iot_thread] [INFO]  Received: 0   Queued: 0   Processed: 0   Dropped: 0
   48 11915 [iot_thread] [INFO]  Received: 0   Queued: 0   Processed: 0   Dropped: 0
   49 12915 [iot_thread] [INFO]  Received: 0   Queued: 0   Processed: 0   Dropped: 0
   50 13915 [iot_thread] [INFO]  Received: 0   Queued: 0   Processed: 0   Dropped: 0
   51 14915 [iot_thread] [INFO]  Received: 0   Queued: 0   Processed: 0   Dropped: 0
```

Laden Sie die FreeRTOS OTA-Demo auf dem Microchip Curiosity MZEF herunter, erstellen Sie sie, flashen Sie sie und führen Sie sie aus PIC32

Important

Diese Referenzintegration wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Note

In Absprache mit Microchip entfernen wir den Curiosity PIC32 MZEF (DM320104) aus dem Hauptzweig des FreeRTOS Reference Integration-Repositorys und werden ihn nicht mehr in neuen Releases anbieten. Microchip hat eine offizielle Mitteilung veröffentlicht, dass der PIC32 MZEF (DM320104) nicht mehr für neue Designs empfohlen wird. Auf die PIC32 MZEF-Projekte und den Quellcode kann weiterhin über die Tags der vorherigen Version zugegriffen werden. Microchip empfiehlt Kunden, das Curiosity PIC32MZ-EF-2.0-Entwicklungsboard () für neue Designs zu verwenden. DM320209 Die PIC32 MZv1 Plattform befindet sich immer noch in Version 202012.00 des FreeRTOS Reference Integration-Repositorys. Die Plattform wird jedoch von v202107.00 der FreeRTOS Reference nicht mehr unterstützt.

Um den FreeRTOS OTA-Democode herunterzuladen

- Sie können den Quellcode auf der GitHub Website unter https://github.com/FreeRTOS/FreeRTOS herunterladen.

So erstellen Sie die OTA-Update Demo-Anwendung:

1. Folgen Sie den Anweisungen unterErste Schritte mit FreeRTOS, um das `aws_demos` Projekt in die MPLAB X IDE zu importieren, Ihren AWS IoT Endpunkt, Ihre Wi-Fi-SSID und Ihr Passwort sowie einen privaten Schlüssel und ein Zertifikat für Ihr Board zu konfigurieren.
2. Öffnen Sie die `vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/ota_demo_config.h` Datei und geben Sie Ihr Zertifikat ein.

   ```
   [] = "your-certificate-key";
   ```

3. Fügen Sie den Inhalt Ihres Codesignaturzertifikats hier ein:

   ```
   #define otapalconfigCODE_SIGNING_CERTIFICATE [] = "your-certificate-key";
   ```

   Verwenden Sie das gleiche Format wie `aws_clientcredential_keys.h` — jede Zeile muss mit dem neuen Zeilenzeichen ('\n') enden und in Anführungszeichen eingeschlossen sein.

   Ihr Zertifikat sollte beispielsweise so aussehen:

   ```
   "-----BEGIN CERTIFICATE-----\n"
   "MIIBXTCCAQOgAwIBAgIJAM4DeybZcTwKMAoGCCqGSM49BAMCMCExHzAdBgNVBAMM\n"
   "FnRlc3Rf62lnbmVyQGFtYXpvbi5jb20wHhcNMTcxMTAzMTkxODM1WhcNMTgxMTAz\n"
   "MTkxODM2WjAhMR8wHQYDVQBBZZZ0ZXN0X3NpZ25lckBhbWF6b24uY29tMFkwEwYH\n"
   "KoZIzj0CAQYIKoZIzj0DAQcDQgAERavZfvwL1X+E4dIF7dbkVMUn4IrJ1CAsFkc8\n"
   "gZxPzn683H40XMKltDZPEwr9ng78w9+QYQg7ygnr2stz8yhh06MkMCIwCwYDVR0P\n"
   "BAQDAgeAMBMGA1UdJQQMMAoGCCsGAQUFBwMDMAoGCCqGSM49BAMCA0gAMEUCIF0R\n"
   "r5cb7rEUNtWOvGd05MacrgOABfSoVYvBOK9fP63WAqt5h3BaS123coKSGg84twlq\n"
   "TkO/pV/xEmyZmZdV+HxV/OM=\n"
   "-----END CERTIFICATE-----\n";
   ```

4. Installieren Sie Python 3 oder höher.
5. Installieren Sie pyOpenSSL, indem Sie `pip install pyopenssl` ausführen.

6. Kopieren Sie Ihr Codesignierungszertifikat im PEM-Format in den Pfad `demos/ota/bootloader/utility/codesigner_cert_utility/`. Benennen Sie die Zertifikatsdatei in `aws_ota_codesigner_certificate.pem` um.
7. Öffne*`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/aws_demo_config.h`, kommentiere und definiere `CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` oder`CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`. `#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED`
8. Erstellen Sie die Lösung. Prüfen Sie, ob die Erstellung fehlerfrei ausgeführt wurde.
9. Starten Sie einen Terminalemulator und verwenden Sie die folgenden Einstellungen, um sich mit Ihrem Board zu verbinden:

   - Baudrate: 115200
   - Datenbits: 8
   - Parität: Keine
   - Stop-Bits: 1
10. Trennen Sie den Debugger von Ihrem Board und führen Sie das Projekt auf Ihrem Board aus, um sicherzustellen, dass es eine Verbindung zu Wi-Fi und dem AWS IoT MQTT-Nachrichtenbroker herstellen kann.

Wenn Sie das Projekt ausführen, sollte die MPLAB X IDE ein Ausgabefenster öffnen. Vergewissern Sie sich, dass der ICD4Tab ausgewählt ist. Die Ausgabe sollte folgendermaßen aussehen.

```
Bootloader version 00.09.00
[prvBOOT_Init] Watchdog timer initialized.
[prvBOOT_Init] Crypto initialized.

[prvValidateImage] Validating image at Bank : 0
[prvValidateImage] No application image or magic code present at: 0xbd000000
[prvBOOT_ValidateImages] Validation failed for image at 0xbd000000

[prvValidateImage] Validating image at Bank : 1
[prvValidateImage] No application image or magic code present at: 0xbd100000
[prvBOOT_ValidateImages] Validation failed for image at 0xbd100000

[prvBOOT_ValidateImages] Booting default image.
```

```
>0 36246 [IP-task] vDHCPProcess: offer ac140a0eip
                                                    1 36297 [IP-task] vDHCPProcess: offer
 ac140a0eip
                2 36297 [IP-task]

IP Address: 172.20.10.14
3 36297 [IP-task] Subnet Mask: 255.255.255.240
4 36297 [IP-task] Gateway Address: 172.20.10.1
5 36297 [IP-task] DNS Server Address: 172.20.10.1


6 36299 [OTA] OTA demo version 0.9.2
7 36299 [OTA] Creating MQTT Client...
8 36299 [OTA] Connecting to broker...
9 38673 [OTA] Connected to broker.
10 38793 [OTA Task] [prvSubscribeToJobNotificationTopics] OK: $aws/things/devthingota/
jobs/$next/get/accepted
11 38863 [OTA Task] [prvSubscribeToJobNotificationTopics] OK: $aws/things/devthingota/
jobs/notify-next
12 38863 [OTA Task] [OTA_CheckForUpdate] Request #0
13 38964 [OTA] [OTA_AgentInit] Ready.
14 38973 [OTA Task] [prvParseJSONbyModel] Extracted parameter [ clientToken:
 0:devthingota ]
15 38973 [OTA Task] [prvParseJSONbyModel] parameter not present: execution
16 38973 [OTA Task] [prvParseJSONbyModel] parameter not present: jobId
17 38973 [OTA Task] [prvParseJSONbyModel] parameter not present: jobDocument
18 38973 [OTA Task] [prvParseJSONbyModel] parameter not present: streamname
19 38973 [OTA Task] [prvParseJSONbyModel] parameter not present: files
20 38975 [OTA Task] [prvParseJSONbyModel] parameter not present: filepath
21 38975 [OTA Task] [prvParseJSONbyModel] parameter not present: filesize
22 38975 [OTA Task] [prvParseJSONbyModel] parameter not present: fileid
23 38975 [OTA Task] [prvParseJSONbyModel] parameter not present: certfile
24 38975 [OTA Task] [prvParseJSONbyModel] parameter not present: sig-sha256-ecdsa
25 38975 [OTA Task] [prvParseJobDoc] Ignoring job without ID.
26 38975 [OTA Task] [prvOTA_Close] Context->0x8003b620
27 38975 [OTA Task] [prvPAL_Abort] Abort - OK
28 39964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
29 40964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
30 41964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
31 42964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
32 43964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
33 44964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
34 45964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
35 46964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
```

```
36 47964 [OTA] State: Ready  Received: 1   Queued: 1   Processed: 1   Dropped: 0
```

Der Terminalemulator sollte den folgenden Text anzeigen:

```
AWS Validate: no valid signature in descr: 0xbd000000
AWS Validate: no valid signature in descr: 0xbd100000


>AWS Launch:  No Map performed. Running directly from address: 0x9d000020?
AWS Launch:  wait for app at: 0x9d000020
WILC1000: Initializing...
0 0

>[None] Seed for randomizer: 1172751941
1 0 [None] Random numbers: 00004272 00003B34 00000602 00002DE3
Chip ID 1503a0

[spi_cmd_rsp][356][nmi spi]: Failed cmd response read, bus error...

[spi_read_reg][1086][nmi spi]: Failed cmd response, read reg (0000108c)...

[spi_read_reg][1116]Reset and retry 10 108c

Firmware ver. : 4.2.1

Min driver ver : 4.2.1

Curr driver ver: 4.2.1

WILC1000: Initialization successful!

Start Wi-Fi Connection...
Wi-Fi Connected
2 7219 [IP-task] vDHCPProcess: offer c0a804beip
3 7230 [IP-task] vDHCPProcess: offer c0a804beip
4 7230 [IP-task]

IP Address: 192.168.4.190
5 7230 [IP-task] Subnet Mask: 255.255.240.0
6 7230 [IP-task] Gateway Address: 192.168.0.1
7 7230 [IP-task] DNS Server Address: 208.67.222.222
```

```
8 7232 [OTA] OTA demo version 0.9.0
9 7232 [OTA] Creating MQTT Client...
10 7232 [OTA] Connecting to broker...
11 7232 [OTA] Sending command to MQTT task.
12 7232 [MQTT] Received message 10000 from queue.
13 8501 [IP-task] Socket sending wakeup to MQTT task.
14 10207 [MQTT] Received message 0 from queue.
15 10256 [IP-task] Socket sending wakeup to MQTT task.
16 10256 [MQTT] Received message 0 from queue.
17 10256 [MQTT] MQTT Connect was accepted. Connection established.
18 10256 [MQTT] Notifying task.
19 10257 [OTA] Command sent to MQTT task passed.
20 10257 [OTA] Connected to broker.
21 10258 [OTA Task] Sending command to MQTT task.
22 10258 [MQTT] Received message 20000 from queue.
23 10306 [IP-task] Socket sending wakeup to MQTT task.
24 10306 [MQTT] Received message 0 from queue.
25 10306 [MQTT] MQTT Subscribe was accepted. Subscribed.
26 10306 [MQTT] Notifying task.
27 10307 [OTA Task] Command sent to MQTT task passed.
28 10307 [OTA Task] [OTA] Subscribed to topic: $aws/things/Microchip/jobs/$next/get/
accepted

29 10307 [OTA Task] Sending command to MQTT task.
30 10307 [MQTT] Received message 30000 from queue.
31 10336 [IP-task] Socket sending wakeup to MQTT task.
32 10336 [MQTT] Received message 0 from queue.
33 10336 [MQTT] MQTT Subscribe was accepted. Subscribed.
34 10336 [MQTT] Notifying task.
35 10336 [OTA Task] Command sent to MQTT task passed.
36 10336 [OTA Task] [OTA] Subscribed to topic: $aws/things/Microchip/jobs/notify-next

37 10336 [OTA Task] [OTA] Check For Update #0
38 10336 [OTA Task] Sending command to MQTT task.
39 10336 [MQTT] Received message 40000 from queue.
40 10366 [IP-task] Socket sending wakeup to MQTT task.
41 10366 [MQTT] Received message 0 from queue.
42 10366 [MQTT] MQTT Publish was successful.
43 10366 [MQTT] Notifying task.
44 10366 [OTA Task] Command sent to MQTT task passed.
45 10376 [IP-task] Socket sending wakeup to MQTT task.
46 10376 [MQTT] Received message 0 from queue.
47 10376 [OTA Task] [OTA] Set job doc parameter [ clientToken: 0:Microchip ]
48 10376 [OTA Task] [OTA] Missing job parameter: execution
```

```
49 10376 [OTA Task] [OTA] Missing job parameter: jobId
50 10376 [OTA Task] [OTA] Missing job parameter: jobDocument
51 10378 [OTA Task] [OTA] Missing job parameter: ts_ota
52 10378 [OTA Task] [OTA] Missing job parameter: files
53 10378 [OTA Task] [OTA] Missing job parameter: streamname
54 10378 [OTA Task] [OTA] Missing job parameter: certfile
55 10378 [OTA Task] [OTA] Missing job parameter: filepath
56 10378 [OTA Task] [OTA] Missing job parameter: filesize
57 10378 [OTA Task] [OTA] Missing job parameter: sig-sha256-ecdsa
58 10378 [OTA Task] [OTA] Missing job parameter: fileid
59 10378 [OTA Task] [OTA] Missing job parameter: attr
60 10378 [OTA Task] [OTA] Returned buffer to MQTT Client.
61 11367 [OTA] [OTA] Queued: 1   Processed: 1   Dropped: 0
62 12367 [OTA] [OTA] Queued: 1   Processed: 1   Dropped: 0
63 13367 [OTA] [OTA] Queued: 1   Processed: 1   Dropped: 0
64 14367 [OTA] [OTA] Queued: 1   Processed: 1   Dropped: 0
65 15367 [OTA] [OTA] Queued: 1   Processed: 1   Dropped: 0
66 16367 [OTA] [OTA] Queued: 1   Processed: 1   Dropped: 0
```

Diese Ausgabe zeigt, dass der Microchip Curiosity PIC32 MZEF eine Verbindung zu den MQTT-Themen herstellen AWS IoT und diese abonnieren kann, die für OTA-Updates erforderlich sind. Die `Missing job parameter`-Meldungen sind erwartungsgemäß, da keine OTA-Update-Jobs ausstehen.

Laden Sie die FreeRTOS OTA-Demo herunter, erstellen Sie sie, flashen Sie sie und führen Sie sie auf dem Espressif aus ESP32

> **Important**
>
> Diese Referenzintegration wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

1. Laden Sie die FreeRTOS-Quelle von herunter. GitHub Anweisungen finden Sie in der Datei README.md. Erstellen Sie in Ihrer IDE ein Projekt, das alle erforderlichen Quellen und Bibliotheken enthält.

2. Folgen Sie den Anweisungen unter Erste Schritte mit Espressif, um die erforderliche GCC-basierte Toolchain einzurichten.
3. Öffne*freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h, kommentiere und definiere CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED oder. #define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED
4. Erstellen Sie das Demo-Projekt, indem Sie make im Verzeichnis vendors/espressif/boards/esp32/aws_demos ausführen. Sie können das Demoprogramm flashen und seine Ausgabe überprüfen, indem Sie make flash monitor ausführen (wie in Erste Schritte mit Espressif beschrieben).
5. Vor der Ausführung der OTA-Update-Demo:
   - Öffne*freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h, kommentiere und definiere CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED oderCONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED. #define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED
   - Öffnen vendors/*vendor*/boards/*board*/aws_demos/config_files/ota_demo_config.h und kopieren Sie Ihr SHA-256/ECDSA-Codesignaturzertifikat in:

     ```
     #define otapalconfigCODE_SIGNING_CERTIFICATE [] = "your-certificate-key";
     ```

Laden Sie die FreeRTOS OTA-Demo auf dem Renesas N herunter, erstellen Sie sie, flashen Sie sie und führen Sie sie aus RX65

> **Important**
>
> Diese Referenzintegration wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

In diesem Kapitel erfahren Sie, wie Sie die FreeRTOS OTA-Demoanwendungen auf dem Renesas N herunterladen, erstellen, flashen und ausführen. RX65

Themen

- Richten Sie Ihre Betriebsumgebung ein
- Richten Sie Ihre AWS Ressourcen ein
- Importieren, konfigurieren Sie die Header-Datei und erstellen Sie aws_demos und boot_loader

Richten Sie Ihre Betriebsumgebung ein

Die Verfahren in diesem Abschnitt verwenden die folgenden Umgebungen:

- IDE: e 2 Studio 7.8.0, e 2 Studio 2020-07
- Toolchains: CCRX-Compiler v3.0.1
- Zielgeräte: RSKRX65 N-2 MB
- Debugger: E 2, E 2 Lite-Emulator
- Software: Renesas Flash-Programmierer, Renesas Secure Flash Programmer.exe, Tera Term

Um Ihre Hardware einzurichten

1. Connect den E 2 Lite-Emulator und die serielle USB-Schnittstelle mit Ihrem RX65 N-Board und PC.
2. Connect die Stromquelle an das RX65 N an.

Richten Sie Ihre AWS Ressourcen ein

1. Um die FreeRTOS-Demos ausführen zu können, benötigen Sie ein AWS Konto mit einem IAM-Benutzer, der berechtigt ist, auf Dienste zuzugreifen. AWS IoT Falls Sie dies noch nicht getan haben, folgen Sie den Schritten unter. Richten Sie Ihr AWS Konto und Ihre Berechtigungen ein
2. Folgen Sie den Schritten unter, um OTA-Updates einzurichtenVoraussetzungen für OTA-Updates. Folgen Sie insbesondere den Schritten unterVoraussetzungen für OTA-Updates mit MQTT.
3. Öffnen Sie die AWS IoT -Konsole.
4. Wählen Sie im linken Navigationsbereich Verwalten und dann Dinge aus, um ein Ding zu erstellen.

Ein Ding ist eine Repräsentation eines Geräts oder einer logischen Entität in AWS IoT. Es kann ein physisches Gerät oder ein Sensor sein (beispielsweise eine Glühbirne oder ein Wandschalter). Es kann sich auch um eine logische Einheit wie eine Instanz einer Anwendung oder eine physische Entität handeln AWS IoT, die keine Verbindung zu Geräten herstellt, die dies tun (z. B. ein Auto mit Motorsensoren oder einem Bedienfeld). AWS IoT bietet ein Verzeichnis für Dinge, mit dem Sie Ihre Dinge verwalten können.

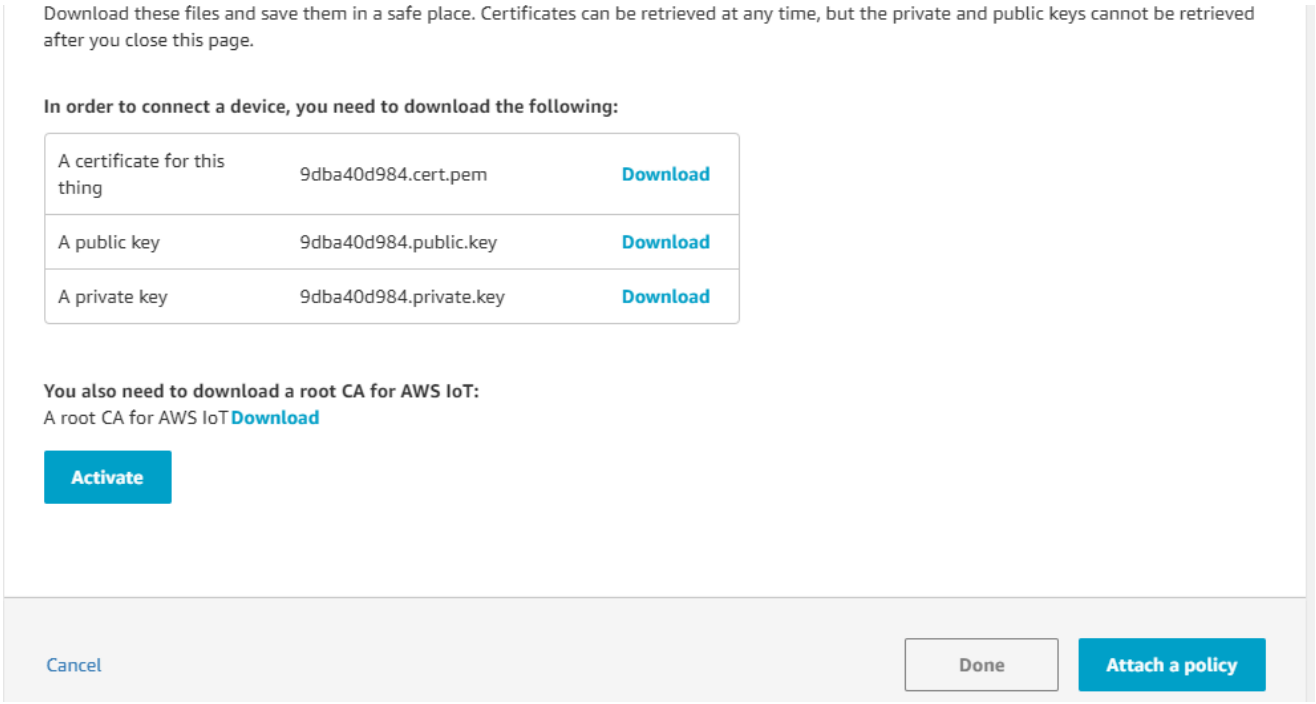
a. Wähle „Erstellen“ und dann „Ein einzelnes Ding erstellen“.

b. Gib einen Namen für dein Ding ein und wähle dann Weiter.

c. Wählen Sie Create certificate (Zertifikat erstellen).

d. Laden Sie die drei erstellten Dateien herunter und wählen Sie dann Aktivieren.

e. Wählen Sie Attach a policy (Richtlinie anfügen) aus.



f. Wählen Sie die Richtlinie aus, in der Sie sie erstellt habenGeräterichtlinie.

Jedes Gerät, das mithilfe von MQTT ein OTA-Update erhält, muss als Ding in registriert sein AWS IoT und dem Ding muss eine Richtlinie wie die aufgelistete angehängt sein. Weitere Informationen zu den Elementen finden Sie unter den `"Action"`- und `"Resource"`-Objekten der AWS IoT Core-Richtlinienaktionen und AWS IoT Core-Aktionsressourcen.

Hinweise

- Die `iot:Connect` Berechtigungen ermöglichen es Ihrem Gerät, eine Verbindung AWS IoT über MQTT herzustellen.
- Die `iot:Subscribe`- und `iot:Publish`-Berechtigungen für die Themen von AWS IoT -Aufgaben (`.../jobs/*`) ermöglichen es dem verbundenen Gerät, Auftragsbenachrichtigungen und Auftragsdokumente zu empfangen und den Abschlussstatus einer Auftragsausführung zu veröffentlichen.
- Die `iot:Subscribe` und `iot:Publish` Berechtigungen zu den Themen AWS IoT OTA-Streams (`.../streams/*`) ermöglichen es dem verbundenen Gerät, OTA-Aktualisierungsdaten von abzurufen. AWS IoT Diese Berechtigungen sind zum Ausführen von Firmware-Updates über MQTT erforderlich.
- Die `iot:Receive` Berechtigungen ermöglichen es AWS IoT Core , Nachrichten zu diesen Themen auf dem angeschlossenen Gerät zu veröffentlichen. Diese Berechtigung wird bei jeder Zustellung einer MQTT-Nachricht überprüft. Sie können diese Berechtigung verwenden, um den Zugriff auf Clients zu widerrufen, die derzeit ein Thema abonniert haben.

5. Um ein Codesignaturprofil zu erstellen und ein Codesignaturzertifikat zu registrieren. AWS

   a. Informationen zur Erstellung der Schlüssel und der Zertifizierung finden Sie in Abschnitt 7.3 „Generieren von SHA256 ECDSA-Schlüsselpaaren mit OpenSSL“ in der Designrichtlinie für MCU Firmware-Updates von Renesas.

   b. Öffnen Sie die AWS IoT -Konsole. Wählen Sie im linken Navigationsbereich Verwalten und dann Jobs aus. Wählen Sie Job erstellen und dann OTA-Aktualisierungsjob erstellen aus.

   c. Wählen Sie unter Zu aktualisierende Geräte auswählen die Option Auswählen und dann das Objekt aus, das Sie zuvor erstellt haben. Klicken Sie auf Weiter.

   d. Gehen Sie auf der Seite Create a FreeRTOS OTA update job wie folgt vor:

      i. Wählen Sie für Wählen Sie das Protokoll für die Firmware-Imageübertragung die Option MQTT aus.

      ii. Wählen Sie für Wählen Sie Ihr Firmware-Image aus und signieren Sie die Option Ein neues Firmware-Image für mich signieren.

      iii. Wählen Sie unter Code Signing-Profil die Option Create aus.

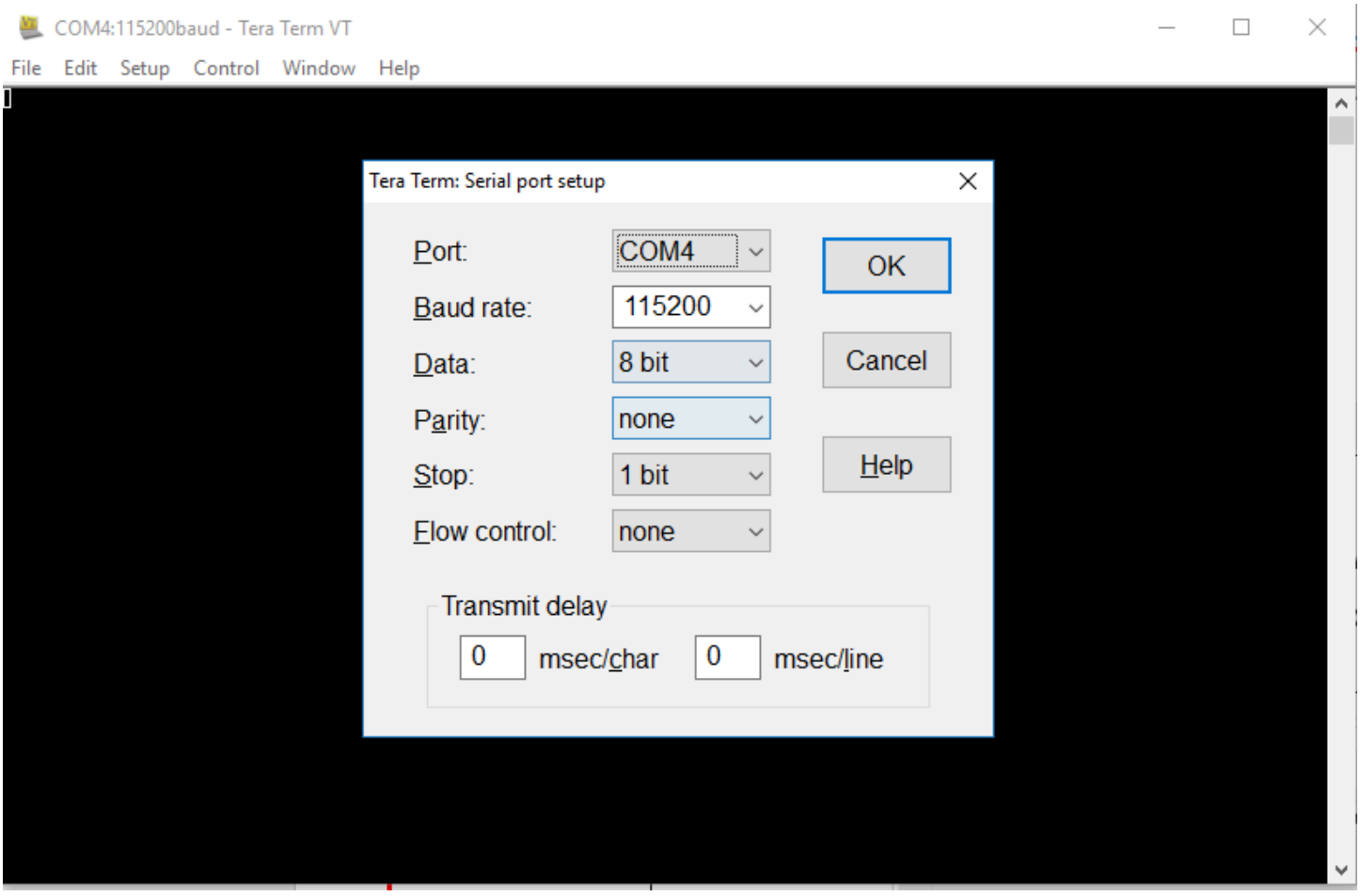
iv. Geben Sie im Fenster Codesignaturprofil erstellen einen Profilnamen ein. Wählen Sie für die Gerätehardwareplattform Windows Simulator aus. Wählen Sie für das Codesignaturzertifikat Import aus.

v. Suchen Sie nach dem Zertifikat (`secp256r1.crt`), dem privaten Schlüssel des Zertifikats (`secp256r1.key`) und der Zertifikatskette (`ca.crt`).

vi. Geben Sie einen Pfadnamen des Codesignaturzertifikats auf dem Gerät ein. Wählen Sie die Option Erstellen aus.

6. Gehen Sie wie unter beschrieben vor AWS IoT, um Zugriff auf das Codesigning für zu gewähren. Gewähren Sie Zugriff auf die Codesignatur für AWS IoT

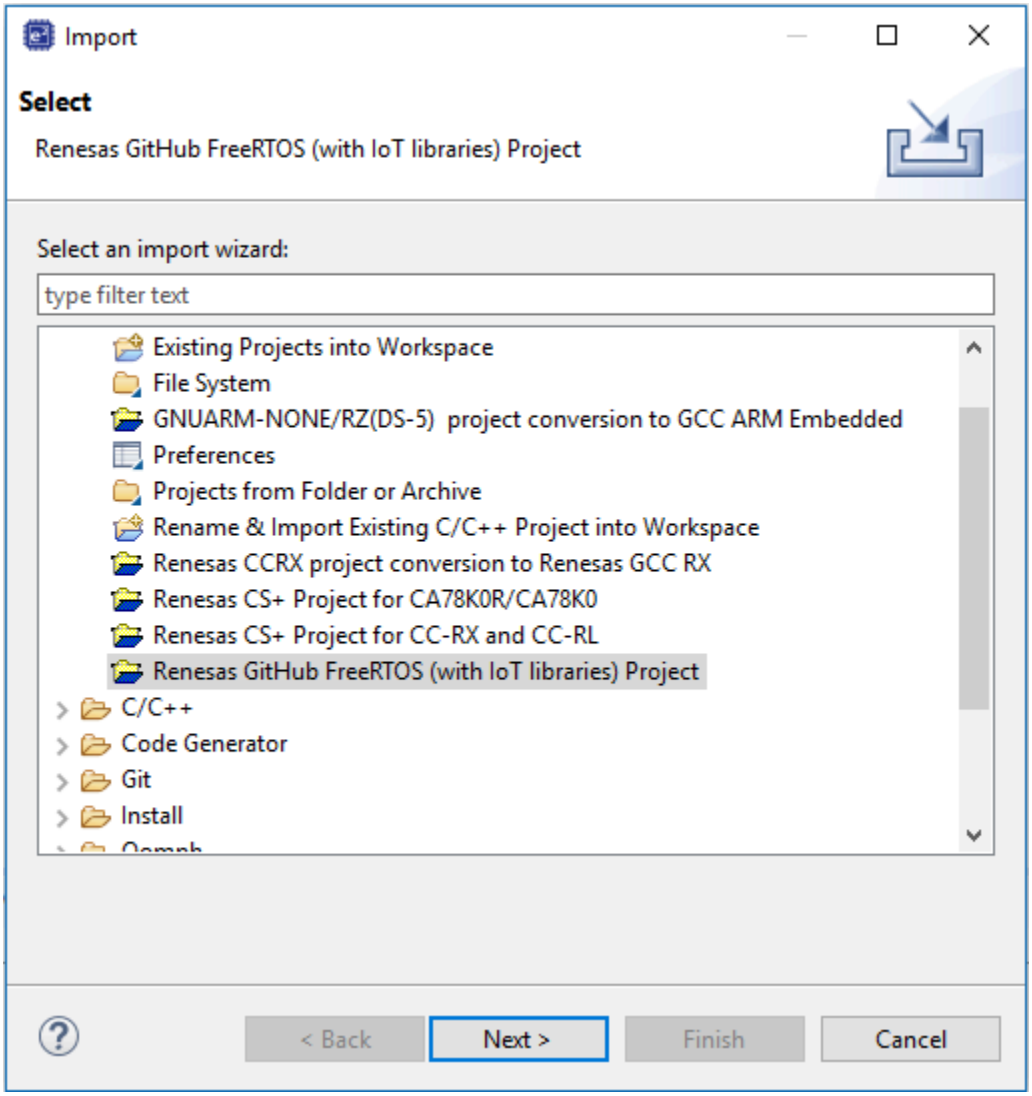
Wenn Sie Tera Term nicht auf Ihrem PC installiert haben, können Sie es von https://ttssh2.osdn.jp/index.html.en herunterladen und wie hier gezeigt einrichten. Stellen Sie sicher, dass Sie den seriellen USB-Anschluss Ihres Geräts an Ihren PC anschließen.



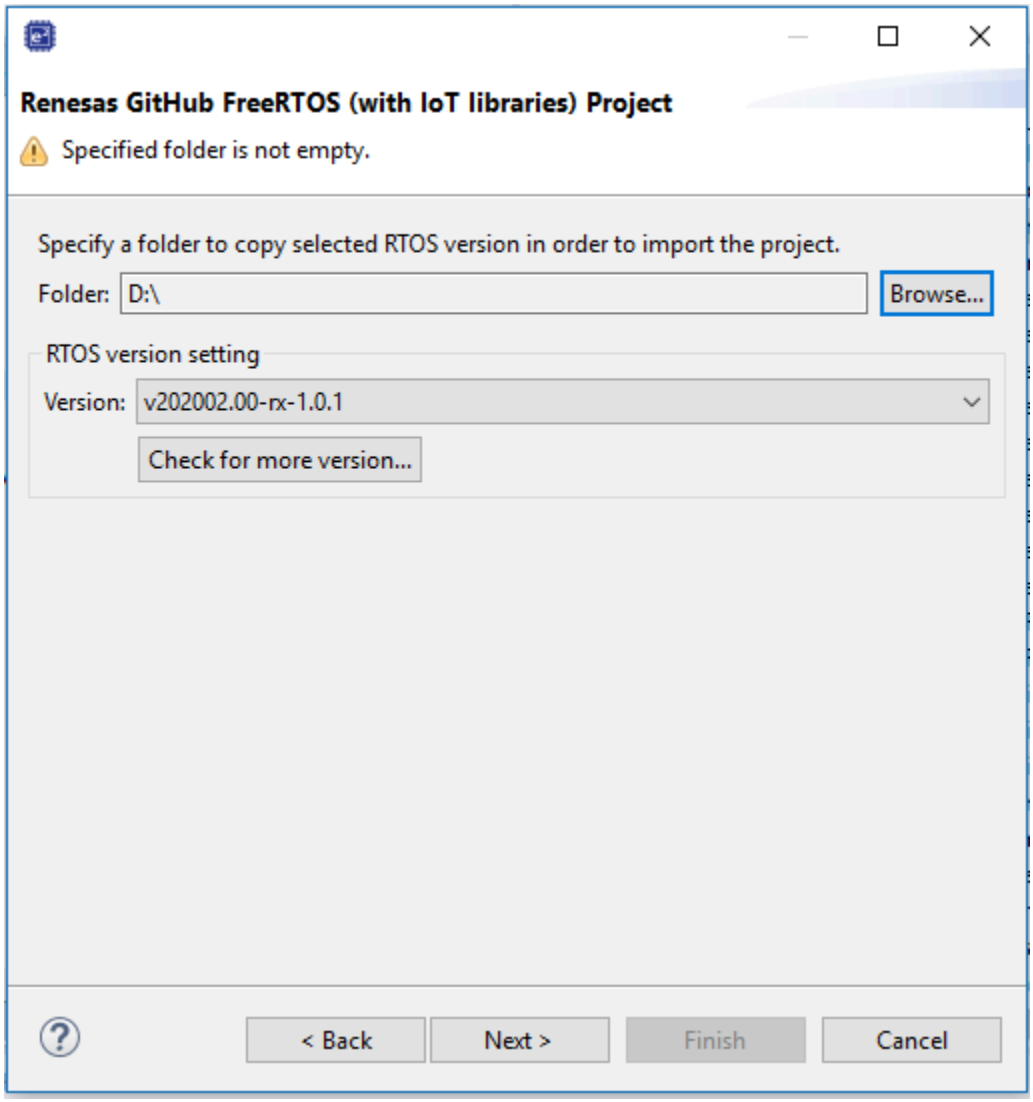
Importieren, konfigurieren Sie die Header-Datei und erstellen Sie aws_demos und boot_loader

Zu Beginn wählen Sie die neueste Version des FreeRTOS-Pakets aus. Diese wird automatisch aus dem Projekt heruntergeladen GitHub und in das Projekt importiert. Auf diese Weise können Sie sich auf die Konfiguration von FreeRTOS und das Schreiben von Anwendungscode konzentrieren.

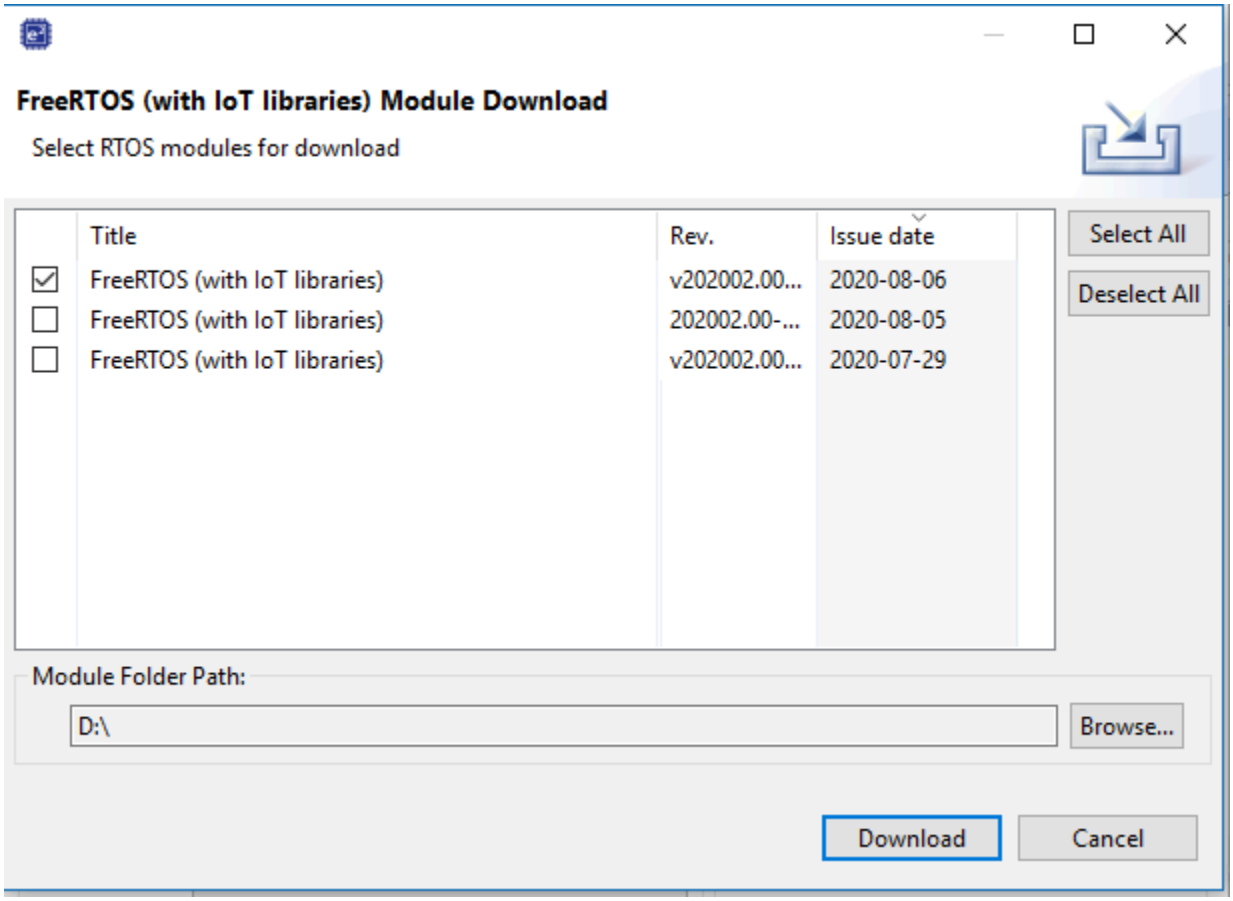
1. Starten Sie e [2 Studio].
2. Wählen Sie in Datei und dann Importieren aus.
3. Wählen Sie das Renesas GitHub FreeRTOS (mit IoT-Bibliotheken) -Projekt aus.



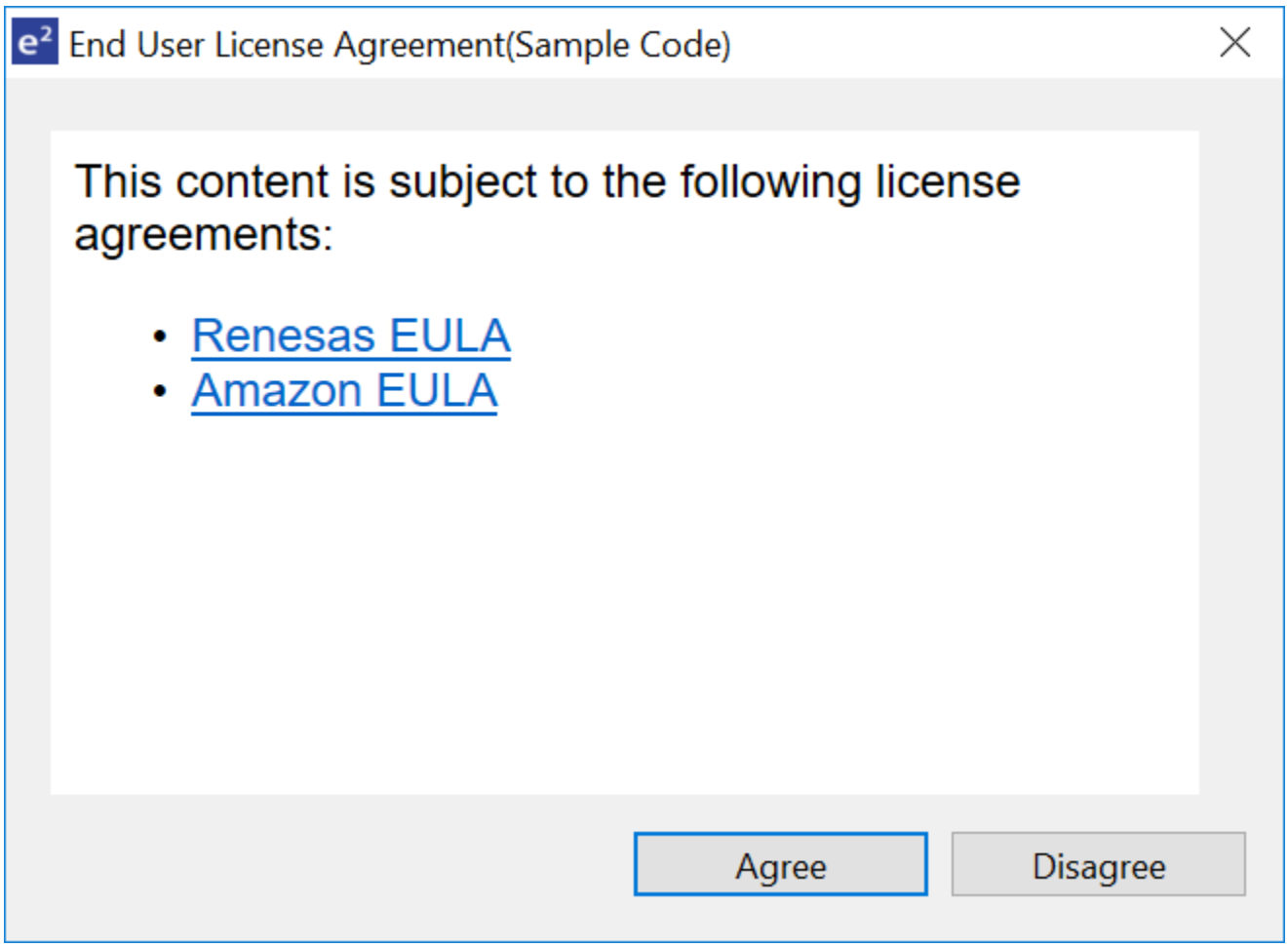
4. Wählen Sie Nach weiterer Version suchen..., um das Download-Dialogfeld aufzurufen.



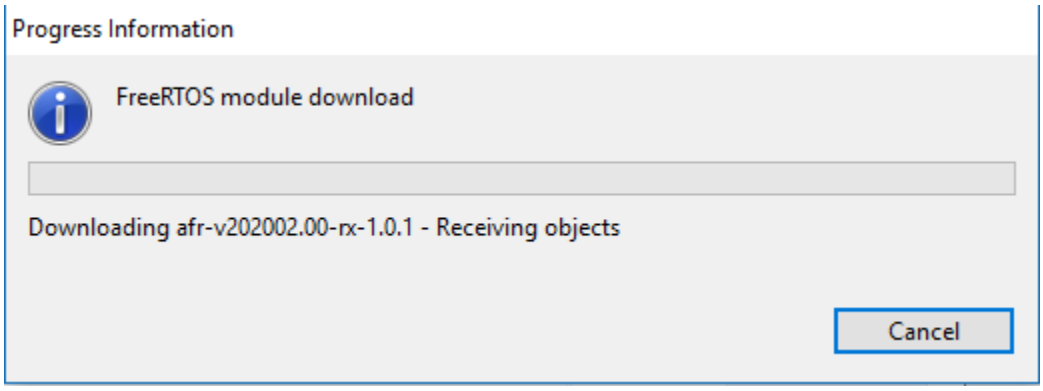
5. Wählen Sie das neueste Paket aus.



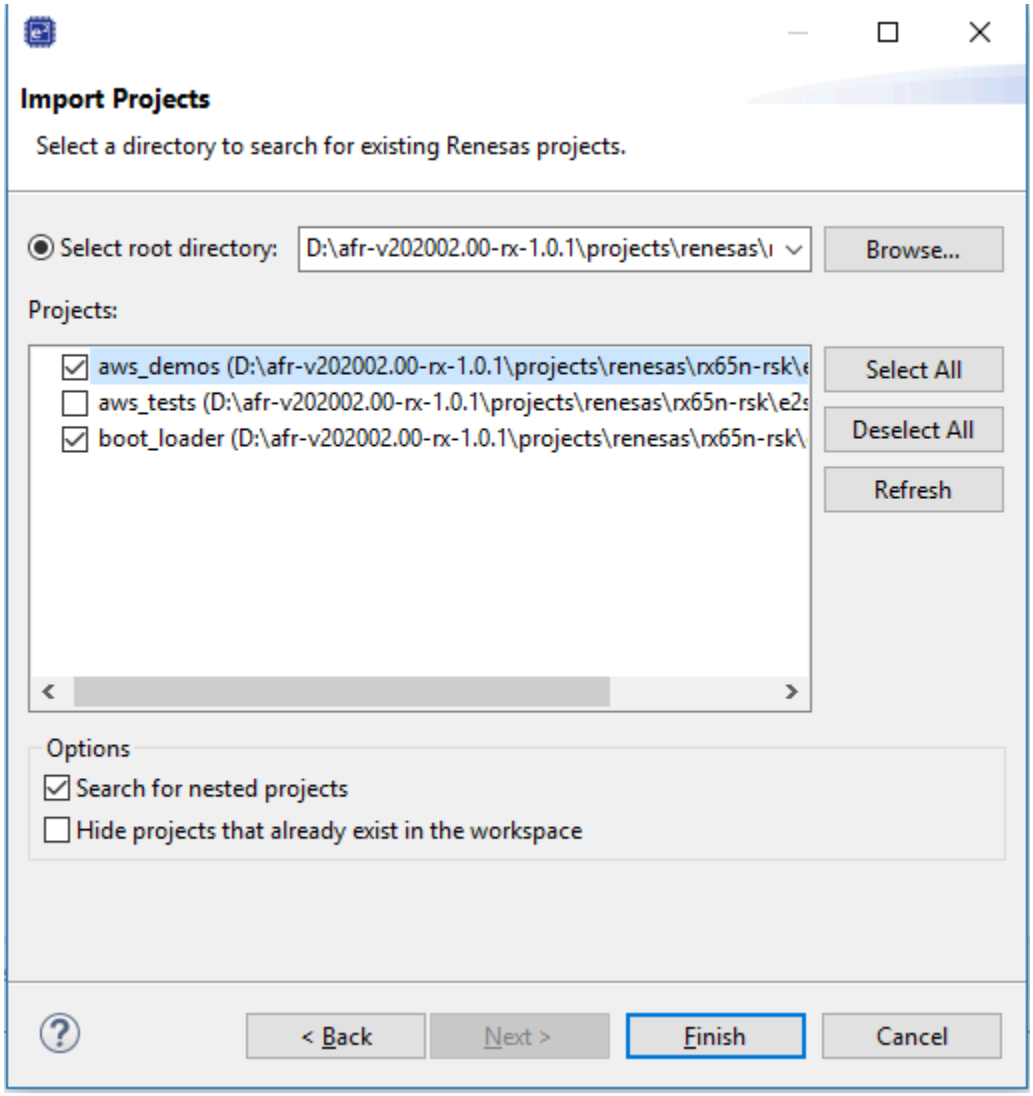
6. Wählen Sie Zustimmen, um die Endbenutzer-Lizenzvereinbarung zu akzeptieren.



7. Warten Sie, bis der Download abgeschlossen ist.



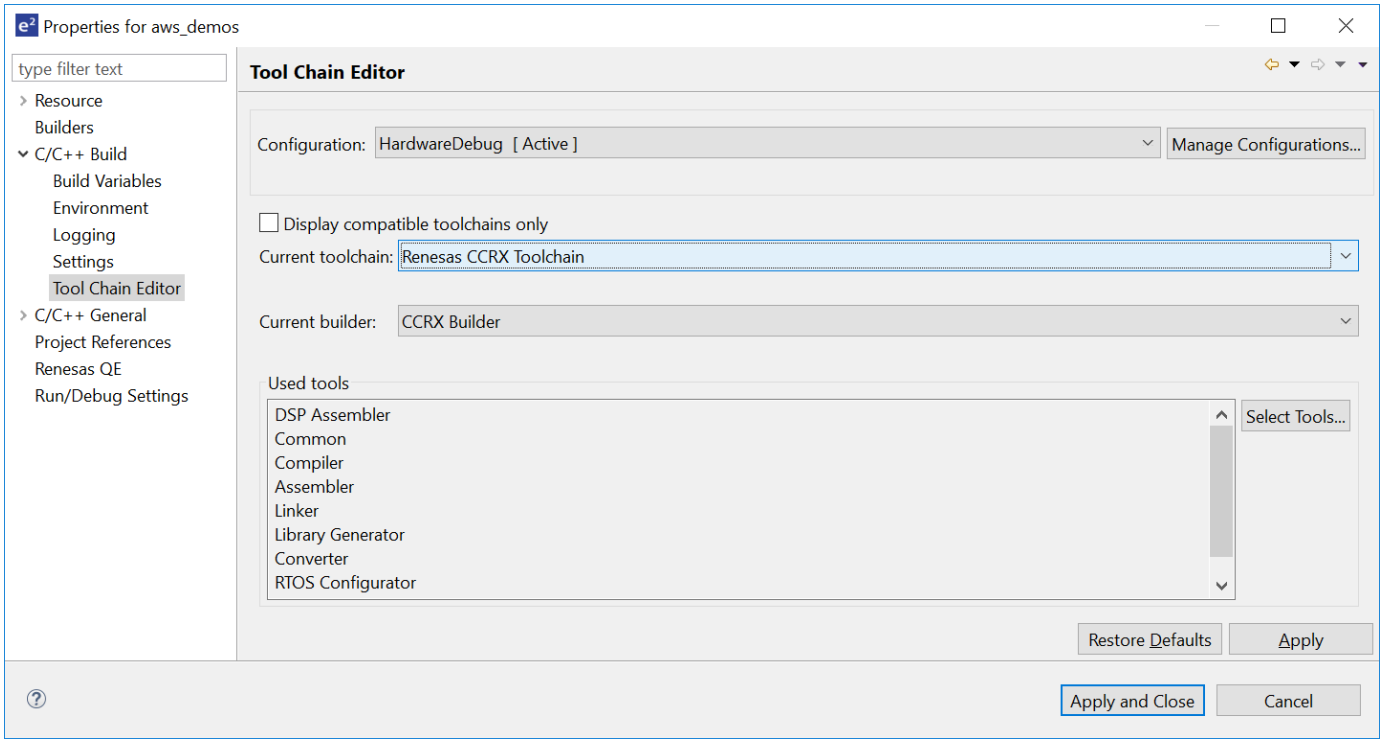
8. Wählen Sie die Projekte aws_demos und boot_loader aus und klicken Sie dann auf Fertig stellen, um sie zu importieren.



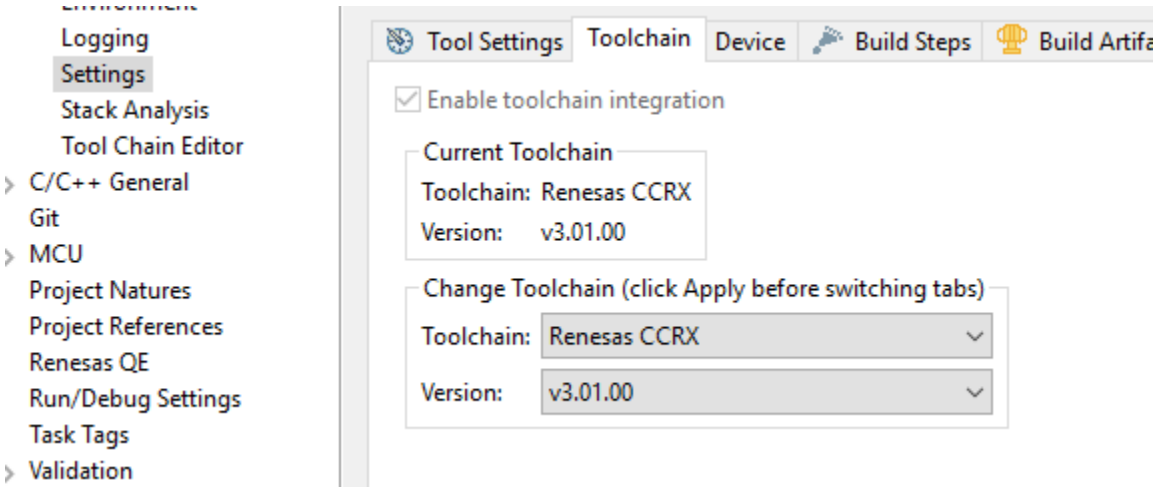
9. Öffnen Sie für beide Projekte die Projekteigenschaften. Wählen Sie im Navigationsbereich Tool Chain Editor aus.

    a. Wählen Sie die Aktuelle Toolchain aus.

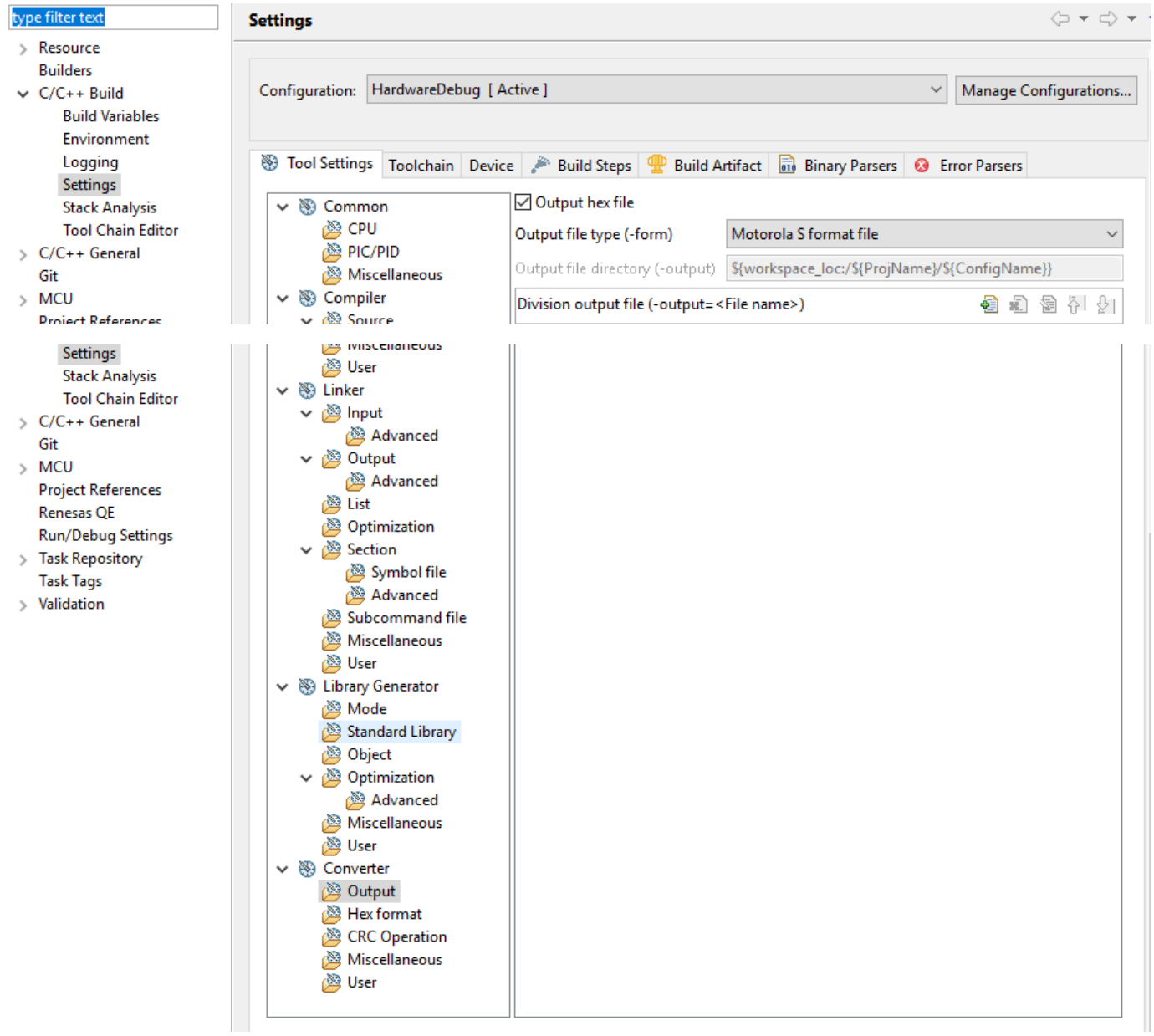
    b. Wählen Sie den aktuellen Builder.



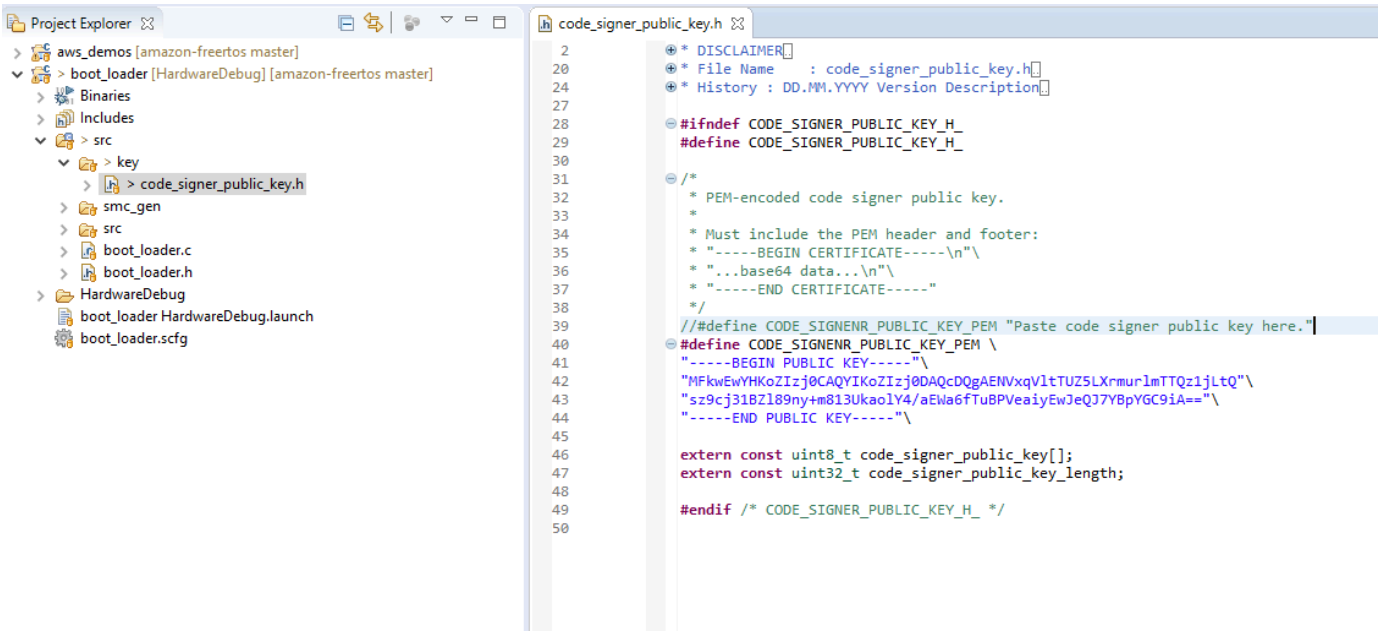
10. Wählen Sie im Navigationsbereich Settings (Einstellungen). Wählen Sie die Registerkarte Toolchain und dann die Toolketten-Version aus.



Wählen Sie die Registerkarte Werkzeugeinstellungen, erweitern Sie den Konverter und wählen Sie dann Ausgabe. Vergewissern Sie sich, dass im Hauptfenster die Option Hex-Datei ausgeben ausgewählt ist, und wählen Sie dann den Ausgabedateityp.



11. Öffnen Sie im Bootloader-Projekt den öffentlichen Schlüssel `projects\renesas\rx65n-rsk\e2studio\boot_loader\src\key\code_signer_public_key.h` und geben Sie ihn ein. Informationen zum Erstellen eines öffentlichen Schlüssels finden Sie unter So implementieren Sie FreeRTOS OTA mithilfe von Amazon Web Services auf RX65 N und in Abschnitt 7.3 „Generieren von SHA256 ECDSA-Schlüsselpaaren mit OpenSSL“ in der Designrichtlinie für MCU Firmware-Updates von Renesas.



Erstellen `boot_loader.mot` Sie dann das zu erstellende Projekt.

12. Öffnen Sie das `aws_demos` Projekt.

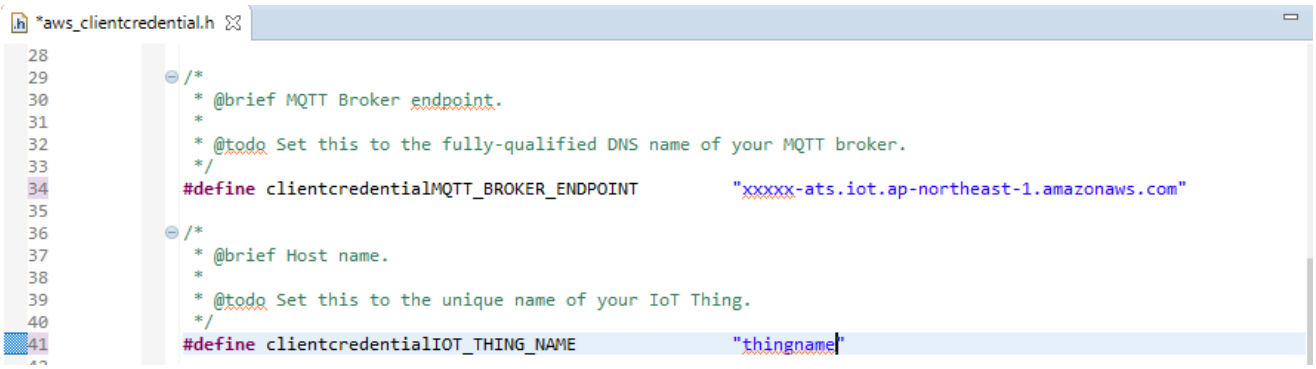
    a. Öffnen Sie die AWS IoT -Konsole.

    b. Wählen Sie im linken Navigationsbereich die Option Einstellungen aus. Notieren Sie sich Ihren benutzerdefinierten Endpunkt im Textfeld Gerätedatenendpunkt.

    c. Wählen Sie „Verwalten“ und anschließend „Dinge“. Notieren Sie sich den Namen AWS IoT des Dings auf Ihrem Board.

    d. Öffnen Sie das `aws_demos` Projekt `demos/include/aws_clientcredential.h` und geben Sie die folgenden Werte an.

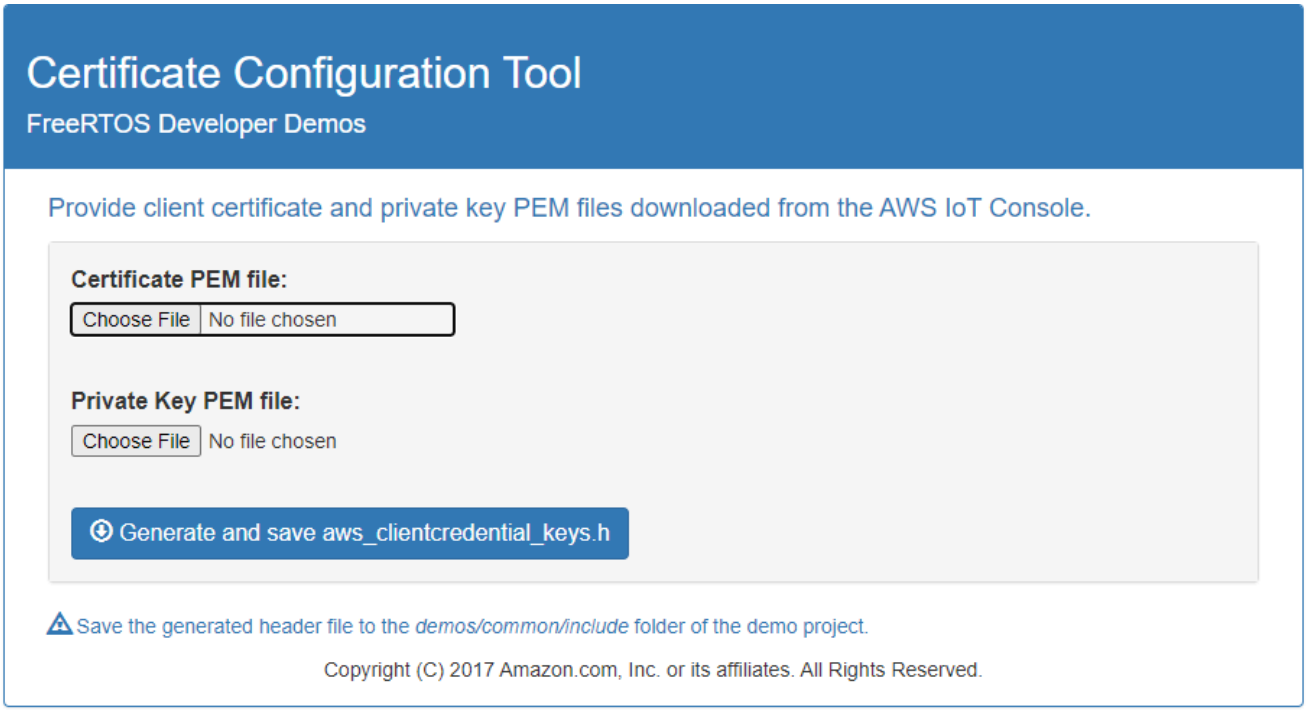
```
#define clientcredentialMQTT_BROKER_ENDPOINT[] = "Your AWS IoT endpoint";
#define clientcredentialIOT_THING_NAME "The AWS IoT thing name of your board"
```



e. Öffnen Sie die `tools/certificate_configuration/CertificateConfigurator.html` Datei.

f. Importieren Sie die Zertifikats-PEM-Datei und die PEM-Datei mit dem privaten Schlüssel, die Sie zuvor heruntergeladen haben.

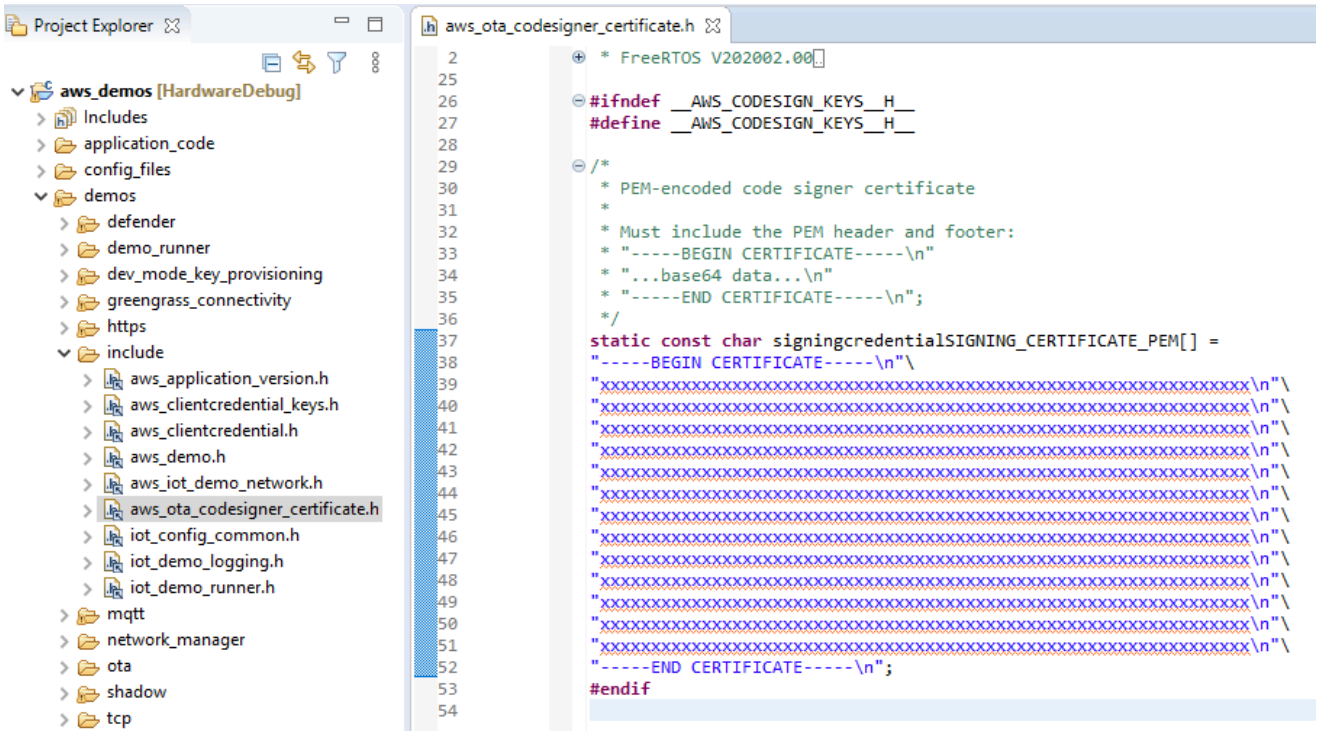
g. Wählen Sie Generate and save aws_clientcredentials al_keys.h und ersetzen Sie diese Datei im Verzeichnis. `demos/include/`



h. Öffnen Sie die Datei und geben Sie diese Werte an. `vendors/renesas/boards/rx65n-rsk/aws_demos/config_files/ota_demo_config.h`

```
#define otapalconfigCODE_SIGNING_CERTIFICATE [] = "your-certificate-key";
```

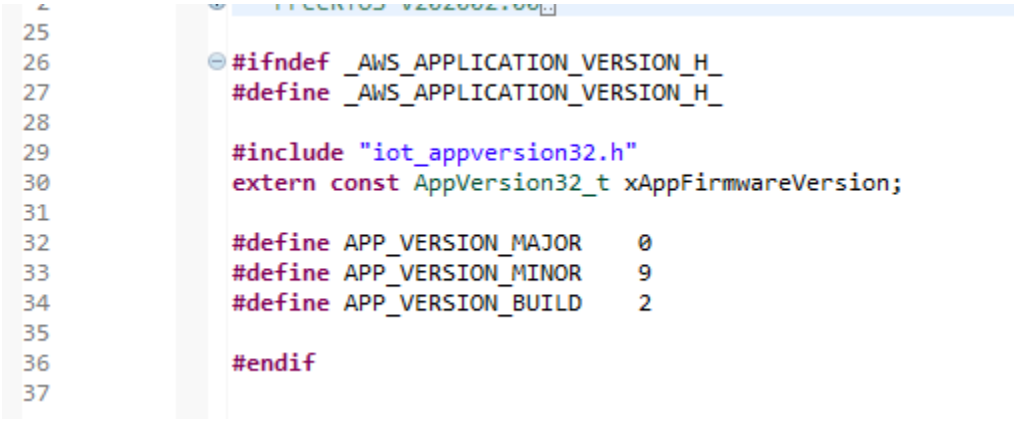
Wo *`your-certificate-key`* ist der Wert aus der Datei`secp256r1.crt`. Denken Sie daran, nach jeder Zeile der Zertifizierung „\“ hinzuzufügen. Weitere Informationen zur Erstellung der `secp256r1.crt` Datei finden Sie unter So implementieren Sie FreeRTOS OTA mithilfe von Amazon Web Services auf RX65 N und in Abschnitt 7.3 „Generieren von SHA256 ECDSA-Schlüsselpaaren mit OpenSSL“ in der Designrichtlinie für MCU Firmware-Updates von Renesas.



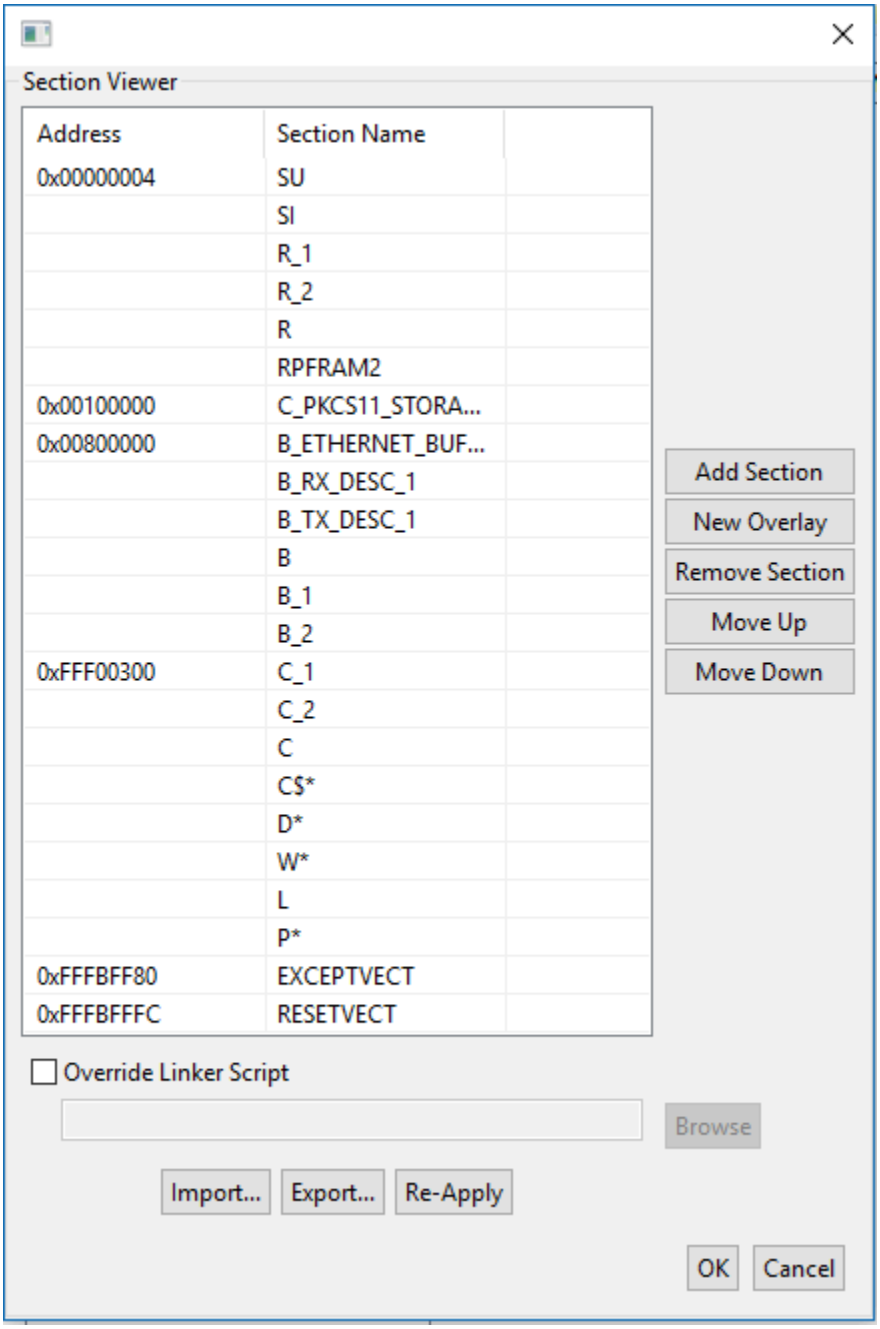
13. Aufgabe A: Installieren Sie die erste Version der Firmware

    a. Öffnen Sie die `vendors/renesas/boards/board/aws_demos/config_files/aws_demo_config.h` Datei, kommentieren Sie sie `#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED` und definieren Sie entweder `CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` oder`CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`.

    b. Öffnen Sie die `demos/include/ aws_application_version.h` Datei und stellen Sie die ursprüngliche Version der Firmware auf ein`0.9.2`.



    c. Ändern Sie die folgenden Einstellungen im Section Viewer.



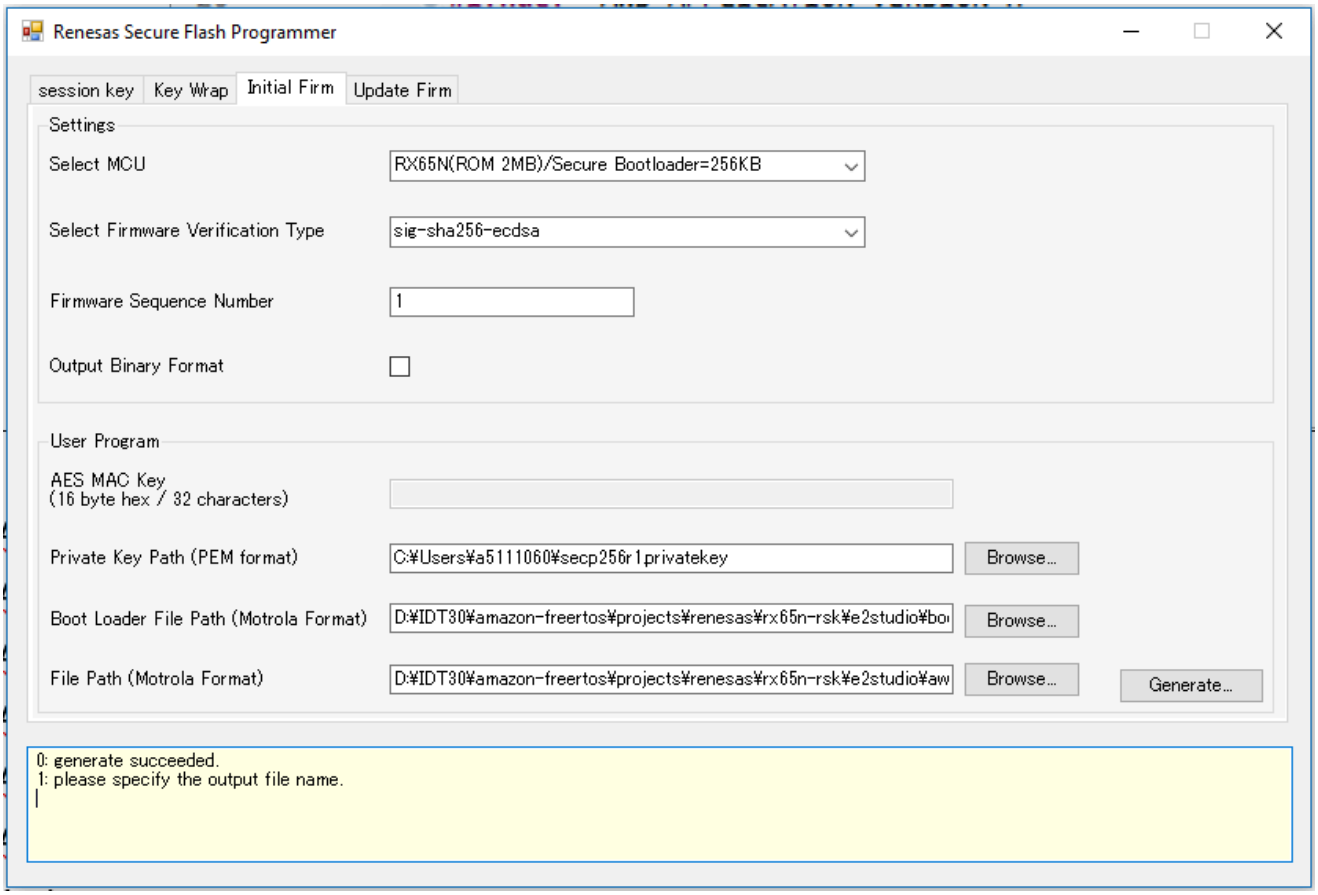
d. Wählen Sie Build, um die `aws_demos.mot` Datei zu erstellen.

14. Erstellen Sie die Datei `userprog.mot` mit dem Renesas Secure Flash Programmer. `userprog.mot`ist eine Kombination aus und. `aws_demos.mot` `boot_loader.mot` Sie können diese Datei auf den RX65 N-RSK flashen, um die erste Firmware zu installieren.

    a. Laden Sie https://github.com/renesas/Amazon-FreeRTOS-Tools herunter und öffnen Sie. `Renesas Secure Flash Programmer.exe`

    b. Wählen Sie die Registerkarte Initial Firm und legen Sie dann die folgenden Parameter fest:

- Pfad des privaten Schlüssels — Der Speicherort von`secp256r1.privatekey`.
- Bootloader-Dateipfad — Der Speicherort von `boot_loader.mot` (`projects\renesas\rx65n-rsk\e2studio\boot_loader\HardwareDebug`).
- Dateipfad — Der Speicherort von `aws_demos.mot` (`projects\renesas\rx65n-rsk\e2studio\aws_demos\HardwareDebug`).



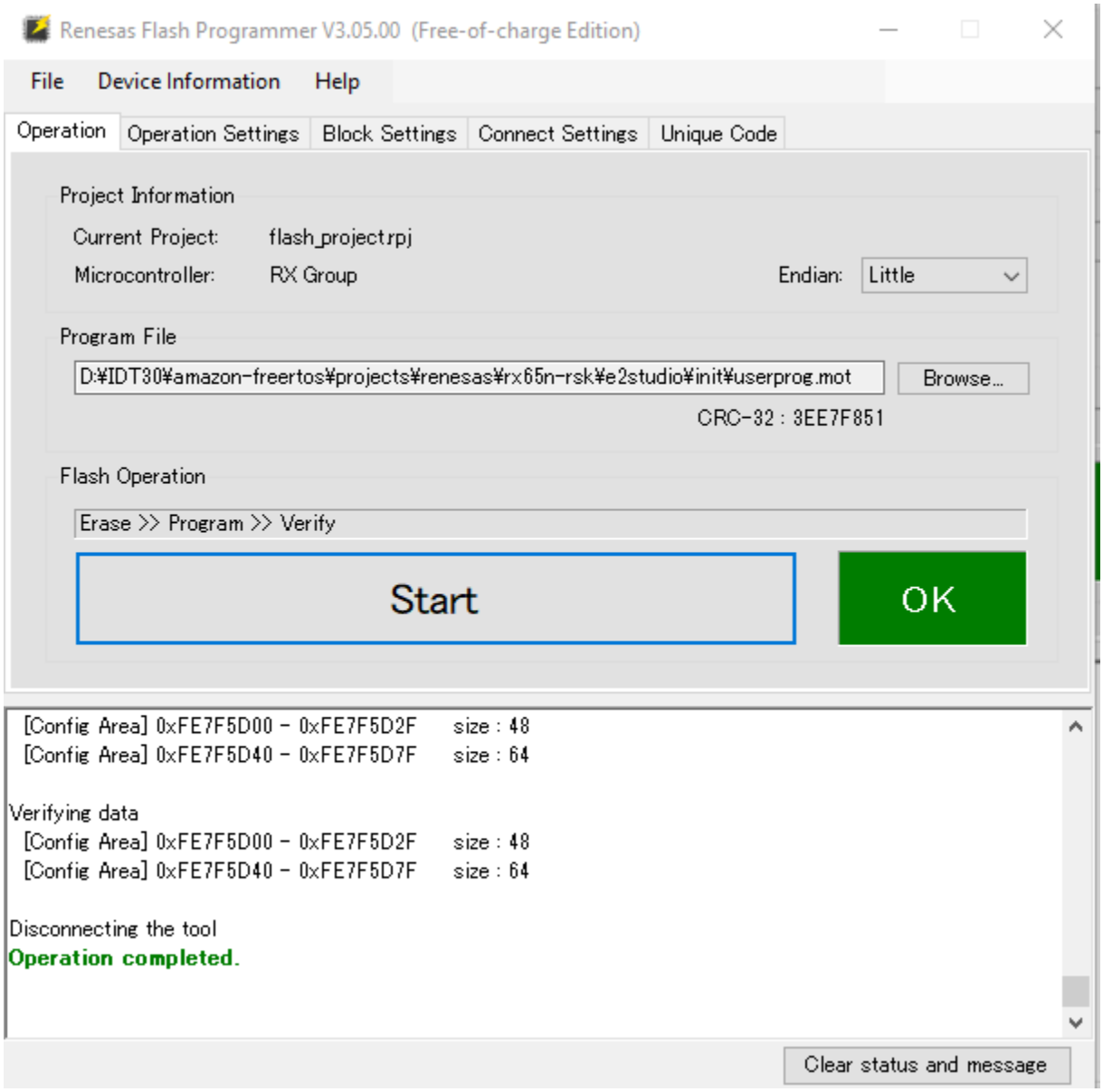
c. Erstellen Sie ein Verzeichnis mit dem Namen `init_firmware` Generate `userprog.mot` und speichern Sie es im Verzeichnis. `init_firmware` Stellen Sie sicher, dass die Generierung erfolgreich war.

15. Flashen Sie die erste Firmware auf dem RX65 N-RSK.

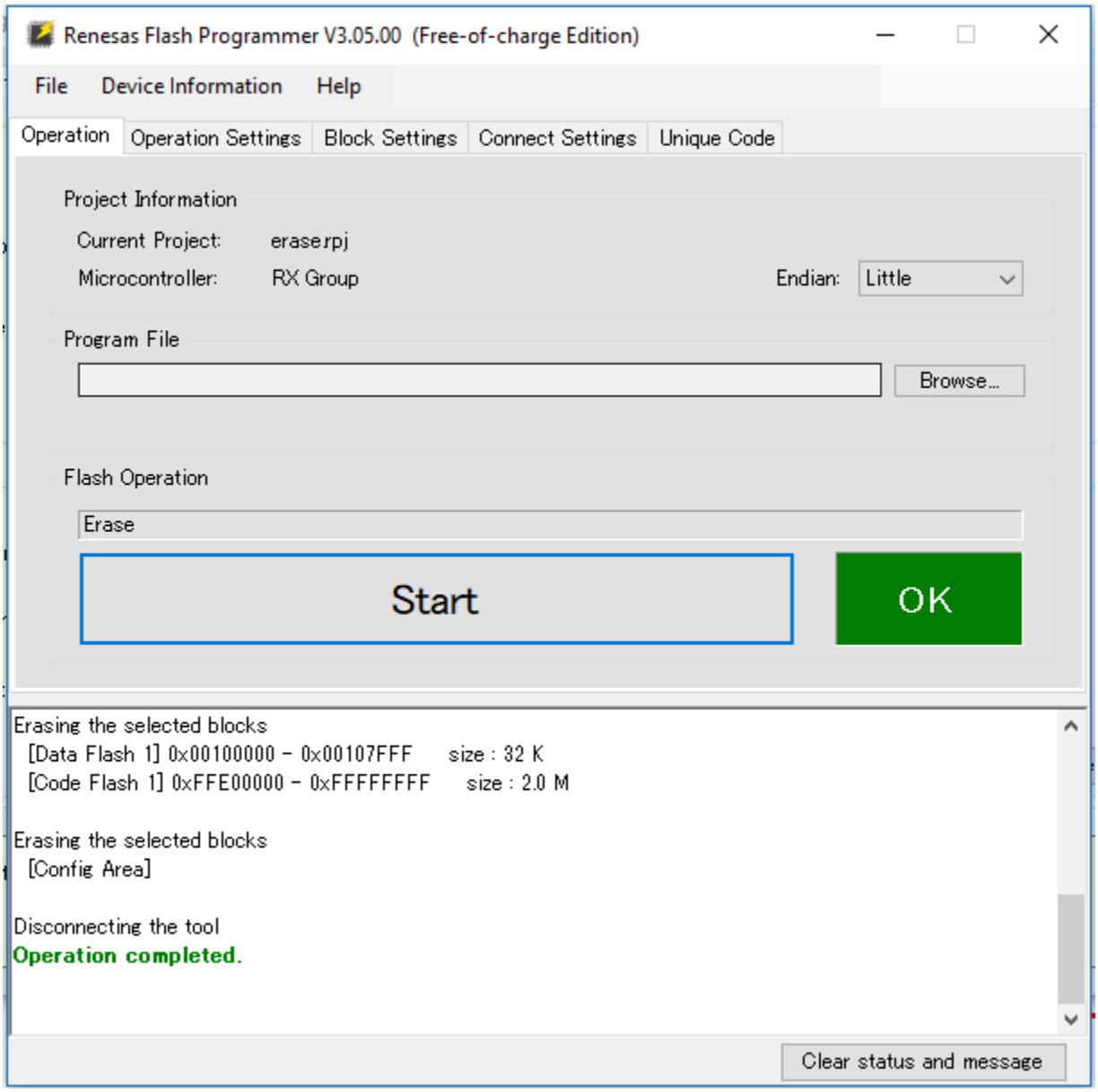
a. [Laden Sie die neueste Version des Renesas Flash Programmer (Programming GUI) von - .html herunter. https://www.renesas.com/tw/ en/products/software-tools/tools/programmer/ renesas flash-programmer-programming-gui]

b. Öffnen Sie die `vendors\renesas\rx_mcu_boards\boards\rx65n-rsk\aws_demos\flash_project\erase_from_bank\ erase.rpj` Datei, um Daten auf der Bank zu löschen.

c. Wählen Sie Start, um die Bank zu löschen.



d. Um zu flashen, wählen Sie Durchsuchen... **`userprog.mot`** und navigieren Sie zum `init_firmware` Verzeichnis, wählen Sie die `userprog.mot` Datei aus und wählen Sie Start.



16. Version 0.9.2 (erste Version) der Firmware wurde auf Ihrem N-RSK installiert. RX65 Das RX65 N-RSK-Board wartet jetzt auf OTA-Updates. Wenn Sie Tera Term auf Ihrem PC geöffnet haben, sehen Sie bei der Ausführung der ersten Firmware etwa Folgendes.

```
------------------------------------------------
RX65N secure boot program
------------------------------------------------
Checking flash ROM status.
bank 0 status = 0xff [LIFECYCLE_STATE_BLANK]
bank 1 status = 0xfc [LIFECYCLE_STATE_INSTALLING]
bank info = 1. (start bank = 0)
start installing user program.
```

```
copy secure boot (part1) from bank0 to bank1...OK
copy secure boot (part2) from bank0 to bank1...OK
update LIFECYCLE_STATE from [LIFECYCLE_STATE_INSTALLING] to [LIFECYCLE_STATE_VALID]
bank1(temporary area) block0 erase (to update LIFECYCLE_STATE)...OK
bank1(temporary area) block0 write (to update LIFECYCLE_STATE)...OK
swap bank...
-------------------------------------------------
RX65N secure boot program
-------------------------------------------------
Checking flash ROM status.
bank 0 status = 0xf8 [LIFECYCLE_STATE_VALID]
bank 1 status = 0xff [LIFECYCLE_STATE_BLANK]
bank info = 0. (start bank = 1)
integrity check scheme = sig-sha256-ecdsa
bank0(execute area) on code flash integrity check...OK
jump to user program
#0 1 [ETHER_RECEI] Deferred Interrupt Handler Task started
1 1 [ETHER_RECEI] Network buffers: 3 lowest 3
2 1 [ETHER_RECEI] Heap: current 234192 lowest 234192
3 1 [ETHER_RECEI] Queue space: lowest 8
4 1 [IP-task] InitializeNetwork returns OK
5 1 [IP-task] xNetworkInterfaceInitialise returns 0
6 101 [ETHER_RECEI] Heap: current 234592 lowest 233392
7 2102 [ETHER_RECEI] prvEMACHandlerTask: PHY LS now 1
8 3001 [IP-task] xNetworkInterfaceInitialise returns 1
9 3092 [ETHER_RECEI] Network buffers: 2 lowest 2
10 3092 [ETHER_RECEI] Queue space: lowest 7
11 3092 [ETHER_RECEI] Heap: current 233320 lowest 233320
12 3193 [ETHER_RECEI] Heap: current 233816 lowest 233120
13 3593 [IP-task] vDHCPProcess: offer c0a80a09ip
14 3597 [ETHER_RECEI] Heap: current 233200 lowest 233000
15 3597 [IP-task] vDHCPProcess: offer c0a80a09ip
16 3597 [IP-task] IP Address: 192.168.10.9
17 3597 [IP-task] Subnet Mask: 255.255.255.0
18 3597 [IP-task] Gateway Address: 192.168.10.1
19 3597 [IP-task] DNS Server Address: 192.168.10.1
20 3600 [Tmr Svc] The network is up and running
21 3622 [Tmr Svc] Write certificate...
22 3697 [ETHER_RECEI] Heap: current 232320 lowest 230904
23 4497 [ETHER_RECEI] Heap: current 226344 lowest 225944
24 5317 [iot_thread] [INFO ][DEMO][5317] ---------STARTING DEMO---------

25 5317 [iot_thread] [INFO ][INIT][5317] SDK successfully initialized.
```

```
26 5317 [iot_thread] [INFO ][DEMO][5317] Successfully initialized the demo. Network
 type for the demo: 4
27 5317 [iot_thread] [INFO ][MQTT][5317] MQTT library successfully initialized.
28 5317 [iot_thread] [INFO ][DEMO][5317] OTA demo version 0.9.2

29 5317 [iot_thread] [INFO ][DEMO][5317] Connecting to broker...

30 5317 [iot_thread] [INFO ][DEMO][5317] MQTT demo client identifier is rx65n-gr-
rose (length 13).
31 5325 [ETHER_RECEI] Heap: current 206944 lowest 206504
32 5325 [ETHER_RECEI] Heap: current 206440 lowest 206440
33 5325 [ETHER_RECEI] Heap: current 206240 lowest 206240
38 5334 [ETHER_RECEI] Heap: current 190288 lowest 190288
39 5334 [ETHER_RECEI] Heap: current 190088 lowest 190088
40 5361 [ETHER_RECEI] Heap: current 158512 lowest 158168
41 5363 [ETHER_RECEI] Heap: current 158032 lowest 158032
42 5364 [ETHER_RECEI] Network buffers: 1 lowest 1
43 5364 [ETHER_RECEI] Heap: current 156856 lowest 156856
44 5364 [ETHER_RECEI] Heap: current 156656 lowest 156656
46 5374 [ETHER_RECEI] Heap: current 153016 lowest 152040
47 5492 [ETHER_RECEI] Heap: current 141464 lowest 139016
48 5751 [ETHER_RECEI] Heap: current 140160 lowest 138680
49 5917 [ETHER_RECEI] Heap: current 138280 lowest 138168
59 7361 [iot_thread] [INFO ][MQTT][7361] Establishing new MQTT connection.
62 7428 [iot_thread] [INFO ][MQTT][7428] (MQTT connection 81cfc8, CONNECT operation
 81d0e8) Wait complete with result SUCCESS.
63 7428 [iot_thread] [INFO ][MQTT][7428] New MQTT connection 4e8c established.
64 7430 [iot_thread] [OTA_AgentInit_internal] OTA Task is Ready.
65 7430 [OTA Agent T] [prvOTAAgentTask] Called handler. Current State [Ready] Event
 [Start] New state [RequestingJob]
66 7431 [OTA Agent T] [INFO ][MQTT][7431] (MQTT connection 81cfc8) SUBSCRIBE
 operation scheduled.
67 7431 [OTA Agent T] [INFO ][MQTT][7431] (MQTT connection 81cfc8, SUBSCRIBE
 operation 818c48) Waiting for operation completion.
68 7436 [ETHER_RECEI] Heap: current 128248 lowest 127992
69 7480 [OTA Agent T] [INFO ][MQTT][7480] (MQTT connection 81cfc8, SUBSCRIBE
 operation 818c48) Wait complete with result SUCCESS.
70 7480 [OTA Agent T] [prvSubscribeToJobNotificationTopics] OK: $aws/things/rx65n-
gr-rose/jobs/$next/get/accepted
71 7481 [OTA Agent T] [INFO ][MQTT][7481] (MQTT connection 81cfc8) SUBSCRIBE
 operation scheduled.
72 7481 [OTA Agent T] [INFO ][MQTT][7481] (MQTT connection 81cfc8, SUBSCRIBE
 operation 818c48) Waiting for operation completion.
```

```
73 7530 [OTA Agent T] [INFO ][MQTT][7530] (MQTT connection 81cfc8, SUBSCRIBE
 operation 818c48) Wait complete with result SUCCESS.
74 7530 [OTA Agent T] [prvSubscribeToJobNotificationTopics] OK: $aws/things/rx65n-
gr-rose/jobs/notify-next
75 7530 [OTA Agent T] [prvRequestJob_Mqtt] Request #0
76 7532 [OTA Agent T] [INFO ][MQTT][7532] (MQTT connection 81cfc8) MQTT PUBLISH
 operation queued.
77 7532 [OTA Agent T] [INFO ][MQTT][7532] (MQTT connection 81cfc8, PUBLISH
 operation 818b80) Waiting for operation completion.
78 7552 [OTA Agent T] [INFO ][MQTT][7552] (MQTT connection 81cfc8, PUBLISH
 operation 818b80) Wait complete with result SUCCESS.
79 7552 [OTA Agent T] [prvOTAAgentTask] Called handler. Current State
 [RequestingJob] Event [RequestJobDocument] New state [WaitingForJob]
80 7552 [OTA Agent T] [prvParseJSONbyModel] Extracted parameter [ clientToken:
 0:rx65n-gr-rose ]
81 7552 [OTA Agent T] [prvParseJSONbyModel] parameter not present: execution
82 7552 [OTA Agent T] [prvParseJSONbyModel] parameter not present: jobId
83 7552 [OTA Agent T] [prvParseJSONbyModel] parameter not present: jobDocument
84 7552 [OTA Agent T] [prvParseJSONbyModel] parameter not present: afr_ota
85 7552 [OTA Agent T] [prvParseJSONbyModel] parameter not present: protocols
86 7552 [OTA Agent T] [prvParseJSONbyModel] parameter not present: files
87 7552 [OTA Agent T] [prvParseJSONbyModel] parameter not present: filepath
99 7651 [ETHER_RECEI] Heap: current 129720 lowest 127304
100 8430 [iot_thread] [INFO ][DEMO][8430] State: Ready  Received: 1   Queued: 0
 Processed: 0   Dropped: 0
101 9430 [iot_thread] [INFO ][DEMO][9430] State: WaitingForJob  Received: 1
 Queued: 0   Processed: 0   Dropped: 0
102 10430 [iot_thread] [INFO ][DEMO][10430] State: WaitingForJob  Received: 1
 Queued: 0   Processed: 0   Dropped: 0
103 11430 [iot_thread] [INFO ][DEMO][11430] State: WaitingForJob  Received: 1
 Queued: 0   Processed: 0   Dropped: 0
104 12430 [iot_thread] [INFO ][DEMO][12430] State: WaitingForJob  Received: 1
 Queued: 0   Processed: 0   Dropped: 0
105 13430 [iot_thread] [INFO ][DEMO][13430] State: WaitingForJob  Received: 1
 Queued: 0   Processed: 0   Dropped: 0
106 14430 [iot_thread] [INFO ][DEMO][14430] State: WaitingForJob  Received: 1
 Queued: 0   Processed: 0   Dropped: 0
107 15430 [iot_thread] [INFO ][DEMO][15430] State: WaitingForJob  Received: 1
 Queued: 0   Processed: 0   Dropped: 0
```

17. Aufgabe B: Aktualisieren Sie die Version Ihrer Firmware

    a. Öffnen Sie die `demos/include/aws_application_version.h` Datei und erhöhen Sie den `APP_VERSION_BUILD` Token-Wert auf`0.9.3`.

b. Erstellen Sie das Projekt neu.

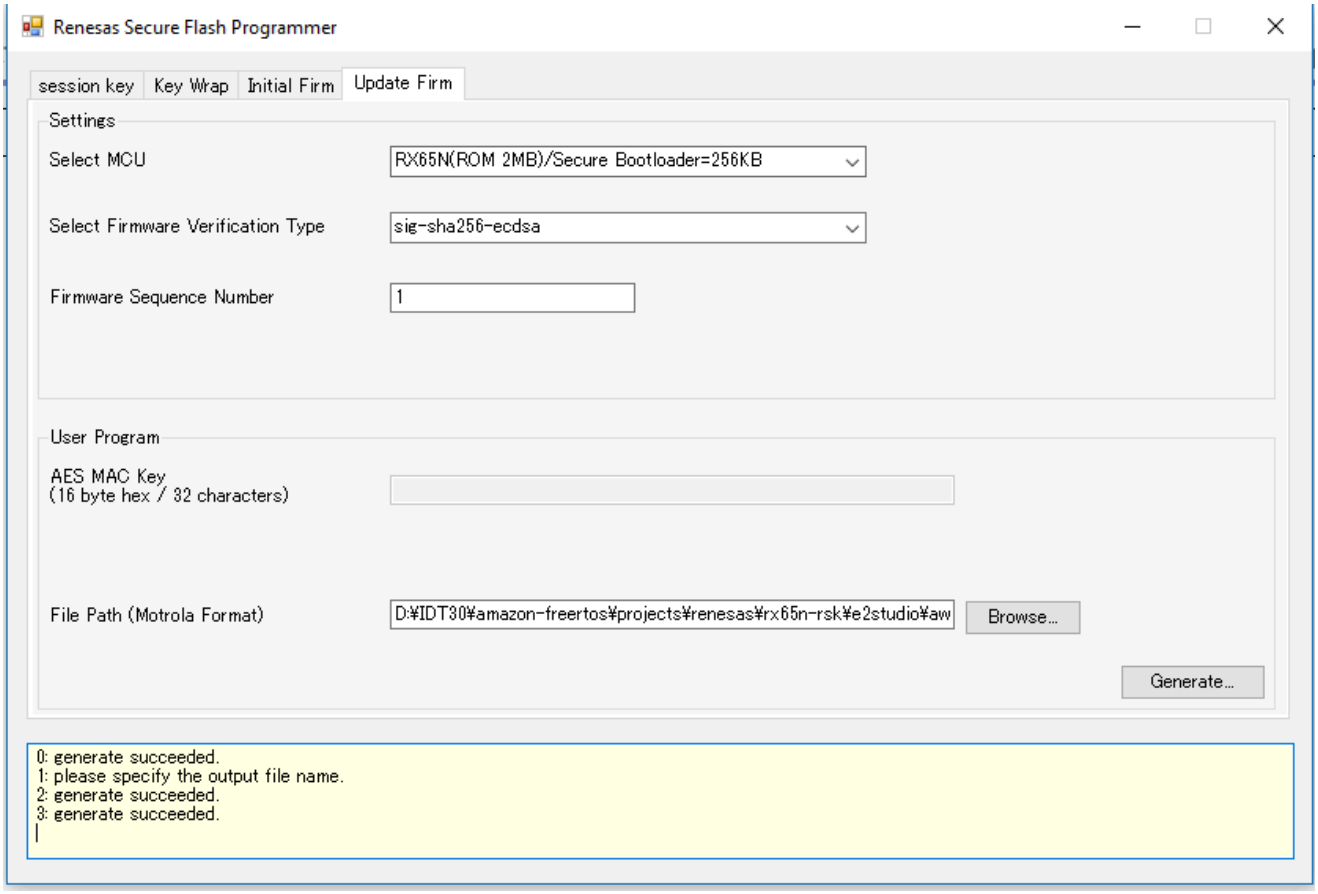
18. Erstellen Sie die `userprog.rsu` Datei mit dem Renesas Secure Flash Programmer, um die Version Ihrer Firmware zu aktualisieren.

    a. Öffnen Sie die `Amazon-FreeRTOS-Tools\Renesas Secure Flash Programmer.exe` Datei.

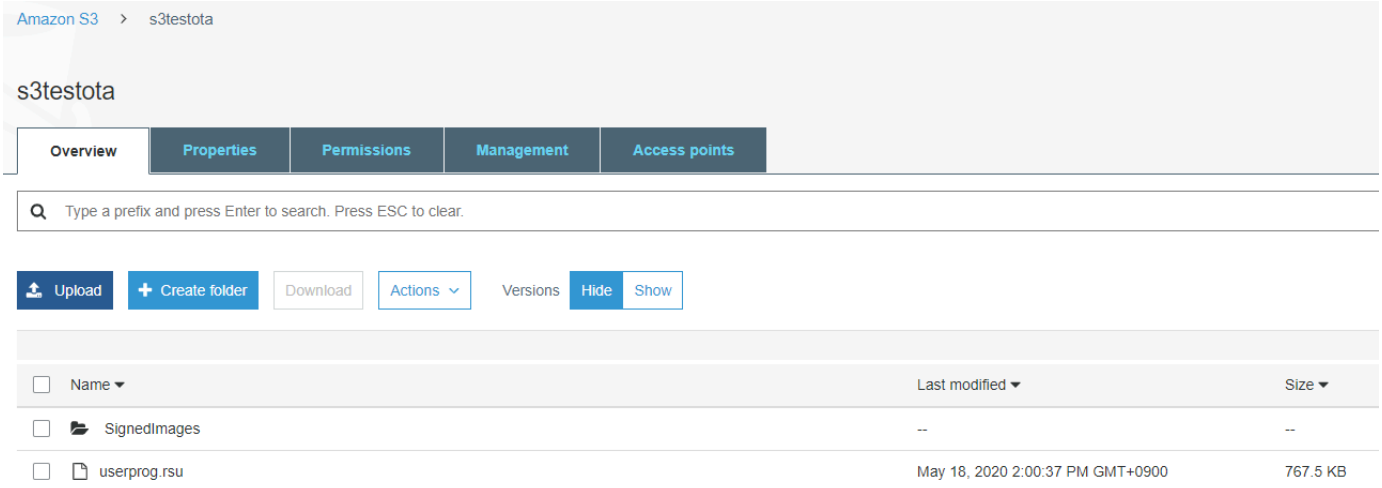
    b. Wählen Sie die Registerkarte Update Firm und legen Sie die folgenden Parameter fest:

    - Dateipfad — Der Speicherort der `aws_demos.mot` Datei (`projects\renesas\rx65n-rsk\e2studio\aws_demos\HardwareDebug`).

    c. Erstellen Sie ein Verzeichnis mit dem Namen `update _firmware`. Generieren `userprog.rsu` und speichern Sie sie im `update_firmware` Verzeichnis. Stellen Sie sicher, dass die Generierung erfolgreich war.



19. Laden Sie das Firmware-Update,`userproj.rsu`, in einen Amazon S3 S3-Bucket hoch, wie unter beschriebenErstellen Sie einen Amazon S3 S3-Bucket, um Ihr Update zu speichern.



20. Erstellen Sie einen Job zur Aktualisierung der Firmware auf dem RX65 N-RSK.

    AWS IoT Jobs ist ein Dienst, der ein oder mehrere verbundene Geräte über einen ausstehenden Job informiert. Ein Job kann verwendet werden, um eine Flotte von Geräten zu verwalten, Firmware und Sicherheitszertifikate auf Geräten zu aktualisieren oder administrative Aufgaben wie das Neustarten von Geräten und das Durchführen von Diagnosen durchzuführen.

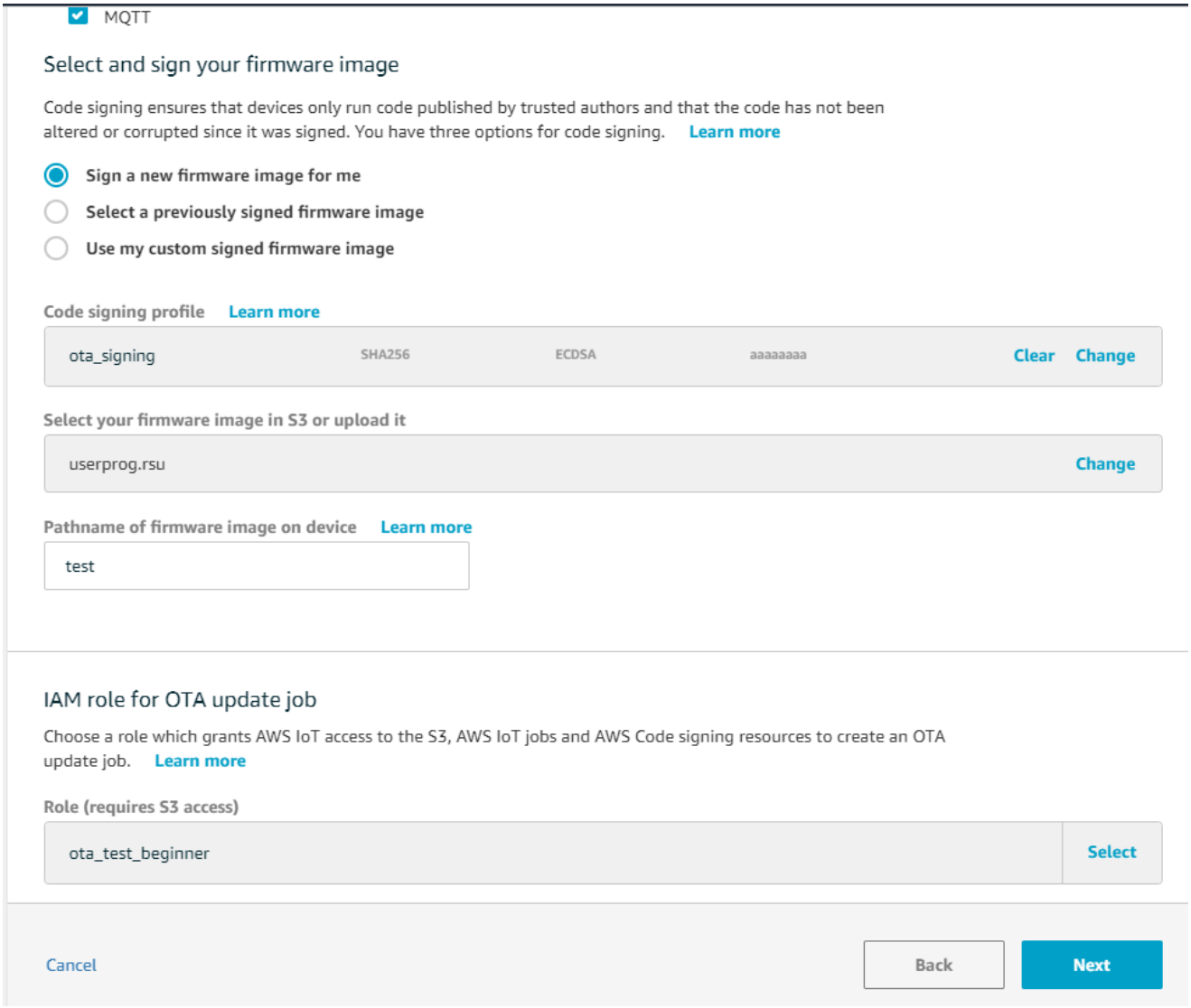
    a. Melden Sie sich bei der AWS IoT -Konsole an. Wählen Sie im Navigationsbereich „Verwalten“ und anschließend „Jobs“ aus.

    b. Wählen Sie Job erstellen und dann OTA-Aktualisierungsjob erstellen aus. Wählen Sie eine Sache aus und wählen Sie dann Weiter.

    c. Erstellen Sie einen FreeRTOS OTA-Aktualisierungsjob wie folgt:

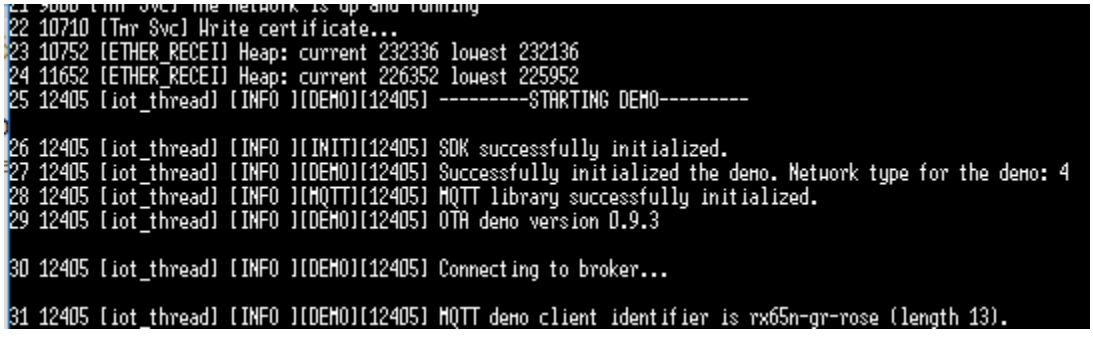
    - Wählen Sie MQTT.
    - Wählen Sie das Codesignaturprofil aus, das Sie im vorherigen Abschnitt erstellt haben.
    - Wählen Sie das Firmware-Image aus, das Sie in einen Amazon S3 S3-Bucket hochgeladen haben.
    - Geben Sie als Pfadname des Firmware-Images auf dem Gerät ein**`test`**.
    - Wählen Sie die IAM-Rolle aus, die Sie im vorherigen Abschnitt erstellt haben.

    d. Wählen Sie Weiter aus.

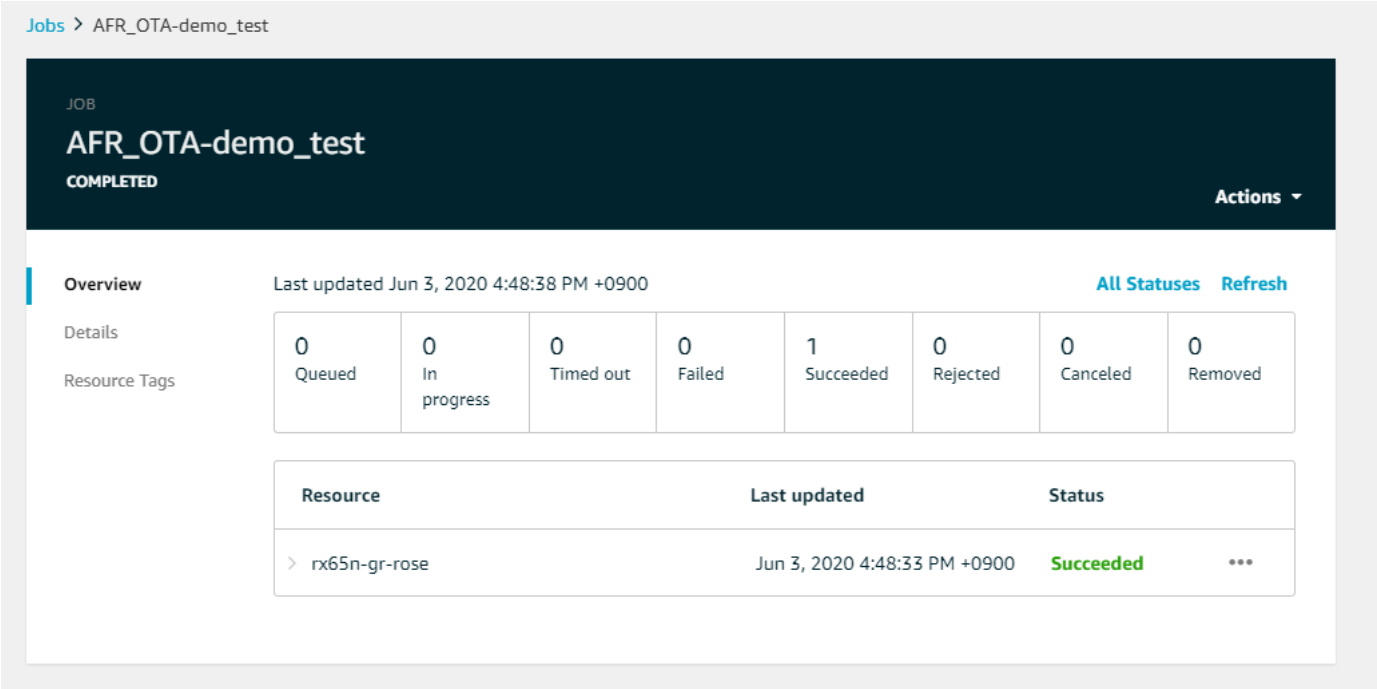


e. Geben Sie eine ID ein und wählen Sie dann Create.

21. Öffnen Sie Tera Term erneut, um zu überprüfen, ob die Firmware erfolgreich auf die OTA-Demoversion 0.9.3 aktualisiert wurde.



22. Stellen Sie auf der AWS IoT Konsole sicher, dass der Auftragsstatus Erfolgreich lautet.



Tutorial: Führen Sie OTA-Updates auf Espressif ESP32 mit FreeRTOS Bluetooth Low Energy durch

> **Important**
>
> Diese Referenzintegration wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Dieses Tutorial zeigt Ihnen, wie Sie einen ESP32 Espressif-Mikrocontroller aktualisieren, der mit einem MQTT Bluetooth Low Energy-Proxy auf einem Android-Gerät verbunden ist. Es aktualisiert das Gerät mithilfe von AWS IoT Over-the-air (OTA-) Aktualisierungsjobs. Das Gerät stellt AWS IoT mithilfe der Amazon Cognito Cognito-Anmeldeinformationen, die in der Android-Demo-App eingegeben wurden, eine Verbindung her. Ein autorisierter Betreiber initiiert das OTA-Update von der Cloud aus. Wenn das Gerät über die Android-Demo-App eine Verbindung herstellt, wird das OTA-Update initiiert und die Firmware auf dem Gerät aktualisiert.

FreeRTOS-Versionen 2019.06.00 Major und höher beinhalten Bluetooth Low Energy MQTT-Proxyunterstützung, die für die Wi-Fi-Bereitstellung und sichere Verbindungen zu Diensten verwendet

werden kann. AWS IoT Mithilfe der Bluetooth Low Energy-Funktion können Sie Geräte mit geringem Stromverbrauch erstellen, die zur Konnektivität mit einem Mobilgerät gekoppelt werden können, ohne dass WLAN erforderlich ist. Geräte können mithilfe von MQTT kommunizieren, indem sie sich über Android oder iOS Bluetooth Low Energy verbinden SDKs , die Generic Access Profile (GAP) und Generic Attributes (GATT) -Profile verwenden.

Hier sind die Schritte, die wir befolgen werden, um OTA-Updates über Bluetooth Low Energy zuzulassen:

1. Speicher konfigurieren: Erstellen Sie einen Amazon S3 S3-Bucket und Richtlinien und konfigurieren Sie einen Benutzer, der Updates durchführen kann.
2. Erstellen Sie ein Codesignaturzertifikat: Erstellen Sie ein Signaturzertifikat und ermöglichen Sie es dem Benutzer, Firmware-Updates zu signieren.
3. Amazon Cognito Cognito-Authentifizierung konfigurieren: Erstellen Sie einen Anmeldeinformationsanbieter, einen Benutzerpool und Anwendungszugriff auf den Benutzerpool.
4. FreeRTOS konfigurieren: Bluetooth Low Energy, Client-Anmeldeinformationen und das öffentliche Code-Signing-Zertifikat einrichten.
5. Eine Android-App konfigurieren: Richten Sie einen Anbieter für Anmeldeinformationen und einen Benutzerpool ein und stellen Sie die Anwendung auf einem Android-Gerät bereit.
6. Führen Sie das OTA-Aktualisierungsskript aus: Verwenden Sie das OTA-Aktualisierungsskript, um ein OTA-Update zu initiieren.

Weitere Informationen zur Funktionsweise der Updates finden Sie unterFreeRTOS RTOS-Updates Over-the-Air. Weitere Informationen zur Einrichtung der Bluetooth Low Energy MQTT-Proxyfunktion finden Sie im Beitrag Using Bluetooth Low Energy with FreeRTOS on ESP32 Espressif von Richard Kang.

Voraussetzungen

Um die Schritte in diesem Tutorial durchzuführen, benötigen Sie die folgenden Ressourcen:

- Ein ESP32 Entwicklungsboard.
- Ein MicroUSB-zu-USB-A-Kabel.
- Ein AWS Konto (das kostenlose Kontingent ist ausreichend).
- Ein Android-Telefon mit Android v 6.0 oder höher und Bluetooth-Version 4.2 oder höher.

Auf Ihrem Entwicklungscomputer benötigen Sie:

- Ausreichender Festplattenspeicher (~500 MB) für die Xtensa-Toolchain und den FreeRTOS-Quellcode und Beispiele.
- Android Studio ist installiert.
- Das AWS CLIinstallierte.
- Python3 installiert.
- Das boto3 AWS Software Developer Kit (SDK) für Python.

Bei den Schritten in diesem Tutorial wird davon ausgegangen, dass die Xtensa-Toolchain, ESP-IDF und FreeRTOS-Code im Verzeichnis in Ihrem Home-Verzeichnis installiert sind. `/esp` Sie müssen zu Ihrer Variablen etwas hinzufügen. `~/esp/xtensa-esp32-elf/bin` `$PATH`

Schritt 1: Speicher konfigurieren

1. Erstellen Sie einen Amazon S3 S3-Bucket, um Ihr Update zu speichernmit aktivierter Versionierung zum Speichern der Firmware-Images.
2. Erstellen einer OTA-Update-Servicerolleund fügen Sie der Rolle die folgenden verwalteten Richtlinien hinzu:
   - AWSIotProtokollierung
   - AWSIotRuleActions
   - AWSIotThingsRegistration
   - AWSFreeRTOSOTAUpdate
3. Erstellen Sie einen Benutzer, der OTA-Updates durchführen kann. Dieser Benutzer kann Firmware-Updates für IoT-Geräte im Konto signieren und bereitstellen und hat Zugriff auf OTA-Updates auf allen Geräten. Der Zugriff sollte auf vertrauenswürdige Entitäten beschränkt sein.
4. Folgen Sie den Schritten Erstellen einer OTA-Benutzerrichtlinie und hängen Sie es an Ihren Benutzer an.

Schritt 2: Erstellen Sie das Codesignaturzertifikat

1. Erstellen Sie einen Amazon S3 S3-Bucket mit aktivierter Versionierung, um die Firmware-Images zu speichern.

2. Erstellen Sie ein Codesignaturzertifikat, das zum Signieren der Firmware verwendet werden kann. Notieren Sie sich das Zertifikat Amazon Resource Name (ARN), wenn das Zertifikat importiert wird.

```
aws acm import-certificate --profile=ota-update-user --certificate file://
ecdsasigner.crt --private-key file://ecdsasigner.key
```

Beispielausgabe:

```
{
"CertificateArn": "arn:aws:acm:us-east-1:<account>:certificate/<certid>"
}
```

Sie werden den ARN später verwenden, um ein Signaturprofil zu erstellen. Wenn Sie möchten, können Sie das Profil jetzt mit dem folgenden Befehl erstellen:

```
aws signer put-signing-profile --profile=ota-update-user --profile-
name esp32Profile --signing-material certificateArn=arn:aws:acm:us-
east-1:account:certificate/certid --platform AmazonFreeRTOS-Default --signing-
parameters certname=/cert.pem
```

Beispielausgabe:

```
{
"arn": "arn:aws:signer::<account>:/signing-profiles/esp32Profile"
}
```

Schritt 3: Amazon Cognito Cognito-Authentifizierungskonfiguration

Erstellen Sie eine Richtlinie AWS IoT

1. Melden Sie sich bei der AWS IoT -Konsole an.
2. Wählen Sie in der oberen rechten Ecke der Konsole Mein Konto aus. Notieren Sie sich unter Kontoeinstellungen Ihre 12-stellige Konto-ID.
3. Wählen Sie im linken Navigationsbereich die Option Einstellungen aus. Notieren Sie sich unter Gerätedatenendpunkt den Endpunktwert. Der Endpunkt sollte ungefähr so lautenxxxxxxxxxxxxxx.iot.us-west-2.amazonaws.com. In diesem Beispiel ist die AWS Region „us-west-2".

4. Wählen Sie im linken Navigationsbereich Secure, dann Policies und anschließend Create aus. Wenn Sie in Ihrem Konto keine Richtlinien haben, wird die Meldung „Sie haben noch keine Richtlinien“ angezeigt, und Sie können „Richtlinie erstellen“ auswählen.
5. Geben Sie einen Namen für Ihre Richtlinie ein, zum Beispiel „esp32_mqtt_proxy_iot_policy“.
6. Wählen Sie im Abschnitt Anweisungen hinzufügen die Option Erweiterter Modus aus. Kopieren Sie die folgende JSON und fügen Sie sie in das Fenster des Richtlinien-Editors ein. Ersetze es `aws-account-id` durch deine Konto-ID und `aws-region` durch deine Region (z. B. „us-west-2").

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:Connect",
            "Resource": "arn:aws:iot:us-east-1:123456789012:*"
        },
        {
            "Effect": "Allow",
            "Action": "iot:Publish",
            "Resource": "arn:aws:iot:us-east-1:123456789012:*"
        },
        {
            "Effect": "Allow",
            "Action": "iot:Subscribe",
            "Resource": "arn:aws:iot:us-east-1:123456789012:*"
        },
        {
            "Effect": "Allow",
            "Action": "iot:Receive",
            "Resource": "arn:aws:iot:us-east-1:123456789012:*"
        }
    ]
}
```

7. Wählen Sie Erstellen aus.

Erstelle ein Ding AWS IoT

1. Melden Sie sich bei der AWS IoT -Konsole an.
2. Wählen Sie im linken Navigationsbereich Manage (Verwalten) und dann Things (Objekte).
3. Wählen Sie in der oberen rechten Ecke die Option Erstellen aus. Wenn Sie in Ihrem Konto keine Dinge registriert haben, wird die Meldung „Sie haben noch keine Dinge“ angezeigt und Sie können „Etwas registrieren“ auswählen.
4. Wählen Sie auf der Seite „ AWS IoT Dinge erstellen“ die Option „Eine einzelne Sache erstellen“ aus.
5. Geben Sie auf der Seite Gerät zur Registrierung von Dingen hinzufügen einen Namen für Ihr Ding ein (z. B. „esp32-ble“). Nur alphanumerische Zeichen, Bindestriche (-) und Unterstriche (_) sind zulässig. Wählen Sie Weiter aus.
6. Wählen Sie auf der Seite Zertifikat für Ihr Ding hinzufügen unter Zertifikat überspringen und Sache erstellen die Option Ding ohne Zertifikat erstellen aus. Da wir die mobile BLE-Proxy-App verwenden, die Amazon Cognito Cognito-Anmeldeinformationen für die Authentifizierung und Autorisierung verwendet, ist kein Gerätezertifikat erforderlich.

Erstellen Sie einen Amazon Cognito App-Client

1. Melden Sie sich bei der Amazon Cognito-Konsole an.
2. Wählen Sie im Navigationsbanner oben rechts die Option Benutzerpool erstellen aus.
3. Geben Sie den Poolnamen ein (z. B. „esp32_mqtt_proxy_user_pool“).
4. Wählen Sie Review defaults.
5. Wählen Sie unter App-Clients die Option App-Client hinzufügen und anschließend App-Client hinzufügen aus.
6. Geben Sie einen Namen für den App-Client ein (zum Beispiel „mqtt_app_client“).
7. Stellen Sie sicher, dass die Option Clientgeheimnis generieren ausgewählt ist.
8. Wählen Sie Create app client.
9. Wählen Sie Return to pool details (Zurück zu den Pool-Details).
10. Wählen Sie auf der Seite „Überprüfen“ des Benutzerpools die Option Pool erstellen aus. Sie sollten die Meldung „Ihr Benutzerpool wurde erfolgreich erstellt“ sehen. Notieren Sie sich die Pool-ID.
11. Wählen Sie im Navigationsbereich App-Clients aus.

12. Wählen Sie „Details anzeigen“. Notieren Sie sich die App-Client-ID und das App-Client-Geheimnis.

Amazon-Cognito-Identitätspool erstellen

1. Melden Sie sich bei der Amazon Cognito-Konsole an.
2. Wählen Sie Neuen Identitätspool erstellen.
3. Geben Sie einen Namen für den Identitätspool ein (z. B. „mqtt_proxy_identity_pool“).
4. Erweitern Sie Authentifizierungsanbieter.
5. Wählen Sie den Tab Cognito.
6. Geben Sie die Benutzerpool-ID und die App-Client-ID ein, die Sie sich in den vorherigen Schritten notiert haben.
7. Wählen Sie Pool erstellen.
8. Wählen Sie auf der nächsten Seite Allow aus, um neue Rollen für authentifizierte und nicht authentifizierte Identitäten zu erstellen.
9. Notieren Sie sich die Identitätspool-ID, die im folgenden Format vorliegt. `us-east-1:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`

Fügen Sie der authentifizierten Identität eine IAM-Richtlinie hinzu

1. Öffnen Sie die Amazon-Cognito-Konsole.
2. Wählen Sie den Identitätspool aus, den Sie gerade erstellt haben (z. B. „mqtt_proxy_identity_pool“).
3. Wählen Sie Edit identity pool (Identitäten-Pool bearbeiten).
4. Notieren Sie sich die der authentifizierten Rolle zugewiesene IAM-Rolle (z. B. „Cognito_MQTT_Proxy_Identity_PoolAuth_Role“).
5. Öffnen Sie die IAM-Konsole.
6. Wählen Sie im Navigationsbereich Roles aus.
7. Suchen Sie nach der zugewiesenen Rolle (z. B. „Cognito_MQTT_Proxy_Identity_PoolAuth_Role“) und wählen Sie sie dann aus.
8. Wählen Sie Inline-Richtlinie hinzufügen und anschließend JSON aus.
9. Geben Sie die folgende Richtlinie ein:

JSON

```
{
       "Version":"2012-10-17",
       "Statement": [
       {
          "Effect": "Allow",
          "Action": [
             "iot:AttachPolicy",
             "iot:AttachPrincipalPolicy",
             "iot:Connect",
             "iot:Publish",
             "iot:Subscribe"
          ],
          "Resource": "*"
       }]
    }
```

10. Wählen Sie Review policy (Richtlinie überprüfen) aus.
11. Geben Sie einen Richtliniennamen ein (z. B. „mqttProxyCognitoRichtlinie“).
12. Wählen Sie Richtlinie erstellen aus.

Schritt 4: Amazon FreeRTOS konfigurieren

1. Laden Sie die neueste Version des Amazon FreeRTOS FreeRTOS-Codes aus dem GitHub FreeRTOS-Repo herunter.
2. Um die OTA-Update-Demo zu aktivieren, folgen Sie den Schritten unter. Erste Schritte mit dem Espressif ESP32 - DevKit C und dem ESP-WROVER-KIT
3. Nehmen Sie diese zusätzlichen Änderungen in den folgenden Dateien vor:

   a. Öffnen `vendors/espressif/boards/esp32/aws_demos/config_files/aws_demo_config.h` und definieren`CONFIG_OTA_UPDATE_DEMO_ENABLED`.

   b. Öffnen `vendors/espressif/boards/esp32/aws_demos/common/config_files/aws_demo_config.h` und wechseln `democonfigNETWORK_TYPES` zu`AWSIOT_NETWORK_TYPE_BLE`.

   c. Öffnen Sie `demos/include/aws_clientcredential.h` und geben Sie Ihre Endpunkt-URL für ein`clientcredentialMQTT_BROKER_ENDPOINT`.

Geben Sie den Namen Ihres Dings für ein `clientcredentialIOT_THING_NAME` (zum Beispiel „esp32-ble“). Zertifikate müssen nicht hinzugefügt werden, wenn Sie Amazon Cognito Cognito-Anmeldeinformationen verwenden.

d. Öffnen `vendors/espressif/boards/esp32/aws_demos/config_files/aws_iot_network_config.h` und ändern `configSUPPORTED_NETWORKS` und nur `configENABLED_NETWORKS` `AWSIOT_NETWORK_TYPE_BLE` zum Einbeziehen.

e. Öffnen Sie die `vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/ota_demo_config.h` Datei und geben Sie Ihr Zertifikat ein.

```
#define otapalconfigCODE_SIGNING_CERTIFICATE [] = "your-certificate-key";
```

Die Anwendung sollte starten und die Demoversion drucken:

```
11 13498 [iot_thread] [INFO ][DEMO][134980] Successfully initialized the demo.
 Network type for the demo: 2
12 13498 [iot_thread] [INFO ][MQTT][134980] MQTT library successfully initialized.
13 13498 [iot_thread] OTA demo version 0.9.20
14 13498 [iot_thread] Creating MQTT Client...
```

Schritt 5: Konfigurieren Sie eine Android-App

1. Laden Sie das Android Bluetooth Low Energy SDK und eine Beispiel-App aus dem amazon-freertos-ble-android GitHub -sdk-Repo herunter.
2. Öffnen Sie die Datei `app/src/main/res/raw/awsconfiguration.json` und geben Sie die Pool-ID, die Region und AppClientSecret die Anweisungen im folgenden JSON-Beispiel ein. AppClientId

```
{
  "UserAgent": "MobileHub/1.0",
  "Version": "1.0",
  "CredentialsProvider": {
    "CognitoIdentity": {
      "Default": {
        "PoolId": "Cognito->Manage Identity Pools->Federated Identities->mqtt_proxy_identity_pool->Edit Identity Pool->Identity Pool ID",
        "Region": "Your region (for example us-east-1)"
      }
```

```
    }
  },

  "IdentityManager": {
    "Default": {}
  },

  "CognitoUserPool": {
    "Default": {
      "PoolId": "Cognito-> Manage User Pools -> esp32_mqtt_proxy_user_pool ->
 General Settings -> PoolId",
      "AppClientId": "Cognito-> Manage User Pools -> esp32_mqtt_proxy_user_pool ->
 General Settings -> App clients ->Show Details",
      "AppClientSecret": "Cognito-> Manage User Pools -> esp32_mqtt_proxy_user_pool
 -> General Settings -> App clients ->Show Details",
      "Region": "Your region (for example us-east-1)"
    }
  }
}
```

3. Öffnen Sie den zuvor erstellten Richtliniennamen (z. B.*esp32_mqtt_proxy_iot_policy*) `app/src/main/java/software/amazon/freertos/DemoConstants.java` und geben Sie auch die Region (z. B.*us-east-1*) ein.
4. Erstellen und installieren Sie die Demo-App.
   a. Wählen Sie in Android Studio Build und dann Make Module App aus.
   b. Wählen Sie „Ausführen“ und dann „App ausführen“. Sie können in Android Studio zum Logcat-Fenster wechseln, um Protokollnachrichten zu überwachen.
   c. Erstellen Sie auf dem Android-Gerät über den Anmeldebildschirm ein Konto.
   d. Erstellen eines Benutzers. Wenn ein Benutzer bereits existiert, geben Sie die Anmeldeinformationen ein.
   e. Erlauben Sie der Amazon FreeRTOS-Demo, auf den Standort des Geräts zuzugreifen.
   f. Suchen Sie nach Bluetooth Low Energy-Geräten.
   g. Stellen Sie den Schieberegler für das gefundene Gerät auf Ein.
   h. Drücken Sie Y an der Debug-Konsole für die ESP32 serielle Schnittstelle.
   i. Wählen Sie Pair & Connect.

5. Je mehr... Der Link wird aktiv, nachdem die Verbindung hergestellt wurde. Der Verbindungsstatus sollte sich im Logcat des Android-Geräts auf „BLE_CONNECTED“ ändern, wenn die Verbindung hergestellt ist:

```
2019-06-06 20:11:32.160 23484-23497/software.amazon.freertos.demo I/FRD: BLE
 connection state changed: 0; new state: BLE_CONNECTED
```

6. Bevor die Nachrichten übertragen werden können, handeln das Amazon FreeRTOS FreeRTOS-Gerät und das Android-Gerät die MTU aus. Sie sollten die folgende Ausgabe in Logcat sehen:

```
2019-06-06 20:11:46.720 23484-23497/software.amazon.freertos.demo I/FRD:
 onMTUChanged : 512 status: Success
```

7. Das Gerät stellt eine Verbindung zur App her und beginnt, MQTT-Nachrichten über den MQTT-Proxy zu senden. Um zu bestätigen, dass das Gerät kommunizieren kann, stellen Sie sicher, dass sich der Wert der charakteristischen Daten von MQTT_CONTROL auf 01 ändert:

```
2019-06-06 20:12:28.752 23484-23496/software.amazon.freertos.demo D/FRD: <-<-<-
 Writing to characteristic: MQTT_CONTROL with data: 01
2019-06-06 20:12:28.839 23484-23496/software.amazon.freertos.demo D/FRD:
 onCharacteristicWrite for: MQTT_CONTROL; status: Success; value: 01
```

8. Wenn die Geräte gekoppelt sind, wird auf der Konsole eine Aufforderung angezeigt. ESP32 Um BLE zu aktivieren, drücken Sie Y. Die Demo funktioniert erst, wenn Sie diesen Schritt ausführen.

```
E (135538) BT_GATT: GATT_INSUF_AUTHENTICATION: MITM Required
W (135638) BT_L2CAP: l2cble_start_conn_update, the last connection update command
 still pending.
E (135908) BT_SMP: Value for numeric comparison = 391840
15 13588 [InputTask] Numeric comparison:391840
16 13589 [InputTask] Press 'y' to confirm
17 14078 [InputTask] Key accepted
W (146348) BT_SMP: FOR LE SC LTK IS USED INSTEAD OF STK
18 16298 [iot_thread] Connecting to broker...
19 16298 [iot_thread] [INFO ][MQTT][162980] Establishing new MQTT connection.
20 16298 [iot_thread] [INFO ][MQTT][162980] (MQTT connection 0x3ffd5754, CONNECT
 operation 0x3ffd586c) Waiting for operation completion.
21 16446 [iot_thread] [INFO ][MQTT][164450] (MQTT connection 0x3ffd5754, CONNECT
 operation 0x3ffd586c) Wait complete with result SUCCESS.
22 16446 [iot_thread] [INFO ][MQTT][164460] New MQTT connection 0x3ffc0ccc
 established.
```

```
23 16446 [iot_thread] Connected to broker.
```

Schritt 6: Führen Sie das OTA-Aktualisierungsskript aus

1. Führen Sie die folgenden Befehle aus, um die Voraussetzungen zu installieren:

   ```
   pip3 install boto3
   ```

   ```
   pip3 install pathlib
   ```

2. Inkrementieren Sie die FreeRTOS-Anwendungsversion in. `demos/include/aws_application_version.h`
3. Erstellen Sie eine neue .bin-Datei.
4. Laden Sie das Python-Skript start_ota.py herunter. Um den Inhalt der Hilfe für das Skript zu sehen, führen Sie den folgenden Befehl in einem Terminalfenster aus:

   ```
   python3 start_ota.py -h
   ```

   Dies sollte etwa wie folgt aussehen:

   ```
   usage: start_ota.py [-h] --profile PROFILE [--region REGION]
                       [--account ACCOUNT] [--devicetype DEVICETYPE] --name NAME
                       --role ROLE --s3bucket S3BUCKET --otasigningprofile
                       OTASIGNINGPROFILE --signingcertificateid
                       SIGNINGCERTIFICATEID [--codelocation CODELOCATION]
   Script to start OTA update
   optional arguments:
   -h, --help            show this help message and exit
   --profile PROFILE     Profile name created using aws configure
   --region REGION       Region
   --account ACCOUNT     Account ID
   --devicetype DEVICETYPE thing|group
   --name NAME           Name of thing/group
   --role ROLE           Role for OTA updates
   --s3bucket S3BUCKET   S3 bucket to store firmware updates
   --otasigningprofile OTASIGNINGPROFILE
                         Signing profile to be created or used
   --signingcertificateid SIGNINGCERTIFICATEID
                         certificate id (not arn) to be used
   ```

```
--codelocation CODELOCATION
                      base folder location (can be relative)
```

5. Wenn Sie die bereitgestellte CloudFormation Vorlage zum Erstellen von Ressourcen verwendet haben, führen Sie den folgenden Befehl aus:

```
python3 start_ota_stream.py --profile otausercf --name esp32-ble --role
 ota_ble_iot_role-sample --s3bucket afr-ble-ota-update-bucket-sample --
otasigningprofile abcd --signingcertificateid certificateid
```

   Sie sollten den Start des Updates in der ESP32 Debug-Konsole sehen:

```
38 2462 [OTA Task] [prvParseJobDoc] Job was accepted. Attempting to start transfer.
---
49 2867 [OTA Task] [prvIngestDataBlock] Received file block 1, size 1024
50 2867 [OTA Task] [prvIngestDataBlock] Remaining: 1290
51 2894 [OTA Task] [prvIngestDataBlock] Received file block 2, size 1024
52 2894 [OTA Task] [prvIngestDataBlock] Remaining: 1289
53 2921 [OTA Task] [prvIngestDataBlock] Received file block 3, size 1024
54 2921 [OTA Task] [prvIngestDataBlock] Remaining: 1288
55 2952 [OTA Task] [prvIngestDataBlock] Received file block 4, size 1024
56 2953 [OTA Task] [prvIngestDataBlock] Remaining: 1287
57 2959 [iot_thread] State: Active  Received: 5   Queued: 5   Processed: 5
 Dropped: 0
```

6. Wenn das OTA-Update abgeschlossen ist, wird das Gerät gemäß den Anforderungen des OTA-Aktualisierungsvorgangs neu gestartet. Es versucht dann, mit der aktualisierten Firmware eine Verbindung herzustellen. Wenn das Upgrade erfolgreich war, ist die aktualisierte Firmware als aktiv markiert und Sie sollten die aktualisierte Version in der Konsole sehen:

```
13 13498 [iot_thread] OTA demo version 0.9.21
```

AWS IoT Device Shadow-Demoanwendung

> **Important**
>
> Diese Demo wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-

FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Einführung

Diese Demo zeigt, wie Sie die AWS IoT Device Shadow-Bibliothek verwenden, um eine Verbindung zum AWS Device Shadow-Dienst herzustellen. Es verwendet denCoreMQTT-Bibliothek, um eine MQTT-Verbindung mit TLS (Mutual Authentication) zum AWS IoT MQTT-Broker herzustellen, und den CoreJSON-Bibliotheksparser, um vom Shadow-Dienst empfangene Shadow-Dokumente zu analysieren. AWS Die Demo zeigt grundlegende Shadow-Operationen, z. B. wie man ein Shadow-Dokument aktualisiert und wie man ein Shadow-Dokument löscht. Die Demo zeigt auch, wie man eine Callback-Funktion in der CoreMQTT-Bibliothek registriert, um Nachrichten wie den Shadow `/update` und `/update/delta` Nachrichten, die vom AWS IoT Device Shadow-Dienst gesendet werden, zu verarbeiten.

Diese Demo ist nur als Lernübung gedacht, da die Anfrage zur Aktualisierung des Shadow-Dokuments (Status) und die Aktualisierungsantwort von derselben Anwendung ausgeführt werden. In einem realistischen Produktionsszenario würde eine externe Anwendung per Fernzugriff eine Aktualisierung des Gerätestatus anfordern, auch wenn das Gerät derzeit nicht verbunden ist. Das Gerät bestätigt die Aktualisierungsanforderung, sobald es verbunden ist.

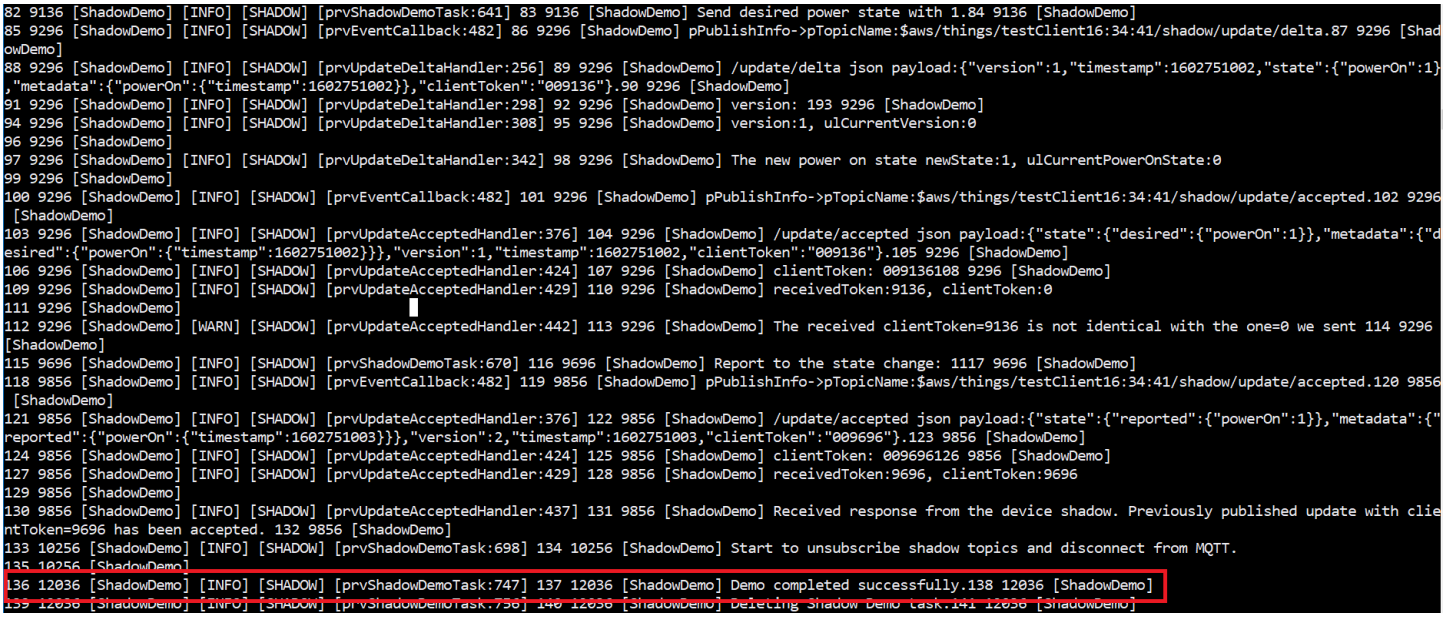
> **Note**
>
> Folgen Sie den Schritten unter, um die FreeRTOS-Demos einzurichten und auszuführen. Erste Schritte mit FreeRTOS

Funktionalität

Die Demo erstellt eine einzelne Anwendungsaufgabe, die eine Reihe von Beispielen durchläuft, die Shadow `/update` und `/update/delta` Callbacks demonstrieren, um das Umschalten des Status eines Remote-Geräts zu simulieren. Es sendet ein Shadow-Update mit dem neuen `desired` Status und wartet darauf, dass das Gerät seinen `reported` Status als Reaktion auf den neuen Status ändert. `desired` Darüber hinaus wird ein `/update` Shadow-Callback verwendet, um die sich ändernden Shadow-Zustände zu drucken. Diese Demo verwendet auch eine sichere MQTT-Verbindung zum AWS IoT MQTT-Broker und geht davon aus, dass im Geräteshadow ein `power0n` Status vorliegt.

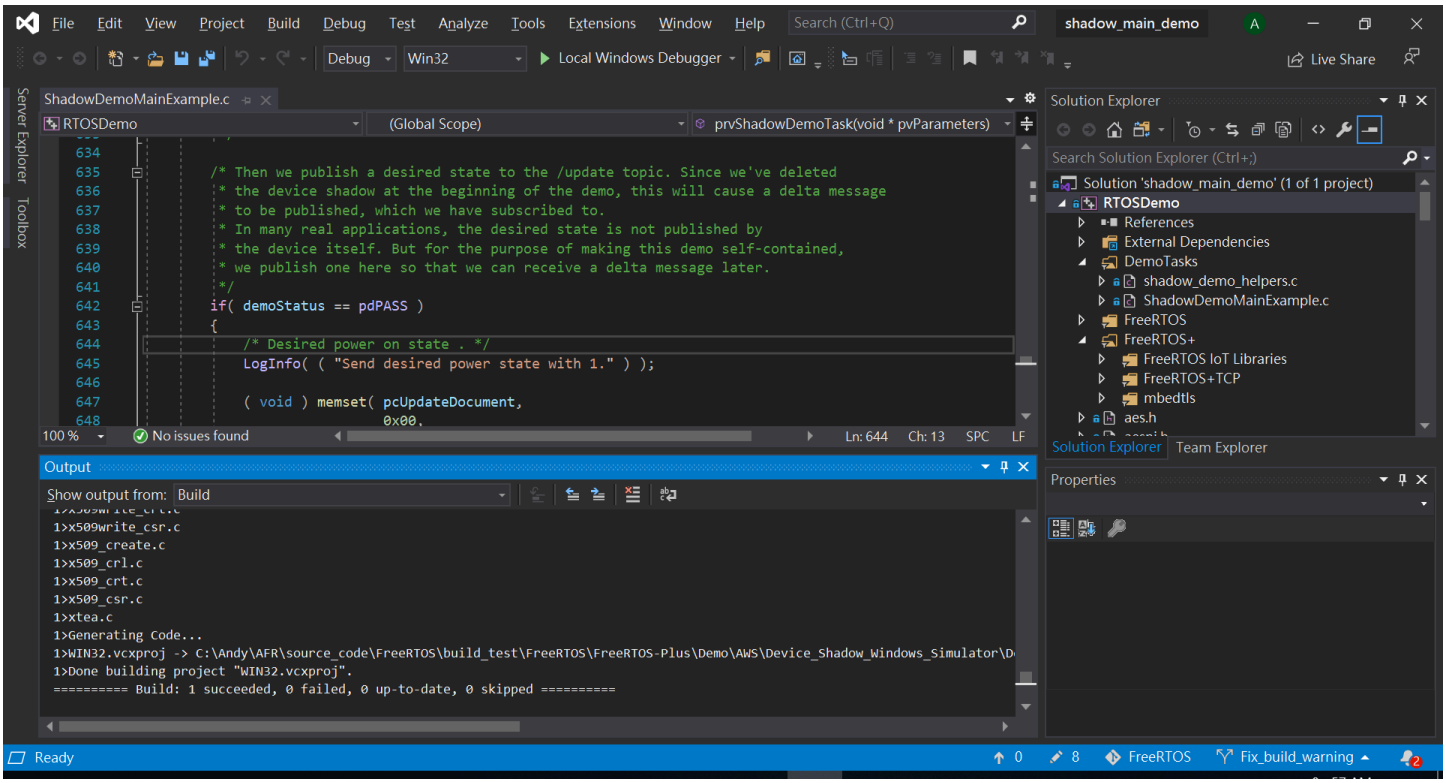
Die Demo führt die folgenden Operationen aus:

1. Stellen Sie mithilfe der Hilfsfunktionen in `shadow_demo_helpers.c` eine MQTT-Verbindung her.
2. Stellen Sie MQTT-Themenzeichenfolgen für Device Shadow-Operationen zusammen, indem Sie Makros verwenden, die von der AWS IoT Device Shadow-Bibliothek definiert sind.
3. Veröffentlichen Sie das MQTT-Thema, das zum Löschen eines Device-Shadows verwendet wurde, um alle vorhandenen Device-Shadow zu löschen.
4. Abonnieren Sie die MQTT-Themen für `/update/delta` `/update/accepted` und zur `/update/rejected` Verwendung von Hilfsfunktionen in. `shadow_demo_helpers.c`
5. Veröffentlichen Sie den gewünschten Status der `powerOn` Verwendung von Hilfsfunktionen in`shadow_demo_helpers.c`. Dadurch wird eine `/update/delta` Nachricht an das Gerät gesendet.
6. Verarbeiten Sie eingehende MQTT-Nachrichten und ermitteln Sie mithilfe einer in `prvEventCallback` der Device Shadow-Bibliothek (`Shadow_MatchTopic`) definierten Funktion, ob die Nachricht mit dem AWS IoT Device Shadow verknüpft ist. Wenn es sich bei der Nachricht um eine `/update/delta` Device-Shadow-Nachricht handelt, veröffentlicht die Haupt-Demofunktion eine zweite Nachricht, auf die der gemeldete Status aktualisiert wird. `powerOn` Wenn eine `/update/accepted` Nachricht empfangen wird, stellen Sie sicher, dass sie dieselbe `clientToken` Nachricht enthält, die zuvor in der Aktualisierungsnachricht veröffentlicht wurde. Das wird das Ende der Demo bedeuten.



Die Demo finden Sie in der Datei *freertos*/demos/device_shadow_for_aws/shadow_demo_main.c oder auf GitHub.

Der folgende Screenshot zeigt die erwartete Ausgabe, wenn die Demo erfolgreich ist.



Connect zum AWS IoT MQTT-Broker her

Um eine Verbindung zum AWS IoT MQTT-Broker herzustellen, verwenden wir dieselbe Methode wie `MQTT_Connect()` in der. Demo zur gegenseitigen CoreMQTT-Authentifizierung

Löschen Sie das Schattendokument

Um das Shadow-Dokument zu löschen, rufen Sie `xPublishToTopic` mit einer leeren Nachricht auf und verwenden Sie dabei Makros, die in der AWS IoT Device Shadow-Bibliothek definiert sind. Dies wird verwendet`MQTT_Publish`, um zum `/delete` Thema zu veröffentlichen. Der folgende Codeabschnitt zeigt, wie dies in der Funktion gemacht wird`prvShadowDemoTask`.

```
/* First of all, try to delete any Shadow document in the cloud. */
returnStatus = PublishToTopic( SHADOW_TOPIC_STRING_DELETE( THING_NAME ),
                               SHADOW_TOPIC_LENGTH_DELETE( THING_NAME_LENGTH ),
                               pcUpdateDocument,
                               0U );
```

Abonnieren Sie Schattenthemen

Abonnieren Sie die Device Shadow-Themen, um Benachrichtigungen vom AWS IoT Broker über Shadow-Änderungen zu erhalten. Die Device Shadow-Themen werden anhand von Makros zusammengestellt, die in der Device Shadow-Bibliothek definiert sind. Der folgende Codeabschnitt zeigt, wie dies in der `prvShadowDemoTask` Funktion erfolgt.

```
/* Then try to subscribe shadow topics. */
if( returnStatus == EXIT_SUCCESS )
{
    returnStatus = SubscribeToTopic(
                     SHADOW_TOPIC_STRING_UPDATE_DELTA( THING_NAME ),
                     SHADOW_TOPIC_LENGTH_UPDATE_DELTA( THING_NAME_LENGTH ) );
}

if( returnStatus == EXIT_SUCCESS )
{
    returnStatus = SubscribeToTopic(
                     SHADOW_TOPIC_STRING_UPDATE_ACCEPTED( THING_NAME ),
                     SHADOW_TOPIC_LENGTH_UPDATE_ACCEPTED( THING_NAME_LENGTH ) );
}

if( returnStatus == EXIT_SUCCESS )
{
    returnStatus = SubscribeToTopic(
                     SHADOW_TOPIC_STRING_UPDATE_REJECTED( THING_NAME ),
                     SHADOW_TOPIC_LENGTH_UPDATE_REJECTED( THING_NAME_LENGTH ) );
}
```

Shadow-Updates senden

Um ein Shadow-Update zu senden, ruft die Demo `xPublishToTopic` mit einer Nachricht im JSON-Format auf und verwendet Makros, die von der Device Shadow-Bibliothek definiert wurden. Dies wird verwendet`MQTT_Publish`, um zum `/delete` Thema zu veröffentlichen. Der folgende Codeabschnitt zeigt, wie dies in der `prvShadowDemoTask` Funktion gemacht wird.

```
#define SHADOW_REPORTED_JSON    \
    "{"                         \
    "\"state\":{"               \
    "\"reported\":{"            \
```

```
    "\"powerOn\":%01d"          \
    "}"                         \
    "},"                        \
    "\"clientToken\":\"%06lu\"" \
    "}"
snprintf( pcUpdateDocument,
          SHADOW_REPORTED_JSON_LENGTH + 1,
          SHADOW_REPORTED_JSON,
           ( int ) ulCurrentPowerOnState,
           ( long unsigned ) ulClientToken );

xPublishToTopic( SHADOW_TOPIC_STRING_UPDATE( THING_NAME ),
                 SHADOW_TOPIC_LENGTH_UPDATE( THING_NAME_LENGTH ),
                 pcUpdateDocument,
                 ( SHADOW_DESIRED_JSON_LENGTH + 1 ) );
```

Verarbeiten Sie Shadow-Delta-Nachrichten und Shadow-Update-Nachrichten

Die Benutzer-Callback-Funktion, die mithilfe der `MQTT_Init` Funktion in der CoreMQTT-Clientbibliothek registriert wurde, benachrichtigt uns über ein eingehendes Paketereignis. Sehen Sie sich die Callback-Funktion an. prvEventCallback GitHub

Die Callback-Funktion bestätigt, dass das eingehende Paket vom Typ ist`MQTT_PACKET_TYPE_PUBLISH`, und verwendet die Device Shadow Library-API, `Shadow_MatchTopic` um zu bestätigen, dass es sich bei der eingehenden Nachricht um eine Shadow-Nachricht handelt.

Wenn es sich bei der eingehenden Nachricht um eine Shadow-Nachricht vom Typ handelt`ShadowMessageTypeUpdateDelta`, rufen wir prvUpdateDeltaHandler auf, um diese Nachricht zu verarbeiten. Der Handler `prvUpdateDeltaHandler` verwendet die CoreJSON-Bibliothek, um die Nachricht zu analysieren, um den Deltawert für den `powerOn` Status zu ermitteln, und vergleicht diesen mit dem aktuellen Gerätestatus, der lokal verwaltet wird. Wenn diese unterschiedlich sind, wird der lokale Gerätestatus aktualisiert, um den neuen Wert des `powerOn` Status aus dem Schattendokument widerzuspiegeln.

Wenn es sich bei der eingehenden Nachricht um eine Shadow-Nachricht mit Typ handelt`ShadowMessageTypeUpdateAccepted`, rufen wir prvUpdateAcceptedHandler auf, um diese Nachricht zu bearbeiten. Der Handler `prvUpdateAcceptedHandler` analysiert die Nachricht mithilfe der CoreJSON-Bibliothek, um sie `clientToken` aus der Nachricht abzurufen. Diese Handlerfunktion überprüft, ob das Client-Token aus der JSON-Nachricht mit dem von der Anwendung

verwendeten Client-Token übereinstimmt. Wenn es nicht übereinstimmt, protokolliert die Funktion eine Warnmeldung.

Secure Sockets Echo-Client-Demo

> **Important**
>
> Diese Demo wird im Amazon-FreeRTOS-Repository gehostet, das veraltet ist. Wir empfehlen, dass Sie hier beginnen, wenn Sie ein neues Projekt erstellen. Wenn Sie bereits ein vorhandenes FreeRTOS-Projekt haben, das auf dem inzwischen veralteten Amazon-FreeRTOS-Repository basiert, finden Sie weitere Informationen unter. Leitfaden zur Migration des Amazon-FreerTOS Github-Repositorys

Das folgende Beispiel verwendet eine einzelne RTOS-Task. Der Quellcode für dieses Beispiel befindet sich unter `demos/tcp/aws_tcp_echo_client_single_task.c`.

Bevor Sie beginnen, stellen Sie sicher, dass Sie FreeRTOS auf Ihren Mikrocontroller heruntergeladen und die FreeRTOS-Demo-Projekte erstellt und ausgeführt haben. Sie können FreeRTOS von klonen oder herunterladen. GitHub Anweisungen finden Sie in der Datei README.md.

So führen Sie die Demo aus:

> **Note**
>
> Folgen Sie den Schritten unter, um die FreeRTOS-Demos einzurichten und auszuführen. Erste Schritte mit FreeRTOS
> Die TCP-Server- und Client-Demos werden derzeit von den Cypress CYW9439 07 AEVAL1 F und CYW9549 07 F Development Kits nicht unterstützt. AEVAL1

1. Folgen Sie den Anweisungen unter Einrichten des TLS Echo Servers im FreeRTOS Porting Guide.

   Ein TLS-Echo-Server sollte ausgeführt werden und den Port 9000 überwachen.

   Während der Einrichtung sollten Sie vier Dateien erstellt haben:

   - `client.pem` (Clientzertifikat)

- `client.key` (privater Clientschlüssel)
- `server.pem` (Serverzertifikat)
- `server.key` (privater Serverschlüssel)

2. Verwenden Sie das Tool `tools/certificate_configuration/CertificateConfigurator.html` zum Kopieren des Clientzertifikats (`client.pem`) und des privaten Clientschlüssels (`client.key`) nach `aws_clientcredential_keys.h`.
3. Öffnen Sie die `FreeRTOSConfig.h` Datei.
4. Legen Sie die Variablen `configECHO_SERVER_ADDR0`, `configECHO_SERVER_ADDR1`, `configECHO_SERVER_ADDR2` und `configECHO_SERVER_ADDR3` für die vier Ganzzahlen fest, aus denen die IP-Adresse besteht, auf welcher der TLS Echo Server ausgeführt wird.
5. Legen Sie die Variable `configTCP_ECHO_CLIENT_PORT` auf `9000` fest, den Port, auf dem der TLS Echo Server verwendet wird.
6. Setzen Sie die Variable `configTCP_ECHO_TASKS_SINGLE_TASK_TLS_ENABLED` auf `1`.
7. Verwenden Sie das Tool `tools/certificate_configuration/PEMfileToCString.html` zum Kopieren des Serverzertifikats (`server.pem`) nach `cTlsECHO_SERVER_CERTIFICATE_PEM` in der Datei `aws_tcp_echo_client_single_task.c`.
8. Öffne*`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/aws_demo_config.h`, kommentiere und definiere `CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` oder. `#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED` `CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`

Der Mikrocontroller und der TLS Echo Server sollten sich im selben Netzwerk befinden. Wenn die Demo gestartet wird (`main.c`), sollten die Protokollnachricht `Received correct string from echo server` angezeigt werden.

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.