aws

Guide de l’utilisateur

FreeRTOS



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

FreeRTOS: Guide de l’utilisateur

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

Les marques et la présentation commerciale d’Amazon ne peuvent être utilisées en relation avec un produit ou un service qui n’est pas d’Amazon, d’une manière susceptible de créer une confusion parmi les clients, ou d’une manière qui dénigre ou discrédite Amazon. Toutes les autres marques commerciales qui ne sont pas la propriété d’Amazon appartiennent à leurs propriétaires respectifs, qui peuvent ou non être affiliés ou connectés à Amazon, ou sponsorisés par Amazon.

Table of Contents

Qu'est-ce que FreeRTOS ? 1
Inscrivez-vous pour un Compte AWS 1
Télécharger le code source de FreeRTOS 1
FreeRTOS-qualified plateformes matérielles 2
Ressources supplémentaires 3
Versions de FreeRTOS 3
Support à long terme pour FreeRTOS 3
Plan de maintenance étendu FreeRTOS 4
Architecture FreeRTOS 4
Flux de travail du développement 5
Principes fondamentaux du noyau FreeRTOS 6
Le planificateur du noyau FreeRTOS 6
Allocation mémoire du noyau 7
Gérer la mémoire des applications 8
Coordination inter-tâches 8
Files d'attente 9
Sémaphores et mutex 9
Direct-to-task notifications 10
Tampons mémoire de flux 10
Tampons de messages 12
Support du multitraitement symétrique (SMP) 14
Minuteurs de logiciels 14
Prise en charge d'une alimentation basse 15
`FreeRTOSConfig.h` 15
AWS IoT SDK de périphérique pour Embedded C 16
Comprendre les API FreeRTOS Common IO 17
Bibliothèques 17
Common IO - basique 17
E/S communes - BLE 19
Common IO pour Amazon Common Software 19
Qu'est-ce que l'ACS ? 20
Programme de qualification 20
Commencez avec FreeRTOS 21
Commencez avec Quick Connect 21

Explorez les bibliothèques FreeRTOS 21
Construisez un environnement sûr et robuste AWS IoT product 22
Développez votre AWS IoT vigie 22
AWS IoT Device Tester pour FreeRTOS 23
Suite de qualifications FreeRTOS 23
Comprendre les suites de tests personnalisées 24
Versions prises en charge d'IDT pour FreeRTOS 25
Dernière version de IDT pour FreeRTOS 25
Versions IDT antérieures 28
Versions IDT non prises en charge 33
Télécharger IDT pour FreeRTOS 68
Télécharger IDT manuellement 69
Téléchargez IDT par programmation 69
IDT avec suite de qualification FreeRTOS 2.0 (FRQ 2.0) 75
Configurer les prérequis de qualification LTS 76
Premier test de votre carte microcontrôleur 84
Utiliser l'interface utilisateur IDT pour exécuter la suite de qualification FreeRTOS 101
Exécutez la suite de qualification FreeRTOS 2.0 117
Afficher les résultats IDT pour FreeRTOS 120
Interpréter l'IDT pour les résultats FreeRTOS 121
Afficher l'IDT pour FreeRtoSlogs 124
IDT avec suite de qualification FreeRTOS 1.0 (FRQ 1.0) 124
Configuration des prérequis de qualification 1.0 126
Premier test de votre carte microcontrôleur 129
Utiliser l'interface utilisateur IDT pour exécuter la suite de qualification FreeRTOS 149
Exécuter des tests Bluetooth Low Energy 158
Exécutez la suite de qualification FreeRTOS 164
Afficher les résultats de l'IDT pour FreeRTOS 170
Interpréter l'IDT pour les résultats FreeRTOS 171
Afficher les logs IDT pour FreeRTOS 174
Développez et exécutez vos propres suites de tests IDT 174
Téléchargez la dernière version d'IDT pour FreeRTOS 175
Workflow de la suite de tests 175
Tutoriel : création et exécution de l'exemple de suite de tests IDT 176
Tutoriel : Développement d'une suite de tests IDT simple 182
Versions de la suite de tests 269

Résolution des erreurs 270
Résoudre les problèmes de configuration de l'appareil 271
Résoudre les erreurs liées au délai d'expiration 285
Fonctionnalité cellulaire et AWS frais 286
Politique de génération de rapports de qualification 286
AWS Politique gérée pour AWS IoT Device Tester 286
Stratégie gérée 287
Mises à jour des politiques 294
Politique de prise en charge 296
Sécurité dans AWS 298
Gestion de l’identité et des accès 298
Public ciblé 299
Authentification par des identités 299
Gestion de l’accès à l’aide de politiques 301
Comment FreeRTOS fonctionne avec IAM 303
Identity-based exemples de politiques 308
Résolution des problèmes 311
Validation de conformité 313
Résilience 314
Sécurité de l’infrastructure 314
Amazon-FreeRTOS Guide de migration du référentiel Github 316
Annexe 316
Archivage 323
Archive du guide de l'utilisateur de FreeRTOS 323
Contenu du précédent guide de l'utilisateur de FreeRTOS 323
Commencez avec FreeRTOS 323
Over-the-Air mises à jour 519
Bibliothèques FreeRTOS 606
Démos de FreeRTOS 675
..... dccxcviii

Qu'est-ce que FreeRTOS ?

Développé en partenariat avec les principaux fabricants de puces au monde sur une période de 15 ans, et désormais téléchargé toutes les 170 secondes, FreeRTOS est un système d'exploitation en temps réel (RTOS) leader du marché pour les microcontrôleurs et les petits microprocesseurs. Distribué gratuitement sous la licence open source du MIT, FreeRTOS inclut un noyau et un ensemble croissant de bibliothèques adaptées à une utilisation dans tous les secteurs de l'industrie. FreeRTOS est construit avec la fiabilité et la facilité d'utilisation en ligne de mire.

FreeRTOS inclut des bibliothèques pour la connectivité, la sécurité et les mises à jour par voie hertzienne (OTA). FreeRTOS inclut également des applications de démonstration qui montrent les fonctionnalités FreeRTOS sur les cartes qualifiées.

FreeRTOS est un projet open source. Vous pouvez télécharger le code source, apporter des modifications ou des améliorations, ou signaler des problèmes sur le GitHub site à l'adresse https://github.com/FreeRTOS/FreeRTOS.

Nous publions le code FreeRTOS sous la licence open source du MIT, afin que vous puissiez l'utiliser dans des projets commerciaux et personnels.

Nous accueillons également les contributions à la documentation FreeRTOS (guide de l'utilisateur de FreeRTOS, guide de portage de FreeRTOS et guide de qualification de FreeRTOS). Pour consulter la source Markdown de la documentation, consultez https://github.com/awsdocs/aws-freertos-docs. Il est publié sous licence Creative Commons (CC BY-ND).

Inscrivez-vous pour un Compte AWS

Pour commencer AWS, vous avez besoin d'un Compte AWS. Pour plus d'informations sur la création d'un Compte AWS, voir Getting started with an Compte AWS dans le Guide de Gestion de compte AWS référence.

Télécharger le code source de FreeRTOS

Téléchargez les derniers packages FreeRTOS et Long Term Support (LTS) depuis la page Téléchargements sur freertos.org.

FreeRTOS-qualified plateformes matérielles

Les plateformes matérielles suivantes sont qualifiées pour FreeRTOS :

- Kit d'approvisionnement sans contact ATECC608A pour AWS IoT
- Kit de développement Cypress CYW943907AEVAL1F
- Kit de développement Cypress CYW954907AEVAL1F
- Kit Cypress CY8CKIT-064S0S2-4343W
- Espressif ESP32-DevKitC
- Espressif ESP-WROVER-KIT
- Espressif ESP-WROOM-32SE
- Espressif ESP32-S2-Saola-1
- Kit de connectivité IoT Infineon XMC4800
- Kit de démarrage Marvell MW320 AWS IoT
- Kit de démarrage Marvell MW322 AWS IoT
- MediaTek Kit de développement MT7697hx
- Ensemble Microchip Curiosity PIC32MZEF
- Nordic n RF52840-DK
- NuMaker-IoT-M487
- Module IoT NXP LPC54018
- Solution de sécurité OPTIGA Trust X
- Module IoT RSK RX65N de Renesas
- Nœud IoT du kit de découverte STM32L4 de STMicroelectronics
- Instruments texans CC3220SF-LAUNCHXL
- Microsoft Windows 7 ou version ultérieure, avec au moins un double cœur et une connexion Ethernet câblée
- Kit IoT industriel Xilinx Avnet MicroZed

Les appareils qualifiés sont également répertoriés dans le catalogue d'appareils des partenaires AWS.

Pour plus d'informations sur la qualification d'un nouvel appareil, consultez le guide de qualification FreeRTOS.

Ressources supplémentaires

Ces ressources peuvent être utiles pour vous.

- Pour de la documentation FreeRTOS supplémentaire, consultez freertos.org.
- Pour toute question concernant FreeRTOS destinée à l'équipe d'ingénierie de FreeRTOS, vous pouvez ouvrir un problème sur la page FreeRTOS. GitHub
- Pour toute question technique concernant FreeRTOS, consultez les forums de la communauté FreeRTOS.
- Pour plus d'informations sur la connexion des appareils à AWS IoT, consultez la section Device Provisioning dans le Guide du AWS IoT Core développeur.
- Pour obtenir un support technique pour AWS, consultez la section AWS Support.
- Pour toute question concernant la AWS facturation, les services liés aux comptes, les événements, les abus ou tout autre problème lié à ce service AWS, consultez la page Contactez-nous.

Versions de FreeRTOS

Les bibliothèques individuelles utilisent des numéros de version de style x.y.z, similaires à la gestion des versions sémantiques. x est le numéro de version principale, y le numéro de version secondaire, et à partir de 2022, z est un numéro de correctif. Avant 2022, z était un numéro de version ponctuel, ce qui exigeait que les premières bibliothèques LTS aient un numéro de correctif de la forme « x.y.z LTS Patch 2 ».

Les packages de bibliothèque utilisent les numéros de version d'horodatage de style yyyymm.x. yyyy est l'année, mm le mois et x un numéro de séquence facultatif indiquant l'ordre de sortie dans le mois. Dans le cas du package LTS, x est un numéro de correctif séquentiel pour cette version LTS. Les bibliothèques individuelles contenues dans un package sont celles de la dernière version de cette bibliothèque à cette date. Pour le package LTS, il s'agit de la dernière version de correctif des bibliothèques LTS initialement publiée en tant que version LTS à cette date.

Support à long terme pour FreeRTOS

Les versions de FreeRTOS Long Term Support (LTS) reçoivent des corrections de sécurité et de bogues critiques (le cas échéant) pendant au moins deux ans après leur publication. Grâce à cette maintenance continue, vous pouvez intégrer des corrections de bogues tout au long du cycle de

développement et de déploiement sans avoir à vous soucier des coûteuses interruptions liées à la mise à jour vers les nouvelles versions majeures des bibliothèques FreeRTOS.

Avec FreeRTOS LTS, vous disposez de l'ensemble complet de bibliothèques nécessaires pour créer des produits intégrés et connectés sécurisés. LTS permet de réduire les coûts de maintenance et de test associés à la mise à jour des bibliothèques sur vos appareils déjà en production.

FreeRTOS LTS inclut le noyau FreeRTOS et les bibliothèques IoT : FreeRTOS+TCP, CoreMQTT, CoreHTTP, CorePKCS11, CoreJSON, OTA, Jobs et Device Shadow. AWS IoT AWS IoT AWS IoT Device Defender AWS IoT Pour plus d'informations, consultez les bibliothèques FreeRTOS LTS.

Plan de maintenance étendu FreeRTOS

AWS propose également un plan de maintenance étendu (EMP) de FreeRTOS, qui fournit des correctifs de sécurité et des corrections de bogues critiques sur la version de FreeRTOS Long Term Support (LTS) que vous avez choisie pour un maximum de dix années supplémentaires. Avec FreeRTOS EMP, vos appareils à longue durée de vie basés sur FreeRTOS peuvent compter sur une version dont les fonctionnalités sont stables et qui reçoit des mises à jour de sécurité pendant des années. Vous recevez des notifications en temps opportun concernant les correctifs à venir sur les bibliothèques FreeRTOS, afin que vous puissiez planifier le déploiement des correctifs de sécurité sur vos appareils Internet des objets (IoT).

Pour en savoir plus sur FreeRTOS EMP, consultez la page Fonctionnalités.

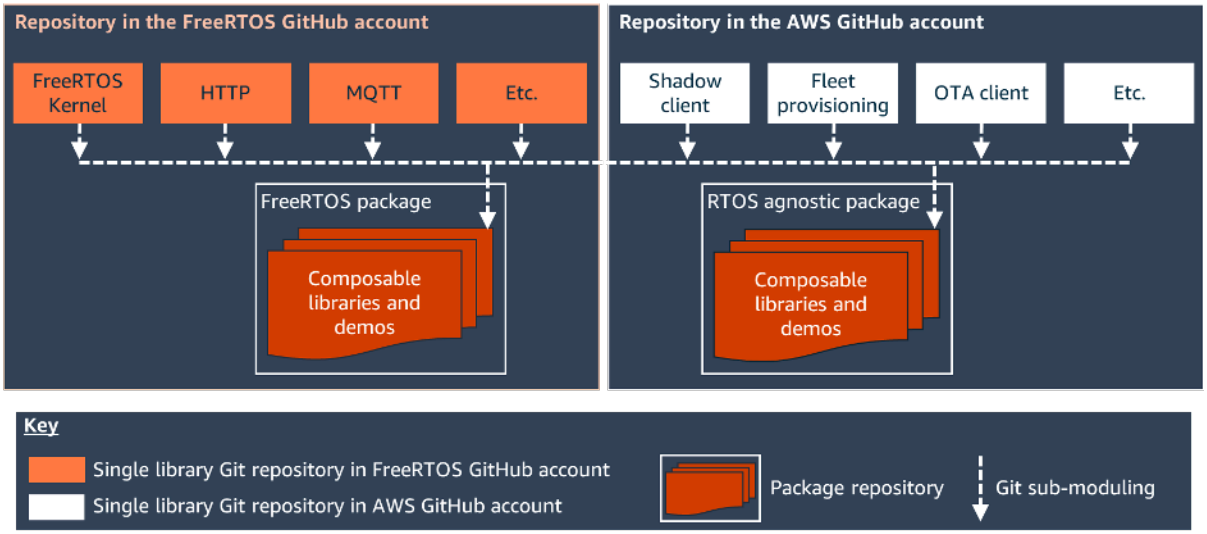
Architecture FreeRTOS

FreeRTOS contient deux types de référentiels, les référentiels à bibliothèque unique et les référentiels de packages. Chaque référentiel de bibliothèque contient le code source d'une bibliothèque sans aucun projet de construction ni exemple. Les référentiels de packages contiennent plusieurs bibliothèques et peuvent contenir des projets préconfigurés illustrant l'utilisation de la bibliothèque.

Bien que les référentiels de packages contiennent plusieurs bibliothèques, ils ne contiennent pas de copies de ces bibliothèques. Les référentiels de packages font plutôt référence aux bibliothèques qu'ils contiennent en tant que sous-modules git. L'utilisation de sous-modules garantit l'existence d'une source de vérité unique pour chaque bibliothèque individuelle.

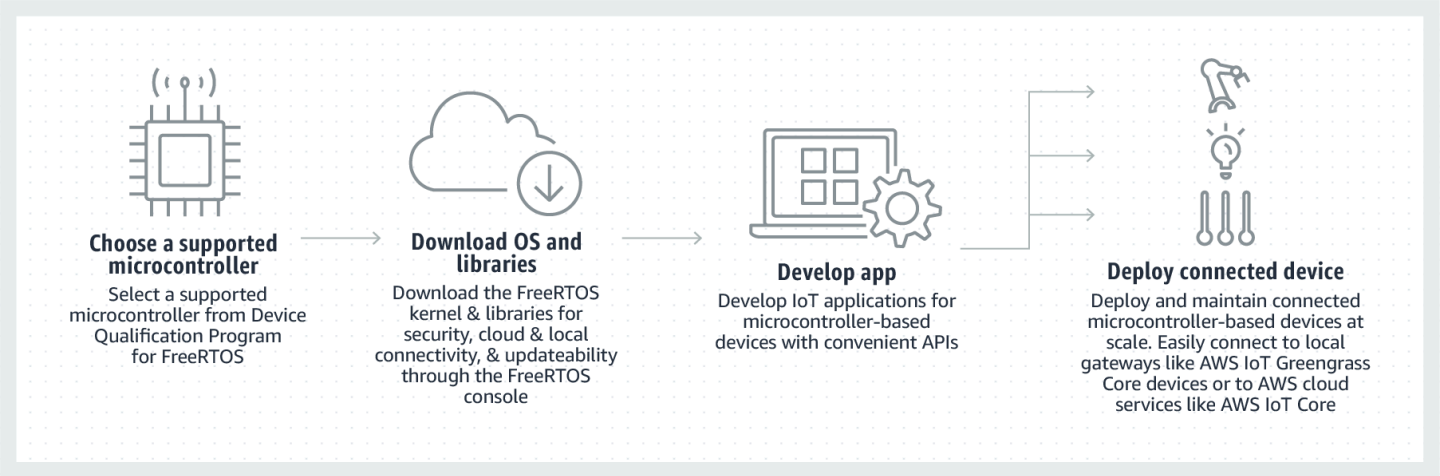
Les référentiels git des bibliothèques individuels sont répartis entre deux GitHub organisations. Les référentiels contenant des bibliothèques spécifiques à FreeRTOS (telles que FreeRTOS+TCP) ou

des bibliothèques génériques (telles que CoreMQTT, qui est indépendant du cloud car il fonctionne avec n'importe quel broker MQTT) font partie de l'organisation FreeRTOS. GitHub Des référentiels contenant des bibliothèques AWS IoT spécifiques (tels que le client de mise à jour AWS IoT sans fil) se trouvent dans l'organisation. AWS GitHub Le schéma suivant explique la structure.



Flux de travail du développement

Vous commencez le développement en téléchargeant FreeRTOS. Vous décompressez le package et l'importez dans votre IDE. Vous pouvez ensuite développer une application sur votre plateforme matérielle sélectionnée, et fabriquer et déployer ces appareils à l'aide du processus de développement adapté à votre appareil. Les appareils déployés peuvent se connecter au AWS IoT service ou AWS IoT Greengrass faire partie d'une solution IoT complète.



Principes fondamentaux du noyau FreeRTOS

Le noyau FreeRTOS est un système d'exploitation en temps réel qui prend en charge de nombreuses architectures. Ses principes fondamentaux sont idéaux pour créer des applications de microcontrôleurs embarqués. Il fournit :

- Un planificateur multitâche.
- Plusieurs options d'allocation mémoire (y compris la possibilité de créer des systèmes complètement alloués de façon statique).
- Des primitives de coordination inter-tâches, y compris les notifications de tâche, les files d'attente de messages, les différents types de sémaphores, et les tampons mémoire de flux et de messages.
- Support du multitraitement symétrique (SMP) sur les microcontrôleurs multicœurs.

Le noyau FreeRTOS n'exécute jamais d'opérations non déterministes, comme parcourir une liste liée, à l'intérieur d'une interruption ou section critique. Le noyau FreeRTOS inclut une implémentation efficace d'un minuteur logiciel qui n'utilise pas de temps UC, sauf en cas de besoin d'un minuteur. Les tâches bloquées ne nécessitent pas de maintenance périodique fastidieuse. Direct-to-task les notifications permettent une signalisation rapide des tâches, pratiquement sans surcharge de RAM. Ils peuvent être utilisés dans la plupart des scénarios de signalisation inter-tâches et d'interruption de tâches.

Le noyau FreeRTOS est conçu pour être petit, simple et facile à utiliser. L'image binaire d'un noyau RTOS typique est de l'ordre de 4 000 à 9 000 octets.

Pour obtenir la documentation la plus récente sur le noyau FreeRTOS, consultez. FreeRTOS.org FreeRTOS.org propose un certain nombre de tutoriels et de guides détaillés sur l'utilisation du noyau FreeRTOS, notamment un guide de démarrage rapide du noyau FreeRTOS et l'implémentation plus approfondie du RTOS dans la documentation FreeRTOS.

Le planificateur du noyau FreeRTOS

Une application embarquée qui utilise un noyau RTOS peut être structurée sous la forme d'un ensemble de tâches indépendantes. Chaque tâche s'exécute dans son propre contexte, sans dépendance à l'égard d'autres tâches. Une seule tâche de l'application est en cours d'exécution à

un moment donné dans le temps. Le planificateur RTOS en temps réel détermine à quel moment chaque tâche doit être exécutée. Chaque tâche est fournie avec sa propre pile. Lorsqu'une tâche est échangée afin qu'une autre tâche puisse s'exécuter, le contexte d'exécution de la tâche est enregistré dans la pile des tâches afin de pouvoir être restauré lorsque la même tâche sera à nouveau échangée ultérieurement pour reprendre son exécution.

Pour fournir un comportement déterministe en temps réel, le planificateur de tâche FreeRTOS permet d'attribuer des priorités strictes. RTOS garantit que la tâche à la priorité la plus haute en mesure de s'exécuter reçoit le temps de traitement. Cette attribution nécessite que le temps de traitement soit partagé entre les tâches de priorité égale, si elles sont prêtes à être exécutées simultanément. FreeRTOS crée aussi une tâche inactive qui s'exécute uniquement lorsqu'aucune autre tâche n'est prête à être exécutée.

Allocation mémoire du noyau

Le noyau RTOS a besoin de RAM chaque fois qu'une tâche, une file d'attente ou un autre objet RTOS est créé. La RAM peut être allouée :

- Statiquement au moment de la compilation.
- Dynamiquement depuis le segment RTOS par les fonctions de création d'objet de l'API RTOS.

Lorsque les objets RTOS sont créés de manière dynamique, l'utilisation des fonctions `malloc()` et `free()` de la bibliothèque standard C n'est pas toujours appropriée pour un certain nombre de raisons :

- Ils peuvent ne pas être disponibles sur les systèmes intégrés.
- Ils occupent un espace code précieux.
- Ils ne sont généralement pas thread-safe.
- Ils ne sont pas déterministes.

Pour ces raisons, FreeRTOS conserve l'API d'allocation mémoire dans sa couche portable. La couche portable est en dehors des fichiers source qui implémentent les principales fonctionnalités RTOS. Par conséquent, vous pouvez fournir une implémentation spécifique à l'application adaptée au système en temps réel que vous développez. Lorsque le noyau RTOS nécessite de la mémoire RAM, il appelle `pvPortMalloc()` au lieu de `malloc()`. Lorsque la mémoire RAM est libérée, le noyau RTOS appelle `vPortFree()` au lieu de `free()`.

Gérer la mémoire des applications

Lorsque les applications ont besoin de mémoire, elles peuvent l'allouer depuis le segment FreeRTOS. FreeRTOS offre plusieurs modèles de gestion de segment qui varient en complexité et en fonctionnalités. Vous pouvez également fournir votre propre implémentation du segment.

Le noyau FreeRTOS comprend cinq implémentations de segment :

`heap_1`

Est l'implémentation la plus simple. Ne permet pas que la mémoire soit libérée.

`heap_2`

Permet que la mémoire soit libérée, mais ne fusionne pas les blocs libres adjacents.

`heap_3`

Encapsule les fonctions standard `free()` et `malloc()` pour la sécurité des threads.

`heap_4`

Fusionne les blocs libres adjacents afin d'éviter la fragmentation. Inclut une option de placement à adresse absolue.

`heap_5`

Est similaire à heap_4. Peut étendre le segment sur plusieurs zones mémoire non adjacentes.

Coordination inter-tâches

Cette section contient des informations sur les primitives FreeRTOS.

Rubriques

- Files d'attente
- Sémaphores et mutex
- Direct-to-task notifications
- Tampons mémoire de flux
- Tampons de messages

Files d’attente

Les files d'attente constituent la forme principale de la communication inter-tâches. Elles peuvent être utilisées pour envoyer des messages entre les tâches et entre les interruptions et les tâches. Dans la plupart des cas, elles sont utilisées comme buffers de type FIFO, les nouvelles données étant envoyées à la fin de la file d'attente. (Les données peuvent également être envoyées au début de la file d'attente.) Les messages sont envoyés par copie via les files d'attente, ce qui signifie que les données (qui peuvent être un pointeur vers des tampons mémoire de plus grande taille) sont elles-mêmes copiées dans la file d'attente au lieu de simplement stocker une référence aux données.

Les API de file d'attente permettent de spécifier une durée de blocage. Lorsqu'une tâche tente de lire à partir d'une file d'attente vide, la tâche est placée dans l'état Blocked jusqu'à ce que les données deviennent disponibles dans la file d'attente ou que la durée de blocage se soit écoulée. Les tâches de l'état Blocked ne consomment pas de temps UC, ce qui permet à d'autres tâches de s'exécuter. De même, quand une tâche tente d'écrire sur une file d'attente complète, la tâche est placée dans l'état Blocked jusqu'à ce que l'espace devienne disponible dans la file d'attente ou que la durée de blocage se soit écoulée. Si plusieurs tâches sont bloquées sur la même file d'attente, la tâche avec la priorité la plus haute est débloquée en premier.

D'autres primitives FreeRTOS, telles que les notifications directes de tâches et les tampons de flux et de messages, offrent des alternatives légères aux files d'attente dans de nombreux scénarios de conception.

Sémaphores et mutex

Le noyau FreeRTOS fournit des sémaphores binaires, des sémaphores de comptabilisation et des mutex, à des fins d'exclusion mutuelle et de synchronisation.

Les sémaphores binaires ne peuvent avoir que deux valeurs. Ils sont un bon choix pour la mise en œuvre de la synchronisation (soit entre les tâches, soit entre les tâches et une interruption). Les sémaphores de comptabilisation acceptent plus de deux valeurs. Ils permettent à plusieurs tâches de partager des ressources ou d'effectuer des opérations de synchronisation plus complexes.

Les mutex sont des sémaphores binaires qui incluent un mécanisme d'héritage de priorité. Cela signifie que si une tâche à la priorité élevée est bloquée lors de la tentative d'obtention d'un mutex détenu par une tâche de moindre priorité, la priorité de la tâche contenant le jeton est temporairement élevée à celle de la tâche de blocage. Ce mécanisme est conçu pour s'assurer que la tâche à la priorité la plus élevée est conservée dans l'état Blocked le moins de temps possible, afin de minimiser l'inversion de priorité qui s'est produite.

Direct-to-task notifications

Les notifications aux tâches permettent d'interagir avec d'autres tâches et de se synchroniser avec les routines ISR (Interrupt Service Routine), sans qu'un objet de communication distinct comme un sémaphore soit nécessaire. Chaque tâche RTOS a une valeur de notification 32 bits utilisée pour stocker le contenu de la notification, le cas échéant. Une notification de tâche RTOS est un événement envoyé directement à une tâche qui peut débloquer la tâche de réception et, éventuellement, mettre à jour la valeur de notification de la tâche de réception.

Les notifications aux tâches RTOS peuvent être utilisées comme une alternative légère et plus rapide aux sémaphores binaires et aux sémaphores de comptabilisation, ainsi que, dans certains cas, aux files d'attente. Les notifications aux tâches présentent des avantages de vitesse et de plan RAM par rapport à d'autres fonctions qui peuvent être utilisées pour exécuter des fonctionnalités équivalentes. Cependant, les notifications de tâche ne peuvent être utilisées que lorsqu'il n'y a qu'une seule tâche qui puisse être le destinataire de l'événement.

Tampons mémoire de flux

Les tampons mémoire de flux autorisent qu'un flux d'octets soit transmis d'une routine ISR à une tâche, ou d'une tâche à une autre. Un flux d'octets peut avoir une longueur arbitraire et n'a pas nécessairement de début ou de fin. N'importe quel nombre d'octets peut être écrit en une seule fois et n'importe quel nombre d'octets être lu en une seule fois. Vous activez la fonctionnalité de tampon de flux en incluant le fichier source `stream_buffer.c` dans votre projet.

Les tampons mémoire de flux présument qu'il n'y a qu'une seule tâche ou interruption qui écrit dans le tampon mémoire (l'auteur) et qu'une seule tâche ou interruption lit à partir du tampon mémoire (le lecteur). Le fait que l'auteur et le lecteur soient des tâches ou des routines ISR différentes présente toutes les conditions de sécurité, contrairement au fait d'avoir plusieurs auteurs ou lecteurs.

L'implémentation de tampons mémoire de flux utilise les notifications directes aux tâches. Par conséquent, l'appel d'une API de tampon mémoire de flux qui place la tâche appelante dans l'état Blocked peut changer l'état de notification et la valeur de la tâche appelante.

Envoi de données

`xStreamBufferSend()` est utilisée pour envoyer les données au tampon mémoire de flux d'une tâche. `xStreamBufferSendFromISR()` est utilisée pour envoyer les données à un tampon mémoire de flux d'une routine ISR.

`xStreamBufferSend()` autorise la spécification d'une durée de blocage. Si la fonction `xStreamBufferSend()` est appelée avec une durée de blocage différente de zéro pour écrire sur un tampon mémoire de flux et que le tampon mémoire est plein, la tâche est placée dans l'état Blocked jusqu'à ce que l'espace devienne disponible ou que la durée de blocage expire.

`sbSEND_COMPLETED()` et `sbSEND_COMPLETED_FROM_ISR()` sont des macros qui sont appelées (en interne par l'API FreeRTOS) lorsque les données sont écrites dans un tampon mémoire de flux. Cela permet d'accepter le handle du tampon mémoire de flux qui a été mis à jour. Ces deux macros vérifient s'il existe une tâche bloquée dans le tampon mémoire de flux en attente de données et, si tel est le cas, suppriment la tâche de l'état Blocked.

Vous pouvez modifier ce comportement par défaut en fournissant votre propre implémentation de `sbSEND_COMPLETED()` dans `FreeRTOSConfig.h`. C'est utile quand un tampon mémoire de flux est utilisé pour transmettre les données entre les noyaux sur un processeur multicœur. Dans ce scénario, `sbSEND_COMPLETED()` peut être mis en œuvre pour générer une interruption dans l'autre cœur de l'UC et la routine ISR peut ensuite utiliser l'API `xStreamBufferSendCompletedFromISR()`pour contrôler et, si nécessaire débloquer, une tâche en attente de données.

Réception de données

`xStreamBufferReceive()` permet de lire les données à partir du tampon mémoire de flux d'une tâche. `xStreamBufferReceiveFromISR()` permet de lire les données à partir du tampon mémoire de flux d'une routine ISR.

`xStreamBufferReceive()` autorise la spécification d'une durée de blocage. Si la fonction `xStreamBufferReceive()` est appelée avec une durée de blocage différente de zéro pour lire depuis un tampon mémoire de flux et que le tampon est vide, la tâche est placée dans l'état Blocked jusqu'à ce qu'une quantité de données spécifiée devienne disponible dans le tampon mémoire de flux ou que la durée de blocage expire.

La quantité de données qui doivent être dans le tampon mémoire de flux avant qu'une tâche ne soit débloquée est appelée niveau de déclenchement du tampon mémoire de flux. Une tâche bloquée avec un niveau déclencheur égal à 10 est débloquée lorsqu'au moins 10 octets sont écrits dans le tampon mémoire ou que la durée de blocage de la tâche expire. Si la durée de blocage d'une tâche en lecture expire avant que le niveau déclencheur ne soit atteint, la tâche reçoit toutes les données écrites sur le tampon mémoire. Le niveau déclencheur d'une tâche doit être défini sur une valeur comprise entre 1 et la taille du tampon mémoire de flux. Le niveau déclencheur d'un tampon mémoire

de flux est défini quand `xStreamBufferCreate()` est appelé. Il peut être modifié en appelant `xStreamBufferSetTriggerLevel()`.

`sbRECEIVE_COMPLETED()` et `sbRECEIVE_COMPLETED_FROM_ISR()` sont des macros qui sont appelées (en interne par l'API FreeRTOS) lorsque les données sont lues à partir d'un tampon de flux. Les macros vérifient s'il existe une tâche bloquée sur le tampon de flux en attente d'espace disponible au sein du tampon mémoire et, si tel est le cas, suppriment la tâche de l'état Blocked. Vous pouvez modifier le comportement par défaut de `sbRECEIVE_COMPLETED()` en fournissant une autre implémentation dans `FreeRTOSConfig.h`.

Tampons de messages

Les tampons mémoire de messages permettent que des messages discrets de longueur variable soient transmis d'une routine ISR à une tâche, ou d'une tâche à une autre. Par exemple, les messages de longueur 10, 20 et 123 octets peuvent être tous écrits dans le même tampon mémoire de messages, ainsi qu'y être lus. Un message de 10 octets peut uniquement être lu sous la forme d'un message de 10 octets, et pas par octets individuels. Les tampons mémoire de messages sont conçus sur l'implémentation de tampons mémoire du flux. Vous pouvez activer la fonctionnalité de tampons mémoire de messages en incluant la source `stream_buffer.c` dans votre projet.

Les tampons mémoire de messages présument qu'il n'y a qu'une seule tâche ou interruption qui écrit dans le tampon (l'auteur) et qu'une seule tâche ou interruption lit à partir de la mémoire tampon (le lecteur). Le fait que l'auteur et le lecteur soient des tâches ou des routines ISR différentes présente toutes les conditions de sécurité, contrairement au fait d'avoir plusieurs auteurs ou lecteurs.

L'implémentation de tampons mémoire de messages utilise les notifications directes aux tâches. Par conséquent, l'appel d'une API de tampon mémoire de flux qui place la tâche appelante dans l'état Blocked peut changer l'état de notification et la valeur de la tâche appelante.

Pour permettre aux tampons mémoire de messages de gérer les messages de taille variable, la longueur de chaque message est écrite dans le tampon mémoire de messages avant le message lui-même. La longueur est stockée dans une variable de type `size_t`, soit généralement 4 octets sur une architecture 32 bits. Par conséquent, écrire un message de 10 octets dans un tampon de messages consomme réellement 14 octets d'espace tampon. De même, écrire un message de 100 octets dans un tampon mémoire de messages consomme réellement 104 octets d'espace tampon.

Envoi de données

`xMessageBufferSend()` permet d'envoyer des données à un tampon mémoire de messages depuis une tâche. `xMessageBufferSendFromISR()` permet d'envoyer les données à un tampon de messages depuis une routine ISR.

`xMessageBufferSend()` autorise la spécification d'une durée de blocage. Si la fonction `xMessageBufferSend()` est appelée avec une durée de blocage différente de zéro pour écrire sur un tampon mémoire de messages et que le tampon mémoire est plein, la tâche est placée dans l'état Blocked jusqu'à ce que l'espace devienne disponible dans le tampon mémoire de messages ou que la durée de blocage expire.

`sbSEND_COMPLETED()` et `sbSEND_COMPLETED_FROM_ISR()` sont des macros qui sont appelées (en interne par l'API FreeRTOS) lorsque les données sont écrites dans un tampon mémoire de flux. Cela permet d'accepter un paramètre unique, qui est le handle du tampon mémoire de flux qui a été mis à jour. Ces deux macros vérifient s'il existe une tâche bloquée dans le tampon mémoire de flux en attente de données et, si tel est le cas, suppriment la tâche de l'état Blocked.

Vous pouvez modifier ce comportement par défaut en fournissant votre propre implémentation de `sbSEND_COMPLETED()` dans `FreeRTOSConfig.h`. C'est utile quand un tampon mémoire de flux est utilisé pour transmettre les données entre les noyaux sur un processeur multicœur. Dans ce scénario, `sbSEND_COMPLETED()` peut être mis en œuvre pour générer une interruption dans l'autre cœur de l'UC et la routine ISR peut ensuite utiliser l'API `xStreamBufferSendCompletedFromISR()`pour contrôler et, si nécessaire débloquer, une tâche qui était en attente de données.

Réception de données

`xMessageBufferReceive()` permet de lire les données à partir d'un tampon mémoire de messages d'une tâche. `xMessageBufferReceiveFromISR()` permet de lire les données à partir d'un tampon mémoire de messages d'une routine ISR. `xMessageBufferReceive()` autorise la spécification d'une durée de blocage. Si la fonction `xMessageBufferReceive()` est appelée avec une durée de blocage différente de zéro pour lire depuis un tampon mémoire de messages et que le tampon est vide, la tâche est placée dans l'état Blocked jusqu'à ce que les données deviennent disponibles ou que la durée de blocage expire.

`sbRECEIVE_COMPLETED()` et `sbRECEIVE_COMPLETED_FROM_ISR()` sont des macros qui sont appelées (en interne par l'API FreeRTOS) lorsque les données sont lues à partir d'un tampon de flux. Les macros vérifient s'il existe une tâche bloquée sur le tampon de flux en attente d'espace

disponible au sein du tampon mémoire et, si tel est le cas, suppriment la tâche de l'état Blocked. Vous pouvez modifier le comportement par défaut de `sbRECEIVE_COMPLETED()` en fournissant une autre implémentation dans `FreeRTOSConfig.h`.

Support du multitraitement symétrique (SMP)

La prise en charge du protocole SMP dans le noyau FreeRTOS permet à une instance du noyau FreeRTOS de planifier des tâches sur plusieurs cœurs de processeur identiques. Les architectures de base doivent être identiques et partager la même mémoire.

L'API FreeRTOS reste sensiblement la même entre les versions monocœur et SMP, à l'exception de ces API supplémentaires. Par conséquent, une application écrite pour la version monocœur FreeRTOS doit être compilée avec la version SMP avec un minimum ou aucun effort. Cependant, il peut y avoir des problèmes fonctionnels, car certaines hypothèses qui étaient vraies pour les applications monocœurs pourraient ne plus être vraies pour les applications multicœurs.

Une hypothèse courante est qu'une tâche de priorité inférieure ne peut pas être exécutée alors qu'une tâche de priorité plus élevée est en cours d'exécution. Cela était vrai pour un système monocœur, mais ce n'est plus le cas pour les systèmes multicœurs, car plusieurs tâches peuvent s'exécuter simultanément. Si l'application s'appuie sur des priorités relatives des tâches pour fournir une exclusion mutuelle, elle peut observer des résultats inattendus dans un environnement multicœur.

Une autre hypothèse courante est que les ISR ne peuvent pas fonctionner simultanément entre eux ou avec d'autres tâches. Cela n'est plus vrai dans un environnement multicœur. Le rédacteur de l'application doit garantir une exclusion mutuelle appropriée lors de l'accès aux données partagées entre les tâches et les ISR.

Minuteurs de logiciels

Un minuteur logiciel permet qu'une fonction soit exécutée à un moment défini dans le futur. La fonction exécutée par le minuteur est appelée fonction de rappel du minuteur. Le temps entre le démarrage d'un minuteur et sa fonction de rappel en cours d'exécution est appelé période du minuteur. Le noyau FreeRTOS offre une implémentation efficace du minuteur logiciel pour les raisons suivantes :

- Il n'exécute pas les fonctions de rappel à partir d'un contexte d'interruption.
- Il ne consomme pas de temps de traitement, sauf si un minuteur a réellement expiré.

- Il n'ajoute pas de surcharge de traitement à l'interruption du tic-tac.
- Il ne parcourt aucune structure de liste de liens tandis que les interruptions sont désactivées.

Prise en charge d'une alimentation basse

Comme la plupart des systèmes d'exploitation embarqués, le noyau FreeRTOS utilise un minuteur matériel pour générer des interruptions régulières de tics, qui permettent de mesurer le temps. L'économie d'alimentation des implémentations de minuteur matériel classiques est limitée par la nécessité de quitter régulièrement l'état d'alimentation faible pour traiter les interruptions du tic-tac, et d'y accéder à nouveau. Si la fréquence des interruptions de tics est trop élevée, l'énergie et le temps nécessaires pour entrer dans un état d'alimentation basse et le quitter pour chaque tic-tac l'emportent sur tout gain potentiel d'économie d'alimentation, à l'exception des modes d'économie d'alimentation les plus légers.

Pour résoudre cette restriction, FreeRTOS inclut un mode minuteur sans tic-tac pour les applications à alimentation faible. Le mode inactif FreeRTOS sans tic-tac arrête l'interruption régulière du tic-tac pendant les périodes d'inactivité (périodes où il n'y a pas de tâches d'application capables de s'exécuter), puis apporte une correction à la valeur du nombre de tics-tacs RTOS quand l'interruption du tic-tac est redémarrée. L'arrêt de l'interruption du tic-tac permet au microcontrôleur de demeurer dans un état d'économie d'alimentation profonde jusqu'à ce qu'une interruption se produise ou qu'il soit temps pour le noyau RTOS de faire passer une tâche à l'état Ready.

Configurer le noyau (FreeRTOSConfig.h)

Vous pouvez configurer le noyau FreeRTOS pour une carte et une application spécifiques avec le fichier d'en-tête `FreeRTOSConfig.h`. Chaque application développée sur le noyau doit avoir un fichier d'en-tête `FreeRTOSConfig.h` dans son chemin d'inclusion de préprocesseur. `FreeRTOSConfig.h` est propre à l'application et doit être placé dans un répertoire d'applications, et non pas dans l'un des répertoires de code source du noyau FreeRTOS.

Les `FreeRTOSConfig.h` fichiers des applications de démonstration et de test de FreeRTOS se trouvent dans et. *`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/FreeRTOSConfig.h` *`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_tests/config_files/FreeRTOSConfig.h`

Pour obtenir la liste des paramètres de configuration disponibles dans lesquels vous pouvez les spécifier`FreeRTOSConfig.h`, consultez FreeRTOS.org.

AWS IoT SDK de périphérique pour Embedded C

> **Note**
>
> Ce SDK est destiné à être utilisé par des développeurs de logiciels embarqués expérimentés.

Le Kit SDK des appareils AWS IoT pour Embedded C (C-SDK) est une collection de fichiers source C sous licence open source du MIT qui peuvent être utilisés dans des applications intégrées pour connecter en toute sécurité des appareils IoT à AWS IoT Core. Il inclut un client MQTT, un client HTTP, un analyseur JSON, AWS IoT Device Shadow, AWS IoT Jobs, AWS IoT Fleet Provisioning et des bibliothèques. AWS IoT Device Defender Ce SDK est distribué sous forme source et peut être intégré au micrologiciel client avec le code d'application, d'autres bibliothèques et un système d'exploitation (OS) de votre choix.

Kit SDK des appareils AWS IoT pour Embedded C Il est généralement destiné aux appareils aux ressources limitées qui nécessitent un environnement d'exécution en langage C optimisé. Vous pouvez utiliser le kit SDK sur n'importe quel système d'exploitation et l'héberger sur n'importe quel type de processeur (par exemple, microcontrôleurs et MPU). Toutefois, si vos appareils disposent de suffisamment de mémoire et de ressources de traitement, nous vous recommandons d'utiliser l'un des kits SDK pour AWS IoT appareils les plus performants.

Pour plus d'informations, consultez les ressources suivantes :

- AWS IoT SDK de périphérique pour Embedded C
- AWS IoT SDK du périphérique pour Embedded C on GitHub
- AWS IoT SDK du périphérique pour Embedded C Readme
- AWS IoT SDK de périphérique pour les échantillons C intégrés

Comprendre les API FreeRTOS Common IO

Les API d'E/S communes agissent comme des couches d'abstraction matérielle (HAL) qui fournissent une interface commune entre les pilotes et le code d'application de niveau supérieur. FreeRTOS Common IO fournit un ensemble d'API standard pour accéder aux périphériques série courants sur les cartes de référence prises en charge ; les implémentations de ces API ne sont pas incluses. Ces API communes communiquent et interagissent avec ces périphériques et permettent à votre code de fonctionner sur toutes les plates-formes. Sans Common IO, l'écriture de code pour fonctionner avec des appareils de bas niveau est spécifique au fournisseur de silicium.

> **Note**
>
> FreeRTOS n'a pas besoin d'implémenter les API Common IO pour fonctionner, mais il tentera d'utiliser les API Common IO comme moyen d'interfacer avec les périphériques spécifiques d'une carte basée sur un microcontrôleur au lieu d'API spécifiques au fournisseur.

En général, les pilotes de périphériques sont indépendants du système d'exploitation sous-jacent et sont spécifiques à une configuration matérielle donnée. La HAL ignore les détails du fonctionnement d'un pilote spécifique et fournit une API uniforme pour contrôler de tels périphériques. Vous pouvez utiliser les mêmes API pour accéder à différents pilotes de périphériques sur plusieurs cartes de référence basées sur des microcontrôleurs (MCU).

Bibliothèques

FreeRTOS fournit actuellement deux bibliothèques Common IO : Common IO - basic et Common IO - BLE.

Common IO - basique

Présentation de

Common IO - basic fournit des API qui gèrent les I/O périphériques de base et les fonctions que vous pouvez trouver sur les MCU-based cartes mères. Le référentiel Common IO - basic est disponible sur GitHub.

Périphériques pris en charge

- ADC

- GPIO
- I2C
- PWM
- SPI
- UART
- Chien de garde
- Éclair
- RTC
- REFUSER
- Réinitialisations
- I2S
- Compteur de performance
- Informations sur la plate-forme matérielle

Fonctionnalités prises en charge

- Synchrone read/write

  La fonction ne revient pas tant que la quantité de données demandée n'est pas transférée.

- Asynchrone read/write

  La fonction revient immédiatement et le transfert de données s'effectue de manière asynchrone. Lorsque l'exécution de la fonction est terminée, un rappel utilisateur enregistré est invoqué.

Spécificités relatives au périphérique

- I2C

  Combinez plusieurs opérations en une seule transaction. Permet d'écrire puis de lire des actions dans une seule et même transaction.

- SPI

  Transférez les données entre le primaire et le secondaire, ce qui signifie que l'écriture et la lecture se font simultanément.

Référence des API

Pour une référence complète sur les API, consultez le manuel Common IO - basic API reference.

E/S communes - BLE

Présentation de

Common IO - BLE fournit une abstraction de la pile Bluetooth Low Energy du fabricant. Il fournit les interfaces suivantes qui peuvent être utilisées pour contrôler le dispositif et effectuer des opérations GAP et GATT. Le référentiel Common IO - BLE est disponible sur GitHub.

Gestionnaire de périphériques Bluetooth :

Cela fournit une interface permettant de contrôler le périphérique Bluetooth, d'effectuer des opérations de découverte de l'appareil et d'autres tâches liées à la connectivité.

Gestionnaire d'adaptateurs BLE :

Cela fournit une interface pour les fonctions de l'API GAP spécifiques à BLE.

Gestionnaire d'adaptateurs Bluetooth Classic :

Cela fournit une interface pour contrôler les fonctionnalités classiques BT d'un appareil.

Serveur GATT :

Cela fournit une interface permettant d'utiliser la fonctionnalité de serveur Bluetooth GATT.

Client du GATT :

Cela fournit une interface permettant d'utiliser la fonction client Bluetooth GATT.

Interface de connexion A2DP :

Cela fournit une interface pour le profil de source A2DP pour le périphérique local.

Référence des API

Pour une référence complète de l'API, consultez la référence de l'API Common IO - BLE.

Common IO pour Amazon Common Software

Les API Common IO font partie des implémentations requises par Amazon Common Software for Devices, en particulier pour être implémentées dans un kit de portage d'appareils (DPK) d'un fournisseur.

Qu'est-ce que l'ACS ?

Amazon Common Software (ACS) for Devices est un logiciel qui vous permet d'intégrer plus rapidement les SDK Amazon Device sur vos appareils. ACS fournit une couche d'intégration d'API unifiée, des composants pré-validés et économes en mémoire pour les fonctions courantes telles que la connectivité, un kit de portage de périphériques (DPK) et des suites de tests multiniveaux.

Programme de qualification

Le programme de qualification Amazon Common Software for Devices vérifie qu'une version de l'ACS DPK (Device Porting Kit) exécutée sur une carte de développement spécifique basée sur un microcontrôleur est compatible avec les meilleures pratiques publiées par le programme et suffisamment robuste pour réussir les ACS-mandated tests spécifiés par le programme de qualification.

Les fournisseurs qualifiés dans le cadre de ce programme sont répertoriés sur la page des fournisseurs de chipsets ACS.

Pour plus d'informations sur les conditions d'éligibilité, contactez ACS for Devices.

Commencez avec FreeRTOS

Rubriques

- Commencez avec Quick Connect
- Explorez les bibliothèques FreeRTOS
- Construisez un environnement sûr et robuste AWS IoT product
- Développez votre AWS IoT vigie

Commencez avec Quick Connect

Pour explorer rapidement AWS IoT, commencez par les démos de AWS Quick Connect. Les démos Quick Connect sont simples à configurer et à connecter à un forum qualifié FreeRTOS fourni par un partenaire. AWS IoT

Suivez le AWS IoT didacticiel de mise en route pour mieux comprendre la AWS IoT AWS IoT console. Vous pouvez modifier le code source de démonstration fourni avec les démos de Quick Connect à l'aide du système de construction du forum choisi et des outils pour vous connecter à votre AWS compte. Le flux de données provenant de la AWS IoT console de votre compte est désormais visible.

Explorez les bibliothèques FreeRTOS

Une fois que vous aurez compris comment un appareil IoT AWS IoT fonctionne ensemble, vous pouvez commencer à explorer les bibliothèques FreeRTOS et les bibliothèques Long-Term Support (LTS).

Certaines bibliothèques couramment utilisées pour les appareils basés sur FreeRTOS sont les suivantes AWS IoT :

- Noyau FreeRTOS
- Noyau MQTT
- AWS IoT Over-the-Air (OTA)

Visitez freertos.org pour accéder à de la documentation technique et à des démonstrations spécifiques à la bibliothèque.

Construisez un environnement sûr et robuste AWS IoT product

Reportez-vous à la section Intégrations AWS IoT FreeRTOS en vedette pour découvrir les meilleures pratiques permettant de rendre les logiciels pour appareils IoT plus sûrs et plus robustes. Ces intégrations FreeRTOS IoT sont conçues pour améliorer la sécurité en combinant le logiciel FreeRTOS et une carte fournie par un partenaire dotée de fonctionnalités de sécurité matérielles. Utilisez-les en production tels quels ou utilisez-les comme modèle pour vos propres designs.

Développez votre AWS IoT vigie

Pour créer un projet d'application pour votre AWS IoT produit, procédez comme suit :

1. Téléchargez la dernière version de FreeRTOS ou de Long Term Support (LTS) sur freertos.org, ou clonez depuis le dépôt. FreeRTOS-LTS GitHub Vous pouvez également intégrer les bibliothèques FreeRTOS requises dans votre projet à partir de la chaîne d'outils du fournisseur du microcontrôleur, le cas échéant.
2. Suivez le guide de portage de FreeRTOS pour créer un projet, configurer l'environnement de développement et intégrer les bibliothèques FreeRTOS dans votre projet. Utilisez le FreeRTOS-Libraries-Integration-Tests GitHub référentiel pour valider le portage.

AWS IoT Device Tester pour FreeRTOS

L'IDT pour FreeRTOS est un outil permettant de qualifier le débit de données avec le système d'exploitation FreeRTOS. Le testeur de périphériques (IDT) ouvre d'abord une connexion USB ou UART avec un appareil. Il fait ensuite clignoter une image de FreeRTOS configuré pour tester le fonctionnement de l'appareil dans diverses conditions. AWS IoT Device Tester les suites sont extensibles et IDT est utilisé pour l'orchestration des AWS IoT tests clients.

IDT pour FreeRTOS s'exécute sur un ordinateur hôte (Windows, macOS ou Linux) connecté à l'appareil testé. IDT configure et orchestre les cas de test, et agrège les résultats. Il fournit également une interface de ligne de commande pour gérer l'exécution des tests.

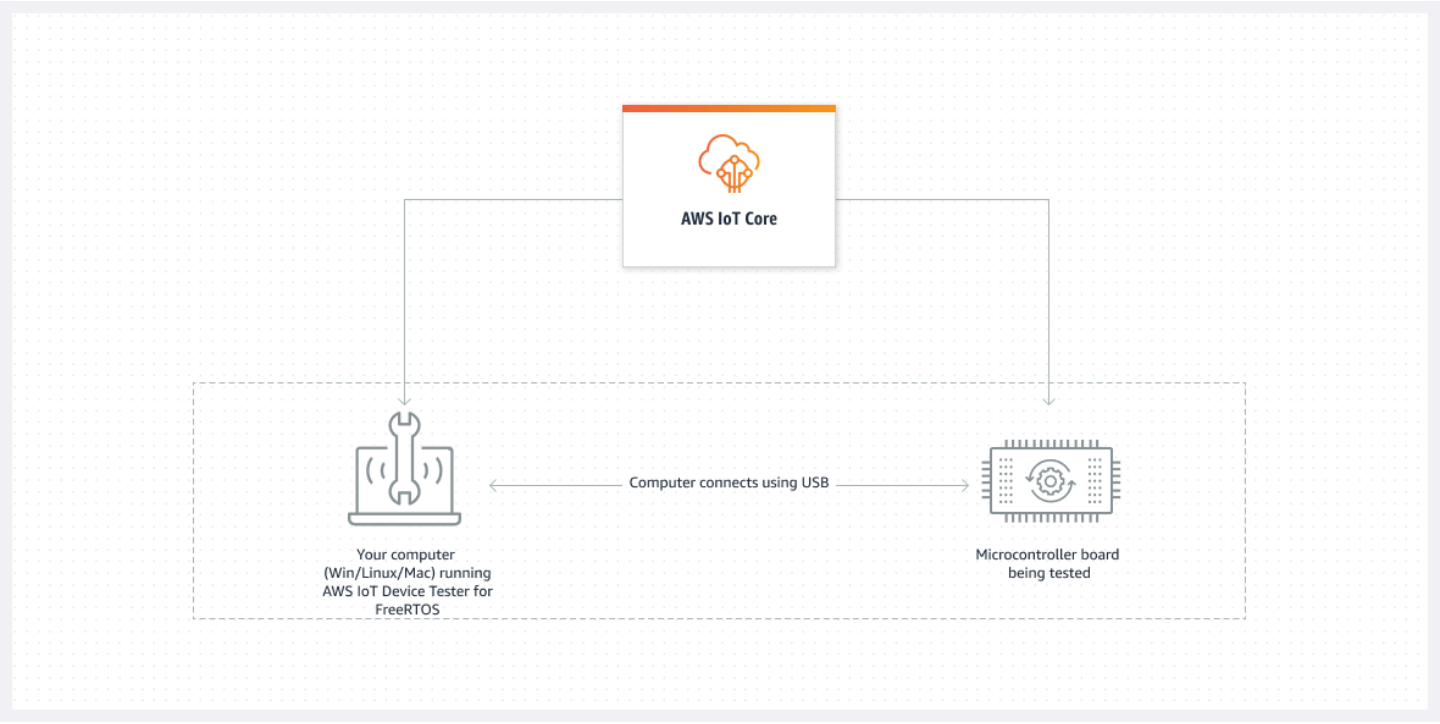
Suite de qualifications FreeRTOS

IDT for FreeRTOS vérifie le port de FreeRTOS sur votre microcontrôleur et vérifie s'il peut communiquer efficacement de manière fiable et sécurisée. AWS IoT Plus précisément, il vérifie si les interfaces de la couche de portage pour les bibliothèques FreeRTOS sont correctement implémentées. Il effectue également des tests de bout en bout avec. AWS IoT Core Par exemple, il vérifie si votre forum peut envoyer et recevoir des messages MQTT et les traiter correctement.

La suite de qualification FreeRTOS (FRQ) 2.0 utilise des cas de tests et Device Advisor définis dans le Guide FreeRTOS-Libraries-Integration-Tests de qualification FreeRTOS.

IDT for FreeRTOS génère des rapports de test que vous pouvez soumettre AWS à Partner Network (APN) pour inclure vos appareils FreeRTOS dans le catalogue d'appareils partenaires. AWS Pour de plus amples informations, veuillez consulter AWS Device Qualification Program.

Le schéma suivant montre la configuration de l'infrastructure de test pour la qualification FreeRTOS.



IDT pour FreeRTOS organise les ressources de test en suites de tests et en groupes de tests :

- Une suite de tests est un ensemble de groupes de tests utilisés pour vérifier qu'un appareil fonctionne avec des versions spécifiques de FreeRTOS.
- Un groupe de test est l'ensemble des cas de test individuels liés à une fonctionnalité particulière, telle que la messagerie BLE et MQTT.

Pour de plus amples informations, consultez Versions de la suite de tests.

Comprendre les suites de tests personnalisées

IDT pour FreeRTOS combine une configuration standardisée et un format de résultat avec un environnement de suite de tests. Cet environnement vous permet de développer des suites de tests personnalisées pour vos appareils et leurs logiciels. Vous pouvez ajouter des tests personnalisés pour votre propre validation interne ou les fournir à vos clients pour la vérification des appareils.

La façon dont vous configurez les suites de tests personnalisées détermine les configurations de paramètres que vous devez fournir à vos utilisateurs pour exécuter vos suites de tests personnalisées. Pour de plus amples informations, veuillez consulter Développez et exécutez vos propres suites de tests IDT.

Versions prises en charge de AWS IoT Device Tester

Cette rubrique répertorie les versions prises en charge de AWS IoT Device Tester for FreeRTOS. À titre de bonne pratique, nous vous recommandons d'utiliser la dernière version d'IDT pour FreeRTOS compatible avec votre version cible de FreeRTOS. Chaque version d'IDT pour FreeRTOS possède une ou plusieurs versions correspondantes de FreeRTOS qu'elle prend en charge. Nous vous recommandons de télécharger une nouvelle version d'IDT pour FreeRTOS lorsqu'une nouvelle version de FreeRTOS sera publiée.

En téléchargeant le logiciel, vous acceptez le contrat de AWS IoT Device Tester licence contenu dans l'archive de téléchargement.

> **Note**
>
> Lorsque vous utilisez AWS IoT Device Tester FreeRTOS, nous vous recommandons de passer à la dernière version du correctif. FreeRTOS-LTS

> **Important**
>
> Depuis octobre 2022, AWS IoT Device Tester AWS IoT FreeRTOS Qualification (FRQ) 1.0 ne génère pas de rapports de qualification signés. Vous ne pouvez pas qualifier de nouveaux appareils AWS IoT FreeRTOS pour qu'ils figurent dans le catalogue des appareils partenaires dans AWS le cadre du programme de qualification des appareils utilisant AWS les versions IDT FRQ 1.0. Bien que vous ne puissiez pas qualifier les appareils FreeRTOS avec IDT FRQ 1.0, vous pouvez continuer à tester vos appareils FreeRTOS avec FRQ 1.0. Nous vous recommandons d'utiliser IDT FRQ 2.0 pour qualifier et répertorier les appareils FreeRTOS dans le catalogue d'appareils partenaires.AWS

Dernière version de AWS IoT Device Tester pour FreeRTOS

Utilisez les liens suivants pour télécharger les dernières versions d'IDT pour FreeRTOS.

Dernière version de AWS IoT Device Tester pour FreeRTOS

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Liens de téléchargement	Date de sortie	Notes de mise à jour
IDT v4.9.0	FRQ_2.5.0	• 202112,00 • 202212,00 • 202212,01 • Tous les correctifs de FreeRTOS 202210-LTS qui utilisent les bibliothèques FreeRTOS LTS.	• Linux • macOS • Fenêtres	04/04/2023.04	• Prend en charge les tests par rapport à FreeRTOS 202112, 202212, 202212.01 et à tous les correctifs de FreeRTOS 202210-LTS qui utilisent les bibliothèques FreeRTOS. Pour plus d'informations, consultez README.md. Vous devez inclure la version du correctif pour FreeRTOS-

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Liens de téléchargement	Date de sortie	Notes de mise à jour
					LTS dans votre `manifest.yml` . • Durée d'exécution améliorée des tests OTA E2E. • Limite le nombre d'appareils répertoriés dans la liste `device.json` à 1. • Correctifs de bogues mineurs et améliorations.

> **Note**
>
> Nous ne recommandons pas l'exécution d'IDT par plusieurs utilisateurs à partir d'un emplacement partagé, tel qu'un répertoire NFS ou un dossier partagé réseau Windows. Cette pratique peut entraîner des pannes ou une corruption des données. Nous vous recommandons d'extraire le package IDT sur une unité locale et d'exécuter le fichier binaire IDT sur votre station de travail locale.

Versions IDT antérieures pour FreeRTOS

Les versions antérieures suivantes d'IDT pour FreeRTOS sont également prises en charge.

Versions antérieures de AWS IoT Device Tester pour FreeRTOS

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Liens de téléchargement	Date de sortie	Notes de mise à jour
IDT v4.8.1	FRQ_2.4.0	• 202112,00 • 202212,00 • 202212,01 • Tous les correctifs de FreeRTOS 202210-LTS qui utilisent les bibliothèques FreeRTOS LTS.	• Linux • macOS • Fenêtres	23/01/2023	• Voir README.MD pour plus d'informations. Vous devez inclure la version du correctif pour FreeRTOS-LTS dans votre `manifest.yml` . • Correctifs de bogues mineurs et améliorations.
IDT v4.6.0	FRQ_2.3.0	• 202112,00 • 202212,00 • 202212,01 • 202210-LTS qui utilisent les	• Linux • macOS • Fenêtres	2022.11.16	• Voir README.MD pour plus d'informations. Vous devez inclure la

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Liens de téléchargement	Date de sortie	Notes de mise à jour
		bibliothèques FreeRTOS LTS.			version du correctif pour FreeRTOS-LTS dans votre `manifest.yml` . • Pour plus d'informations sur le contenu de la version FreeRTOS 202210-LTS, consultez le fichier sur. CHANGELOG.md GitHub • Permet de configurer et d'exécuter AWS IoT Device Tester FreeRTOS via une interface utilisate

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Liens de téléchargement	Date de sortie	Notes de mise à jour
					ur basée sur le Web. Pour commencer, consultez Interface utilisateur pour IDT pour la suite de qualification FreeRTOS 2.0 (FRQ 2.0). • Ajoute une option permettant de conserver les copies modifiées du code source créées et utilisées lors de l'exécution pour le débogage post-test. Pour plus

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Liens de téléchargement	Date de sortie	Notes de mise à jour
					d'informations, consultez Configurer les paramètres de création, de flash et de test. • Ajoute le support du SDK client IDT pour Java. Pour plus d'informations sur le SDK du client IDT, consultez. Développez et exécutez vos propres suites de tests IDT

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Liens de téléchargement	Date de sortie	Notes de mise à jour
IDT v4.5.11	FRQ_2.2.0	• 202112,00 • 202212,00 • 202212,01 • 202210-LTS qui utilisent les bibliothèques FreeRTOS LTS.	• Linux • macOS • Fenêtres	2022.10.14	• Voir README.MD pour plus d'informations. Vous devez inclure la version du correctif pour FreeRTOS-LTS dans votre `manifest.yml` . • Pour plus d'informations sur le contenu de la version FreeRTOS 202210-LTS, consultez le fichier sur. CHANGELOG.md GitHub • Correctifs de bogues mineurs et

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Liens de téléchargement	Date de sortie	Notes de mise à jour
					améliorations.

Pour de plus amples informations, veuillez consulter Comprendre la politique de support pour AWS IoT Device Tester.

Versions IDT non prises en charge pour FreeRTOS

Cette section répertorie les versions non prises en charge d'IDT pour FreeRTOS. Les versions non prises en charge ne reçoivent pas de corrections de bogues ou de mises à jour. Pour de plus amples informations, veuillez consulter Comprendre la politique de support pour AWS IoT Device Tester.

Les versions suivantes de ne IDT-FreeRTOS sont plus prises en charge.

Versions non prises en charge de AWS IoT Device Tester pour FreeRTOS

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Date de sortie	Notes de mise à jour
IDT v4.5.10	FRQ_2.1.4	• 202112,00 • 202012-LTS qui utilisent les bibliothèques FreeRTOS LTS.	2022.09.02	• Pour plus d'informations sur ce qui est inclus dans la version FreeRTOS 202012-LTS, consultez le fichier sur. CHANGELOG.md GitHub • Résolution d'un problème

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Date de sortie	Notes de mise à jour
				affectant le groupe de `OTA End to End` test. • Supprimé `FullTransportInterfacePlainText` de la course lors des courses de qualification. Le texte brut peut toujours être exécuté en tant que groupe de test de développement à l'aide de l'`-\-group-id` indicateur. • Amélioration de la journalisation et de la lisibilité de la console et de la sortie des fichiers. • Correctifs de bogues

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Date de sortie	Notes de mise à jour
				mineurs et améliorations.

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Date de sortie	Notes de mise à jour
IDT v4.5.9	FRQ_2.1.3	• 202112,00 • 202012.04 -LTS qui utilisent les bibliothèques FreeRTOS LTS.	17/08/2022	• Pour plus d'informations sur le contenu de la version FreeRTOS 202012.04-LTS, consultez le fichier sur. CHANGELOG .md GitHub • Résolution d'un problème affectant le groupe de `FreeRTOSIntegrity` test. • Groupe de `FullCloudIoT` test mis à jour en supprimant le cas de test « MQTT Connect Exponential Backoff Retries ». • Correctifs de bogues

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Date de sortie	Notes de mise à jour
				mineurs et améliorations.

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Date de sortie	Notes de mise à jour
IDT v4.5.6	FRQ_2.1.2	• 202112,00 • 202012.04 -LTS qui utilisent les bibliothèques FreeRTOS LTS.	29/06/2022	• Pour plus d'informations sur le contenu de la version FreeRTOS 202012.04-LTS, consultez le fichier sur. CHANGELOG .md GitHub • Ajoute un nouveau groupe de test par rapport `FullCloud IoT` auquel la carte est testée AWS IoT Core Device Advisor. • Résolution d'un problème affectant les scénarios de test OTA E2E. • Correctifs de bogues mineurs et améliorations.

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Date de sortie	Notes de mise à jour
IDT v4.5.5	FRQ_2.1.1	• 202112,00 • 202012.04 -LTS qui utilisent les bibliothèques FreeRTOS LTS.	2022.06.06	• Pour plus d'informations sur le contenu de la version FreeRTOS 202012.04-LTS, consultez le fichier sur. CHANGELOG .md GitHub • Ajoute un nouveau groupe de test par rapport `FullCloud IoT` auquel la carte est testée AWS IoT Core Device Advisor. • Résolution d'un problème affectant les scénarios de test FreeRTOSVersion et FreeRTOSIntegrity. • Correctifs de bogues

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Date de sortie	Notes de mise à jour
				mineurs et améliorations.
IDT v4.5.5	FRQ_2.1.0	• 202107,00 • 202112,00 • 202012.04 -LTS qui utilisent les bibliothèques FreeRTOS LTS.	2022.05.31	• Pour plus d'informations sur le contenu de la version FreeRTOS 202012.04-LTS, consultez le fichier sur. CHANGELOG .md GitHub • Ajoute un nouveau groupe de test par rapport `FullCloud IoT` auquel la carte est testée AWS IoT Core Device Advisor. • Correctifs de bogues mineurs et améliorations.

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Date de sortie	Notes de mise à jour
IDT v4.5.4	FRQ_2.0.0	• 202112,00 • 202012.04 -LTS qui utilisent les bibliothèques FreeRTOS LTS.	2022.05.09	• Pour plus d'informations sur le contenu de la version FreeRTOS 202012.04-LTS, consultez le fichier sur. CHANGELOG .md GitHub • Supprime du référentiel l'obligation de qualifier les cartes utilisant uniquement les versions d'Amazon FreeRTOS. `aws/amazon-freertos` GitHub • Correctifs de bogues mineurs et améliorations.

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Date de sortie	Notes de mise à jour
IDT v4.5.2	FRQ_1.6.2	202107,00	25/01/2022	• Pour plus d'informations sur le contenu de la version 202107.00 de FreeRTOS, consultez le fichier sur. CHANGELOG .md GitHub • Implément e le nouvel orchestra teur de tests IDT pour configurer des suites de tests personnal isées. Pour plus d'informa tions, voir Configurer l'orchestrateur de test IDT. • Correctifs de bogues mineurs et améliorations.

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Date de sortie	Notes de mise à jour
IDT v4.0.3	FRQ_1.5.1	202012,00	2021.07.30	• Support pour la qualification des appareils dont les informations d'identification sont verrouillées sur un module de sécurité matériel. • Correctifs de bogues mineurs et améliorations.

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Date de sortie	Notes de mise à jour
IDT v4.3.0	FRQ_1.6.1	202107,00	26/07/2021	• Pour plus d'informations sur le contenu de la version 202107.00 de FreeRTOS, consultez le fichier sur. CHANGELOG.md GitHub • Permet de configurer et d'exécuter AWS IoT Device Tester FreeRTOS via une interface utilisateur basée sur le Web. Pour commencer, consultez Utiliser l'interface utilisateur IDT pour exécuter la suite de qualification FreeRTOS.

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Date de sortie	Notes de mise à jour
IDT v4.1.0	FRQ_1.6.0	202107,00	2021.07.21	• Pour plus d'informations sur le contenu de la version 202107.00 de FreeRTOS, consultez le fichier sur. CHANGELOG.md GitHub • Supprime les cas de test suivants de la qualification OTA : • Agent OTA • Nom de fichier OTA manquant • Nombre maximal de blocs configurés par OTA • Supprime le groupe de `Both` test OTA Dataplane de la qualification OTA. Dans le `device.js`

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Date de sortie	Notes de mise à jour
				on fichier, la `OTADataPlaneProtocol` configuration accepte désormais uniquement `HTTP` ou en `MQTT` tant que valeurs prises en charge. • Implémente les modifications suivantes dans la `freertosFileConfiguration` configuration du `userdata.json` fichier pour les modifications apportées au code source de FreeRTOS : • Remplace le nom de fichier spécifié pour

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Date de sortie	Notes de mise à jour
				`otaAgentTestsConfig` et `otaAgentDemosConfig` par `aws_ota_agent_config.h` `ota_config.h` . • Ajoute une nouvelle configuration `otaDemosConfig` facultative pour spécifier le chemin du nouveau `ota_demo_config.h` fichier. • Ajoute un nouveau champ `testStartDelayms` `userdata.json` pour spécifier le

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Date de sortie	Notes de mise à jour
				délai entre le moment où un appareil est flashé pour exécuter un groupe de test FreeRTOS et le moment où il commence à exécuter des tests. La valeur doit être exprimée en millisecondes. Ce délai peut être utilisé pour donner à IDT une chance de se connecter afin qu'aucune sortie de test ne soit manquée.

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Date de sortie	Notes de mise à jour
IDT v4.0.1	FRQ_1.4.1	202012,00	2021.01.19	• Pour plus d'informations sur le contenu de la version 202012.00 de FreeRTOS, consultez le fichier dans. CHANGELOG.md GitHub • Introduit des cas de test OTA (Over-the-air) E2E (de bout en bout) supplémentaires. • Prend en charge la qualification des cartes de développement exécutant FreeRTOS 202012.00 qui utilisent les bibliothèques FreeRTOS LTS. • Ajoute la prise en charge de

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Date de sortie	Notes de mise à jour
				la qualification des cartes de développement FreeRTOS utilisant la connectivité cellulaire. • Corrige un bogue dans la configuration du serveur Echo. • Vous permet de développer et d'exécuter vos propres suites de tests personnalisées à l'aide AWS IoT Device Tester de FreeRTOS. Pour de plus amples informations, veuillez consulter Développez et exécutez vos propres suites de tests IDT.

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Date de sortie	Notes de mise à jour
				• Fournit des applications IDT signées par code, vous n'avez donc pas besoin d'accorder d'autorisations lorsque vous l'exécutez sous Windows ou macOS. • La logique d'analyse des résultats du test BLE a été affinée.

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Date de sortie	Notes de mise à jour
IDT v3.4.0	FRQ_1.3.0	202011,01	05/11/2020	• Pour plus de détails, consultez le CHANGELOG.mdfichier dans GitHub. • Correction d'un bug selon lequel « RSA » n'était pas une option de configuration PKCS11 valide. • Correction d'un bug en raison duquel les compartiments Amazon S3 n'étaient pas nettoyés correctement après les tests OTA. • Mises à jour pour prendre en charge les nouveaux cas de test au sein du groupe de test FullMQTT.

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Date de sortie	Notes de mise à jour
IDT v3.3.0	FRQ_1.2.0	202007,00	2020,09,17	• Pour plus de détails, consultez le CHANGELOG.mdfichier dans GitHub. • De nouveaux tests de bout en bout pour valider la fonctionnalité de suspension et de reprise des mises à jour par Over The Air (OTA). • Correction d'un bug empêchant les utilisateurs de la région eu-central-1 de réussir la validation de configuration pour les tests OTA. • `--update-idt` Paramètre ajouté à la `run-suite` commande.

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Date de sortie	Notes de mise à jour
				Vous pouvez utiliser cette option pour définir la réponse à l'invite de mise à jour IDT. • `--update-managed-policy` Paramètre ajouté à la `run-suite` commande. Vous pouvez utiliser cette option pour définir la réponse à l'invite de mise à jour des politiques gérées. • Améliorations internes et corrections de bogues, notamment : • Pour les mises à jour automatiques de la suite

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Date de sortie	Notes de mise à jour
				de tests, améliorations apportées à la mise à niveau du fichier de configuration.

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Date de sortie	Notes de mise à jour
IDT v3.0.2	FRQ_1.0.1	202002.00		• Pour plus d'informations, consultez le CHANGELOG.mdfichier dans GitHub. • Ajoute la mise à jour automatique des suites de tests dans IDT. IDT peut désormais télécharger les dernières suites de tests disponibles pour votre version de FreeRTOS. Avec cette fonctionnalité, vous pouvez : • Télécharger les dernières suites de tests à l'aide de la commande `upgrade-`

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Date de sortie	Notes de mise à jour
				`test-suite` . • Télécharger les dernières suites de tests en définissant un indicateur lorsque vous démarrez IDT. Utilisez l'`-u` *`flag`*option où *`flag`* peut être « y » pour toujours télécharger ou « n » pour utiliser la version existante. Lorsqu'il existe plusieurs versions de suite de tests disponibles,

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Date de sortie	Notes de mise à jour
				la dernière version est utilisée, sauf si vous spécifiez un ID de suite de tests au démarrage d'IDT. • Utilisez la nouvelle `list-supported-versions` option pour répertorier les versions de FreeRTOS et de la suite de tests prises en charge par la version installée d'IDT. • Répertorier les cas de test dans un groupe et exécuter

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Date de sortie	Notes de mise à jour
				des tests individuels. Les suites de tests sont versionnées à l'aide d'un format `major.minor.patch` commençant par 1.0.0. • Ajoute la `list-supported-products` commande : répertorie les versions de FreeRTOS et de suites de tests prises en charge par la version installée d'IDT. • Ajoute une `list-test-cases` commande — Répertorie les scénarios de test disponibles dans un groupe de test.

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Date de sortie	Notes de mise à jour
				• Ajoute l'`test-id`option pour la `run-suite` commande : utilisez cette option pour exécuter des scénarios de test individuels dans un groupe de test.

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Date de sortie	Notes de mise à jour
IDT v1.7.1	FRQ_1.0.0	202002.00		• Pour plus de détails, consultez le CHANGELOG.mdfichier dans GitHub. • Prend en charge la méthode de signature de code personnalisée pour les cas de test OTA de bout en bout, afin que vous puissiez utiliser vos propres commandes et scripts de signature de code pour signer des charges utiles OTA. • Ajoute une vérification préalable des ports série avant le début des tests. Les

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Date de sortie	Notes de mise à jour
				tests échouent rapidement avec un message d'erreur amélioré si le port série est mal configuré dans le fichier `device.json` . • Ajout d'une politique AWS gérée `AWSIoTDeviceTesterForFreeRTOSFullAccess` avec les autorisations requises pour s'exécuter AWS IoT Device Tester. Si les nouvelles versions nécessitent des autorisations supplémentaires, nous les ajoutons à

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Date de sortie	Notes de mise à jour
				cette politique gérée afin que vous n'ayez pas à mettre à jour manuellement vos autorisations IAM. • Le fichier nommé `AFQ_Report.xml` dans le répertoire des résultats est maintenant `FRQ_Report.xml` .

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Date de sortie	Notes de mise à jour
IDT v1.6.2	FRQ_1.0.0	202002.00		• Prend en charge les tests optionnels pour OTA via HTTPS afin de qualifier vos cartes de développement FreeRTOS. • Supporte le point de terminaison AWS IoT ATS lors des tests. • Prend en charge la fonctionnalité permettant d'informer les utilisateurs de la dernière version IDT avant le début de la suite de tests.

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Date de sortie	Notes de mise à jour
IDT v1.5.2	FRQ_1.0.0	201910.00		• Permet la qualification des appareils FreeRTOS dotés d'un élément sécurisé (clé intégrée). • Supporte les ports configurables du serveur Echo pour les Secure Sockets et les groupes de Wi-Fi test. • Supporte l'indicateur multiplicateur de délai d'attente pour augmenter les délais d'attente, ce qui est utile lorsque vous résolvez des erreurs liées au délai d'expiration. • Ajout d'un correctif de

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Date de sortie	Notes de mise à jour
				bogue pour l'analyse de journaux. • Prend en charge le point de terminais on Iot ats dans les tests.
IDT v1.4.1	FRQ_1.0.0	201908.00		• Ajout de la prise en charge de la nouvelle bibliothèque PKCS11 et des mises à jour de cas de test. • Introduction de codes d'erreur exploitab les. Pour de plus amples informations, consultez Codes d'erreur IDT. • Mise à jour de la stratégie IAM utilisée pour exécuter IDT.

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Date de sortie	Notes de mise à jour
IDT v1.3.2	FRQ_1.0.0	201906,00		• Ajout de la prise en charge des tests Bluetooth Low Energy (BLE). • Amélioration de l'expérience utilisateur pour les commandes de l'interface de ligne de commande (CLI) IDT. • Mise à jour de la stratégie IAM utilisée pour exécuter IDT.
IDT-FreeRTOS v1.2	FRQ_1.0.0	• FreeRTOS v1.4.8 • FreeRTOS v1.4.9		Ajout du support pour tester les appareils FreeRTOS avec le système de construction CMAKE.
IDT-FreeRTOS v1.1	FRQ_1.0.0			

AWS IoT Device Tester Version	Versions des suites de tests	Versions de FreeRTOS prises en charge	Date de sortie	Notes de mise à jour
IDT-FreeRTOS v1.0	FRQ_1.0.0			

Télécharger IDT pour FreeRTOS

Cette rubrique décrit les options de téléchargement d'IDT pour FreeRTOS. Vous pouvez utiliser l'un des liens de téléchargement de logiciels suivants ou suivre les instructions pour télécharger IDT par programmation.

> **Important**
>
> Depuis octobre 2022, AWS IoT Device Tester AWS IoT FreeRTOS Qualification (FRQ) 1.0 ne génère pas de rapports de qualification signés. Vous ne pouvez pas qualifier les nouveaux appareils AWS IoT FreeRTOS à répertorier dans le catalogue des appareils partenaires dans AWS le cadre du programme de qualification des appareils utilisant AWS les versions IDT FRQ 1.0. Bien que vous ne puissiez pas qualifier les appareils FreeRTOS avec IDT FRQ 1.0, vous pouvez continuer à tester vos appareils FreeRTOS avec FRQ 1.0. Nous vous recommandons d'utiliser IDT FRQ 2.0 pour qualifier et répertorier les appareils FreeRTOS dans le catalogue d'appareils partenaires.AWS

Rubriques

- Télécharger IDT manuellement
- Téléchargez IDT par programmation

En téléchargeant le logiciel, vous acceptez le contrat de AWS IoT Device Tester licence contenu dans l'archive de téléchargement.

> **Note**
>
> IDT ne prend pas en charge son exécution par plusieurs utilisateurs à partir d'un emplacement partagé, tel qu'un répertoire NFS ou un dossier partagé réseau Windows. Nous

vous recommandons d'extraire le package IDT sur une unité locale et d'exécuter le fichier binaire IDT sur votre station de travail locale.

Télécharger IDT manuellement

Cette rubrique répertorie les versions prises en charge d'IDT pour FreeRTOS. À titre de bonne pratique, nous vous recommandons d'utiliser la dernière version compatible avec votre version cible de FreeRTOS. AWS IoT Device Tester Les nouvelles versions de FreeRTOS peuvent nécessiter le téléchargement d'une nouvelle version de. AWS IoT Device Tester Vous recevez une notification lorsque vous lancez un test s'il n' AWS IoT Device Tester est pas compatible avec la version de FreeRTOS que vous utilisez.

Consultez Versions prises en charge de AWS IoT Device Tester

Téléchargez IDT par programmation

IDT fournit une opération d'API que vous pouvez utiliser pour récupérer une URL où vous pouvez télécharger IDT par programmation. Vous pouvez également utiliser cette opération d'API pour vérifier si vous disposez de la dernière version d'IDT. Cette opération d'API possède le point de terminaison suivant.

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

Pour appeler cette opération d'API, vous devez être autorisé à effectuer l'**`iot-device-tester:LatestIdt`**action. Incluez votre AWS signature, avec `iot-device-tester` comme nom du service

Requête d'API

HostOs — Le système d'exploitation de la machine hôte. Sélectionnez l'une des options suivantes :

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

TestSuiteType — Le type de suite de tests. Choisissez l'option suivante :

FR— IDT pour FreeRTOS

ProductVersion

(Facultatif) La version de FreeRTOS. Le service renvoie la dernière version compatible d'IDT pour cette version de FreeRTOS. Si vous ne spécifiez pas cette option, le service renvoie la dernière version d'IDT.

Réponse de l'API

La réponse de l'API est au format suivant. `DownloadURL`Inclut un fichier zip.

```
{
    "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
    }
 }
```

Exemples

Vous pouvez vous référer aux exemples suivants pour télécharger IDT par programmation.
Ces exemples utilisent des informations d'identification que vous stockez dans les variables d'`AWS_SECRET_ACCESS_KEY`environnement `AWS_ACCESS_KEY_ID` et. Pour suivre les meilleures pratiques de sécurité, ne stockez pas vos informations d'identification dans votre code.

Example

Exemple : téléchargement à l'aide de la version 7.75.0 ou ultérieure de cURL (Mac et Linux)

Si vous utilisez la version 7.75.0 ou ultérieure de cURL, vous pouvez utiliser le `aws-sigv4` drapeau pour signer la demande d'API. Cet exemple utilise jq pour analyser l'URL de téléchargement à partir de la réponse.

> **Warning**
>
> L'`aws-sigv4`indicateur nécessite que les paramètres de requête de la requête CURL GET soient dans l'ordre de HostOs/ProductVersion/TestSuiteTypeou HostOs/TestSuiteType. Les

commandes non conformes entraîneront une erreur lors de l'obtention de signatures non concordantes pour la chaîne canonique par l'API Gateway.
Si le paramètre facultatif ProductVersionest inclus, vous devez utiliser une version de produit prise en charge, comme indiqué dans Versions prises en charge de AWS IoT Device Tester for FreeRTOS.

- *us-west-2*Remplacez-le par votre Région AWS. Pour la liste des codes de région, voir Points de terminaison régionaux.
- *linux*Remplacez-le par le système d'exploitation de votre machine hôte.
- *202107.00*Remplacez-le par votre version de 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

Exemple : téléchargement à l'aide d'une version antérieure de cURL (Mac et Linux)

Vous pouvez utiliser la commande cURL suivante avec une AWS signature que vous signez et calculez. Pour plus d'informations sur la façon de signer et de calculer une AWS signature, consultez Signature des demandes AWS d'API.

- *linux*Remplacez-le par le système d'exploitation de votre machine hôte.
- Remplacez *Timestamp* par la date et l'heure, par exemple**20220210T004606Z**.
- Remplacez *Date* par la date, telle que**20220210**.
- *AWSRegion*Remplacez-le par votre Région AWS. Pour la liste des codes de région, voir Points de terminaison régionaux.
- *AWSSignature*Remplacez-le par la AWS signature que vous générez.

```
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

Exemple : téléchargement à l'aide d'un script Python

Cet exemple utilise la bibliothèque de requêtes Python. Cet exemple est adapté de l'exemple Python pour signer une demande d' AWS API dans la référence AWS générale.

- Remplacez *us-west-2* par votre région. Pour la liste des codes de région, voir Points de terminaison régionaux.
- *linux*Remplacez-le par le système d'exploitation de votre machine hôte.

```
# 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 avec suite de qualification FreeRTOS 2.0 (FRQ 2.0)

La suite de qualification FreeRTOS 2.0 est une version mise à jour de la suite de qualification FreeRTOS. Nous recommandons aux développeurs d'utiliser FRQ 2.0 car il contient des scénarios de test pertinents pour qualifier les appareils qui exécutent les bibliothèques FreeRTOS Long Term Support (LTS).

IDT pour FreeRTOS vérifie le port de FreeRTOS sur votre microcontrôleur et s'il communique efficacement avec. AWS IoT Plus précisément, il vérifie les interfaces de la couche de portage avec les bibliothèques FreeRTOS et si les référentiels de test FreeRTOS sont correctement implémentés. Il effectue également des tests de bout en bout avec. AWS IoT CoreLes tests exécutés par IDT pour FreeRTOS sont définis dans le référentiel FreeRTOS. GitHub

IDT pour FreeRTOS exécute des tests sous forme d'applications intégrées qu'il fait clignoter sur le microcontrôleur testé. Les images binaires de l'application incluent FreeRTOS, les interfaces FreeRTOS portées et les pilotes de périphériques embarqués. Le but des tests est de vérifier que les interfaces FreeRTOS portées fonctionnent correctement au-dessus des pilotes de votre appareil.

IDT for FreeRTOS génère des rapports de test que vous pouvez soumettre AWS IoT pour que votre matériel soit répertorié dans le AWS catalogue des appareils partenaires. Pour de plus amples informations, veuillez consulter AWS Device Qualification Program.

IDT pour FreeRTOS s'exécute sur un ordinateur hôte (Windows, macOS ou Linux) connecté à l'appareil testé. IDT configure et orchestre les cas de test et agrège les résultats. Il fournit également une interface de ligne de commande pour gérer l'exécution des tests.

Afin de tester votre appareil, IDT for FreeRTOS crée des ressources telles que des AWS IoT objets, des groupes FreeRTOS, des fonctions Lambda. Pour créer ces ressources, IDT pour FreeRTOS

utilise les informations d'identification configurées dans AWS le pour effectuer des appels d'API en `config.json` votre nom. Ces ressources sont allouées à différents moments d'un test.

Lorsque vous exécutez IDT pour FreeRTOS sur votre ordinateur hôte, il exécute les étapes suivantes :

1. Chargement et validation des informations d'identification et de la configuration de votre appareil.
2. Tests sélectionnés avec les ressources locales et les ressources cloud requises.
3. Élimination des ressources locales et des ressources cloud.
4. Génération de rapports de tests indiquant si votre carte a réussi les tests obligatoires pour la qualification.

Configurer les prérequis de qualification LTS

Cette section décrit les prérequis pour tester les microcontrôleurs avec. AWS IoT Device Tester

Préparez-vous à la qualification FreeRTOS

> **Note**
>
> AWS IoT Device Tester for FreeRTOS recommande vivement d'utiliser le dernier correctif de la version la plus récente. FreeRTOS-LTS

IDT pour FRQ 2.0 est une qualification pour FreeRTOS. Avant de lancer IDT FRQ 2.0 pour la qualification, vous devez terminer la qualification de votre plateau dans le guide de qualification FreeRTOS. Pour porter des bibliothèques, les tester et les configurer`manifest.yml`, consultez la section Portage des bibliothèques FreeRTOS dans le Guide de portage de FreeRTOS. FRQ 2.0 contient un processus de qualification différent. Consultez les derniers changements en matière de qualification dans le guide de qualification FreeRTOS pour plus de détails.

Le FreeRTOS-Libraries-Integration-Testsréférentiel doit être présent pour qu'IDT puisse s'exécuter. Consultez la section README.mdsur la façon de cloner et de porter ce référentiel vers votre projet source. FreeRTOS-Libraries-Integration-Tests doit inclure le fichier `manifest.yml` situé à la racine de votre projet, pour que IDT puisse fonctionner.

> **Note**
>
> IDT dépend de l'implémentation de`UNITY_OUTPUT_CHAR`. Les journaux des résultats des tests et les journaux des appareils ne doivent pas être entrelacés. Voir la section Implémentation des macros de journalisation des bibliothèques dans le Guide de portage de FreeRTOS pour plus de détails.

Télécharger IDT pour FreeRTOS

Chaque version de FreeRTOS possède une version correspondante d'IDT pour FreeRTOS afin d'effectuer des tests de qualification. Téléchargez la version appropriée d'IDT pour FreeRTOS à partir des versions prises en charge de pour FreeRTOS. AWS IoT Device Tester

Extrayez IDT pour FreeRTOS vers un emplacement du système de fichiers où vous disposez d'autorisations de lecture et d'écriture. Microsoft Windows ayant une limite de caractères pour la longueur du chemin, extrayez IDT pour FreeRTOS dans un répertoire racine tel que ou. `C:\` `D:\`

> **Note**
>
> Plusieurs utilisateurs ne doivent pas exécuter IDT à partir d'un emplacement partagé, tel qu'un répertoire NFS ou un dossier partagé sur le réseau Windows. Cela entraînera des pannes ou une corruption des données. Nous vous recommandons d'extraire le package IDT sur un disque local.

Télécharger Git

Git doit être installé sur IDT comme condition préalable pour garantir l'intégrité du code source.

Suivez les instructions du GitHubguide pour installer Git. Pour vérifier la version actuellement installée de Git, entrez la commande `git --version` sur le terminal.

> **Warning**
>
> IDT utilise Git pour s'aligner sur le statut propre ou sale d'un répertoire. Si Git n'est pas installé, les groupes de `FreeRTOSIntegrity` test échoueront ou ne s'exécuteront pas

comme prévu. Si IDT renvoie une erreur telle que `git executable not found` ou`git command not found`, installez ou réinstallez Git et réessayez.

Rubriques

- Créez un AWS compte
- AWS IoT Device Tester politique gérée
- (Facultatif) Installez le AWS Command Line Interface

Créez un AWS compte

> **Note**
>
> La suite de qualification IDT complète n'est prise en charge que dans les cas suivants Régions AWS
>
> - USA Est (Virginie du Nord)
> - USA Ouest (Oregon)
> - Asie Pacifique (Tokyo)
> - Europe (Irlande)

Afin de tester votre appareil, IDT for FreeRTOS crée des ressources telles que des AWS IoT objets, des groupes FreeRTOS et des fonctions Lambda. Pour créer ces ressources, IDT for FreeRTOS nécessite que vous créiez et configuriez AWS un compte, ainsi qu'une politique IAM qui accorde à IDT for FreeRTOS l'autorisation d'accéder aux ressources en votre nom lors de l'exécution des tests.

Les étapes suivantes consistent à créer et à configurer votre AWS compte.

1. Si vous avez déjà un AWS compte, passez à l'étape suivante. Sinon, créez un AWS compte.
2. Suivez les étapes décrites dans Création de rôles IAM. N'ajoutez pas d'autorisations ou de politiques pour le moment.
3. Pour exécuter des tests de qualification OTA, passez à l'étape 4. Sinon, passez à l'étape 5.
4. Associez la politique intégrée d'autorisations OTA IAM à votre rôle IAM.

a.

> **Important**
>
> Le modèle de stratégie suivant accorde à IDT l'autorisation de créer des rôles, de créer des stratégies et d'attacher des stratégies à des rôles. IDT pour FreeRTOS utilise ces autorisations pour les tests qui créent des rôles. Bien que le modèle de politique ne fournisse pas de privilèges d'administrateur à l'utilisateur, les autorisations peuvent être utilisées pour obtenir un accès administrateur à votre AWS compte.

b. Suivez les étapes ci-dessous pour associer les autorisations nécessaires à votre rôle IAM :

i. Sur la page Autorisations, choisissez Ajouter des autorisations.

ii. Choisissez Créer une politique en ligne.

iii. Sélectionnez l'onglet JSON et copiez les autorisations suivantes dans la zone de texte JSON. Utilisez le modèle sous La plupart des régions si vous n'êtes pas dans la région de Chine. Si vous vous trouvez dans la région de Chine, utilisez le modèle sous Régions de Pékin et du 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

Le modèle de politique suivant peut être utilisé dans les régions de Pékin et de Ningxia.

iv. Lorsque vous avez terminé, sélectionnez Review policy (Examiner une politique).

v. Entrez IDTFreeRTOSIAMPermissionscomme nom de la politique.

vi. Choisissez Create Policy (Créer une politique).

5. AWSIoTDeviceTesterForFreeRTOSFullAccessAttachez-vous à votre rôle IAM.

   a. Pour associer les autorisations nécessaires à votre rôle IAM :

      i. Sur la page Autorisations, choisissez Ajouter des autorisations.

      ii. Choisissez Attacher des politiques.

      iii. Recherchez la AWSIoTDeviceTesterForFreeRTOSFullAccesspolitique. Cochez la case.

   b. Choisissez Ajouter des autorisations.

6. Exportez les informations d'identification pour IDT. Voir Obtenir les informations d'identification du rôle IAM pour l'accès à la CLI pour plus de détails.

AWS IoT Device Tester politique gérée

La politique `AWSIoTDeviceTesterForFreeRTOSFullAccess` gérée contient les AWS IoT Device Tester autorisations suivantes pour la vérification des versions, les fonctionnalités de mise à jour automatique et la collecte de métriques.

- `iot-device-tester:SupportedVersion`

  Accorde AWS IoT Device Tester l'autorisation de récupérer la liste des produits pris en charge, des suites de tests et des versions IDT.

- `iot-device-tester:LatestIdt`

  Accorde AWS IoT Device Tester l'autorisation de récupérer la dernière version d'IDT disponible au téléchargement.

- `iot-device-tester:CheckVersion`

  Accorde AWS IoT Device Tester l'autorisation de vérifier la compatibilité des versions pour IDT, les suites de tests et les produits.

- `iot-device-tester:DownloadTestSuite`

  AWS IoT Device Tester Autorise le téléchargement des mises à jour de la suite de tests.

- `iot-device-tester:SendMetrics`

  Accorde AWS l'autorisation de collecter des statistiques relatives à l'utilisation AWS IoT Device Tester interne.

(Facultatif) Installez le AWS Command Line Interface

Vous préférerez peut-être utiliser le AWS CLI pour effectuer certaines opérations. Si vous ne l'avez pas AWS CLI installé, suivez les instructions de la section Installer le AWS CLI.

Configurez le AWS CLI pour la AWS région que vous souhaitez utiliser en l'exécutant à aws configure partir d'une ligne de commande. Pour plus d'informations sur les AWS régions qui prennent en charge IDT pour FreeRTOS,AWS consultez Régions et points de terminaison. Pour plus d'informations sur la aws configure section Configuration rapide avec aws configure.

Premier test de votre carte microcontrôleur

Vous pouvez utiliser IDT pour FreeRTOS pour tester votre implémentation des bibliothèques FreeRTOS. Après avoir porté les bibliothèques FreeRTOS pour les pilotes de périphériques de votre carte, exécutez-les pour exécuter les tests AWS IoT Device Tester de qualification sur votre carte microcontrôleur.

Ajouter des couches de transfert pour les bibliothèques

Pour transférer FreeRTOS sur votre appareil, consultez le Guide de portage de FreeRTOS. Lors de l'implémentation du référentiel de tests FreeRTOS et du portage des couches FreeRTOS, vous devez fournir `manifest.yml` un chemin d'accès à chaque bibliothèque, y compris le référentiel de tests. Ce fichier se trouvera dans le répertoire racine de votre code source. Consultez les instructions du fichier manifeste pour plus de détails.

Configurez votre AWS informations d'identification pour AWS IoT Device Tester pour communiquer avec le AWS Cloud

Vous devez configurer vos AWS informations d'identification AWS IoT Device Tester pour pouvoir communiquer avec le AWS Cloud. Pour plus d'informations, voir Configurer les AWS informations d'identification et la région pour le développement. Les AWS informations d'identification valides sont spécifiées dans le fichier *devicetester_extract_location*/devicetester_freertos_*[win|mac|linux]*/configs/config.json de configuration.

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

"auth": {
```

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

L'`auth`attribut du `config.json` fichier possède un champ de méthode qui contrôle AWS l'authentification et peut être déclaré en tant que fichier ou environnement. La définition du champ sur environnement extrait vos AWS informations d'identification des variables d'environnement de votre machine hôte. La définition du champ sur fichier permet d'importer un profil spécifié à partir du fichier de `.aws/credentials` configuration.

Rubriques

- Créer un pool d'appareils dans IDT pour FreeRTOS
- Configurer les paramètres de création, de flash et de test

Créer un pool d'appareils dans IDT pour FreeRTOS

Les appareils à tester sont organisés dans des groupes. Chaque groupe d'appareils se compose d'un ou de plusieurs appareils identiques. Vous pouvez configurer IDT pour FreeRTOS afin de tester un seul appareil ou plusieurs appareils dans un pool. Pour accélérer le processus de qualification, IDT for FreeRTOS peut tester des appareils avec les mêmes spécifications en parallèle. Il utilise une méthode séquentielle pour exécuter un groupe de tests différents pour chaque appareil d'un groupe.

Le `device.json` fichier possède un tableau à son niveau supérieur. Chaque attribut du tableau correspond à un nouveau pool de périphériques. Chaque pool de périphériques possède un attribut de tableau d'appareils, dans lequel plusieurs appareils sont déclarés. Dans le modèle, il existe un pool de périphériques et un seul appareil dans ce pool de périphériques. Vous pouvez ajouter un ou plusieurs appareils à un groupe d'appareils en modifiant la section `devices` du modèle `device.json` dans le dossier `configs`.

> **Note**
>
> Tous les appareils d'un même pool doivent avoir les mêmes spécifications techniques et le même SKU. Pour permettre des compilations parallèles du code source pour différents groupes de test, IDT pour FreeRTOS copie le code source dans un dossier de résultats à l'intérieur du dossier extrait d'IDT pour FreeRTOS. Vous devez référencer le chemin du code source dans votre commande de build ou de flash à l'aide de la `testdata.sourcePath`

variable. IDT pour FreeRTOS remplace cette variable par un chemin temporaire du code source copié. Pour de plus amples informations, veuillez consulter IDT pour les variables FreeRTOS.

Voici un exemple de `device.json` fichier qui a été utilisé pour créer un pool de périphériques avec plusieurs appareils.

```
[
    {
        "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"
              }
            ]
          }
        ]
    }
]
```

Les attributs suivants sont utilisés dans le fichier `device.json` :

`id`

Un ID alphanumérique défini par l'utilisateur qui identifie de façon unique un groupe d'appareils. Les appareils appartenant à un groupe doivent être du même type. Lorsque vous exécutez une suite de tests, les appareils du groupe sont utilisés pour paralléliser la charge de travail.

`sku`

Une valeur alphanumérique qui identifie de façon unique la carte que vous testez. La référence est utilisée pour effectuer le suivi des cartes qualifiées.

> **Note**
>
> Si vous souhaitez mettre votre carte en vente dans le catalogue des appareils AWS partenaires, le SKU que vous spécifiez ici doit correspondre au SKU que vous avez utilisé lors du processus de mise en vente.

`features`

Tableau contenant les fonctionnalités prises en charge par l'appareil. AWS IoT Device Tester utilise ces informations pour sélectionner les tests de qualification à exécuter.

Les valeurs prises en charge sont :

`Wifi`

Indique si votre carte possède Wi-Fi des capacités.

`Cellular`

Indique si votre carte possède des fonctionnalités cellulaires.

`PKCS11`

Indique l'algorithme de chiffrement de clé publique que la carte prend en charge. PKCS11 est requis pour la qualification. Les valeurs prises en charge sont `ECC`RSA, et`Both`. `Both`indique que le tableau prend en charge les deux `ECC` et`RSA`.

`KeyProvisioning`

Indique la méthode d'écriture d'un certificat X.509 client de confiance sur votre tableau.

Les valeurs valides sont `Import`Onboard, `Both` et`No`. `Onboard`Both, ou l'approvisionnement `No` des clés est requis pour la qualification. `Import`à elle seule, n'est pas une option valide pour la qualification.

- À utiliser `Import` uniquement si votre forum autorise l'importation de clés privées. `Import`La sélection n'est pas une configuration valide pour la qualification et ne doit être utilisée qu'à des fins de test, en particulier pour les cas de test PKCS11. `Onboard`, `Both` ou `No` est requis pour la qualification.
- À utiliser `Onboard` si votre carte prend en charge les clés privées intégrées (par exemple, si votre appareil possède un élément sécurisé ou si vous préférez générer votre propre paire

de clés et votre propre certificat). Veillez à ajouter un élément `secureElementConfig` dans chacune des sections de l'appareil et indiquez le chemin absolu vers le fichier de clé publique dans le champ `publicKeyAsciiHexFilePath`.

- À utiliser `Both` si votre tableau prend en charge à la fois l'importation de clés privées et la génération de clés intégrée pour le provisionnement des clés.
- À utiliser `No` si votre tableau ne prend pas en charge l'approvisionnement en clés. `Non`'est une option valide que lorsque votre appareil est également préconfiguré.

`OTA`

Indique si la carte prend en charge la fonctionnalité de mise à jour à distance (OTA). L'attribut `OtaDataPlaneProtocol` indique le protocole de plan de données OTA pris en charge par le périphérique. L'OTA avec le protocole de plan de données HTTP ou MQTT est requis pour la qualification. Pour ignorer l'exécution des tests OTA pendant les tests, définissez la fonctionnalité OTA sur `No` et l'`OtaDataPlaneProtocol`attribut sur`None`. Il ne s'agira pas d'une course de qualification.

`BLE`

Indique si la carte prend en charge Bluetooth Low Energy (BLE).

`devices.id`

Un identificateur unique défini par l'utilisateur pour l'appareil testé.

`devices.connectivity.serialPort`

Le port série de l'ordinateur hôte utilisé pour se connecter aux appareils testés.

`devices.secureElementConfig.PublicKeyAsciiHexFilePath`

Obligatoire si votre planche n'est PAS `pre-provisioned` ou n'`PublicDeviceCertificateArn`est pas fournie. Comme `Onboard` il s'agit d'un type obligatoire de fourniture de clés, ce champ est actuellement obligatoire pour le groupe de FullTransportInterfaceTLS test. Si votre appareil l'est`pre-provisioned`, `PublicKeyAsciiHexFilePath` il est facultatif et n'a pas besoin d'être inclus.

Le bloc suivant est un chemin absolu vers le fichier contenant la clé publique en octets hexadécimaux extraite de la clé `Onboard` privée.

```
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
```

Si votre clé publique est au format .der, vous pouvez l'encoder directement en hexadécimal pour générer le fichier hexadécimal.

Pour générer le fichier hexadécimal à partir d'une clé publique .der, entrez la commande suivante : xxd

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

Si votre clé publique est au format .pem, vous pouvez extraire les en-têtes et pieds de page codés en base64 et les décoder au format binaire. Ensuite, vous codez la chaîne binaire en hexadécimal pour générer le fichier hexadécimal.

Pour générer un fichier hexadécimal pour une clé publique .pem, procédez comme suit :

1. Exécutez la base64 commande suivante pour supprimer l'en-tête et le pied de page base64 de la clé publique. La clé décodée, nomméebase64key, est ensuite sortie dans le fichier `pubkey.der` :

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

2. Exécutez la xxd commande suivante pour convertir `pubkey.der` au format hexadécimal. La clé obtenue est enregistrée sous *`outFile`*

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

`devices.secureElementConfig.PublicDeviceCertificateArn`

L'ARN du certificat de votre élément sécurisé qui est téléchargé vers AWS IoT Core. Pour plus d'informations sur le téléchargement de votre certificat vers AWS IoT Core, consultez la section relative aux certificats X.509 clients dans le Guide du AWS IoT développeur.

`devices.secureElementConfig.SecureElementSerialNumber`

(Facultatif) Numéro de série de l'élément sécurisé. Le numéro de série est éventuellement utilisé pour créer des certificats d'appareil pour le provisionnement des clés JITR.

`devices.secureElementConfig.preProvisioned`

(Facultatif) Réglez sur « Oui » si l'appareil possède un élément sécurisé préconfiguré avec des informations d'identification verrouillées, qui ne peut pas importer, créer ou détruire des objets. Si cet attribut est défini sur Oui, vous devez fournir les étiquettes pkcs11 correspondantes.

`devices.secureElementConfig.pkcs11JITPCodeVerifyRootCertSupport`

(Facultatif) Définissez ce paramètre sur Oui si l'implémentation CorePKCS11 de l'appareil prend en charge le stockage pour JITP. Cela permettra le test JITP lors du `codeverify` test du principal PKCS 11 et nécessitera la fourniture d'étiquettes de clé de vérification du code, de certificat JITP et de certificat racine PKCS 11.

`identifiers`

(Facultatif) Un tableau de paires nom-valeur arbitraires. Vous pouvez utiliser ces valeurs dans les commandes de création et flash décrites dans la section suivante.

Configurer les paramètres de création, de flash et de test

IDT pour FreeRTOS crée et affiche automatiquement les tests sur votre carte. Pour activer cela, vous devez configurer IDT pour exécuter les commandes de compilation et de flash pour votre matériel. Les paramètres de création et flash sont configurés dans le modèle de fichier `userdata.json` situé dans le dossier `config`.

Configurer les paramètres pour tester un seul appareil

Les paramètres de création, de flash et de test sont définis dans le fichier `configs/userdata.json`. L'exemple JSON suivant montre comment configurer IDT pour FreeRTOS afin de tester plusieurs appareils :

```
{
    "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>"
     }
  }
```

Voici les attributs utilisés dans le fichier `userdata.json` :

`sourcePath`

Le chemin d'accès à la racine du code source FreeRTOS porté.

`retainModifiedSourceDirectories`

(Facultatif) Vérifie s'il faut conserver les répertoires sources modifiés utilisés lors de la construction et du flashage à des fins de débogage. S'il est défini sur`true`, les répertoires sources modifiés sont nommés RetainedSrc et se trouvent dans les dossiers du journal des résultats de chaque groupe de test exécuté. S'il n'est pas inclus, le champ est défini par défaut sur. `false`

`freeRTOSTestParamConfigPath`

Le chemin d'accès au `test_param_config.h` fichier à FreeRTOS-Libraries-Integration-Tests intégrer. Ce fichier doit utiliser la variable `{{testData.sourcePath}}` placeholder pour le rendre relatif à la racine du code source. AWS IoT Device Tester utilise les paramètres de ce fichier pour configurer les tests.

`freeRTOSTestExecutionConfigPath`

Le chemin d'accès au `test_execution_config.h` fichier à FreeRTOS-Libraries-Integration-Tests intégrer. Ce fichier doit utiliser la variable d'`{{testData.sourcePath}}`espace réservé pour le rendre relatif à la racine du dépôt. AWS IoT Device Tester utilise ce fichier pour contrôler les tests à exécuter.

`freeRTOSVersion`

La version de FreeRTOS, y compris la version de correctif utilisée dans votre implémentation. Voir Versions prises en charge de AWS IoT Device Tester pour FreeRTOS pour les versions de FreeRTOS compatibles avec FreeRTOS. AWS IoT Device Tester

`buildTool`

La commande permettant de créer votre code source. Toutes les références au chemin du code source dans la commande de construction doivent être remplacées par la AWS IoT Device Tester variable`{{testData.sourcePath}}`. Utilisez l'`{{config.idtRootPath}}`espace réservé pour référencer un script de génération relatif au chemin AWS IoT Device Tester racine.

`flashTool`

Commande permettant de flasher une image sur votre appareil. Toutes les références au chemin du code source dans la commande flash doivent être remplacées par la AWS IoT Device Tester variable`{{testData.sourcePath}}`. Utilisez l'`{{config.idtRootPath}}`espace réservé pour référencer un script flash relatif au chemin AWS IoT Device Tester racine.

> **Note**
>
> La nouvelle structure des tests d'intégration avec FRQ 2.0 ne nécessite pas de variables de chemin telles que `{{enableTests}}` et`{{buildImageName}}`. Les tests OTA de bout en bout sont exécutés avec les modèles de configuration fournis dans le FreeRTOS-Libraries-Integration-Tests GitHub référentiel. Si les fichiers du GitHub référentiel sont présents dans votre projet source parent, le code source n'est pas modifié entre les tests. Si une image de compilation différente pour OTA de bout en bout est

nécessaire, vous devez créer cette image dans le script de génération et la spécifier dans le `userdata.json` fichier spécifié ci-dessous`otaConfiguration`.

`testStartDelayms`

Spécifie le nombre de millisecondes que le lanceur de tests FreeRTOS attendra avant de commencer à exécuter les tests. Cela peut être utile si l'appareil testé commence à produire des informations de test importantes avant qu'IDT n'ait la chance de se connecter et de commencer à se connecter en raison de problèmes de réseau ou d'autres problèmes de latence. Cette valeur s'applique uniquement aux groupes de test FreeRTOS, et non aux autres groupes de test qui n'utilisent pas le lanceur de tests FreeRTOS, tels que les tests OTA. Si vous recevez une erreur liée à la valeur 10 attendue mais que vous en avez reçu 5, ce champ doit être défini sur 5000.

`echoServerConfiguration`

Configuration permettant de configurer le serveur d'écho pour le test TLS. Ce champ est obligatoire.

`keyGenerationMethod`

Le serveur Echo est configuré avec cette option. Les options sont EC ou RSA.

`serverPort`

Le numéro de port sur lequel s'exécute le serveur Echo.

`otaConfiguration`

Configuration pour les tests OTA PAL et OTA E2E. Ce champ est obligatoire.

`otaE2EFirmwarePath`

Chemin d'accès à l'image du bin OTA qu'IDT utilise pour les tests OTA de bout en bout.

`otaPALCertificatePath`

Le chemin d'accès au certificat pour le test OTA PAL sur l'appareil. Ceci est utilisé pour vérifier la signature. Par exemple, ecdsa-sha256-signer.crt.pem.

`deviceFirmwarePath`

Le chemin d'accès au nom codé en dur de l'image du microprogramme à démarrer. Si votre appareil n'utilise PAS le système de fichiers pour le démarrage du microprogramme, spécifiez ce champ sous la forme `'NA'`. Si votre appareil utilise le système de fichiers pour le

démarrage du microprogramme, spécifiez le chemin ou le nom de l'image de démarrage du microprogramme.

`codeSigningConfiguration`

`signingMethod`

Méthode de signature de code. Les valeurs possibles sont AWS ou Custom.

> **Note**
>
> Pour les régions de Pékin et de Ningxia, utilisez Custom. AWS la signature de code n'est pas prise en charge dans cette région.

`signerHashingAlgorithm`

Algorithme de hachage pris en charge sur le périphérique. Les valeurs possibles sont `SHA1` ou `SHA256`.

`signerSigningAlgorithm`

Algorithme de signature pris en charge sur le périphérique. Les valeurs possibles sont `RSA` ou `ECDSA`.

`signerCertificate`

Certificat de confiance utilisé pour l'OTA. Pour la méthode de signature AWS du code, utilisez l'Amazon Resource Name (ARN) pour le certificat sécurisé chargé sur le AWS Certificate Manager. Pour la méthode de signature de code personnalisée, utilisez le chemin absolu vers le fichier de certificat du signataire. Pour plus d'informations sur la création d'un certificat sécurisé, voir Création d'un certificat de signature de code.

`untrustedSignerCertificate`

L'ARN ou le chemin de fichier d'un deuxième certificat utilisé dans certains tests OTA en tant que certificat non fiable. Pour plus d'informations sur la création d'un certificat, voir Création d'un certificat de signature de code.

`signerCertificateFileName`

Nom de fichier du certificat de signature de code sur l'appareil. Cette valeur doit correspondre au nom de fichier que vous avez fourni lors de l'exécution de la `aws acm import-certificate` commande.

`compileSignerCertificate`

Valeur booléenne qui détermine le statut du certificat de vérification de signature. Les valeurs valides sont `true` et `false`.

Définissez cette valeur sur true si le certificat de vérification de signature du signataire du code n'est pas fourni ou flashé. Il doit être compilé dans le projet. AWS IoT Device Tester récupère le certificat sécurisé et le compile dans. `aws_codesigner_certificate.h`

`signerPlatform`

Algorithme de signature et de hachage utilisé par AWS Code Signer lors de la création de la tâche de mise à jour OTA. Actuellement, les valeurs possibles pour ce champ sont `AmazonFreeRTOS-TI-CC3220SF` et `AmazonFreeRTOS-Default`.

- Choisissez `AmazonFreeRTOS-TI-CC3220SF`, si `SHA1` et `RSA`.
- Choisissez `AmazonFreeRTOS-Default`, si `SHA256` et `ECDSA`.
- Si vous avez besoin de `SHA256` | `RSA` ou `SHA1` | `ECDSA` pour votre configuration, contactez-nous pour obtenir une assistance complémentaire.
- Configurez `signCommand` si vous avez choisi `Custom` pour `signingMethod`.

`signCommand`

Deux espaces réservés `{{inputImageFilePath}}` et `{{outputSignatureFilePath}}` sont obligatoires dans la commande. `{{inputImageFilePath}}` est le chemin d'accès au fichier de l'image créée par IDT qu'il faut signer. `{{outputSignatureFilePath}}` est le chemin d'accès au fichier de la signature qui sera généré par le script.

`pkcs11LabelConfiguration`

La configuration des étiquettes PKCS11 nécessite au moins un ensemble d'étiquettes d'étiquette de certificat de périphérique, d'étiquette de clé publique et d'étiquette de clé privée pour exécuter les groupes de test PKCS11. Les étiquettes PKCS11 requises sont basées sur la configuration de votre appareil dans le `device.json` fichier. Si le préprovisionnement est défini sur Oui`device.json`, les étiquettes requises doivent être l'une des suivantes en fonction de ce qui est choisi pour la fonctionnalité PKCS11.

- `PreProvisionedEC`
- `PreProvisionedRSA`

Si le préprovisionnement est défini sur Non`device.json`, les étiquettes requises sont les suivantes :

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

Les trois libellés suivants sont obligatoires uniquement si vous sélectionnez Oui pour `pkcs11JITPCodeVerifyRootCertSupport` dans votre `device.json` fichier.

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

Les valeurs de ces champs doivent correspondre aux valeurs définies dans le Guide de portage de FreeRTOS.

`pkcs11LabelDevicePrivateKeyForTLS`

(Facultatif) Cette étiquette est utilisée pour l'étiquette PKCS #11 de la clé privée. Pour les appareils dotés d'un support intégré et d'importation pour l'approvisionnement des clés, cette étiquette est utilisée à des fins de test. Cette étiquette peut être différente de celle définie pour le boîtier préapprovisionné. Si le provisionnement des clés est défini sur Non et que le préprovisionnement est défini sur Oui`device.json`, ce paramètre ne sera pas défini.

`pkcs11LabelDevicePublicKeyForTLS`

(Facultatif) Cette étiquette est utilisée pour l'étiquette PKCS #11 de la clé publique. Pour les appareils dotés d'un support intégré et d'importation pour l'approvisionnement des clés, cette étiquette est utilisée à des fins de test. Cette étiquette peut être différente de celle définie pour le boîtier préapprovisionné. Si le provisionnement des clés est défini sur Non et que le préprovisionnement est défini sur Oui`device.json`, ce paramètre ne sera pas défini.

`pkcs11LabelDeviceCertificateForTLS`

(Facultatif) Cette étiquette est utilisée pour l'étiquette PKCS #11 du certificat de l'appareil. Pour les appareils dotés d'un support intégré et d'importation pour l'approvisionnement des clés, cette étiquette sera utilisée à des fins de test. Cette étiquette peut être différente de celle définie pour le boîtier préapprovisionné. Si le provisionnement des clés est défini sur Non et que le préprovisionnement est défini sur Oui`device.json`, ce paramètre ne sera pas défini.

pkcs11LabelPreProvisionedECDevicePrivateKeyForTLS

(Facultatif) Cette étiquette est utilisée pour l'étiquette PKCS #11 de la clé privée. Pour les appareils dotés d'éléments sécurisés ou présentant des limitations matérielles, celui-ci portera une étiquette différente afin de préserver les AWS IoT informations d'identification. Si votre appareil prend en charge le préapprovisionnement avec une clé EC, fournissez cette étiquette. Lorsque PreProvisioned est défini sur Yes in`device.json`, cette étiquette, ou les deux`pkcs11LabelPreProvisionedRSADevicePrivateKeyForTLS`, doivent être fournies. Cette étiquette peut être différente de celle définie pour les valises à bord et les valises d'importation.

pkcs11LabelPreProvisionedECDevicePublicKeyForTLS

(Facultatif) Cette étiquette est utilisée pour l'étiquette PKCS #11 de la clé publique. Pour les appareils dotés d'éléments sécurisés ou présentant des limitations matérielles, celui-ci portera une étiquette différente afin de préserver les AWS IoT informations d'identification. Si votre appareil prend en charge le préapprovisionnement avec une clé EC, fournissez cette étiquette. Lorsque PreProvisioned est défini sur Yes in`device.json`, cette étiquette, ou les deux`pkcs11LabelPreProvisionedRSADevicePublicKeyForTLS`, doivent être fournies. Cette étiquette peut être différente de celle définie pour les valises à bord et les valises d'importation.

pkcs11LabelPreProvisionedECDeviceCertificateForTLS

(Facultatif) Cette étiquette est utilisée pour l'étiquette PKCS #11 du certificat de l'appareil. Pour les appareils dotés d'éléments sécurisés ou présentant des limitations matérielles, celui-ci portera une étiquette différente afin de préserver les AWS IoT informations d'identification. Si votre appareil prend en charge le préapprovisionnement avec une clé EC, fournissez cette étiquette. Lorsque PreProvisioned est défini sur Yes in`device.json`, cette étiquette, ou les deux`pkcs11LabelPreProvisionedRSADeviceCertificateForTLS`, doivent être fournies. Cette étiquette peut être différente de celle définie pour les valises à bord et les valises d'importation.

pkcs11LabelPreProvisionedRSADevicePrivateKeyForTLS

(Facultatif) Cette étiquette est utilisée pour l'étiquette PKCS #11 de la clé privée. Pour les appareils dotés d'éléments sécurisés ou présentant des limitations matérielles, celui-ci portera une étiquette différente afin de préserver les AWS IoT informations d'identification. Si votre appareil prend en charge le préprovisionnement à l'aide d'une clé RSA, fournissez cette étiquette. Lorsque PreProvisioned est défini sur Yes in`device.json`, cette étiquette, ou les deux`pkcs11LabelPreProvisionedECDevicePrivateKeyForTLS`, doivent être fournies.

pkcs11LabelPreProvisionedRSADevicePublicKeyForTLS

(Facultatif) Cette étiquette est utilisée pour l'étiquette PKCS #11 de la clé publique. Pour les appareils dotés d'éléments sécurisés ou présentant des limitations matérielles, celui-ci portera une étiquette différente afin de préserver les AWS IoT informations d'identification. Si votre appareil prend en charge le préprovisionnement à l'aide d'une clé RSA, fournissez cette étiquette. Lorsque PreProvisioned est défini sur Yes in`device.json`, cette étiquette, ou les deux`pkcs11LabelPreProvisionedECDevicePublicKeyForTLS`, doivent être fournies.

pkcs11LabelPreProvisionedRSADeviceCertificateForTLS

(Facultatif) Cette étiquette est utilisée pour l'étiquette PKCS #11 du certificat de l'appareil. Pour les appareils dotés d'éléments sécurisés ou présentant des limitations matérielles, celui-ci portera une étiquette différente afin de préserver les AWS IoT informations d'identification. Si votre appareil prend en charge le préprovisionnement à l'aide d'une clé RSA, fournissez cette étiquette. Lorsque PreProvisioned est défini sur Yes in`device.json`, cette étiquette, ou les deux`pkcs11LabelPreProvisionedECDeviceCertificateForTLS`, doivent être fournies.

pkcs11LabelCodeVerifyKey

(Facultatif) Cette étiquette est utilisée pour l'étiquette PKCS #11 de la clé de vérification du code. Si votre appareil prend en charge le stockage PKCS #11 du certificat JITP, de la clé de vérification du code et du certificat racine, fournissez cette étiquette. Lorsque `pkcs11JITPCodeVerifyRootCertSupport` la `device.json` valeur Oui est définie sur Oui, cette étiquette doit être fournie.

pkcs11LabelJITPCertificate

(Facultatif) Cette étiquette est utilisée pour l'étiquette PKCS #11 du certificat JITP. Si votre appareil prend en charge le stockage PKCS #11 du certificat JITP, de la clé de vérification du code et du certificat racine, fournissez cette étiquette. Lorsque `pkcs11JITPCodeVerifyRootCertSupport` la `device.json` valeur Oui est définie sur Oui, cette étiquette doit être fournie.

IDT pour les variables FreeRTOS

Les commandes permettant de créer votre code et de flasher l'appareil peuvent nécessiter une connectivité ou d'autres informations sur vos appareils pour fonctionner correctement. AWS IoT Device Tester vous permet de référencer les informations du périphérique dans Flash et de créer des

commandes à l'aide de JsonPath. À l'aide d' JsonPathexpressions simples, vous pouvez récupérer les informations requises spécifiées dans votre `device.json` fichier.

Variables de chemin

IDT pour FreeRTOS définit les variables de chemin suivantes qui peuvent être utilisées dans les lignes de commande et les fichiers de configuration :

`{{testData.sourcePath}}`

Développe le code source du chemin d'accès. Si vous utilisez cette variable, elle doit être utilisée dans les commandes flash et build.

`{{device.connectivity.serialPort}}`

Extension au port série.

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

Extension jusqu'au numéro de série de votre appareil.

`{{config.idtRootPath}}`

S'étend jusqu'au chemin AWS IoT Device Tester racine.

Interface utilisateur pour IDT pour la suite de qualification FreeRTOS 2.0 (FRQ 2.0)

AWS IoT Device Tester for FreeRTOS (IDT for FreeRTOS) inclut une interface utilisateur (UI) Web dans laquelle vous pouvez interagir avec l'application de ligne de commande IDT et les fichiers de configuration associés. Vous utilisez l'interface utilisateur IDT pour FreeRTOS pour créer une nouvelle configuration ou modifier une configuration existante pour votre appareil. Vous pouvez également utiliser l'interface utilisateur pour appeler l'application IDT et exécuter les tests FreeRTOS sur votre appareil.

Pour plus d'informations sur l'utilisation de la ligne de commande pour exécuter des tests de qualification, consultezPremier test de votre carte microcontrôleur.

Cette section décrit les prérequis pour l'interface utilisateur IDT pour FreeRTOS et comment exécuter des tests de qualification à partir de l'interface utilisateur.

Rubriques

- Configurer les prérequis IDT
- Configurer AWS informations d'identification pour utiliser l'interface utilisateur IDT
- Ouvrez l'interface utilisateur IDT pour FreeRTOS
- Création d'une nouvelle configuration
- Modifier une configuration existante
- Exécuter des tests de qualification

Configurer les prérequis IDT

Pour exécuter des tests via l'interface utilisateur AWS IoT Device Tester (IDT) pour FreeRTOS, vous devez remplir les conditions requises sur la Configurer les prérequis de qualification LTS page de qualification IDT FreeRTOS (FRQ) 2.x.

Configurer AWS informations d'identification pour utiliser l'interface utilisateur IDT

Vous devez configurer vos informations d'identification utilisateur IAM pour l' AWS utilisateur que vous avez créé dansCréez un AWS compte. Vous pouvez spécifier vos informations d'identification de deux manières :

- Dans un fichier d'informations d'identification
- En tant que variables d'environnement

Configurer AWS informations d'identification avec un fichier d'informations d'identification

IDT utilise le même fichier d'informations d'identification que l' AWS CLI. Pour de plus amples informations, veuillez consulter Fichiers de configuration et d'informations d'identification.

L'emplacement du fichier d'informations d'identification varie en fonction du système d'exploitation que vous utilisez :

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

Ajoutez vos AWS informations d'identification au `credentials` fichier au format suivant :

```
[default]
aws_access_key_id = your_access_key_id
```

```
aws_secret_access_key = your_secret_access_key
```

> **Note**
>
> Si vous n'utilisez pas le `default` AWS profil, vous devez spécifier le nom du profil dans l'interface utilisateur IDT pour FreeRTOS. Pour plus d'informations sur les profils, consultez Configuration et paramètres des fichiers d'identification.

Configurer AWS informations d'identification avec variables d'environnement

Les variables d'environnement sont des variables gérées par le système d'exploitation et utilisées par les commandes du système. Ils ne sont pas enregistrés si vous fermez la session SSH. L'interface utilisateur IDT pour FreeRTOS utilise les variables d'environnement `AWS_SECRET_ACCESS_KEY` et pour `AWS_ACCESS_KEY_ID` stocker vos informations d'identification. AWS

Pour définir ces variables sous Linux, macOS ou Unix, utilisez export:

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

Pour définir ces variables sous Windows, utilisez set :

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

Ouvrez l'interface utilisateur IDT pour FreeRTOS

Cette rubrique explique comment ouvrir l'interface utilisateur IDT pour FreeRTOS afin d'utiliser la suite de qualification FreeRTOS.

Pour ouvrir l'interface utilisateur IDT pour FreeRTOS

1. Téléchargez une version IDT compatible pour FreeRTOS. Extrayez ensuite l'archive téléchargée dans un répertoire pour lequel vous disposez d'autorisations de lecture et d'écriture.
2. Accédez au répertoire d'installation d'IDT pour FreeRTOS :

   ```
   cd devicetester-extract-location/bin
   ```
3. Exécutez la commande suivante pour ouvrir l'interface utilisateur IDT pour FreeRTOS :

Linux

```
.devicetester_ui_linux_x86-64
```

Windows

```
./devicetester_ui_win_x64-64
```

macOS

```
./devicetester_ui_mac_x86-64
```

> **Note**
>
> Sous macOS, pour autoriser votre système à exécuter l'interface utilisateur, accédez à Préférences système -> Sécurité et confidentialité. Lorsque vous exécuterez les tests, vous devrez peut-être le faire trois fois de plus.

L'interface utilisateur IDT pour FreeRTOS s'ouvre dans votre navigateur par défaut. Les trois dernières versions majeures des navigateurs suivants sont compatibles avec l'interface utilisateur :

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

> **Note**
>
> Pour une meilleure expérience, nous recommandons Google Chrome ou Mozilla Firefox pour accéder à l'interface utilisateur IDT pour FreeRTOS. Microsoft Internet Explorer n'est pas pris en charge par l'interface utilisateur.

> **Important**
>
> Vous devez configurer vos AWS informations d'identification avant d'ouvrir l'interface utilisateur. Si vous n'avez pas configuré vos informations d'identification, fermez la fenêtre du navigateur IDT pour FreeRTOS UI, suivez les étapes Configurer AWS informations d'identification pour utiliser l'interface utilisateur IDT indiquées, puis rouvrez l'interface utilisateur IDT pour FreeRTOS.

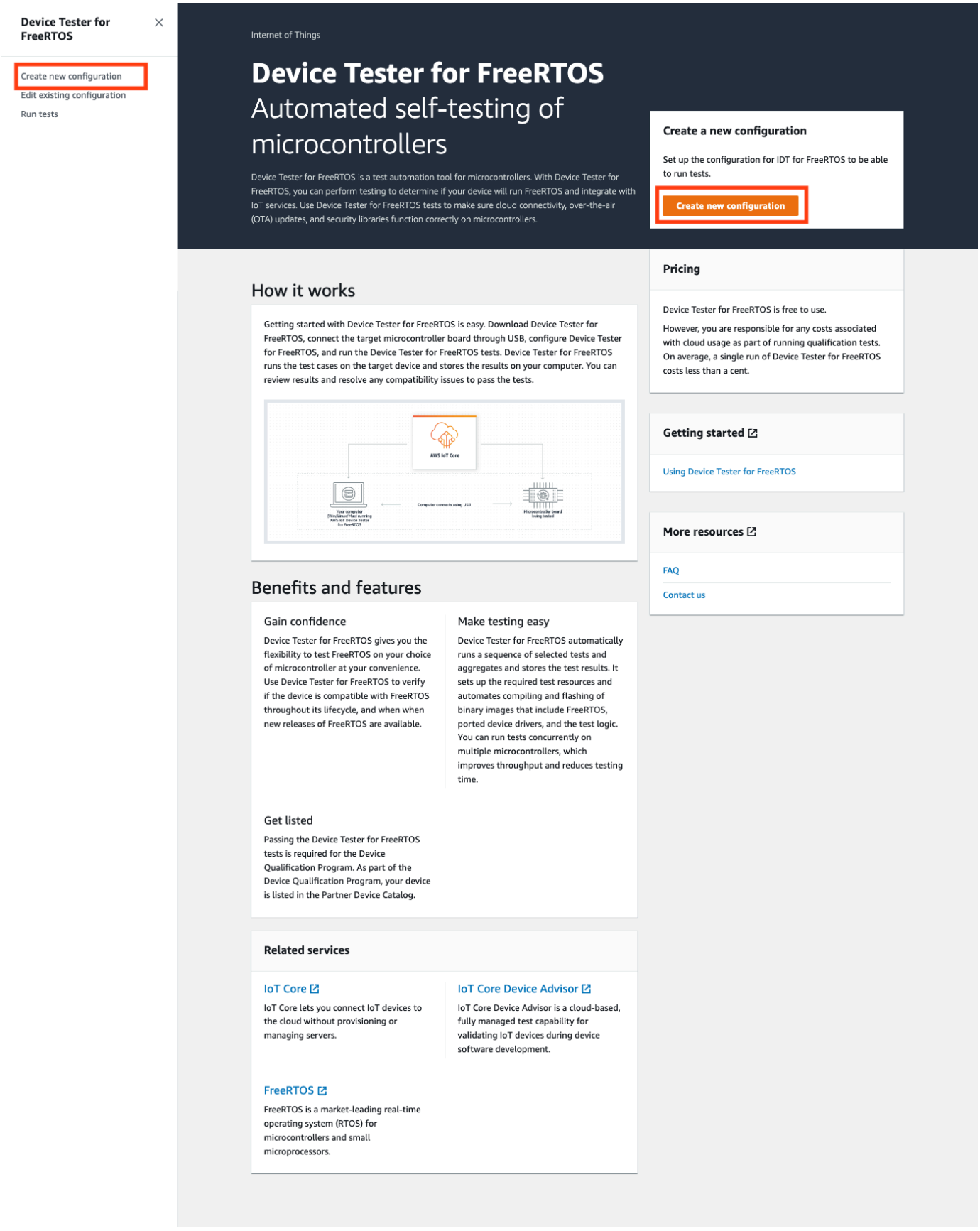
Création d'une nouvelle configuration

Si vous utilisez IDT pour la première fois, vous devez créer une nouvelle configuration pour configurer les fichiers de configuration JSON dont IDT pour FreeRTOS a besoin pour exécuter des tests. Vous pouvez ensuite exécuter des tests ou modifier la configuration créée.

Pour des exemples de `userdata.json` fichiers, et`config.json`,,,`device.json`,,,,,,,,,Premier test de votre carte microcontrôleur,,

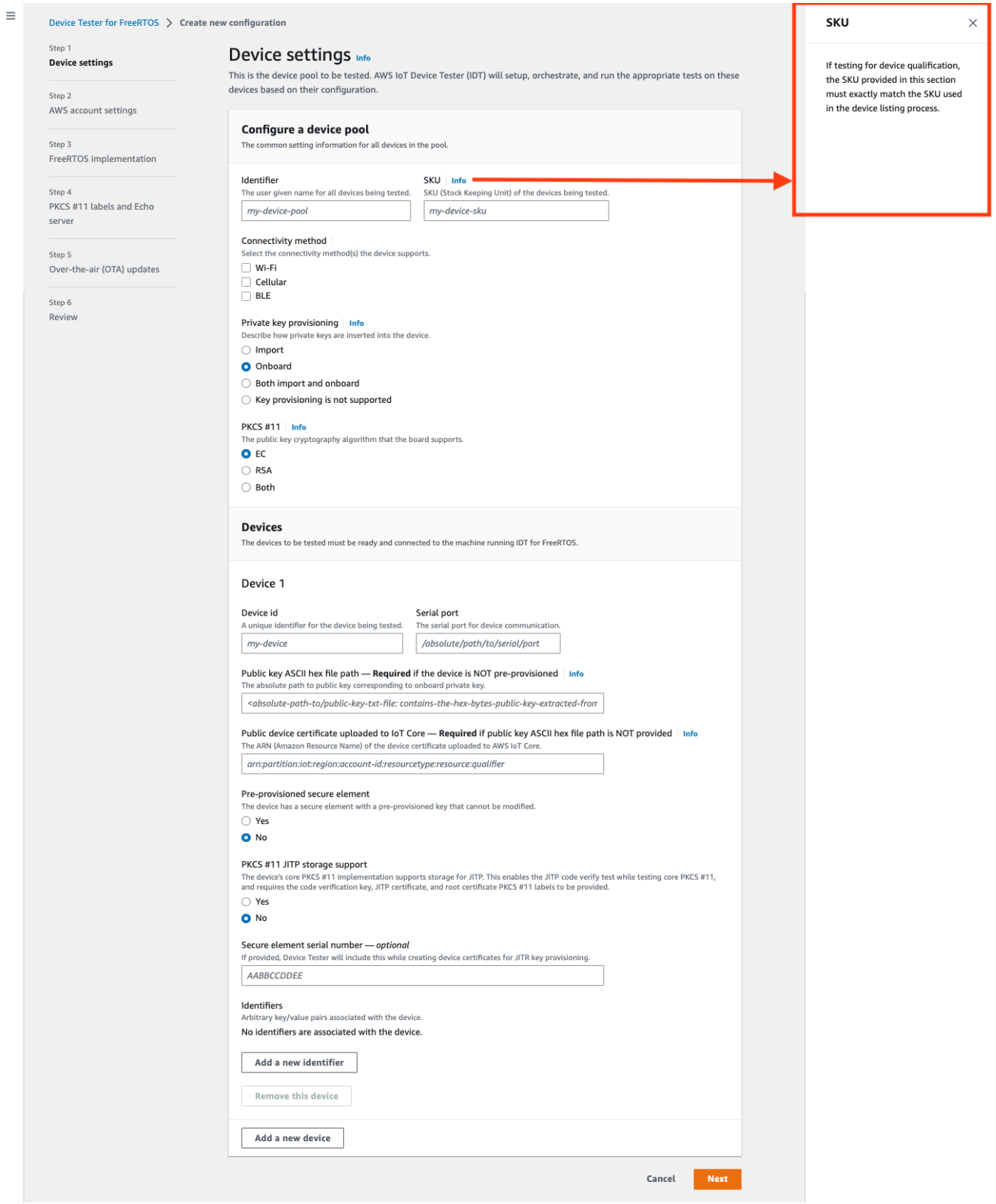
Pour créer une configuration

1. Dans l'interface utilisateur IDT pour FreeRTOS, ouvrez le menu de navigation et choisissez Create new configuration.

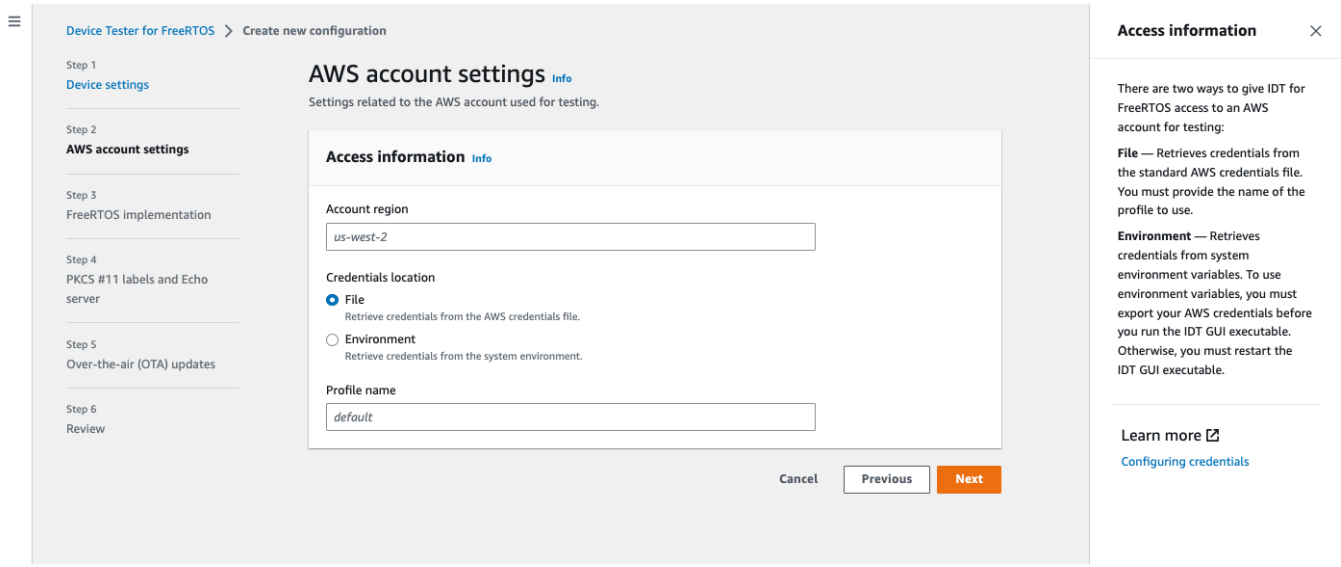

Device Tester for FreeRTOS
Create new configuration
Edit existing configuration
Run tests
Internet of Things
Device Tester for FreeRTOS
Automated self-testing of microcontrollers
Device Tester for FreeRTOS is a test automation tool for microcontrollers. With Device Tester for FreeRTOS, you can perform 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. Suivez l'assistant de configuration pour saisir les paramètres de configuration IDT utilisés pour exécuter les tests de qualification. L'assistant configure les paramètres suivants dans les fichiers de configuration JSON situés dans le *devicetester-extract-location*`/config` répertoire.

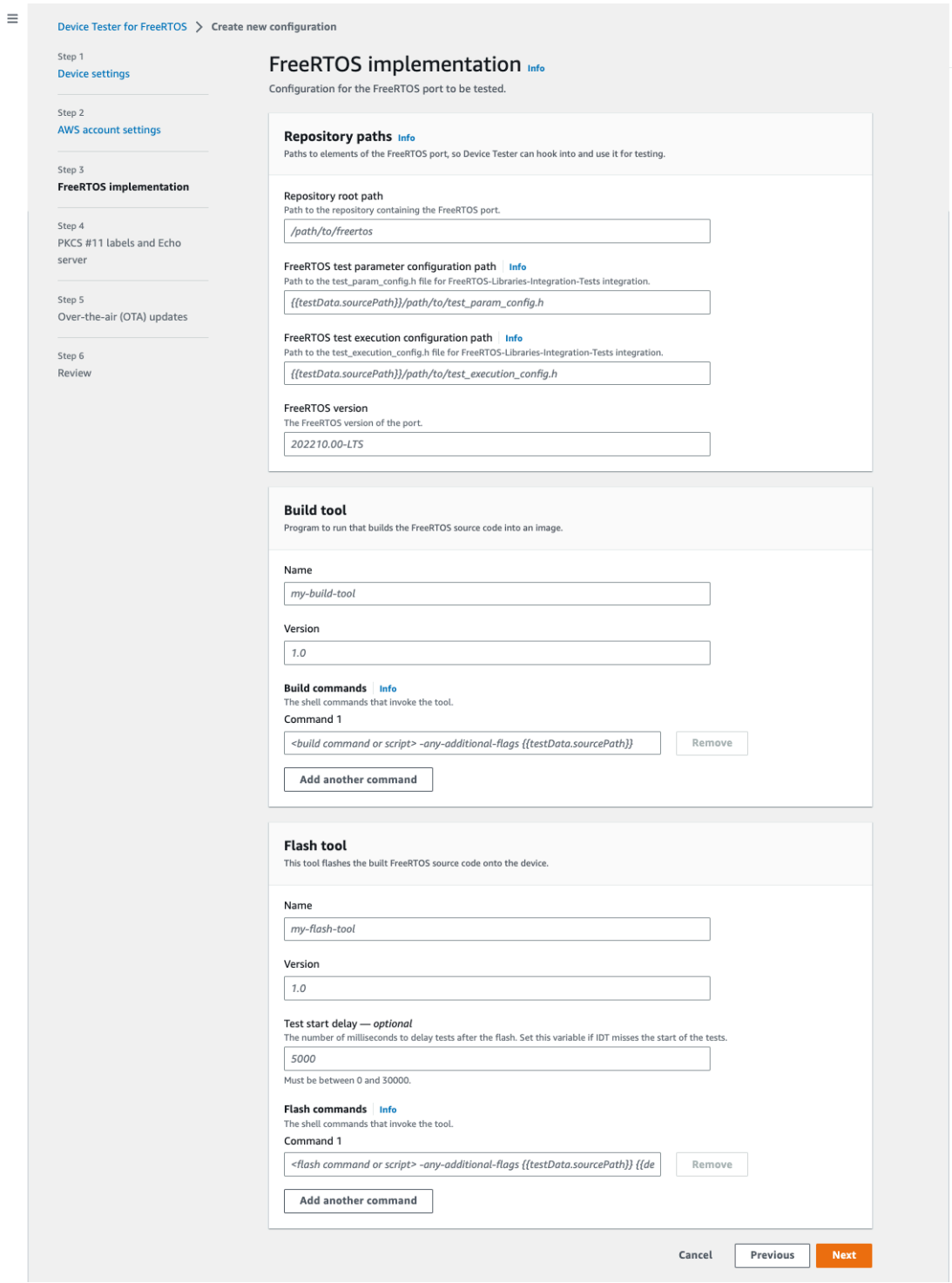
   - Paramètres des appareils : paramètres du pool de périphériques pour les appareils à tester. Ces paramètres sont configurés dans les `sku` champs `id` et, et les appareils sont bloqués pour le pool de périphériques dans le `config.json` fichier.


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.

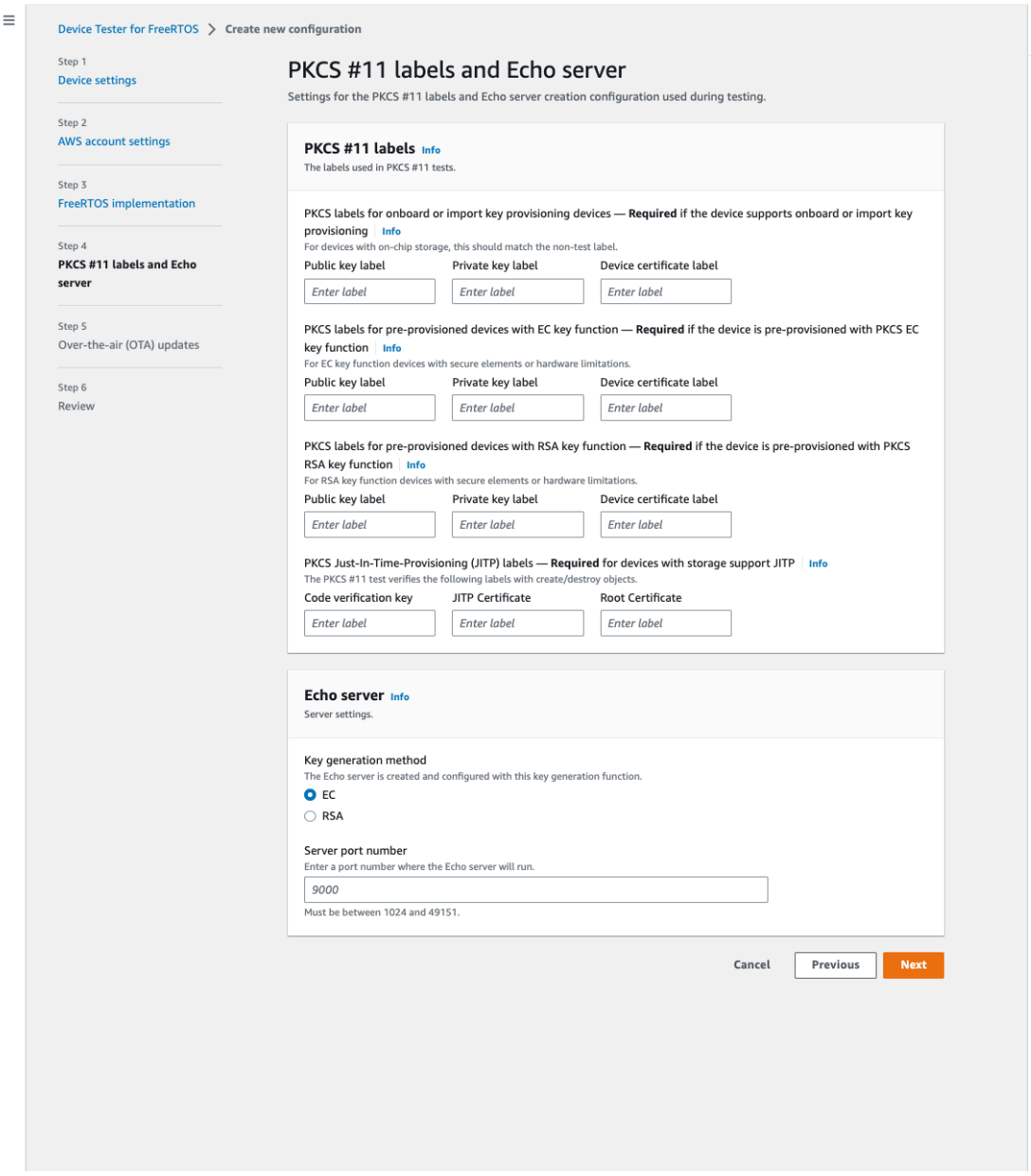
- AWS paramètres du compte — Les Compte AWS informations utilisées par IDT pour FreeRTOS pour AWS créer des ressources lors des tests. Ces paramètres sont configurés dans le `config.json` fichier.



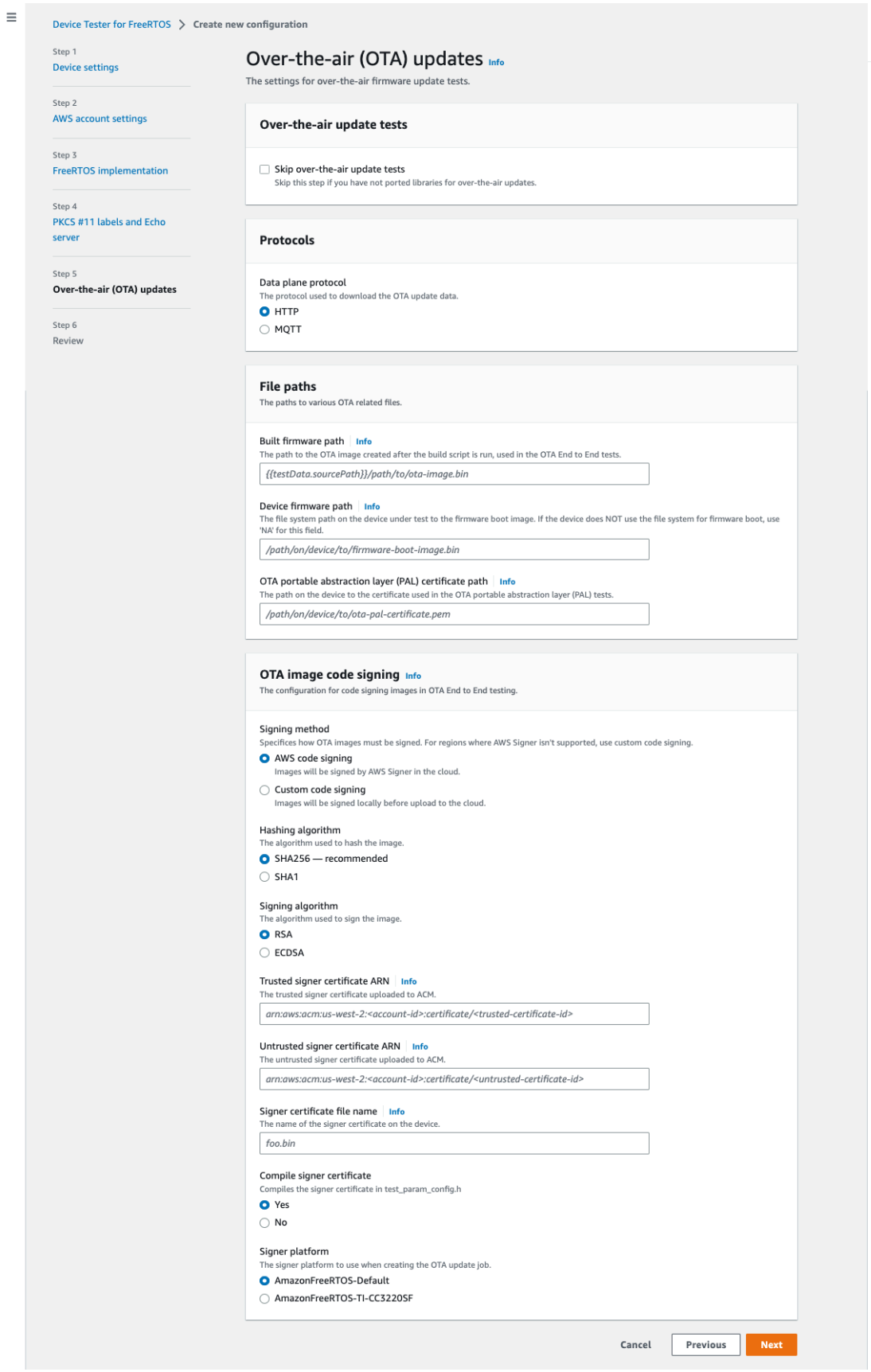
- Implémentation de FreeRTOS — Le chemin absolu vers le référentiel FreeRTOS et le code porté, ainsi que la version de FreeRTOS sur laquelle vous souhaitez exécuter IDT FRQ. Les chemins d'accès aux fichiers d'en-tête d'exécution et de configuration des paramètres depuis le `FreeRTOS-Libraries-Integration-Tests` GitHub référentiel. Les commandes de compilation et de flash pour votre matériel qui permettent à IDT de créer et de tester automatiquement votre carte mère. Ces paramètres sont configurés dans le `userdata.json` fichier.



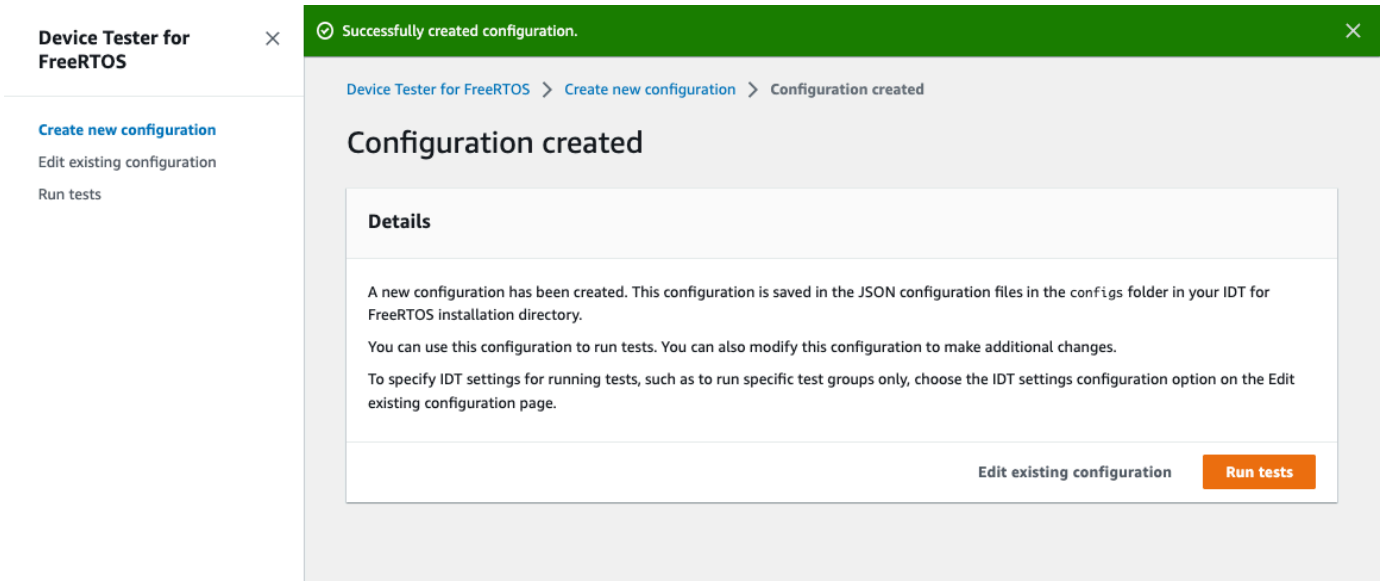
- Étiquettes PKCS #11 et serveur Echo : les étiquettes PKCS #11 qui correspondent aux clés fournies sur votre matériel en fonction de la fonctionnalité clé et de la méthode de fourniture des clés. Les paramètres de configuration du serveur Echo pour les tests de l'interface de transport. Ces paramètres sont configurés dans les `device.json` fichiers `userdata.json` et.



- Over-the-air Mises à jour (OTA) : paramètres qui contrôlent les tests de fonctionnalité OTA. Ces paramètres sont configurés dans le `features` bloc des `userdata.json` fichiers `device.json` et.


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. Sur la page Révision, vérifiez vos informations de configuration.



Après avoir passé en revue votre configuration, pour exécuter vos tests de qualification, choisissez Run tests.

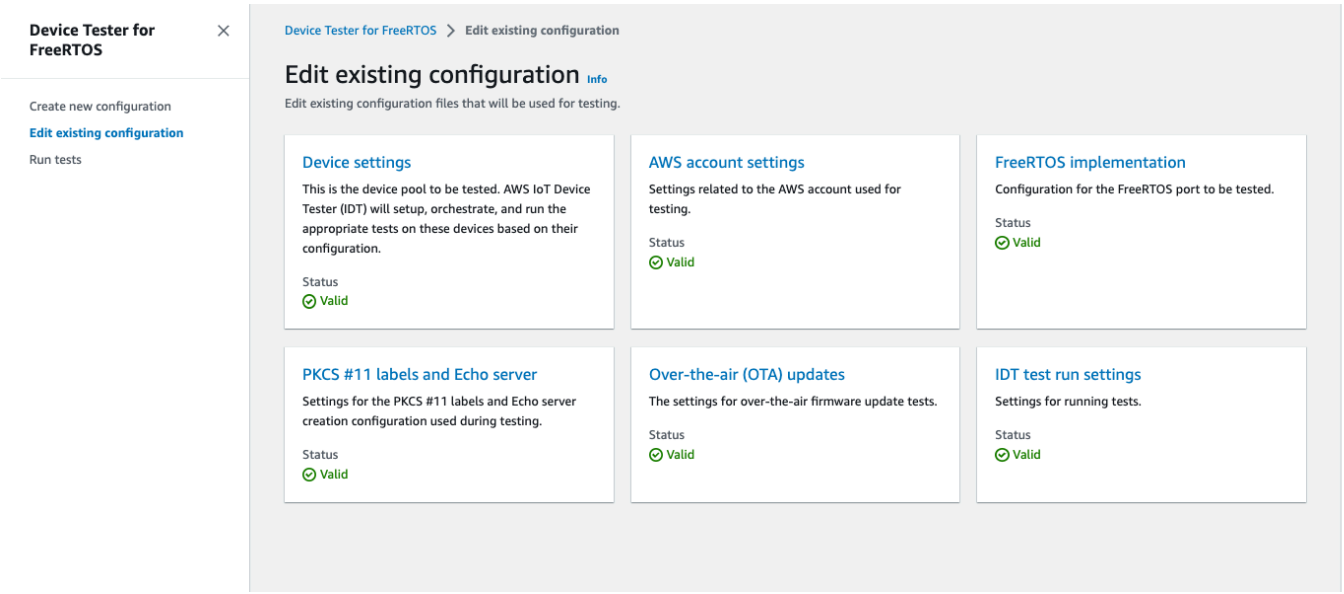
Modifier une configuration existante

Si vous avez déjà configuré des fichiers de configuration pour IDT pour FreeRTOS, vous pouvez utiliser l'interface utilisateur IDT pour FreeRTOS pour modifier votre configuration existante. Les fichiers de configuration existants doivent se trouver dans le *devicetester-extract-location*`/config` répertoire.

Pour modifier une configuration

1. Dans l'interface utilisateur IDT pour FreeRTOS, ouvrez le menu de navigation et choisissez Modifier la configuration existante.

   Le tableau de bord de configuration affiche des informations sur vos paramètres de configuration existants. Si une configuration est incorrecte ou indisponible, le statut de cette configuration est`Error validating configuration`.

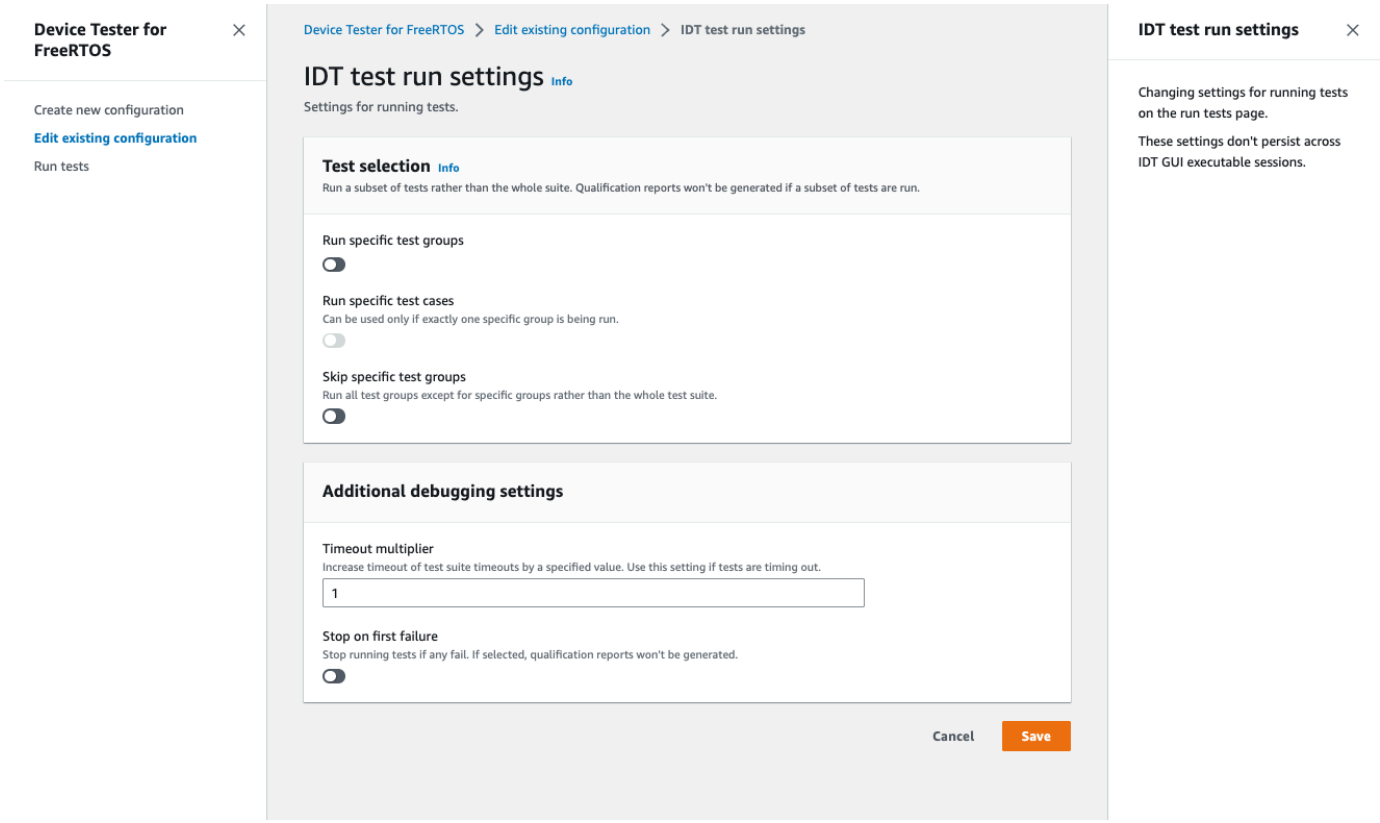


2. Pour modifier un paramètre de configuration existant, procédez comme suit :

   a. Choisissez le nom d'un paramètre de configuration pour ouvrir sa page de paramètres.

   b. Modifiez les paramètres, puis choisissez Enregistrer pour régénérer le fichier de configuration correspondant.

3. Pour modifier les paramètres d'exécution du test IDT pour FreeRTOS, choisissez les paramètres du cycle de test IDT dans la vue d'édition :



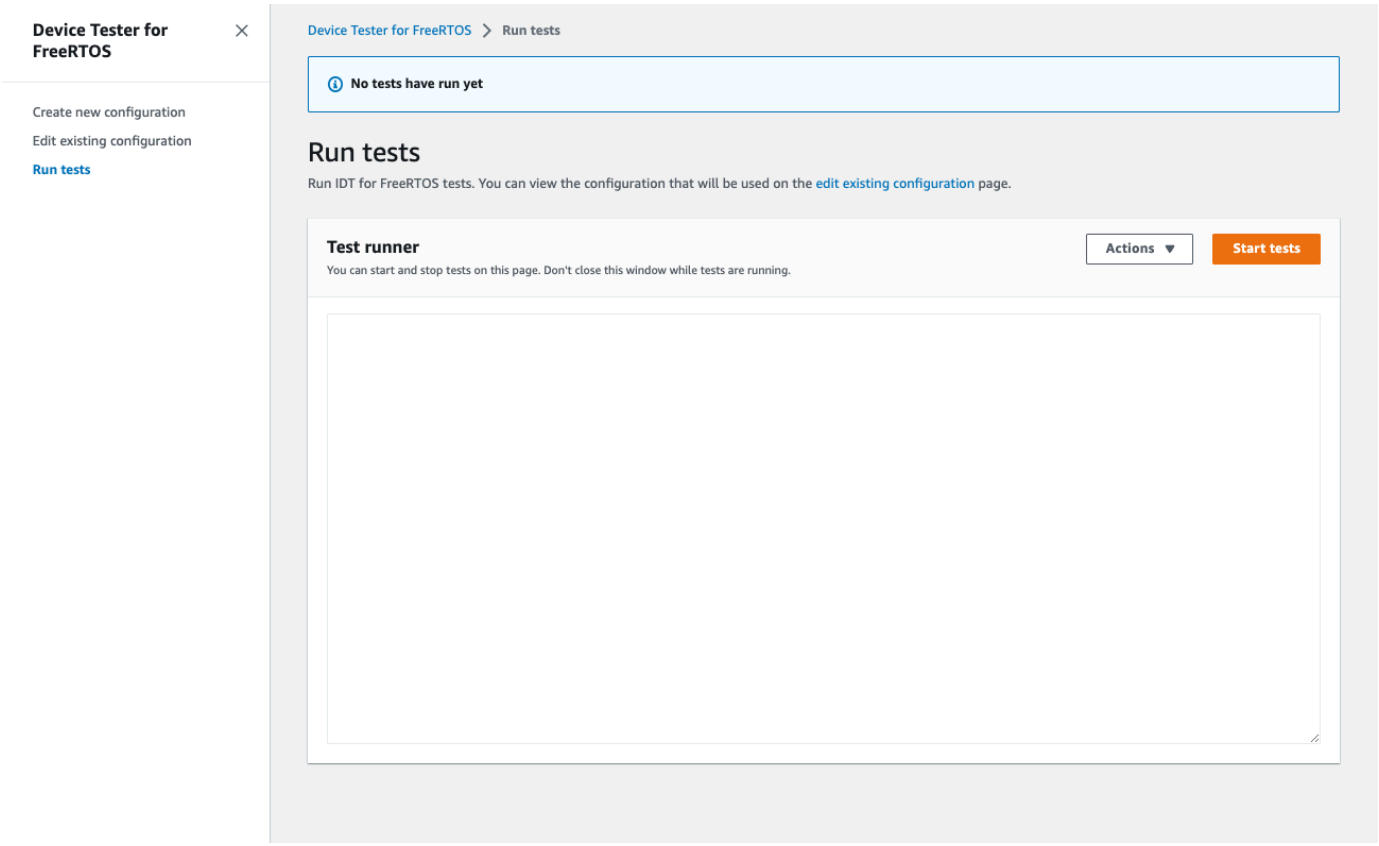
Une fois que vous avez terminé de modifier votre configuration, vérifiez que tous vos paramètres de configuration sont validés. Si le statut de chaque paramètre de configuration est`Valid`, vous pouvez exécuter vos tests de qualification avec cette configuration.

Exécuter des tests de qualification

Après avoir créé une configuration pour l'interface utilisateur IDT pour FreeRTOS, vous pouvez exécuter vos tests de qualification.

Pour exécuter des tests de qualification

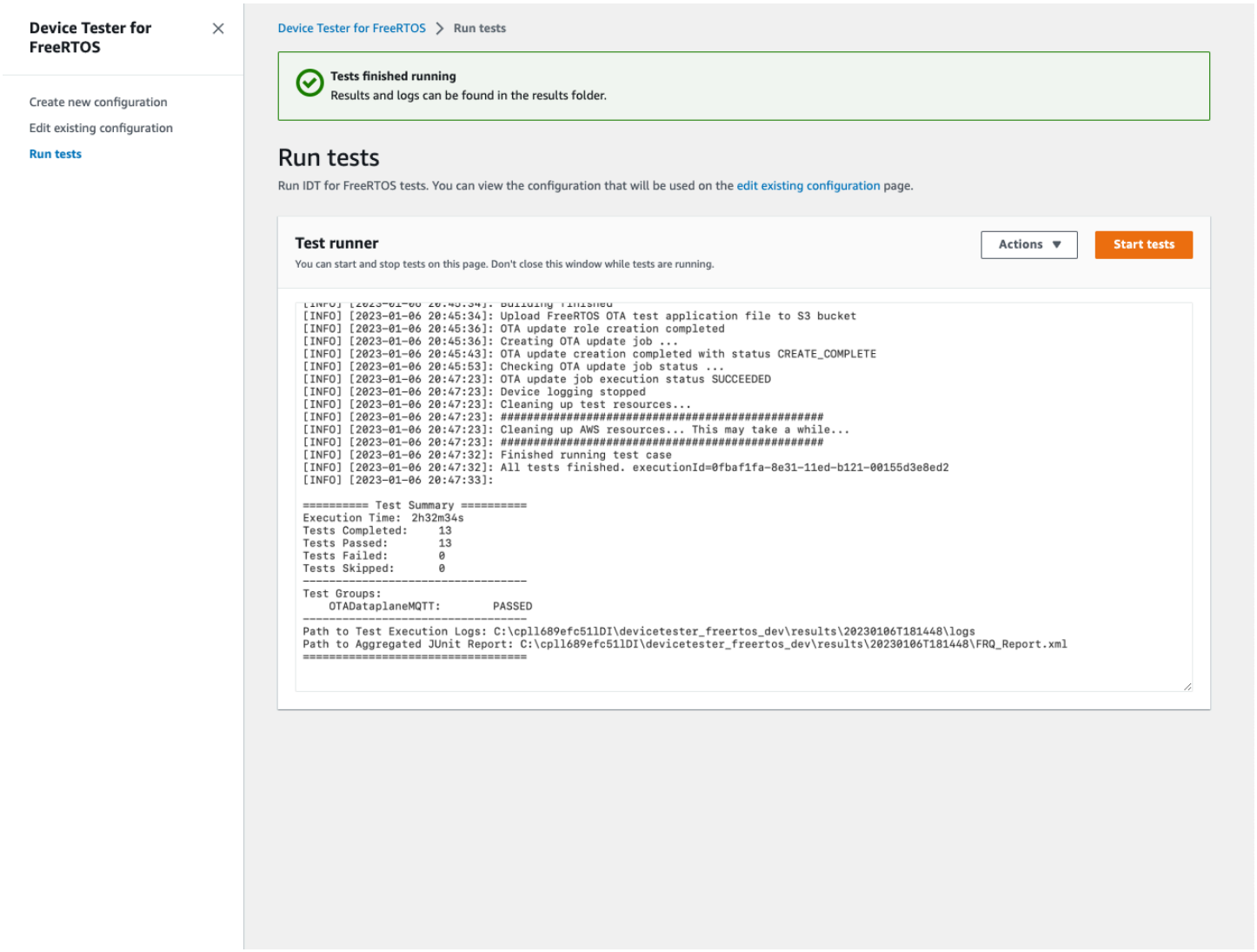
1. Dans le menu de navigation, choisissez Exécuter des tests.
2. Choisissez Démarrer les tests pour démarrer le test. Par défaut, tous les tests applicables sont exécutés pour la configuration de votre appareil. IDT pour FreeRTOS génère un rapport de qualification lorsque tous les tests sont terminés.



IDT pour FreeRTOS exécute les tests de qualification. Il affiche ensuite le résumé du test et les éventuelles erreurs dans la console Test Runner. Une fois le test terminé, vous pouvez consulter les résultats du test et les journaux depuis les emplacements suivants :

- Les résultats des tests se trouvent dans le *devicetester-extract-location*`/results/`*execution-id* répertoire.
- Les journaux de test se trouvent dans le *devicetester-extract-location*`/results/`*execution-id*`/logs` répertoire.

Pour plus d'informations sur les résultats des tests et les journaux, consultez Afficher les résultats IDT pour FreeRTOS etAfficher l'IDT pour FreeRtoSlogs.



Exécutez la suite de qualification FreeRTOS 2.0

Utilisez l'exécutable AWS IoT Device Tester for FreeRTOS pour interagir avec IDT pour FreeRTOS. Les exemples de ligne de commande ci-dessous vous montrent comment exécuter les tests de qualification d'un groupe d'appareils (ensemble d'appareils identiques).

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
```

Exécute une suite de tests sur un groupe d'appareils. Le fichier `userdata.json` doit figurer dans le répertoire *`devicetester_extract_location`*`/devicetester_freertos_`*`[win|mac|linux]`*`/configs/`.

> **Note**
>
> Si vous utilisez IDT pour FreeRTOS sous Windows, utilisez des barres obliques (/) pour spécifier le chemin du fichier. `userdata.json`

Utilisez la commande suivante pour exécuter un groupe de tests spécifique :

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

Les paramètres `suite-id` et `pool-id` sont facultatifs si vous exécutez une seule suite de tests au niveau d'un seul groupe d'appareils (à savoir, si un seul groupe d'appareils est défini dans votre fichier `device.json`).

Utilisez la commande suivante pour exécuter un cas de test spécifique dans un groupe de tests :

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

Vous pouvez utiliser la commande `list-test-cases` pour répertorier les cas de test dans un groupe de tests.

Options de ligne de commande IDT pour FreeRTOS

group-id

(Facultatif) Groupes de tests à exécuter, sous la forme d'une liste séparée par des virgules. Si cette option n'est pas spécifiée, IDT exécute tous les groupes de tests de la suite de tests.

pool-id

(Facultatif) Le groupe d'appareils à tester. Option requise si vous définissez plusieurs groupes d'appareils dans `device.json`. Si vous avez un seul groupe d'appareils, vous pouvez omettre cette option.

suite-id

(Facultatif) Version de la suite de tests à exécuter. Si cette option n'est pas spécifiée, IDT utilise la dernière version dans le répertoire tests de votre système.

test-id

(Facultatif) Tests à exécuter, sous la forme d'une liste séparée par des virgules. Si cette option est spécifiée, `group-id` doit spécifier un groupe unique.

Example

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

h

Utilisez l'option d'aide pour en savoir plus sur les options `run-suite`.

Example

Exemple

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

IDT pour les commandes FreeRTOS

La commande IDT pour FreeRTOS prend en charge les opérations suivantes :

IDT v4.5.2 and later

`help`

Répertorie les informations sur la commande spécifiée.

`list-groups`

Répertorie les groupes dans une suite donnée.

`list-suites`

Répertorie les suites disponibles.

`list-supported-products`

Répertorie les produits et les versions de suite de tests pris en charge.

`list-supported-versions`

Répertorie les versions de FreeRTOS et de suites de tests prises en charge par la version actuelle d'IDT.

`list-test-cases`

Répertorie les cas de test dans un groupe spécifié.

`run-suite`

Exécute une suite de tests sur un groupe d'appareils.

Utilisez l'option `--suite-id` pour spécifier une version de suite de tests ou omettez-la pour utiliser la dernière version sur votre système.

Utilisez l'option `--test-id` pour exécuter un cas de test individuel.

Example

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

> **Note**
>
> Depuis IDT v3.0.0, IDT recherche en ligne des suites de tests plus récentes. Pour de plus amples informations, veuillez consulter Versions de la suite de tests.

Afficher les résultats IDT pour FreeRTOS

Lorsqu'il s'exécute, IDT écrit les erreurs sur la console, les fichiers journaux et les rapports de tests. Après avoir terminé la gamme de test de qualification, IDT écrit un résumé de la série de tests dans la console et génère deux rapports d'essais. Ces rapports se trouvent à l'emplacement *`devicetester-extract-location`*`/results/`*`execution-id`*`/`. Les deux rapports capturent les résultats de l'exécution de la suite de tests de qualification.

`awsiotdevicetester_report.xml`Il s'agit du rapport de test de qualification que vous soumettez AWS pour inscrire votre appareil dans le catalogue des appareils AWS partenaires. Ce rapport contient les éléments suivants :

- La version IDT pour FreeRTOS.
- La version de FreeRTOS qui a été testée.
- Les fonctionnalités de FreeRTOS prises en charge par l'appareil sont basées sur les tests réussis.
- La référente et le nom de l'appareil spécifiés dans le fichier `device.json`.
- Les caractéristiques de l'appareil spécifiées dans le fichier `device.json`.
- Un récapitulatif des résultats des scénarios de test.
- Une répartition des résultats des cas d'utilisation par bibliothèques qui ont été testés en fonction des fonctionnalités de l'appareil.

`FRQ_Report.xml` est un rapport au format standard JUnit XML. Vous pouvez l'intégrer à CI/CD des plateformes telles que Jenkins, Bamboo, etc. Ce rapport contient les éléments suivants :

- Un récapitulatif des résultats des scénarios de test.
- Une répartition des résultats des cas d'utilisation par bibliothèques qui ont été testés en fonction des fonctionnalités de l'appareil.

Interpréter l'IDT pour les résultats FreeRTOS

La section Rapport dans `awsiotdevicetester_report.xml` ou `FRQ_Report.xml` répertorie les résultats des tests exécutés.

La première balise XML `<testsuites>` contient le résumé global de l'exécution des tests. Par exemple :

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

Attributs utilisés dans la **`<testsuites>`** balise

`name`

Nom de la suite de tests.

`time`

Durée, en secondes, nécessaire pour exécuter la suite de qualification.

`tests`

Nombre de scénarios de test exécutés.

`failures`

Nombre de scénarios de tests qui ont été exécutés, mais dont le résultat n'est pas probant.

`errors`

Le nombre de cas de test que IDT pour FreeRTOS n'a pas pu exécuter.

`disabled`

Cet attribut n'est pas utilisé et peut être ignoré.

S'il n'y a aucun échec ou erreur dans le scénario de test, votre appareil répond aux exigences techniques pour exécuter FreeRTOS et peut interagir avec les services. AWS IoT Si vous choisissez de répertorier votre appareil dans le catalogue des appareils AWS partenaires, vous pouvez utiliser ce rapport comme preuve de qualification.

En cas d'erreurs ou d'échec de scénarios de test, vous pouvez identifier les scénarios concernés à l'aide des balises XML `<testsuites>`. Les balises XML `<testsuite>` au sein de la balise `<testsuites>` indiquent le récapitulatif des résultats d'un groupe de tests.

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

Le format est similaire à la balise `<testsuites>`, mais avec un attribut supplémentaire appelé `skipped` qui n'est pas utilisé et qui ne peut pas être ignoré. Chaque balise XML `<testsuite>`inclut des balises `<testcase>` pour chaque scénario de test qui a été exécuté pour un groupe de tests. Par exemple :

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

Attributs utilisés dans la **`<awsproduct>`** balise

`name`

Nom du produit testé.

`version`

Version du produit testé.

`features`

Caractéristiques validées. Les caractéristiques portant la mention `required` sont requises pour pouvoir envoyer votre carte en vue de sa certification. L'extrait suivant montre comment cela apparaît dans le `awsiotdevicetester_report.xml` fichier.

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

Les fonctionnalités marquées comme `optional` ne sont pas requises pour l'éligibilité. Les extraits suivants illustrent des fonctions facultatives.

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

S'il n'y a aucun échec ou erreur de test pour les fonctionnalités requises, votre appareil répond aux exigences techniques pour exécuter FreeRTOS et peut interagir avec les services. AWS IoT Si vous souhaitez répertorier votre appareil dans le catalogue des appareils AWS partenaires, vous pouvez utiliser ce rapport comme preuve de qualification.

En cas d'erreurs ou d'échecs de tests, vous pouvez identifier les tests concernés à l'aide des balises XML `<testsuites>`. Les balises XML `<testsuite>` au sein de la balise `<testsuites>` montrent le récapitulatif des résultats d'un groupe de tests. Par exemple :

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

Le format est similaire à celui de la `<testsuites>` balise, mais possède un `skipped` attribut qui n'est pas utilisé et qui peut être ignoré. Chaque balise XML `<testsuite>` inclut des balises `<testcase>` pour chaque test exécuté pour un groupe de tests. Par exemple :

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

Attributs utilisés dans la **`<testcase>`** balise

`name`

Nom du scénario de test.

`attempts`

Le nombre de fois où IDT pour FreeRTOS a exécuté le scénario de test.

Lorsqu'un test échoue ou qu'une erreur se produit, les balises `<failure>` ou `<error>` sont ajoutées à la balise `<testcase>` avec des informations relatives au dépannage. Par exemple :

```
<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>
```

Pour de plus amples informations, veuillez consulter Résolution des erreurs.

Afficher l'IDT pour FreeRtoSlogs

Vous pouvez trouver les journaux générés par IDT pour FreeRTOS à partir de l'exécution des tests dans. *devicetester-extract-location*`/results/`*execution-id*`/logs` Deux ensembles de journaux sont générés :

- `test_manager.log`

  Contient les journaux générés par IDT pour FreeRTOS (par exemple, la configuration associée aux journaux et la génération de rapports).
- *test_group_id*`/`*test_case_id*`/`*test_case_id*`.log`

  Fichier journal d'un cas de test, y compris la sortie de l'appareil en cours de test. Le fichier journal est nommé en fonction du groupe de test et du cas de test qui a été exécuté.

IDT avec suite de qualification FreeRTOS 1.0 (FRQ 1.0)

> **Important**
>
> Depuis octobre 2022, AWS IoT Device Tester AWS IoT FreeRTOS Qualification (FRQ) 1.0 ne génère pas de rapports de qualification signés. Vous ne pouvez pas qualifier les nouveaux

appareils AWS IoT FreeRTOS à répertorier dans le catalogue des appareils partenaires dans AWS le cadre du programme de qualification des appareils utilisant AWS les versions IDT FRQ 1.0. Bien que vous ne puissiez pas qualifier les appareils FreeRTOS avec IDT FRQ 1.0, vous pouvez continuer à tester vos appareils FreeRTOS avec FRQ 1.0. Nous vous recommandons d'utiliser IDT FRQ 2.0 pour qualifier et répertorier les appareils FreeRTOS dans le catalogue d'appareils partenaires.AWS

Vous pouvez utiliser IDT pour la qualification FreeRTOS afin de vérifier que le système d'exploitation FreeRTOS fonctionne localement sur votre appareil et qu'il peut communiquer avec. AWS IoT Plus précisément, il vérifie que les interfaces de la couche de portage pour les bibliothèques FreeRTOS sont correctement implémentées. Il effectue également des tests de bout en bout avec. AWS IoT Core Il vérifie par exemple que votre carte peut envoyer, recevoir et traiter correctement des messages MQTT. Les tests exécutés par IDT pour FreeRTOS sont définis dans le référentiel FreeRTOS. GitHub

Les tests comme les applications intégrées qui sont flashés sur votre carte. Les images binaires de l'application incluent FreeRTOS, les interfaces FreeRTOS portées par le fournisseur de semi-conducteurs, et les pilotes de périphériques de la carte. Le but des tests est de vérifier que les interfaces FreeRTOS portées fonctionnent correctement au-dessus des pilotes de périphériques.

IDT for FreeRTOS génère des rapports de test que vous pouvez soumettre AWS IoT pour ajouter votre matériel au catalogue d'appareils partenaires. AWS Pour de plus amples informations, veuillez consulter AWS Device Qualification Program.

IDT pour FreeRTOS s'exécute sur un ordinateur hôte (Windows, macOS ou Linux) connecté à la carte à tester. IDT exécute des scénarios de test et regroupe les résultats. Il fournit également une interface de ligne de commande pour gérer l'exécution des tests.

Outre les appareils de test, IDT for FreeRTOS crée des ressources (par exemple, des AWS IoT objets, des groupes FreeRTOS, des fonctions Lambda, etc.) pour faciliter le processus de qualification. Pour créer ces ressources, IDT pour FreeRTOS utilise les informations d'identification configurées dans AWS le pour effectuer des appels d'API en `config.json` votre nom. Ces ressources sont allouées à différents moments d'un test.

Lorsque vous exécutez IDT pour FreeRTOS sur votre ordinateur hôte, il exécute les étapes suivantes :

1. Chargement et validation des informations d'identification et de la configuration de votre appareil.

2. Tests sélectionnés avec les ressources locales et les ressources cloud requises.
3. Élimination des ressources locales et des ressources cloud.
4. Génération de rapports de tests indiquant si votre carte a réussi les tests obligatoires pour la qualification.

Rubriques

- Configuration des prérequis de qualification 1.0
- Premier test de votre carte microcontrôleur
- Utiliser l'interface utilisateur IDT pour exécuter la suite de qualification FreeRTOS
- Exécuter des tests Bluetooth Low Energy
- Exécutez la suite de qualification FreeRTOS
- Afficher les résultats de l'IDT pour FreeRTOS
- Interpréter l'IDT pour les résultats FreeRTOS
- Afficher les logs IDT pour FreeRTOS

Configuration des prérequis de qualification 1.0

Cette section décrit les prérequis pour tester les microcontrôleurs avec. AWS IoT Device Tester

Télécharger FreeRTOS

Vous pouvez télécharger une version de FreeRTOS à l'aide GitHubde la commande suivante :

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

où se <FREERTOS_RELEASE_VERSION>trouve une version de FreeRTOS (par exemple, 202007.00) correspondant à une version IDT répertoriée dans. Versions prises en charge de AWS IoT Device Tester Cela garantit que vous disposez du code source complet, y compris les sous-modules, et que vous utilisez la bonne version d'IDT pour votre version de FreeRTOS, et vice versa.

Pour Windows, la limitation de la longueur du chemin est de 260 caractères. La structure des chemins de FreeRTOS comporte plusieurs niveaux. Par conséquent, si vous utilisez Windows, maintenez les chemins de vos fichiers en dessous de la limite de 260 caractères. Par exemple,

clonez FreeRTOS sur plutôt que sur`C:\FreeRTOS`. `C:\Users\username\programs\projects\myproj\FreeRTOS\`

Qualification FreeRTOS avec les bibliothèques LTS

- Pour que votre microcontrôleur soit désigné comme compatible avec les versions basées sur le support à long terme (LTS) de FreeRTOS dans AWS le catalogue des appareils partenaires, vous devez fournir un fichier manifeste. Pour plus d'informations, consultez la liste de contrôle de qualification FreeRTOS dans le guide de qualification FreeRTOS.
- Afin de valider que votre microcontrôleur prend en charge les versions LTS de FreeRTOS et de le soumettre au Partner Device Catalog, vous devez AWS IoT Device Tester utiliser (IDT) avec AWS la version v1.4.x de la suite de tests FreeRTOS Qualification (FRQ).
- Support pour les versions LTS de FreeRTOS limité à la version 202012.xx de FreeRTOS.

Télécharger IDT pour FreeRTOS

Chaque version de FreeRTOS possède une version correspondante d'IDT pour FreeRTOS pour effectuer des tests de qualification. Téléchargez la version appropriée d'IDT pour FreeRTOS à partir de. Versions prises en charge de AWS IoT Device Tester

Extrayez IDT pour FreeRTOS vers un emplacement du système de fichiers où vous disposez d'autorisations de lecture et d'écriture. Microsoft Windows ayant une limite de caractères pour la longueur du chemin, extrayez IDT pour FreeRTOS dans un répertoire racine tel que ou. `C:\` `D:\`

> **Note**
>
> Nous ne recommandons pas l'exécution d'IDT par plusieurs utilisateurs à partir d'un emplacement partagé, tel qu'un répertoire NFS ou un dossier partagé réseau Windows. Cela peut entraîner des plantages ou une corruption des données. Nous vous recommandons d'extraire le package IDT sur un disque local.

Créez et configurez un AWS compte

Inscrivez-vous pour un Compte AWS

Pour commencer AWS, vous avez besoin d'un Compte AWS. Pour plus d'informations sur la création d'un Compte AWS, voir Getting started with an Compte AWS dans le Guide de Gestion de compte AWS référence.

AWS IoT Device Tester politique gérée

La politique `AWSIoTDeviceTesterForFreeRTOSFullAccess` gérée contient les AWS IoT Device Tester autorisations suivantes pour la vérification des versions, les fonctionnalités de mise à jour automatique et la collecte de métriques.

- `iot-device-tester:SupportedVersion`

  Accorde AWS IoT Device Tester l'autorisation de récupérer la liste des produits pris en charge, des suites de tests et des versions IDT.

- `iot-device-tester:LatestIdt`

  Accorde AWS IoT Device Tester l'autorisation de récupérer la dernière version d'IDT disponible au téléchargement.

- `iot-device-tester:CheckVersion`

  Accorde AWS IoT Device Tester l'autorisation de vérifier la compatibilité des versions pour IDT, les suites de tests et les produits.

- `iot-device-tester:DownloadTestSuite`

  AWS IoT Device Tester Autorise le téléchargement des mises à jour de la suite de tests.

- `iot-device-tester:SendMetrics`

  Accorde AWS l'autorisation de collecter des statistiques relatives à l'utilisation AWS IoT Device Tester interne.

(Facultatif) Installez AWS Command Line Interface

Vous préférerez peut-être utiliser le AWS CLI pour effectuer certaines opérations. Si vous ne l'avez pas AWS CLI installé, suivez les instructions de la section Installer le AWS CLI.

Configurez le AWS CLI pour la AWS région que vous souhaitez utiliser en l'exécutant à aws configure partir d'une ligne de commande. Pour plus d'informations sur les AWS régions qui prennent en charge IDT pour FreeRTOS,AWS consultez Régions et points de terminaison. Pour plus d'informations sur la aws configure section Configuration rapide avec aws configure.

Premier test de votre carte microcontrôleur

Vous pouvez utiliser IDT pour FreeRTOS pour tester lors du portage des interfaces FreeRTOS. Après avoir porté les interfaces FreeRTOS pour les pilotes de périphériques de votre carte, vous pouvez exécuter les tests AWS IoT Device Tester de qualification sur votre carte microcontrôleur.

Ajouter des couches de transfert pour les bibliothèques

Pour transférer FreeRTOS sur votre appareil, suivez les instructions du Guide de portage de FreeRTOS.

Configurez votre AWS informations d'identification

Vous devez configurer vos AWS informations d'identification AWS IoT Device Tester pour pouvoir communiquer avec le AWS Cloud. Pour plus d'informations, voir Configurer les AWS informations d'identification et la région pour le développement. Des AWS informations d'identification valides doivent être spécifiées dans le fichier *devicetester_extract_location*/devicetester_afreertos_*[win|mac|linux]*/configs/config.json de configuration.

Rubriques

- Créer un pool d'appareils dans IDT pour FreeRTOS
- Configurer les paramètres de création, de flash et de test

Créer un pool d'appareils dans IDT pour FreeRTOS

Les appareils à tester sont organisés dans des groupes. Chaque groupe d'appareils se compose d'un ou de plusieurs appareils identiques. Vous pouvez configurer IDT pour FreeRTOS afin de tester un seul appareil dans un pool ou plusieurs appareils dans un pool. Pour accélérer le processus de qualification, IDT for FreeRTOS peut tester des appareils avec les mêmes spécifications en parallèle. Il utilise une méthode séquentielle pour exécuter un groupe de tests différents pour chaque appareil d'un groupe.

Vous pouvez ajouter un ou plusieurs appareils à un groupe d'appareils en modifiant la section `devices` du modèle `device.json` dans le dossier `configs`.

> **Note**
>
> Tous les appareils du même groupe doivent avoir les mêmes spécifications techniques et la même référence.

Pour permettre des compilations parallèles du code source pour différents groupes de test, IDT pour FreeRTOS copie le code source dans un dossier de résultats à l'intérieur du dossier extrait d'IDT pour FreeRTOS. Le chemin du code source dans votre commande build ou flash doit être référencé à l'aide de la `sdkPath` variable `testdata.sourcePath` or. IDT pour FreeRTOS remplace cette variable par un chemin temporaire du code source copié. Pour plus d'informations, voir,IDT pour les variables FreeRTOS.

Voici un exemple de fichier `device.json` utilisé pour créer un groupe comprenant plusieurs appareils :

```
[
    {
        "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"
              }
            ]
          }
```

```
        ]
    }
]
```

Les attributs suivants sont utilisés dans le fichier `device.json` :

`id`

Un ID alphanumérique défini par l'utilisateur qui identifie de façon unique un groupe d'appareils. Les appareils appartenant à un groupe doivent être du même type. Lorsque vous exécutez une suite de tests, les appareils du groupe sont utilisés pour paralléliser la charge de travail.

`sku`

Une valeur alphanumérique qui identifie de façon unique la carte que vous testez. La référence est utilisée pour effectuer le suivi des cartes qualifiées.

> **Note**
>
> Si vous souhaitez mettre votre carte en vente dans le catalogue des appareils AWS partenaires, le SKU que vous spécifiez ici doit correspondre au SKU que vous avez utilisé lors du processus de mise en vente.

`features`

Tableau contenant les fonctionnalités prises en charge par l'appareil. AWS IoT Device Tester utilise ces informations pour sélectionner les tests de qualification à exécuter.

Les valeurs prises en charge sont :

`TCP/IP`

Indique si votre carte supporte une TCP/IP pile et si elle est supportée sur puce (MCU) ou déchargée vers un autre module. TCP/IP est requis pour la qualification.

`WIFI`

Indique si votre carte possède Wi-Fi des capacités. Doit être défini sur `No` s'`Cellular`il est défini sur`Yes`.

`Cellular`

Indique si votre carte est dotée de fonctionnalités cellulaires. Doit être défini sur `No` s'`WIFI`il est défini sur`Yes`. Lorsque cette fonctionnalité est définie sur`Yes`, le FullSecureSockets test sera exécuté à l'aide d'instances EC2 AWS t2.micro, ce qui peut entraîner des coûts supplémentaires pour votre compte. Pour plus d'informations, consultez Tarification Amazon EC2.

`TLS`

Indique si votre carte prend en charge TLS. Le protocole TLS est requis pour la qualification.

`PKCS11`

Indique l'algorithme de chiffrement de clé publique que la carte prend en charge. PKCS11 est requis pour la qualification. Les valeurs prises en charge sont `ECC`, `RSA`, `Both` et `No`. `Both` indique que la carte prend en charge les algorithmes `ECC` et `RSA` .

`KeyProvisioning`

Indique la méthode d'écriture d'un certificat X.509 client de confiance sur votre tableau. Les valeurs valides sont `Import`, `Onboard` et `No`. La mise en service de clés est requise pour la qualification.

- Utilisez `Import` si votre carte autorise l'importation de clés privées. IDT créera une clé privée et l'intégrera au code source de FreeRTOS.
- Utilisez `Onboard` si votre carte prend en charge la génération d'une clé privée embarquée (par exemple, si votre appareil dispose d'un élément sécurisé ou si vous préférez générer vos propres certificat et paire de clés d'appareil). Veillez à ajouter un élément `secureElementConfig` dans chacune des sections de l'appareil et indiquez le chemin absolu vers le fichier de clé publique dans le champ `publicKeyAsciiHexFilePath`.
- Utilisez `No` si votre carte ne prend pas en charge la mise en service de clés.

`OTA`

Indique si la carte prend en charge la fonctionnalité de mise à jour à distance (OTA). L'attribut `OtaDataPlaneProtocol` indique le protocole de plan de données OTA pris en charge par le périphérique. L'attribut est ignoré si la fonctionnalité OTA n'est pas prise en charge par le périphérique. Lorsque `"Both"` est sélectionné, le temps d'exécution du test OTA est prolongé en raison de l'exécution de tests MQTT, HTTP et mixtes.

> **Note**
>
> À partir de IDT v4.1.0, `OtaDataPlaneProtocol` accepte uniquement `HTTP` et en `MQTT` tant que valeurs prises en charge.

`BLE`

Indique si la carte prend en charge Bluetooth Low Energy (BLE).

`devices.id`

Un identificateur unique défini par l'utilisateur pour l'appareil testé.

`devices.connectivity.protocol`

Le protocole de communication utilisé pour communiquer avec cet appareil. Valeur prise en charge : `uart`.

`devices.connectivity.serialPort`

Le port série de l'ordinateur hôte utilisé pour se connecter aux appareils testés.

`devices.secureElementConfig.PublicKeyAsciiHexFilePath`

Chemin absolu vers le fichier contenant la clé publique d'octets hexadécimaux extraite de la clé privée embarquée.

Exemple de format :

```
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
```

Si votre clé publique est au format .der, vous pouvez l'encoder directement en hexadécimal pour générer le fichier hexadécimal.

Exemple de commande pour la clé publique .der pour générer un fichier hexadécimal :

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

Si votre clé publique est au format .pem, vous pouvez extraire la partie codée en base64, la décoder au format binaire, puis l'encoder en hexadécimal pour générer le fichier hexadécimal.

Par exemple, utilisez ces commandes pour générer un fichier hexadécimal pour une clé publique .pem :

1. Retirez la partie codée en base64 de la clé (supprimez l'en-tête et le pied de page) et stockez-la dans un fichier, par exemple nommez-la`base64key`, exécutez cette commande pour la convertir au format .der :

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

2. Exécutez la `xxd` commande pour le convertir au format hexadécimal.

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

`devices.secureElementConfig.SecureElementSerialNumber`

(Facultatif) Numéro de série de l'élément sécurisé. Indiquez ce champ lorsque le numéro de série est imprimé avec la clé publique de l'appareil lorsque vous exécutez le projet FreeRTOS demo/test .

`devices.secureElementConfig.preProvisioned`

(Facultatif) Réglez sur « Oui » si l'appareil possède un élément sécurisé préconfiguré avec des informations d'identification verrouillées, qui ne peut pas importer, créer ou détruire des objets. Cette configuration ne prend effet que lorsqu'`features`elle est `KeyProvisioning` définie sur « Onboard » et sur « ECC ». `PKCS11`

`identifiers`

(Facultatif) Un tableau de paires nom-valeur arbitraires. Vous pouvez utiliser ces valeurs dans les commandes de création et flash décrites dans la section suivante.

Configurer les paramètres de création, de flash et de test

Pour qu'IDT for FreeRTOS puisse être créé et testé automatiquement sur votre carte, vous devez configurer IDT pour exécuter les commandes de compilation et de flash pour votre matériel. Les paramètres de création et flash sont configurés dans le modèle de fichier `userdata.json` situé dans le dossier `config`.

Configurer les paramètres pour tester un seul appareil

Les paramètres de création, de flash et de test sont définis dans le fichier `configs/userdata.json`. Nous prenons en charge la configuration du serveur Echo en chargeant les certificats et les clés du client et du serveur dans le`customPath`. Pour plus d'informations, consultez la section Configuration d'un serveur Echo dans le Guide de portage de FreeRTOS. L'exemple JSON suivant montre comment configurer IDT pour FreeRTOS afin de tester plusieurs appareils :

```
{
    "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"
            }
        ]
    }
}
```

Voici les attributs utilisés dans le fichier `userdata.json` :

`sourcePath`

Le chemin d'accès à la racine du code source FreeRTOS porté. Pour les tests parallèles avec un SDK, ils `sourcePath` peuvent être définis à l'aide du `{{userData.sdkConfiguration.path}}` support. Par exemple :

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

`vendorPath`

Le chemin d'accès au code FreeRTOS spécifique au fournisseur. Pour les tests série, `vendorPath` peut être défini comme un chemin absolu. Par exemple :

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

Pour les tests en parallèle, `vendorPath` peut être défini à l'aide de l'espace réservé `{{testData.sourcePath}}`. Par exemple :

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

La `vendorPath` variable n'est nécessaire que lorsqu'elle est exécutée sans SDK, sinon elle peut être supprimée.

> **Note**
>
> Lorsque vous exécutez des tests en parallèle sans SDK, l'`{{testData.sourcePath}}`espace réservé doit être utilisé dans les champs `vendorPathbuildTool`,`flashTool`. Lors de l'exécution d'un test avec un seul appareil, les chemins absolus doivent être utilisés dans les champs `vendorPath`, `buildTool` et `flashTool`. Lors de l'exécution avec un SDK, l'`{{sdkPath}}`espace réservé doit être utilisé dans les commandes `sourcePathbuildTool`, et`flashTool`.

`sdkConfiguration`

Si vous qualifiez FreeRTOS en apportant des modifications aux fichiers et à la structure des dossiers au-delà de ce qui est requis pour le portage, vous devrez configurer les informations de votre SDK dans ce bloc. Si vous n'êtes pas éligible à un FreeRTOS porté dans un SDK, vous devez omettre complètement ce bloc.

`sdkConfiguration.name`

Le nom du SDK que vous utilisez avec FreeRTOS. Si vous n'utilisez pas de SDK, le `sdkConfiguration` bloc entier doit être omis.

`sdkConfiguration.version`

Version du SDK que vous utilisez avec FreeRTOS. Si vous n'utilisez pas de SDK, le `sdkConfiguration` bloc entier doit être omis.

`sdkConfiguration.path`

Le chemin absolu vers le répertoire du SDK qui contient votre code FreeRTOS. Si vous n'utilisez pas de SDK, le `sdkConfiguration` bloc entier doit être omis.

`buildTool`

Chemin d'accès complet au script de création (.bat ou .sh) qui contient les commandes pour générer le code source. Toutes les références au chemin du code source dans la

commande de construction doivent être remplacées par la AWS IoT Device Tester variable `{{testdata.sourcePath}}` et les références au chemin du SDK doivent être remplacées par`{{sdkPath}}`. Utilisez l'`{{config.idtRootPath}}`espace réservé pour référencer le chemin IDT absolu ou relatif.

`testStartDelayms`

Spécifie le nombre de millisecondes que le lanceur de tests FreeRTOS attendra avant de commencer à exécuter les tests. Cela peut être utile si l'appareil testé commence à émettre des informations de test importantes avant qu'IDT n'ait la chance de se connecter et de commencer à se connecter en raison d'une latence réseau ou autre. La valeur maximale autorisée est de 30 000 ms (30 secondes). Cette valeur s'applique uniquement aux groupes de test FreeRTOS et ne s'applique pas aux autres groupes de test qui n'utilisent pas le lanceur de tests FreeRTOS, tels que les tests OTA.

`flashTool`

Chemin d'accès complet au script flash (.sh ou .bat) qui contient les commandes flash pour votre appareil. Toutes les références au chemin du code source dans la commande flash doivent être remplacées par la variable IDT pour FreeRTOS `{{testdata.sourcePath}}` et toutes les références au chemin de votre SDK doivent être remplacées par la variable IDT pour FreeRTOS. Utilisez l'espace réservé pour référencer le chemin IDT absolu ou relatif. `{{sdkPath}}` `{{config.idtRootPath}}`

`buildImageInfo`

`testsImageName`

Nom du fichier produit par la commande build lors de la création de tests à partir du *`freertos-source`*`/tests` dossier.

`demosImageName`

Nom du fichier produit par la commande build lors de la création de tests à partir du *`freertos-source`*`/demos` dossier.

`clientWifiConfig`

La Wi-Fi configuration du client. Les tests de Wi-Fi bibliothèque nécessitent une carte MCU pour se connecter à deux points d'accès. (Les deux points d'accès peuvent être les mêmes.) Cet attribut configure les Wi-Fi paramètres du premier point d'accès. Certains scénarios de Wi-Fi test supposent que le point d'accès soit suffisamment sécurisé et qu'il ne soit pas ouvert. Assurez-

vous que les deux points d'accès se trouvent sur le même sous-réseau que l'ordinateur hôte exécutant IDT.

`wifi_ssid`

Le Wi-Fi SSID.

`wifi_password`

Le Wi-Fi mot de passe.

`wifiSecurityType`

Type de Wi-Fi sécurité utilisé. Une des valeurs suivantes :

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

> **Note**
>
> Si votre tableau n'est pas compatible Wi-Fi, vous devez tout de même inclure la `clientWifiConfig` section dans votre `device.json` fichier, mais vous pouvez omettre des valeurs pour ces attributs.

`testWifiConfig`

La Wi-Fi configuration de test. Les tests de Wi-Fi bibliothèque nécessitent une carte MCU pour se connecter à deux points d'accès. (Les deux points d'accès peuvent être les mêmes.) Cet attribut configure le Wi-Fi paramètre du second point d'accès. Certains scénarios de Wi-Fi test supposent que le point d'accès soit suffisamment sécurisé et qu'il ne soit pas ouvert. Assurez-vous que les deux points d'accès se trouvent sur le même sous-réseau que l'ordinateur hôte exécutant IDT.

`wifiSSID`

Le Wi-Fi SSID.

`wifiPassword`

Le Wi-Fi mot de passe.

`wifiSecurityType`

Type de Wi-Fi sécurité utilisé. Une des valeurs suivantes :

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

> **Note**
>
> Si votre tableau n'est pas compatible Wi-Fi, vous devez tout de même inclure la `testWifiConfig` section dans votre `device.json` fichier, mais vous pouvez omettre des valeurs pour ces attributs.

`echoServerCertificateConfiguration`

L'espace réservé à la génération de certificats de serveur Echo configurable pour les tests de socket sécurisés. Ce champ est obligatoire.

`certificateGenerationMethod`

Spécifie si le certificat de serveur est généré automatiquement ou fourni manuellement.

`customPath`

`certificateGenerationMethod`Il s'agit de « Personnalisé », `certificatePath` et `privateKeyPath` ils sont obligatoires.

`certificatePath`

Spécifie le chemin du fichier pour le certificat de serveur.

`privateKeyPath`

Spécifie le chemin du fichier pour la clé privée.

`eccCurveFormat`

Spécifie le format de courbe pris en charge par la carte. Obligatoire lorsque `PKCS11` le paramètre est réglé sur « ecc » dans`device.json`. Les valeurs valides sont « P224 », « P256 », « P384 » ou « P521 ».

`echoServerConfiguration`

Les ports configurables du serveur Echo pour WiFi les tests de sockets sécurisés. Ce champ est facultatif.

`securePortForSecureSocket`

Port utilisé pour configurer un serveur echo avec TLS pour le test des sockets sécurisés. La valeur par défaut est 33333. Assurez-vous que le port configuré n'est pas bloqué par un pare-feu ou votre réseau d'entreprise.

`insecurePortForSecureSocket`

Port utilisé pour configurer le serveur echo sans TLS pour le test des sockets sécurisés. La valeur par défaut utilisée dans le test est 33334. Assurez-vous que le port configuré n'est pas bloqué par un pare-feu ou votre réseau d'entreprise.

`insecurePortForWiFi`

Le port utilisé pour configurer le serveur Echo sans TLS à des fins de WiFi test. La valeur par défaut utilisée dans le test est 33335. Assurez-vous que le port configuré n'est pas bloqué par un pare-feu ou votre réseau d'entreprise.

`otaConfiguration`

Configuration OTA. [Facultatif]

`otaFirmwareFilePath`

Chemin d'accès complet vers l'image OTA créée après la génération. Par exemple, `{{testData.sourcePath}}/`*`relative-path/to/ota/image/from/source/root`*.

`deviceFirmwareFileName`

Le chemin complet du fichier sur le périphérique MCU où se trouve le microprogramme OTA. Certains appareils n'utilisent pas ce champ, mais vous devez tout de même fournir une valeur.

`otaDemoConfigFilePath`

Chemin d'accès complet à `aws_demo_config.h`, disponible dans *`afr-source`*`/vendors/vendor/boards/board/aws_demos/config_files/`. Ces fichiers sont inclus dans le modèle de code de portage fourni par FreeRTOS.

`codeSigningConfiguration`

Configuration de signature de code.

`signingMethod`

Méthode de signature de code. Les valeurs possibles sont `AWS` ou `Custom`.

> **Note**
>
> Pour les régions de Pékin et de Ningxia, utilisez`Custom`. `AWS`la signature de code n'est pas prise en charge dans ces régions.

`signerHashingAlgorithm`

Algorithme de hachage pris en charge sur le périphérique. Les valeurs possibles sont `SHA1` ou `SHA256`.

`signerSigningAlgorithm`

Algorithme de signature pris en charge sur le périphérique. Les valeurs possibles sont `RSA` ou `ECDSA`.

`signerCertificate`

Certificat de confiance utilisé pour l'OTA.

Pour la méthode de signature du AWS code, utilisez l'Amazon Resource Name (ARN) pour le certificat sécurisé chargé sur le AWS Certificate Manager.

Pour la méthode de signature de code personnalisée, utilisez le chemin d'accès absolu vers le fichier de certificat du signataire.

Pour plus d'informations sur la création d'un certificat approuvé, consultez Créer un certificat de signature de code.

`signerCertificateFileName`

Le nom de fichier du certificat de signature de code sur l'appareil. Cette valeur doit correspondre au nom de fichier que vous avez fourni lors de l'exécution de la `aws acm import-certificate` commande.

Pour de plus amples informations, veuillez consulter Créer un certificat de signature de code.

`compileSignerCertificate`

Défini `true` si le certificat de vérification de signature du signataire de code n'est pas fourni ou flashé, il doit donc être compilé dans le projet. AWS IoT Device Tester récupère le certificat sécurisé et le compile dans. `aws_codesigner_certifiate.h`

`untrustedSignerCertificate`

L'ARN ou le chemin de fichier d'un deuxième certificat utilisé dans certains tests OTA en tant que certificat non fiable. Pour plus d'informations sur la création d'un certificat, voir Création d'un certificat de signature de code.

`signerPlatform`

Algorithme de signature et de hachage utilisé par AWS Code Signer lors de la création de la tâche de mise à jour OTA. Actuellement, les valeurs possibles pour ce champ sont `AmazonFreeRTOS-TI-CC3220SF` et `AmazonFreeRTOS-Default`.

- Choisissez `AmazonFreeRTOS-TI-CC3220SF`, si `SHA1` et `RSA`.
- Choisissez `AmazonFreeRTOS-Default`, si `SHA256` et `ECDSA`.

Si vous avez besoin de `SHA256 | RSA` ou `SHA1 | ECDSA` pour votre configuration, contactez-nous pour obtenir une assistance complémentaire.

Configurez `signCommand` si vous avez choisi `Custom` pour `signingMethod`.

`signCommand`

Commande utilisée pour effectuer la signature de code personnalisée. Vous pouvez trouver le modèle dans le répertoire `/configs/script_templates`.

Deux espaces réservés `{{inputImageFilePath}}` et `{{outputSignatureFilePath}}` sont obligatoires dans la commande. `{{inputImageFilePath}}` est le chemin d'accès au fichier de l'image créée par IDT qu'il faut signer. `{{outputSignatureFilePath}}` est le chemin d'accès au fichier de la signature qui sera généré par le script.

`cmakeConfiguration`

Configuration CMake [Facultatif]

> **Note**
>
> Pour exécuter les cas de test CMake, vous devez fournir le nom de la carte, le nom du fournisseur et `frToolchainPath` ou `compilerName`. Vous pouvez également fournir

le `cmakeToolchainPath` si vous avez un chemin personnalisé vers la chaîne d'outils CMake.

`boardName`

Nom de la carte testée. Le nom de la carte doit être le même que le nom du dossier sous *`path/to/afr/source/code`*`/vendors/`*`vendor`*`/boards/`*`board`*.

`vendorName`

Le nom du fournisseur pour le tableau testé. Le fournisseur doit être le même que le nom du dossier sous *`path/to/afr/source/code`*`/vendors/`*`vendor`*.

`compilerName`

Le nom du compilateur.

`frToolchainPath`

Le chemin d'accès complet au compilateur de la chaîne d'outils

`cmakeToolchainPath`

Le chemin d'accès qualifié complet à la chaîne d'outils CMake. Ce champ est facultatif

`freertosFileConfiguration`

Configuration des fichiers FreeRTOS qu'IDT modifie avant d'exécuter les tests.

`required`

Cette section indique les tests requis dont vous avez déplacé les fichiers de configuration, par exemple PKCS11, TLS, etc.

`configName`

Nom du test en cours de configuration.

`filePath`

Le chemin absolu vers les fichiers de configuration dans le *`freertos`* dépôt. Utilisez la `{{testData.sourcePath}}` variable pour définir le chemin.

`optional`

Cette section spécifie les tests facultatifs dont vous avez déplacé les fichiers de configuration WiFi, par exemple OTA, etc.

`configName`

Nom du test en cours de configuration.

`filePath`

Le chemin absolu vers les fichiers de configuration dans le *`freertos`* dépôt. Utilisez la `{{testData.sourcePath}}` variable pour définir le chemin.

> **Note**
>
> Pour exécuter les cas de test CMake, vous devez fournir le nom de la carte, le nom du fournisseur et `afrToolchainPath` ou `compilerName`. Vous pouvez également fournir `cmakeToolchainPath` si vous avez un chemin personnalisé vers la chaîne d'outils CMake.

IDT pour les variables FreeRTOS

Les commandes permettant de créer votre code et de flasher l'appareil peuvent nécessiter une connectivité ou d'autres informations sur vos appareils pour fonctionner correctement. AWS IoT Device Tester vous permet de référencer les informations du périphérique dans Flash et de créer des commandes à l'aide de JsonPath. À l'aide d' JsonPathexpressions simples, vous pouvez récupérer les informations requises spécifiées dans votre `device.json` fichier.

Variables de chemin

IDT pour FreeRTOS définit les variables de chemin suivantes qui peuvent être utilisées dans les lignes de commande et les fichiers de configuration :

`{{testData.sourcePath}}`

Développe le code source du chemin d'accès. Si vous utilisez cette variable, elle doit être utilisée dans les commandes flash et build.

`{{sdkPath}}`

S'étend jusqu'à la valeur de votre `userData.sdkConfiguration.path` lorsqu'elle est utilisée dans les commandes de compilation et de flash.

`{{device.connectivity.serialPort}}`

Extension au port série.

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

Extension jusqu'au numéro de série de votre appareil.

{{enableTests}}

Valeur entière indiquant si la génération est destinée à des tests (valeur 1) ou à des démonstrations (valeur 0).

{{buildImageName}}

Nom de fichier de l'image générée par la commande build.

{{otaCodeSignerPemFile}}

Fichier PEM pour le signataire du code OTA.

{{config.idtRootPath}}

S'étend jusqu'au chemin AWS IoT Device Tester racine. Cette variable remplace le chemin absolu pour IDT lorsqu'il est utilisé par les commandes build et flash.

Utiliser l'interface utilisateur IDT pour exécuter la suite de qualification FreeRTOS

À partir de IDT v4.3.0, AWS IoT Device Tester FreeRTOS (IDT-FreeRTOS) inclut une interface utilisateur Web qui vous permet d'interagir avec le fichier exécutable en ligne de commande IDT et les fichiers de configuration associés. Vous pouvez utiliser l' IDT-FreeRTOS interface utilisateur pour créer une nouvelle configuration afin d'exécuter des tests IDT ou pour modifier une configuration existante. Vous pouvez également utiliser l'interface utilisateur pour appeler l'exécutable IDT et exécuter des tests.

L' IDT-FreeRTOS interface utilisateur fournit les fonctions suivantes :

- Simplifiez la configuration des fichiers de configuration pour les IDT-FreeRTOS tests.
- Simplifiez l'utilisation IDT-FreeRTOS pour exécuter des tests de qualification.

Pour plus d'informations sur l'utilisation de la ligne de commande pour exécuter des tests de qualification, consultezPremier test de votre carte microcontrôleur.

Cette section décrit les conditions requises pour utiliser l' IDT-FreeRTOS interface utilisateur et vous montre comment commencer à exécuter des tests de qualification dans l'interface utilisateur.

Rubriques

- Configurez les prérequis pour exécuter la suite de qualification FreeRTOS
- Commencez avec l' IDT-FreeRTOS interface utilisateur

Configurez les prérequis pour exécuter la suite de qualification FreeRTOS

Cette section décrit les prérequis pour tester les microcontrôleurs avec. AWS IoT Device Tester

Rubriques

- Utiliser un navigateur Web compatible
- Télécharger FreeRTOS
- Télécharger IDT pour FreeRTOS
- Créez et configurez un AWS compte
- AWS IoT Device Tester politique gérée

Utiliser un navigateur Web compatible

L' IDT-FreeRTOS interface utilisateur prend en charge les navigateurs Web suivants.

Navigateur	Version
Google Chrome	Les trois dernières versions majeures
Mozilla Firefox	Les trois dernières versions majeures
Microsoft Edge	Les trois dernières versions majeures
Apple Safari pour macOS	Les trois dernières versions majeures

Nous vous recommandons d'utiliser Google Chrome ou Mozilla Firefox pour une meilleure expérience.

> **Note**
>
> L' IDT-FreeRTOS interface utilisateur ne prend pas en charge Microsoft Internet Explorer.

Télécharger FreeRTOS

Vous pouvez télécharger une version de FreeRTOS à l'aide GitHubde la commande suivante :

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

où se <FREERTOS_RELEASE_VERSION>trouve une version de FreeRTOS (par exemple, 202007.00) correspondant à une version IDT répertoriée dans. Versions prises en charge de AWS IoT Device Tester Cela garantit que vous disposez du code source complet, y compris les sous-modules, et que vous utilisez la bonne version d'IDT pour votre version de FreeRTOS, et vice versa.

Pour Windows, la limitation de la longueur du chemin est de 260 caractères. La structure des chemins de FreeRTOS comporte plusieurs niveaux. Par conséquent, si vous utilisez Windows, maintenez les chemins de vos fichiers en dessous de la limite de 260 caractères. Par exemple, clonez FreeRTOS sur plutôt que sur`C:\FreeRTOS`. `C:\Users\username\programs\projects\myproj\FreeRTOS\`

Considérations relatives à la qualification LTS (qualification pour FreeRTOS utilisant des bibliothèques LTS)

- Pour que votre microcontrôleur soit désigné comme compatible avec les versions basées sur le support à long terme (LTS) de FreeRTOS dans AWS le catalogue des appareils partenaires, vous devez fournir un fichier manifeste. Pour plus d'informations, consultez la liste de contrôle de qualification FreeRTOS dans le guide de qualification FreeRTOS.
- Afin de valider que votre microcontrôleur prend en charge les versions LTS de FreeRTOS et de le soumettre au Partner Device Catalog, vous devez AWS IoT Device Tester utiliser (IDT) avec AWS la version v1.4.x de la suite de tests FreeRTOS Qualification (FRQ).
- Support pour les versions LTS de FreeRTOS limité à la version 202012.xx de FreeRTOS.

Télécharger IDT pour FreeRTOS

Chaque version de FreeRTOS possède une version correspondante d'IDT pour FreeRTOS pour effectuer des tests de qualification. Téléchargez la version appropriée d'IDT pour FreeRTOS à partir de. Versions prises en charge de AWS IoT Device Tester

Extrayez IDT pour FreeRTOS vers un emplacement du système de fichiers où vous disposez d'autorisations de lecture et d'écriture. Microsoft Windows ayant une limite de caractères pour la longueur du chemin, extrayez IDT pour FreeRTOS dans un répertoire racine tel que ou. `C:\` `D:\`

> **Note**
>
> Nous vous recommandons d'extraire le package IDT sur un lecteur local. Si vous autorisez plusieurs utilisateurs à exécuter IDT à partir d'un emplacement partagé, tel qu'un répertoire NFS ou un dossier partagé sur le réseau Windows, le système risque de ne pas répondre ou de corrompre les données.

Créez et configurez un AWS compte

Inscrivez-vous pour un Compte AWS

Pour commencer AWS, vous avez besoin d'un Compte AWS. Pour plus d'informations sur la création d'un Compte AWS, voir Getting started with an Compte AWS dans le Guide de Gestion de compte AWS référence.

AWS IoT Device Tester politique gérée

Pour permettre au testeur d'appareils de s'exécuter et de collecter des métriques, la politique `AWSIoTDeviceTesterForFreeRTOSFullAccess` gérée contient les autorisations suivantes :

- `iot-device-tester:SupportedVersion`

  Accorde l'autorisation d'obtenir la liste des versions de FreeRTOS et des versions de suites de tests prises en charge par IDT, afin qu'elles soient disponibles auprès du. AWS CLI
- `iot-device-tester:LatestIdt`

  Accorde l'autorisation d'obtenir la dernière AWS IoT Device Tester version disponible au téléchargement.
- `iot-device-tester:CheckVersion`

  Accorde l'autorisation de vérifier la compatibilité d'une combinaison de produit, de suite de tests et de versions AWS IoT Device Tester .
- `iot-device-tester:DownloadTestSuite`

  Accorde l'autorisation AWS IoT Device Tester de télécharger des suites de tests.

- `iot-device-tester:SendMetrics`

  Accorde l'autorisation de publier des données de statistiques AWS IoT Device Tester d'utilisation.

Commencez avec l' IDT-FreeRTOS interface utilisateur

Cette section explique comment utiliser l' IDT-FreeRTOS interface utilisateur pour créer ou modifier votre configuration, puis comment exécuter des tests.

Rubriques

- Configurer AWS informations d'identification
- Ouvrez l' IDT-FreeRTOS interface utilisateur
- Création d'une nouvelle configuration
- Modifier une configuration existante
- Exécuter des tests de qualification

Configurer AWS informations d'identification

Vous devez configurer les informations d'identification de l' AWS utilisateur que vous avez créé dansCréez et configurez un AWS compte. Vous pouvez spécifier vos informations d'identification de deux manières :

- Dans un fichier d'informations d'identification
- En tant que variables d'environnement

Configurer AWS informations d'identification avec un fichier d'informations d'identification

IDT utilise le même fichier d'informations d'identification que l' AWS CLI. Pour de plus amples informations, veuillez consulter Fichiers de configuration et d'informations d'identification.

L'emplacement du fichier d'informations d'identification varie en fonction du système d'exploitation que vous utilisez :

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

Ajoutez vos AWS informations d'identification au `credentials` fichier au format suivant :

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

> **Note**
>
> Si vous n'utilisez pas le `default` AWS profil, veillez à spécifier le nom du profil dans l' IDT-FreeRTOS interface utilisateur. Pour plus d'informations sur les profils, consultez Configuration et paramètres des fichiers d'identification.

Configurer AWS informations d'identification avec variables d'environnement

Les variables d'environnement sont des variables gérées par le système d'exploitation et utilisées par les commandes du système. Ils ne sont pas enregistrés si vous fermez la session SSH. L' IDT-FreeRTOS interface utilisateur utilise les variables d'`AWS_SECRET_ACCESS_KEY`environnement `AWS_ACCESS_KEY_ID` et pour stocker vos AWS informations d'identification.

Pour définir ces variables sous Linux, macOS ou Unix, utilisez export:

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

Pour définir ces variables sous Windows, utilisez set :

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

Ouvrez l' IDT-FreeRTOS interface utilisateur

Pour ouvrir l' IDT-FreeRTOS interface utilisateur

1. Téléchargez une IDT-FreeRTOS version compatible et extrayez l'archive téléchargée dans un emplacement de votre système de fichiers où vous disposez d'autorisations de lecture et d'écriture.
2. Exécutez la commande suivante pour accéder au répertoire IDT-FreeRTOS d'installation :

   ```
   cd devicetester-extract-location/bin
   ```
3. Exécutez la commande suivante pour ouvrir l' IDT-FreeRTOS interface utilisateur :

Linux

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

Windows

```
./devicetestergui_win_x64-64
```

macOS

```
./devicetestergui_mac_x86-64
```

> **Note**
>
> Sur Mac, pour autoriser votre système à exécuter l'interface utilisateur, accédez à Préférences système -> Sécurité et confidentialité. Lorsque vous exécutez les tests, il se peut que vous deviez le faire trois fois de plus.

L' IDT-FreeRTOS interface utilisateur s'ouvre dans votre navigateur par défaut. Pour plus d'informations sur les navigateurs pris en charge, consultezUtiliser un navigateur Web compatible.

Création d'une nouvelle configuration

Si vous êtes un nouvel utilisateur, vous devez créer une nouvelle configuration pour configurer les fichiers de configuration JSON nécessaires à l' IDT-FreeRTOS exécution des tests. Vous pouvez ensuite exécuter des tests ou modifier la configuration créée.

Pour des exemples de `userdata.json` fichiers, et`config.json`,,`device.json`,,,,,,,,,Premier test de votre carte microcontrôleur,, Pour un exemple de `resource.json` fichier utilisé uniquement pour exécuter des tests Bluetooth Low Energy (BLE), consultezExécuter des tests Bluetooth Low Energy.

Pour créer une nouvelle configuration

1. Dans l' IDT-FreeRTOS interface utilisateur, ouvrez le menu de navigation, puis choisissez Créer une nouvelle configuration.

> **Important**
>
> Vous devez configurer vos AWS informations d'identification avant d'ouvrir l'interface utilisateur. Si vous n'avez pas configuré vos informations d'identification, fermez la fenêtre du navigateur d' IDT-FreeRTOS interface utilisateur, suivez les étapes décritesConfigurer AWS informations d'identification, puis rouvrez l' IDT-FreeRTOS interface utilisateur.

2. Suivez l'assistant de configuration pour saisir les paramètres de configuration IDT utilisés pour exécuter les tests de qualification. L'assistant configure les paramètres suivants dans les fichiers de configuration JSON situés dans le *`devicetester-extract-location`*`/config` répertoire.

   - AWS paramètres : Compte AWS informations IDT-FreeRTOS utilisées pour créer des AWS ressources lors des tests. Ces paramètres sont configurés dans le `config.json` fichier.
   - Référentiel FreeRTOS : chemin absolu vers le référentiel FreeRTOS et le code porté, ainsi que le type de qualification que vous souhaitez effectuer. Ces paramètres sont configurés dans le `userdata.json` fichier.

     Vous devez transférer FreeRTOS pour votre appareil avant de pouvoir exécuter des tests de qualification. Pour plus d'informations, consultez le guide de portage de FreeRTOS
   - Build and flash : commandes build et flash pour votre matériel qui permettent à IDT de créer et de tester automatiquement votre carte mère. Ces paramètres sont configurés dans le `userdata.json` fichier.
   - Appareils : paramètres du pool de périphériques pour les appareils à tester. Ces paramètres sont configurés dans `sku` les champs `id` et dans le `devices` bloc du pool de périphériques du `device.json` fichier.
   - Mise en réseau : paramètres permettant de tester la prise en charge des communications réseau pour vos appareils. Ces paramètres sont configurés dans le `features` bloc du `device.json` fichier et dans les `testWifiConfig` blocs `clientWifiConfig` et du `userdata.json` fichier.
   - Serveur Echo : paramètres de configuration du serveur Echo pour les tests de socket sécurisés. Ces paramètres sont configurés dans le `userdata.json` fichier.

     Pour plus d'informations sur la configuration du serveur Echo, consultezhttps://docs.aws.amazon.com/freertos/latest/portingguide/afr-echo-server.html.

- CMake — (Facultatif) Les paramètres pour exécuter les tests de fonctionnalité de construction de CMake. Cette configuration n'est requise que si vous utilisez CMake comme système de compilation. Ces paramètres sont configurés dans le `userdata.json` fichier.
- BLE —Paramètres permettant d'exécuter des tests de fonctionnalité Bluetooth Low Energy. Ces paramètres sont configurés dans le `features` bloc du `device.json` fichier et dans le `resource.json` fichier.
- OTA : paramètres permettant d'exécuter des tests de fonctionnalité OTA. Ces paramètres sont configurés dans le `features` bloc du `device.json` fichier et dans le `userdata.json` fichier.

3. Sur la page Révision, vérifiez vos informations de configuration.

Après avoir passé en revue votre configuration, pour exécuter vos tests de qualification, choisissez Run tests.

Modifier une configuration existante

Si vous avez déjà configuré des fichiers de configuration pour IDT, vous pouvez utiliser l' IDT-FreeRTOS interface utilisateur pour modifier votre configuration existante. Assurez-vous que vos fichiers de configuration existants sont disponibles dans le *`devicetester-extract-location`*`/config` répertoire.

Pour modifier une nouvelle configuration

1. Dans l' IDT-FreeRTOS interface utilisateur, ouvrez le menu de navigation, puis choisissez Modifier la configuration existante.

   Le tableau de bord de configuration affiche des informations sur vos paramètres de configuration existants. Si une configuration est incorrecte ou indisponible, le statut de cette configuration est`Error validating configuration`.

2. Pour modifier un paramètre de configuration existant, procédez comme suit :

   a. Choisissez le nom d'un paramètre de configuration pour ouvrir sa page de paramètres.

   b. Modifiez les paramètres, puis choisissez Enregistrer pour régénérer le fichier de configuration correspondant.

Une fois que vous avez terminé de modifier votre configuration, vérifiez que tous vos paramètres de configuration sont validés. Si le statut de chaque paramètre de configuration est`Valid`, vous pouvez exécuter vos tests de qualification à l'aide de cette configuration.

Exécuter des tests de qualification

Après avoir créé une configuration pour IDT-FreeRTOS, vous pouvez exécuter vos tests de qualification.

Pour exécuter des tests de qualification

1. Validez votre configuration.
2. Dans le menu de navigation, choisissez Exécuter des tests.
3. Pour démarrer le test, choisissez Démarrer les tests.

IDT-FreeRTOS exécute les tests de qualification et affiche le résumé des tests ainsi que les éventuelles erreurs dans la console Test Runner. Une fois le test terminé, vous pouvez consulter les résultats du test et les journaux depuis les emplacements suivants :

- Les résultats des tests se trouvent dans le *`devicetester-extract-location`*`/results/`*`execution-id`* répertoire.
- Les journaux de test se trouvent dans le *`devicetester-extract-location`*`/results/`*`execution-id`*`/logs` répertoire.

Pour plus d'informations sur les résultats des tests et les journaux, consultez Afficher les résultats de l'IDT pour FreeRTOS etAfficher les logs IDT pour FreeRTOS.

Exécuter des tests Bluetooth Low Energy

Cette section décrit comment configurer et exécuter des tests Bluetooth Low Energy à l'aide AWS IoT Device Tester de FreeRTOS.

Les tests Bluetooth ne sont pas requis pour la qualification principale. Si vous ne souhaitez pas tester votre appareil avec le support Bluetooth FreeRTOS, vous pouvez ignorer cette configuration. Assurez-vous de laisser la fonctionnalité BLE dans device.json définie sur. No

Conditions préalables

- Suivez les instructions de la section Premier test de votre carte microcontrôleur.

- Un Raspberry Pi 4B ou 3B+. (Requis pour exécuter l'application complémentaire Raspberry Pi BLE)
- Une carte micro SD et un adaptateur de carte SD pour le logiciel Raspberry Pi.

Configuration de Raspberry Pi Setup

Pour tester les capacités BLE de l'appareil testé (DUT), vous devez disposer d'un Raspberry Pi modèle 4B ou 3B+.

Pour configurer votre Raspberry Pi afin d'exécuter les tests BLE

1. Téléchargez l'une des images Yocto personnalisées contenant le logiciel requis pour effectuer les tests.

   - Image pour Raspberry Pi 4B
   - Image pour Raspberry Pi 3B+

   > **Note**
   >
   > L'image Yocto ne doit être utilisée qu'à des fins de test avec AWS IoT Device Tester FreeRTOS et à aucune autre fin.

2. Flashez l'image yocto sur la carte SD pour Raspberry Pi.

   - En utilisant un outil d'écriture sur carte SD comme Etcher, flashez le fichier *`image-name`*`.rpi-sd.img` téléchargé sur la carte SD. Comme l'image du système d'exploitation est volumineuse, cette étape peut prendre un peu de temps. Éjectez ensuite votre carte SD de votre ordinateur et insérez la carte microSD dans votre Raspberry Pi.

3. Configurez votre Raspberry Pi.

   a. Pour le premier démarrage, nous vous recommandons de connecter le Raspberry Pi à un moniteur, un clavier et une souris.

   b. Connectez votre Raspberry Pi à une source d'alimentation micro USB.

   c. Connectez-vous à l'aide des informations d'identification par défaut. Pour l'ID utilisateur, saisissez **`root`**. Pour le mot de passe, saisissez **`idtafr`**.

d. À l'aide d'une Wi-Fi connexion Ethernet ou d'une connexion, connectez le Raspberry Pi à votre réseau.

   i. Pour connecter votre Raspberry Pi Wi-Fi, ouvrez `/etc/wpa_supplicant.conf` le Raspberry Pi et ajoutez vos Wi-Fi informations d'identification à la `Network` configuration.

```
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. Exécutez `ifup wlan0` pour démarrer la Wi-Fi connexion. La connexion à votre Wi-Fi réseau peut prendre une minute.

e. Pour une connexion Ethernet, exécutez `ifconfig eth0`. Pour Wi-Fi établir une connexion, lancez`ifconfig wlan0`. Prenez note de l'adresse IP qui apparaît sous la forme `inet addr` dans la sortie de commande. Vous aurez besoin de l'adresse IP ultérieurement dans cette procédure.

f. (Facultatif) Les tests exécutent les commandes sur le Raspberry Pi via SSH à l'aide des informations d'identification par défaut pour l'image yocto. Pour plus de sécurité, nous vous recommandons de configurer l'authentification par clé publique pour SSH et de désactiver SSH basé sur un mot de passe.

   i. Créez une clé SSH à l'aide de la commande OpenSSL`ssh-keygen`. Si vous avez déjà une paire de clés SSK sur votre ordinateur hôte, il est recommandé d'en créer une nouvelle AWS IoT Device Tester pour permettre à FreeRTOS de se connecter à votre Raspberry Pi.

> **Note**
>
> Windows n'est fourni avec une installation de client SSH. Pour plus d'informations sur la façon d'installer un client SSH sur Windows, consultez la section Download SSH Software.

ii. La commande `ssh-keygen` vous invite à indiquer un nom et un chemin d'accès pour stocker la paire de clés. Par défaut, les fichiers de la paire de clés sont nommés `id_rsa` (clé privée) et `id_rsa.pub` (clé publique). Sur Mac OS et Linux, l'emplacement par défaut de ces fichiers est `~/.ssh/`. Sur Windows, l'emplacement par défaut est `C:\Users\`*`user-name`*.

iii. Lorsque vous êtes invité à saisir une expression clé, appuyez sur Entrée pour continuer.

iv. Pour ajouter votre clé SSH sur votre Raspberry Pi afin que AWS IoT Device Tester FreeRTOS puisse se connecter à l'appareil, utilisez `ssh-copy-id` la commande de votre ordinateur hôte. Cette commande ajoute votre clé publique au fichier `~/.ssh/authorized_keys` sur votre Raspberry Pi.

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

v. Lorsque vous êtes invité à saisir un mot de passe, saisissez **`idtafr`**. Il s'agit du mot de passe par défaut de l'image yocto.

> **Note**
>
> La commande `ssh-copy-id` suppose que la clé publique est nommée `id_rsa.pub`. Sur macOS et Linux, l'emplacement par défaut est `~/.ssh/`. Sur Windows, l'emplacement par défaut est `C:\Users\`*`user-name`*`\.ssh`. Si vous avez donné à la clé publique un autre nom ou si vous l'avez stockée à un autre emplacement, vous devez spécifier le chemin d'accès qualifié complet de votre clé publique SSH à l'aide de l'option `-i` pour `ssh-copy-id` (par exemple, `ssh-copy-id -i ~/my/path/myKey.pub`). Pour plus d'informations sur la création de clés SSH et la copie de clés publiques, consultez SSH-COPY-ID.

vi. Pour vérifier que l'authentification par clé publique fonctionne, exécutez `ssh -i` *`/my/path/myKey`* `root@`*`raspberry-pi-device-ip`*.

Si vous n'êtes pas invité à saisir un mot de passe, votre authentification par clé publique fonctionne.

vii. Vérifiez que vous pouvez vous connecter à votre Raspberry Pi à l'aide d'une clé publique, puis désactivez SSH basé sur un mot de passe.

A. Sur le Raspberry Pi, modifiez le fichier `/etc/ssh/sshd_config`.

B. Définissez l'attribut `PasswordAuthentication` sur no.

C. Enregistrez et fermez le fichier `sshd_config`.

D. Rechargez le serveur SSH en exécutant `/etc/init.d/sshd reload`.

g. Créez un fichier `resource.json`.

i. Dans le répertoire dans lequel vous avez extrait AWS IoT Device Tester, créez un fichier nomméresource.json.

ii. Ajoutez les informations suivantes concernant votre Raspberry Pi au fichier, en les *rasp-pi-ip-address* remplaçant par l'adresse IP de votre 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. Si vous n'avez pas choisi d'utiliser l'authentification par clé publique pour SSH, ajoutez ce qui suit à la `connectivity` section du `resource.json` fichier.

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

```
}
```

iv. (Facultatif) Si vous avez choisi d'utiliser l'authentification par clé publique pour SSH, ajoutez les éléments suivants à la section `connectivity` du fichier `resource.json`.

```
"connectivity": {
    "protocol": "ssh",
    "ip": "rasp-pi-ip-address",
    "auth": {
        "method": "pki",
        "credentials": {
            "user": "root",
            "privKeyPath": "location-of-private-key"
        }
    }
}
```

Configuration de l'appareil FreeRTOS

Dans votre fichier `device.json`, définissez la fonctionnalité `BLE` sur `Yes`. Si vous commencez avec un fichier `device.json` qui existait avant que les tests Bluetooth ne soient disponibles, vous devez ajouter la fonctionnalité BLE au tableau `features` :

```
{
    ...
    "features": [
        {
            "name": "BLE",
            "value": "Yes"
        },
    ...
}
```

Exécutez les tests BLE

Une fois que vous avez activé la fonctionnalité BLE dans `device.json`, les tests BLE commencent lorsque vous exécutez `devicetester_`*`[linux | mac | win_x86-64]`* `run-suite` sans spécifier d'ID de groupe.

Si vous souhaitez exécuter les tests BLE séparément, vous pouvez spécifier l'ID de groupe pour BLE : `devicetester_[linux | mac | win_x86-64] run-suite --userdata path-to-userdata/userdata.json --group-id FullBLE`.

Pour des performances optimales, placez votre Raspberry Pi à proximité de l'appareil testé.

Résoudre les problèmes liés aux tests BLE

Vérifiez que vous avez suivi les étapes décrites dans Premier test de votre carte microcontrôleur. Si des tests autres que les tests BLE échouent, le problème n'est probablement pas dû à la configuration Bluetooth.

Exécutez la suite de qualification FreeRTOS

Vous utilisez l'exécutable AWS IoT Device Tester for FreeRTOS pour interagir avec IDT pour FreeRTOS. Les exemples de ligne de commande ci-dessous vous montrent comment exécuter les tests de qualification d'un groupe d'appareils (ensemble d'appareils identiques).

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
```

Exécute une suite de tests sur un groupe d'appareils. Le fichier `userdata.json` doit figurer dans le répertoire *devicetester_extract_location*`/devicetester_afreertos_[win|mac|linux]/configs/`.

> **Note**
>
> Si vous utilisez IDT pour FreeRTOS sous Windows, utilisez des barres obliques (/) pour spécifier le chemin du fichier. `userdata.json`

Utilisez la commande suivante pour exécuter un groupe de tests spécifique :

```
devicetester_[linux | mac | win] run-suite  \
    --suite-id FRQ_1.0.1  \
    --group-id group-id  \
    --pool-id pool-id  \
    --userdata userdata.json
```

Les paramètres `suite-id` et `pool-id` sont facultatifs si vous exécutez une seule suite de tests au niveau d'un seul groupe d'appareils (à savoir, si un seul groupe d'appareils est défini dans votre fichier `device.json`).

Utilisez la commande suivante pour exécuter un cas de test spécifique dans un groupe de tests :

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

Vous pouvez utiliser la commande `list-test-cases` pour répertorier les cas de test dans un groupe de tests.

Options de ligne de commande IDT pour FreeRTOS

group-id

(Facultatif) Groupes de tests à exécuter, sous la forme d'une liste séparée par des virgules. Si cette option n'est pas spécifiée, IDT exécute tous les groupes de tests de la suite de tests.

pool-id

(Facultatif) Le groupe d'appareils à tester. Option requise si vous définissez plusieurs groupes d'appareils dans `device.json`. Si vous avez un seul groupe d'appareils, vous pouvez omettre cette option.

suite-id

(Facultatif) Version de la suite de tests à exécuter. Si cette option n'est pas spécifiée, IDT utilise la dernière version dans le répertoire tests de votre système.

> **Note**
>
> Depuis IDT v3.0.0, IDT recherche en ligne des suites de tests plus récentes. Pour de plus amples informations, veuillez consulter Versions de la suite de tests.

test-id

(Facultatif) Tests à exécuter, sous la forme d'une liste séparée par des virgules. Si cette option est spécifiée, `group-id` doit spécifier un groupe unique.

Example

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

update-idt

(Facultatif) Si ce paramètre n'est pas défini et qu'une version IDT plus récente est disponible, vous serez invité à mettre à jour IDT. Si ce paramètre est défini surY, IDT arrêtera l'exécution du test s'il détecte qu'une version plus récente est disponible. Si ce paramètre est défini surN, IDT poursuivra l'exécution du test.

politique de gestion des mises à jour

(Facultatif) Si ce paramètre n'est pas utilisé et qu'IDT détecte que votre politique gérée n'est pas à jour, vous serez invité à mettre à jour votre politique gérée. Si ce paramètre est défini surY, IDT arrêtera l'exécution du test s'il détecte que votre politique gérée n'est pas à jour. Si ce paramètre est défini surN, IDT poursuivra l'exécution du test.

upgrade-test-suite

(Facultatif) Si cette option n'est pas utilisée et qu'une version de suite de tests plus récente est disponible, vous êtes invité à la télécharger. Pour masquer l'invite, spécifiez y pour toujours télécharger la suite de tests la plus récente ou spécifiez n pour utiliser la suite de tests spécifiée ou la dernière version sur votre système.

Example

Exemple

Pour toujours télécharger et utiliser la suite de tests la plus récente, utilisez la commande suivante.

```
devicetester_[linux | mac | win_x86-64] run-suite --userdata userdata file --
group-id group ID --upgrade-test-suite y
```

Pour utiliser la dernière suite de tests sur votre système, utilisez la commande suivante.

```
devicetester_[linux | mac | win_x86-64] run-suite --userdata userdata file --
group-id group ID --upgrade-test-suite n
```

h

Utilisez l'option d'aide pour en savoir plus sur les options `run-suite`.

Example

Exemple

```
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
```

Le fichier `userdata.json` doit être situé dans le répertoire *devicetester_extract_location*`/devicetester_afreertos_`*[win|mac|linux]*`/configs/`.

> **Note**
>
> Si vous utilisez IDT pour FreeRTOS sous Windows, utilisez des barres obliques (/) pour spécifier le chemin du fichier. `userdata.json`

Utilisez la commande suivante pour exécuter un groupe de tests spécifique.

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

Les valeurs `suite-id` et `pool-id` sont facultatives si vous exécutez une seule suite de tests au niveau d'un seul groupe d'appareils (autrement, si un seul groupe d'appareils est défini dans le fichier `device.json`).

Options de ligne de commande IDT pour FreeRTOS

group-id

(Facultatif) Spécifie le groupe de tests.

pool-id

Spécifie le groupe d'appareils à tester. Si vous avez un seul groupe d'appareils, vous pouvez omettre cette option.

suite-id

(Facultatif) Spécifie la suite de tests à exécuter.

IDT pour les commandes FreeRTOS

La commande IDT pour FreeRTOS prend en charge les opérations suivantes :

IDT v3.0.0 and later

`help`

Répertorie les informations sur la commande spécifiée.

`list-groups`

Répertorie les groupes dans une suite donnée.

`list-suites`

Répertorie les suites disponibles.

`list-supported-products`

Répertorie les produits et les versions de suite de tests pris en charge.

`list-supported-versions`

Répertorie les versions de FreeRTOS et de suites de tests prises en charge par la version actuelle d'IDT.

`list-test-cases`

Répertorie les cas de test dans un groupe spécifié.

`run-suite`

Exécute une suite de tests sur un groupe d'appareils.

Utilisez l'option `--suite-id` pour spécifier une version de suite de tests ou omettez-la pour utiliser la dernière version sur votre système.

Utilisez l'option `--test-id` pour exécuter un cas de test individuel.

Example

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

Pour obtenir la liste complète des options, veuillez consulter Exécutez la suite de qualification FreeRTOS.

> **Note**
>
> Depuis IDT v3.0.0, IDT recherche en ligne des suites de tests plus récentes. Pour de plus amples informations, veuillez consulter Versions de la suite de tests.

IDT v1.7.0 and earlier

`help`

Répertorie les informations sur la commande spécifiée.

`list-groups`

Répertorie les groupes dans une suite donnée.

`list-suites`

Répertorie les suites disponibles.

`run-suite`

Exécute une suite de tests sur un groupe d'appareils.

Test de requalification

À mesure que de nouvelles versions d'IDT pour les tests de qualification FreeRTOS sont publiées, ou que vous mettez à jour des packages ou des pilotes de périphériques spécifiques à vos cartes, vous pouvez utiliser IDT pour FreeRTOS pour tester vos cartes microcontrôleurs. Pour les qualifications ultérieures, assurez-vous de disposer des dernières versions de FreeRTOS et d'IDT pour FreeRTOS et recommencez les tests de qualification.

Afficher les résultats de l'IDT pour FreeRTOS

Lorsqu'il s'exécute, IDT écrit les erreurs sur la console, les fichiers journaux et les rapports de tests. Après avoir terminé la gamme de test de qualification, IDT écrit un résumé de la série de tests dans la console et génère deux rapports d'essais. Ces rapports se trouvent à l'emplacement *`devicetester-extract-location`*`/results/`*`execution-id`*`/`. Les deux rapports capturent les résultats de l'exécution de la suite de tests de qualification.

`awsiotdevicetester_report.xml`Il s'agit du rapport de test de qualification que vous soumettez AWS pour inscrire votre appareil dans le catalogue des appareils AWS partenaires. Ce rapport contient les éléments suivants :

- La version IDT pour FreeRTOS.
- La version de FreeRTOS qui a été testée.
- Les fonctionnalités de FreeRTOS prises en charge par l'appareil sont basées sur les tests réussis.
- La référente et le nom de l'appareil spécifiés dans le fichier `device.json`.
- Les caractéristiques de l'appareil spécifiées dans le fichier `device.json`.
- Un récapitulatif des résultats des scénarios de test.
- Répartition des résultats des scénarios de test par bibliothèques testées en fonction des fonctionnalités de l'appareil (par exemple FullWiFi, FullMQTT, etc.).
- Si cette qualification de FreeRTOS concerne la version 202012.00 qui utilise les bibliothèques LTS.

`FRQ_Report.xml` est un rapport au format standard JUnit XML. Vous pouvez l'intégrer à CI/CD des plateformes telles que Jenkins, Bamboo, etc. Ce rapport contient les éléments suivants :

- Un récapitulatif des résultats des scénarios de test.
- Une répartition des résultats des cas d'utilisation par bibliothèques qui ont été testés en fonction des fonctionnalités de l'appareil.

Interpréter l'IDT pour les résultats FreeRTOS

La section Rapport dans `awsiotdevicetester_report.xml` ou `FRQ_Report.xml` répertorie les résultats des tests exécutés.

La première balise XML `<testsuites>` contient le résumé global de l'exécution des tests. Par exemple :

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

Attributs utilisés dans la **`<testsuites>`** balise

`name`

Nom de la suite de tests.

`time`

Durée, en secondes, nécessaire pour exécuter la suite de qualification.

`tests`

Nombre de scénarios de test exécutés.

`failures`

Nombre de scénarios de tests qui ont été exécutés, mais dont le résultat n'est pas probant.

`errors`

Le nombre de cas de test que IDT pour FreeRTOS n'a pas pu exécuter.

`disabled`

Cet attribut n'est pas utilisé et peut être ignoré.

S'il n'y a aucun échec ou erreur dans le scénario de test, votre appareil répond aux exigences techniques pour exécuter FreeRTOS et peut interagir avec les services. AWS IoT Si vous choisissez de répertorier votre appareil dans le catalogue des appareils AWS partenaires, vous pouvez utiliser ce rapport comme preuve de qualification.

En cas d'erreurs ou d'échec de scénarios de test, vous pouvez identifier les scénarios concernés à l'aide des balises XML `<testsuites>`. Les balises XML `<testsuite>` au sein de la balise `<testsuites>` indiquent le récapitulatif des résultats d'un groupe de tests.

```
<testsuite name="FullMQTT" package="" tests="16" failures="0" time="76"
disabled="0" errors="0" skipped="0">
```

Le format est similaire à la balise `<testsuites>`, mais avec un attribut supplémentaire appelé `skipped` qui n'est pas utilisé et qui ne peut pas être ignoré. Chaque balise XML `<testsuite>`inclut des balises `<testcase>` pour chaque scénario de test qui a été exécuté pour un groupe de tests. Par exemple :

```
<testcase classname="mcu.Full_MQTT" name="AFQP_MQTT_Connect_HappyCase"
attempts="1"></testcase>
```

Attributs utilisés dans la **`<awsproduct>`** balise

`name`

Nom du produit testé.

`version`

Version du produit testé.

`sdk`

Si vous avez exécuté IDT avec un SDK, ce bloc contient le nom et la version de votre SDK. Si vous n'avez pas exécuté IDT avec un SDK, ce bloc contient :

```
<sdk>
    <name>N/A</vame>
    <version>N/A</version>
</sdk>
```

`features`

Caractéristiques validées. Les caractéristiques portant la mention `required` sont requises pour pouvoir envoyer votre carte en vue de sa certification. L'extrait suivant montre comment cela apparaît dans le `awsiotdevicetester_report.xml` fichier.

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

Les fonctionnalités marquées comme `optional` ne sont pas requises pour l'éligibilité. Les extraits suivants illustrent des fonctions facultatives.

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

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

S'il n'y a aucun échec ou erreur de test pour les fonctionnalités requises, votre appareil répond aux exigences techniques pour exécuter FreeRTOS et peut interagir avec les services. AWS IoT Si vous souhaitez répertorier votre appareil dans le catalogue des appareils AWS partenaires, vous pouvez utiliser ce rapport comme preuve de qualification.

En cas d'erreurs ou d'échecs de tests, vous pouvez identifier les tests concernés à l'aide des balises XML `<testsuites>`. Les balises XML `<testsuite>` au sein de la balise `<testsuites>` montrent le récapitulatif des résultats d'un groupe de tests. Par exemple :

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

Le format est similaire à celui de la `<testsuites>` balise, mais possède un `skipped` attribut qui n'est pas utilisé et qui peut être ignoré. Chaque balise XML `<testsuite>` inclut des balises `<testcase>` pour chaque test exécuté pour un groupe de tests. Par exemple :

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

`lts`

True si vous êtes éligible à une version de FreeRTOS qui utilise les bibliothèques LTS, false dans le cas contraire.

Attributs utilisés dans la **`<testcase>`** balise

`name`

Nom du scénario de test.

`attempts`

Le nombre de fois où IDT pour FreeRTOS a exécuté le scénario de test.

Lorsqu'un test échoue ou qu'une erreur se produit, les balises `<failure>` ou `<error>` sont ajoutées à la balise `<testcase>` avec des informations relatives au dépannage. Par exemple :

```
<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>
```

Pour de plus amples informations, veuillez consulter Résolution des erreurs.

Afficher les logs IDT pour FreeRTOS

Vous pouvez trouver les journaux qu'IDT pour FreeRTOS génère à partir de l'exécution des tests dans. *devicetester-extract-location*/results/*execution-id*/logs Deux ensembles de journaux sont générés :

test_manager.log

Contient les journaux générés par IDT pour FreeRTOS (par exemple, la configuration associée aux journaux et la génération de rapports).

***test_group_id*__*test_case_id*.log** (par exemple, **FullMQTT__Full_MQTT.log**)

Fichier journal d'un cas de test, y compris la sortie de l'appareil en cours de test. Le fichier journal est nommé en fonction du groupe de test et du cas de test qui a été exécuté.

Développez et exécutez vos propres suites de tests IDT

À partir de IDT v4.0.0, IDT pour FreeRTOS combine une configuration standardisée et un format de résultat avec un environnement de suites de tests qui vous permet de développer des suites de tests personnalisées pour vos appareils et leurs logiciels. Vous pouvez ajouter des tests personnalisés pour votre propre validation interne ou les fournir à vos clients pour la vérification des appareils.

Utilisez IDT pour développer et exécuter des suites de tests personnalisées, comme suit :

Pour développer des suites de tests personnalisées

- Créez des suites de tests avec une logique de test personnalisée pour l'appareil que vous souhaitez tester.
- Fournissez à IDT vos suites de tests personnalisées aux testeurs. Incluez des informations sur les configurations de paramètres spécifiques de vos suites de tests.

Pour exécuter des suites de tests personnalisées

- Configurez l'appareil que vous souhaitez tester.
- Implémentez les configurations de paramètres requises par les suites de tests que vous souhaitez utiliser.

- Utilisez IDT pour exécuter vos suites de tests personnalisées.
- Consultez les résultats des tests et les journaux d'exécution des tests exécutés par IDT.

Téléchargez la dernière version de AWS IoT Testeur d'appareils pour FreeRTOS

Téléchargez la dernière version d'IDT et extrayez le logiciel dans un emplacement de votre système de fichiers où vous disposez d'autorisations de lecture et d'écriture.

> **Note**
>
> IDT ne prend pas en charge son exécution par plusieurs utilisateurs à partir d'un emplacement partagé, tel qu'un répertoire NFS ou un dossier partagé réseau Windows. Nous vous recommandons d'extraire le package IDT sur une unité locale et d'exécuter le fichier binaire IDT sur votre station de travail locale.
> Pour Windows, la limitation de la longueur du chemin est de 260 caractères. Si vous utilisez Windows, décompressez IDT dans un répertoire racine comme `C:\` ou `D:\` afin que la longueur de vos chemins respecte la limite de 260 caractères.

Workflow de la suite de tests

Les suites de tests sont composées de trois types de fichiers :

- Fichiers de configuration fournissant à IDT des informations sur la manière d'exécuter la suite de tests.
- Testez les fichiers exécutables utilisés par IDT pour exécuter des scénarios de test.
- Des fichiers supplémentaires sont nécessaires pour exécuter les tests.

Suivez les étapes de base suivantes pour créer des tests IDT personnalisés :

1. Créez des fichiers de configuration pour votre suite de tests.
2. Créez des exécutables de cas de test contenant la logique de test de votre suite de tests.
3. Vérifiez et documentez les informations de configuration requises pour que les testeurs exécutent la suite de tests.

4. Vérifiez qu'IDT peut exécuter votre suite de tests et produire les résultats des tests comme prévu.

Pour créer rapidement un exemple de suite personnalisée et l'exécuter, suivez les instructions figurant dansTutoriel : création et exécution de l'exemple de suite de tests IDT.

Pour commencer à créer une suite de tests personnalisée en Python, consultezTutoriel : Développement d'une suite de tests IDT simple.

Tutoriel : création et exécution de l'exemple de suite de tests IDT

Le AWS IoT Device Tester téléchargement inclut le code source d'un exemple de suite de tests. Vous pouvez suivre ce didacticiel pour créer et exécuter l'exemple de suite de tests afin de comprendre comment vous pouvez utiliser FreeRTOS AWS IoT Device Tester pour exécuter des suites de tests personnalisées. Bien que ce didacticiel utilise le protocole SSH, il est utile d'apprendre à l'utiliser AWS IoT Device Tester avec les appareils FreeRTOS.

Dans ce didacticiel, vous allez effectuer les étapes suivantes :

1. Créez la suite d'exemples de tests
2. Utiliser IDT pour exécuter l'exemple de suite de tests

Rubriques

- Configurer les prérequis pour l'exemple de suite de tests
- Configurer les informations de l'appareil pour IDT
- Créez la suite d'exemples de tests
- Utiliser IDT pour exécuter l'exemple de suite de tests
- Résolution des erreurs

Configurer les prérequis pour l'exemple de suite de tests

Pour suivre ce didacticiel, vous aurez besoin des éléments suivants :

- Exigences relatives à l'ordinateur hôte
  - Dernière version de AWS IoT Device Tester
  - Python 3.7 ou version ultérieure

Pour vérifier la version de Python installée sur votre ordinateur, exécutez la commande suivante :

```
python3 --version
```

Sous Windows, si l'utilisation de cette commande renvoie une erreur, utilisez-la à la `python --version` place. Si le numéro de version renvoyé est 3.7 ou supérieur, exécutez la commande suivante dans un terminal Powershell pour la définir `python3` comme alias pour votre `python` commande.

```
Set-Alias -Name "python3" -Value "python"
```

Si aucune information de version n'est renvoyée ou si le numéro de version est inférieur à 3.7, suivez les instructions de la section Télécharger Python pour installer Python 3.7+. Pour plus d'informations, consultez la documentation Python.

- urllib3

  Pour vérifier qu'`urllib3`il est correctement installé, exécutez la commande suivante :

  ```
  python3 -c 'import urllib3'
  ```

  S'il n'`urllib3`est pas installé, exécutez la commande suivante pour l'installer :

  ```
  python3 -m pip install urllib3
  ```

- Exigences relatives aux dispositifs
  - Appareil doté d'un système d'exploitation Linux et d'une connexion réseau au même réseau que votre ordinateur hôte.

    Nous vous recommandons d'utiliser un Raspberry Pi avec le système d'exploitation Raspberry Pi. Assurez-vous de configurer SSH sur votre Raspberry Pi pour vous y connecter à distance.

Configurer les informations de l'appareil pour IDT

Configurez les informations de votre appareil pour qu'IDT exécute le test. Vous devez mettre à jour le `device.json` modèle situé dans le *`<device-tester-extract-location>`*`/configs` dossier avec les informations suivantes.

```
[
  {
    "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>"
            }
          }
        }
      }
    ]
  }
]
```

Dans l'`devices`objet, fournissez les informations suivantes :

`id`

Identifiant unique défini par l'utilisateur pour votre appareil.

`connectivity.ip`

L'adresse IP de votre appareil.

`connectivity.port`

Facultatif. Le numéro de port à utiliser pour les connexions SSH à votre appareil.

`connectivity.auth`

Informations d'authentification pour la connexion.

Cette propriété s'applique uniquement si `connectivity.protocol` est défini sur `ssh`.

`connectivity.auth.method`

Méthode d'authentification utilisée pour accéder à un appareil sur le protocole de connectivité donné.

Les valeurs prises en charge sont :

- `pki`
- `password`

`connectivity.auth.credentials`

Informations d'identification utilisées pour l'authentification.

`connectivity.auth.credentials.user`

Le nom d'utilisateur utilisé pour vous connecter à votre appareil.

`connectivity.auth.credentials.privKeyPath`

Le chemin complet vers la clé privée utilisée pour vous connecter à votre appareil.

Cette valeur s'applique uniquement si `connectivity.auth.method` est défini sur `pki`.

`devices.connectivity.auth.credentials.password`

Le mot de passe utilisé pour vous connecter à votre appareil.

Cette valeur s'applique uniquement si `connectivity.auth.method` est défini sur `password`.

> **Note**
>
> Spécifiez `privKeyPath` uniquement si `method` est défini sur `pki`.
> Spécifiez `password` uniquement si `method` est défini sur `password`.

Créez la suite d'exemples de tests

Le *`<device-tester-extract-location>`*`/samples/python` dossier contient des exemples de fichiers de configuration, du code source et le SDK du client IDT que vous pouvez combiner dans une suite de tests à l'aide des scripts de génération fournis. L'arborescence de répertoires suivante indique l'emplacement de ces exemples de fichiers :

```
<device-tester-extract-location>
### ...
### tests
### samples
#   ### ...
#   ### python
#       ### configuration
#       ### src
#       ### build-scripts
#           ### build.sh
#           ### build.ps1
### sdks
    ### ...
    ### python
        ### idt_client
```

Pour créer la suite de tests, exécutez les commandes suivantes sur votre ordinateur hôte :

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
```

Cela crée l'exemple de suite de tests dans le `IDTSampleSuitePython_1.0.0` dossier situé à l'intérieur du *<device-tester-extract-location>*`/tests` dossier. Passez en revue les fichiers du `IDTSampleSuitePython_1.0.0` dossier pour comprendre comment l'exemple de suite de tests est structuré et pour voir divers exemples d'exécutables de scénarios de test et de fichiers de configuration de test.

> **Note**
>
> L'exemple de suite de tests inclut le code source python. N'incluez pas d'informations sensibles dans le code de votre suite de tests.

Étape suivante : utilisez IDT pour exécuter l'exemple de suite de tests que vous avez créée.

Utiliser IDT pour exécuter l'exemple de suite de tests

Pour exécuter l'exemple de suite de tests, exécutez les commandes suivantes sur votre ordinateur hôte :

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id IDTSampleSuitePython
```

IDT exécute la suite d'exemples de tests et diffuse les résultats sur la console. Lorsque le test est terminé, les informations suivantes s'affichent :

```
========== 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
```

Résolution des erreurs

Utilisez les informations suivantes pour résoudre les problèmes rencontrés lors de l'exécution du didacticiel.

Le scénario de test ne s'exécute pas correctement

- Si le test échoue, IDT diffuse les journaux d'erreurs sur la console afin de vous aider à résoudre les problèmes liés à l'exécution du test. Assurez-vous de remplir toutes les conditions requises pour ce didacticiel.

Impossible de se connecter à l'appareil testé

Vérifiez les paramètres suivants :

- Votre `device.json` fichier contient l'adresse IP, le port et les informations d'authentification corrects.
- Vous pouvez vous connecter à votre appareil via SSH à partir de votre ordinateur hôte.

Tutoriel : Développement d'une suite de tests IDT simple

Une suite de tests combine les éléments suivants :

- Exécutable de test contenant la logique de test
- Fichiers de configuration décrivant la suite de tests

Ce didacticiel vous montre comment utiliser IDT pour FreeRTOS afin de développer une suite de tests Python contenant un seul cas de test. Bien que ce didacticiel utilise le protocole SSH, il est utile d'apprendre à l'utiliser AWS IoT Device Tester avec les appareils FreeRTOS.

Dans ce didacticiel, vous allez effectuer les étapes suivantes :

1. Création d'un répertoire de suites de tests
2. Création de fichiers de configuration
3. Création de l'exécutable du scénario de test
4. Exécutez la suite de tests

Suivez les étapes ci-dessous pour suivre un didacticiel sur le développement d'une suite de tests IDT simple.

Rubriques

- Configurez les prérequis pour une suite de tests IDT simple
- Création d'un répertoire de suites de tests
- Création de fichiers de configuration
- Obtenez le SDK du client IDT
- Création de l'exécutable du scénario de test
- Configurer les informations de l'appareil pour IDT

- Exécutez la suite de tests
- Résolution des erreurs
- Création de fichiers de configuration de la suite de tests IDT
- Configuration de l'orchestrateur de test IDT
- Configuration de la machine d'état IDT
- Créer un fichier exécutable pour le scénario de test IDT
- Utiliser le contexte IDT
- Configuration des paramètres pour les testeurs
- Déboguer et exécuter des suites de tests personnalisées
- Consulter les résultats et les journaux des tests IDT
- Soumettre les statistiques d'utilisation de l'IDT

Configurez les prérequis pour une suite de tests IDT simple

Pour suivre ce didacticiel, vous aurez besoin des éléments suivants :

- Exigences relatives à l'ordinateur hôte
  - Dernière version de AWS IoT Device Tester
  - Python 3.7 ou version ultérieure

    Pour vérifier la version de Python installée sur votre ordinateur, exécutez la commande suivante :

    ```
    python3 --version
    ```

    Sous Windows, si l'utilisation de cette commande renvoie une erreur, utilisez-la à la `python --version` place. Si le numéro de version renvoyé est 3.7 ou supérieur, exécutez la commande suivante dans un terminal Powershell pour la définir `python3` comme alias pour votre `python` commande.

    ```
    Set-Alias -Name "python3" -Value "python"
    ```

    Si aucune information de version n'est renvoyée ou si le numéro de version est inférieur à 3.7, suivez les instructions de la section Télécharger Python pour installer Python 3.7+. Pour plus d'informations, consultez la documentation Python.
  - urllib3

Pour vérifier qu'`urllib3`il est correctement installé, exécutez la commande suivante :

```
python3 -c 'import urllib3'
```

S'il n'`urllib3`est pas installé, exécutez la commande suivante pour l'installer :

```
python3 -m pip install urllib3
```

- Exigences relatives aux dispositifs
  - Appareil doté d'un système d'exploitation Linux et d'une connexion réseau au même réseau que votre ordinateur hôte.

    Nous vous recommandons d'utiliser un Raspberry Pi avec le système d'exploitation Raspberry Pi. Assurez-vous de configurer SSH sur votre Raspberry Pi pour vous y connecter à distance.

Création d'un répertoire de suites de tests

IDT sépare logiquement les cas de test en groupes de tests au sein de chaque suite de tests. Chaque cas de test doit faire partie d'un groupe de test. Pour ce didacticiel, créez un dossier appelé `MyTestSuite_1.0.0` et créez l'arborescence de répertoires suivante dans ce dossier :

```
MyTestSuite_1.0.0
### suite
    ### myTestGroup
        ### myTestCase
```

Création de fichiers de configuration

Votre suite de tests doit contenir les fichiers de configuration requis suivants :

Fichiers requis

`suite.json`

Contient des informations sur la suite de tests. Consultez Configurer suite.json.

`group.json`

Contient des informations sur un groupe de test. Vous devez créer un `group.json` fichier pour chaque groupe de test de votre suite de tests. Consultez Configurer group.json.

`test.json`

Contient des informations sur un scénario de test. Vous devez créer un `test.json` fichier pour chaque scénario de test de votre suite de tests. Consultez Configurer test.json.

1. Dans le `MyTestSuite_1.0.0/suite` dossier, créez un `suite.json` fichier avec la structure suivante :

```
{
    "id": "MyTestSuite_1.0.0",
    "title": "My Test Suite",
    "details": "This is my test suite.",
    "userDataRequired": false
}
```

2. Dans le `MyTestSuite_1.0.0/myTestGroup` dossier, créez un `group.json` fichier avec la structure suivante :

```
{
    "id": "MyTestGroup",
    "title": "My Test Group",
    "details": "This is my test group.",
    "optional": false
}
```

3. Dans le `MyTestSuite_1.0.0/myTestGroup/myTestCase` dossier, créez un `test.json` fichier avec la structure suivante :

```
{
    "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"
            ]
        }
    }
}
```

L'arborescence de votre `MyTestSuite_1.0.0` dossier doit désormais ressembler à ce qui suit :

```
MyTestSuite_1.0.0
### suite
    ### suite.json
    ### myTestGroup
        ### group.json
        ### myTestCase
            ### test.json
```

Obtenez le SDK du client IDT

Vous utilisez le SDK du client IDT pour permettre à IDT d'interagir avec l'appareil testé et de communiquer les résultats des tests. Pour ce didacticiel, vous allez utiliser la version Python du SDK.

Depuis le *<device-tester-extract-location>*`/sdks/python/` dossier, `idt_client` copiez-le dans votre `MyTestSuite_1.0.0/suite/myTestGroup/myTestCase` dossier.

Pour vérifier que le SDK a bien été copié, exécutez la commande suivante.

```
cd MyTestSuite_1.0.0/suite/myTestGroup/myTestCase
python3 -c 'import idt_client'
```

Création de l'exécutable du scénario de test

Les exécutables du scénario de test contiennent la logique de test que vous souhaitez exécuter. Une suite de tests peut contenir plusieurs exécutables de scénarios de test. Pour ce didacticiel, vous ne créerez qu'un seul exécutable de scénario de test.

1. Créez le fichier de suite de tests.

   Dans le `MyTestSuite_1.0.0/suite/myTestGroup/myTestCase` dossier, créez un `myTestCase.py` fichier avec le contenu suivant :

   ```
   from idt_client import *

   def main():
       # Use the client SDK to communicate with IDT
       client = Client()

   if __name__ == "__main__":
       main()
   ```

2. Utilisez les fonctions du SDK client pour ajouter la logique de test suivante à votre `myTestCase.py` fichier :

   a. Exécutez une commande SSH sur le périphérique testé.

   ```
   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. Envoyez le résultat du test à 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()
```

Configurer les informations de l'appareil pour IDT

Configurez les informations de votre appareil pour qu'IDT exécute le test. Vous devez mettre à jour le `device.json` modèle situé dans le *<device-tester-extract-location>*`/configs` dossier avec les informations suivantes.

```
[
  {
    "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>"
              }
            }
          }
        }
      ]
    }
]
```

Dans l'`devices`objet, fournissez les informations suivantes :

`id`

Identifiant unique défini par l'utilisateur pour votre appareil.

`connectivity.ip`

L'adresse IP de votre appareil.

`connectivity.port`

Facultatif. Le numéro de port à utiliser pour les connexions SSH à votre appareil.

`connectivity.auth`

Informations d'authentification pour la connexion.

Cette propriété s'applique uniquement si `connectivity.protocol` est défini sur `ssh`.

`connectivity.auth.method`

Méthode d'authentification utilisée pour accéder à un appareil sur le protocole de connectivité donné.

Les valeurs prises en charge sont :

- `pki`
- `password`

`connectivity.auth.credentials`

Informations d'identification utilisées pour l'authentification.

`connectivity.auth.credentials.user`

Le nom d'utilisateur utilisé pour vous connecter à votre appareil.

`connectivity.auth.credentials.privKeyPath`

Le chemin complet vers la clé privée utilisée pour vous connecter à votre appareil.

Cette valeur s'applique uniquement si `connectivity.auth.method` est défini sur `pki`.

`devices.connectivity.auth.credentials.password`

Le mot de passe utilisé pour vous connecter à votre appareil.

Cette valeur s'applique uniquement si `connectivity.auth.method` est défini sur `password`.

> **Note**
>
> Spécifiez `privKeyPath` uniquement si `method` est défini sur `pki`.
> Spécifiez `password` uniquement si `method` est défini sur `password`.

Exécutez la suite de tests

Après avoir créé votre suite de tests, vous devez vous assurer qu'elle fonctionne comme prévu. Pour ce faire, procédez comme suit pour exécuter la suite de tests avec votre pool d'appareils existant.

1. Copiez votre `MyTestSuite_1.0.0` dossier dans*<device-tester-extract-location>*/`tests`.
2. Exécutez les commandes suivantes :

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id MyTestSuite
```

IDT exécute votre suite de tests et diffuse les résultats sur la console. Lorsque le test est terminé, les informations suivantes s'affichent :

```
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
```

Résolution des erreurs

Utilisez les informations suivantes pour résoudre les problèmes rencontrés lors de l'exécution du didacticiel.

Le scénario de test ne s'exécute pas correctement

Si le test échoue, IDT diffuse les journaux d'erreurs sur la console afin de vous aider à résoudre les problèmes liés à l'exécution du test. Avant de consulter les journaux d'erreurs, vérifiez les points suivants :

- Le SDK du client IDT se trouve dans le bon dossier, comme décrit dans. Obtenez le SDK du client IDT
- Vous répondez à tous les prérequis pour ce didacticiel. Pour de plus amples informations, veuillez consulter Configurez les prérequis pour une suite de tests IDT simple.

Impossible de se connecter à l'appareil testé

Vérifiez les paramètres suivants :

- Votre `device.json` fichier contient l'adresse IP, le port et les informations d'authentification corrects.
- Vous pouvez vous connecter à votre appareil via SSH à partir de votre ordinateur hôte.

Création de fichiers de configuration de la suite de tests IDT

Cette section décrit les formats dans lesquels vous créez des fichiers de configuration que vous incluez lorsque vous écrivez une suite de tests personnalisée.

Fichiers de configuration requis

`suite.json`

Contient des informations sur la suite de tests. Consultez Configurer suite.json.

`group.json`

Contient des informations sur un groupe de test. Vous devez créer un `group.json` fichier pour chaque groupe de test de votre suite de tests. Consultez Configurer group.json.

`test.json`

Contient des informations sur un scénario de test. Vous devez créer un `test.json` fichier pour chaque scénario de test de votre suite de tests. Consultez Configurer test.json.

Fichiers de configuration facultatifs

`test_orchestrator.yaml` ou **`state_machine.json`**

Définit le mode d'exécution des tests lorsque IDT exécute la suite de tests. SEConfigurer test_orchestrator.yaml.

> **Note**
>
> À partir de IDT v4.5.2, vous utilisez le `test_orchestrator.yaml` fichier pour définir le flux de travail de test. Dans les versions précédentes d'IDT, vous utilisiez le `state_machine.json` fichier. Pour plus d'informations sur la machine à états, consultezConfiguration de la machine d'état IDT.

`userdata_schema.json`

Définit le schéma du `userdata.json`fichier que les testeurs peuvent inclure dans leur configuration de configuration. Le `userdata.json` fichier est utilisé pour toute information de configuration supplémentaire requise pour exécuter le test mais absente du `device.json` fichier. Consultez Configurer userdata_schema.json.

Les fichiers de configuration sont placés dans votre dossier*<custom-test-suite-folder>*, comme indiqué ici.

```
<custom-test-suite-folder>
### suite
    ### suite.json
    ### test_orchestrator.yaml
    ### userdata_schema.json
    ### <test-group-folder>
        ### group.json
        ### <test-case-folder>
            ### test.json
```

Configurer suite.json

Le `suite.json` fichier définit les variables d'environnement et détermine si les données utilisateur sont nécessaires pour exécuter la suite de tests. Utilisez le modèle suivant pour configurer votre *<custom-test-suite-folder>*`/suite/suite.json` fichier :

```
{
    "id": "<suite-name>_<suite-version>",
    "title": "<suite-title>",
    "details": "<suite-details>",
    "userDataRequired": true | false,
    "environmentVariables": [
        {
            "key": "<name>",
            "value": "<value>",
        },
        ...
        {
            "key": "<name>",
            "value": "<value>",
        }
```

```
    ]
}
```

Tous les champs qui contiennent des valeurs sont requis, comme indiqué ici :

`id`

Un identifiant unique défini par l'utilisateur pour la suite de tests. La valeur de `id` doit correspondre au nom du dossier de la suite de tests dans lequel se trouve le `suite.json` fichier. Le nom et la version de la suite doivent également répondre aux exigences suivantes :

- *`<suite-name>`*ne peut pas contenir de traits de soulignement.
- *`<suite-version>`*est noté*`x.x.x`*, où x est un nombre.

L'identifiant est indiqué dans les rapports de IDT-generated test.

`title`

Nom défini par l'utilisateur pour le produit ou la fonctionnalité testé par cette suite de tests. Le nom est affiché dans la CLI IDT pour les testeurs.

`details`

Brève description de l'objectif de la suite de tests.

`userDataRequired`

Définit si les testeurs doivent inclure des informations personnalisées dans un `userdata.json` fichier. Si vous définissez cette valeur sur`true`, vous devez également inclure le `userdata_schema.json`fichier dans le dossier de votre suite de tests.

`environmentVariables`

Facultatif. Un tableau de variables d'environnement à définir pour cette suite de tests.

`environmentVariables.key`

Le nom de la variable d'environnement.

`environmentVariables.value`

Valeur de la variable d'environnement.

Configurer group.json

Le `group.json` fichier définit si un groupe de test est obligatoire ou facultatif. Utilisez le modèle suivant pour configurer votre *<custom-test-suite-folder>*`/suite/`*<test-group>*`/group.json` fichier :

```
{
    "id": "<group-id>",
    "title": "<group-title>",
    "details": "<group-details>",
    "optional": true | false,
}
```

Tous les champs qui contiennent des valeurs sont requis, comme indiqué ici :

`id`

Un identifiant unique défini par l'utilisateur pour le groupe de test. La valeur de `id` doit correspondre au nom du dossier du groupe de test dans lequel se trouve le `group.json` fichier et ne doit pas comporter de trait de soulignement (_). L'identifiant est utilisé dans les rapports de IDT-generated test.

`title`

Nom descriptif du groupe de test. Le nom est affiché dans la CLI IDT pour les testeurs.

`details`

Brève description de l'objectif du groupe de test.

`optional`

Facultatif. Définissez sur `true` pour afficher ce groupe de test en tant que groupe facultatif une fois qu'IDT a terminé d'exécuter les tests requis. La valeur par défaut est `false`.

Configurer test.json

Le `test.json` fichier détermine les exécutables du scénario de test et les variables d'environnement utilisées par un scénario de test. Pour plus d'informations sur la création d'exécutables de scénarios de test, consultez. Créer un fichier exécutable pour le scénario de test IDT

Utilisez le modèle suivant pour configurer votre *<custom-test-suite-folder>*`/suite/`*<test-group>*`/`*<test-case>*`/test.json` fichier :

```
{
    "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>",
        }
```

```
    ]
}
```

Tous les champs qui contiennent des valeurs sont requis, comme indiqué ici :

`id`

Un identifiant unique défini par l'utilisateur pour le scénario de test. La valeur de `id` doit correspondre au nom du dossier de scénario de test dans lequel se trouve le `test.json` fichier et ne doit pas comporter de trait de soulignement (_). L'identifiant est utilisé dans les rapports de IDT-generated test.

`title`

Nom descriptif du scénario de test. Le nom est affiché dans la CLI IDT pour les testeurs.

`details`

Brève description de l'objectif du scénario de test.

`requireDUT`

Facultatif. Réglez sur `true` si un appareil est requis pour exécuter ce test, sinon sur`false`. La valeur par défaut est `true`. Les testeurs configureront les appareils qu'ils utiliseront pour exécuter le test dans leur `device.json` fichier.

`requiredResources`

Facultatif. Un tableau qui fournit des informations sur les périphériques de ressources nécessaires pour exécuter ce test.

`requiredResources.name`

Le nom unique à attribuer au périphérique de ressource lors de l'exécution de ce test.

`requiredResources.features`

Un ensemble de fonctionnalités de périphérique de ressources définies par l'utilisateur.

`requiredResources.features.name`

Le nom de la fonctionnalité. Fonctionnalité de l'appareil pour laquelle vous souhaitez utiliser cet appareil. Ce nom est comparé au nom de la fonctionnalité fourni par le testeur dans le `resource.json` fichier.

`requiredResources.features.version`

Facultatif. Version de la fonctionnalité. Cette valeur est comparée à la version de fonctionnalité fournie par le testeur dans le `resource.json` fichier. Si aucune version n'est fournie, la fonctionnalité n'est pas cochée. Si aucun numéro de version n'est requis pour la fonctionnalité, laissez ce champ vide.

`requiredResources.features.jobSlots`

Facultatif. Le nombre de tests simultanés que cette fonctionnalité peut prendre en charge. La valeur par défaut est 1. Si vous souhaitez qu'IDT utilise des appareils distincts pour des fonctionnalités individuelles, nous vous recommandons de définir cette valeur sur1.

`execution.timeout`

Durée (en millisecondes) pendant laquelle IDT attend la fin du test. Pour plus d'informations sur la définition de cette valeur, consultezCréer un fichier exécutable pour le scénario de test IDT.

`execution.os`

Les exécutables du scénario de test à exécuter sont basés sur le système d'exploitation de l'ordinateur hôte qui exécute IDT. Les valeurs prises en charge sont `linux`, `mac` et `win`.

`execution.os.cmd`

Le chemin d'accès au fichier exécutable du scénario de test que vous souhaitez exécuter pour le système d'exploitation spécifié. Cet emplacement doit se trouver dans le chemin du système.

`execution.os.args`

Facultatif. Les arguments à fournir pour exécuter l'exécutable du scénario de test.

`environmentVariables`

Facultatif. Un tableau de variables d'environnement défini pour ce cas de test.

`environmentVariables.key`

Le nom de la variable d'environnement.

`environmentVariables.value`

Valeur de la variable d'environnement.

> **Note**
>
> Si vous spécifiez la même variable d'environnement dans le `test.json` fichier et dans le `suite.json` fichier, la valeur du `test.json` fichier est prioritaire.

Configurer test_orchestrator.yaml

Un orchestrateur de tests est une construction qui contrôle le flux d'exécution de la suite de tests. Il détermine l'état de départ d'une suite de tests, gère les transitions d'état en fonction de règles définies par l'utilisateur et continue de passer par ces états jusqu'à ce qu'il atteigne l'état final.

Si votre suite de tests n'inclut pas d'orchestrateur de test défini par l'utilisateur, IDT générera un orchestrateur de tests pour vous.

L'orchestrateur de test par défaut exécute les fonctions suivantes :

- Permet aux testeurs de sélectionner et d'exécuter des groupes de tests spécifiques, au lieu de recourir à la suite de tests complète.
- Si aucun groupe de test spécifique n'est sélectionné, exécute chaque groupe de test de la suite de tests dans un ordre aléatoire.
- Génère des rapports et imprime un résumé de console présentant les résultats des tests pour chaque groupe de test et chaque cas de test.

Pour plus d'informations sur le fonctionnement de l'orchestrateur de test IDT, consultez. Configuration de l'orchestrateur de test IDT

Configurer userdata_schema.json

Le `userdata_schema.json` fichier détermine le schéma dans lequel les testeurs fournissent les données utilisateur. Les données utilisateur sont requises si votre suite de tests nécessite des informations qui ne figurent pas dans le `device.json` fichier. Par exemple, vos tests peuvent nécessiter des informations d'identification Wi-Fi réseau, des ports ouverts spécifiques ou des certificats qu'un utilisateur doit fournir. Ces informations peuvent être fournies à IDT sous la forme d'un paramètre d'entrée appelé`userdata`, dont la valeur est un `userdata.json` fichier, que les utilisateurs créent dans leur *`<device-tester-extract-location>`*`/config` dossier. Le format du `userdata.json` fichier est basé sur le `userdata_schema.json` fichier que vous incluez dans la suite de tests.

Pour indiquer que les testeurs doivent fournir un `userdata.json` fichier :

1. Dans le `suite.json` fichier, définissez `userDataRequired` sur`true`.
2. Dans votre*`<custom-test-suite-folder>`*, créez un `userdata_schema.json` fichier.
3. Modifiez le `userdata_schema.json` fichier pour créer un schéma JSON IETF Draft v4 valide.

Lorsque IDT exécute votre suite de tests, il lit automatiquement le schéma et l'utilise pour valider le `userdata.json` fichier fourni par le testeur. S'il est valide, le contenu du `userdata.json` fichier est disponible à la fois dans le contexte IDT et dans le contexte de l'orchestrateur de test.

Configuration de l'orchestrateur de test IDT

À partir de IDT v4.5.2, IDT inclut un nouveau composant d'orchestrateur de test. L'orchestrateur de tests est un composant IDT qui contrôle le flux d'exécution de la suite de tests et génère le rapport de test une fois que IDT a terminé d'exécuter tous les tests. L'orchestrateur de tests détermine la sélection des tests et l'ordre dans lequel les tests sont exécutés en fonction des règles définies par l'utilisateur.

Si votre suite de tests n'inclut pas d'orchestrateur de test défini par l'utilisateur, IDT générera un orchestrateur de tests pour vous.

L'orchestrateur de test par défaut exécute les fonctions suivantes :

- Permet aux testeurs de sélectionner et d'exécuter des groupes de tests spécifiques, au lieu de recourir à la suite de tests complète.
- Si aucun groupe de test spécifique n'est sélectionné, exécute chaque groupe de test de la suite de tests dans un ordre aléatoire.
- Génère des rapports et imprime un résumé de console présentant les résultats des tests pour chaque groupe de test et chaque cas de test.

L'orchestrateur de test remplace la machine d'état IDT. Nous vous recommandons vivement d'utiliser l'orchestrateur de tests pour développer vos suites de tests plutôt que la machine à états IDT. L'orchestrateur de test fournit les fonctionnalités améliorées suivantes :

- Utilise un format déclaratif par rapport au format impératif utilisé par la machine d'état IDT. Cela vous permet de spécifier les tests que vous souhaitez exécuter et le moment où vous souhaitez les exécuter.

- Gère la gestion de groupes spécifiques, la génération de rapports, la gestion des erreurs et le suivi des résultats afin que vous n'ayez pas à gérer manuellement ces actions.
- Utilise le format YAML, qui prend en charge les commentaires par défaut.
- Nécessite 80 % d'espace disque en moins que l'orchestrateur de test pour définir le même flux de travail.
- Ajoute une validation préalable pour vérifier que la définition de votre flux de travail ne contient pas d'identifiants de test incorrects ou de dépendances circulaires.

Tester le format de l'orchestrateur

Vous pouvez utiliser le modèle suivant pour configurer votre propre *custom-test-suite-folder*`/suite/test_orchestrator.yaml` fichier :

```
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
```

Tous les champs qui contiennent des valeurs sont requis, comme indiqué ici :

`Aliases`

Facultatif. User-defined chaînes qui correspondent à des expressions de contexte. Les alias vous permettent de générer des noms conviviaux pour identifier les expressions contextuelles dans la configuration de votre orchestrateur de test. Cela est particulièrement utile si vous créez des expressions contextuelles complexes ou des expressions que vous utilisez à plusieurs endroits.

Vous pouvez utiliser des expressions contextuelles pour stocker des requêtes contextuelles qui vous permettent d'accéder aux données d'autres configurations IDT. Pour de plus amples informations, veuillez consulter Accédez aux données dans le contexte.

Example

Exemple

```
Aliases:
    FizzChosen: "'{{$pool.features[?(@.name == 'Fizz')].value[0]}}' == 'yes'"
    BuzzChosen: "'{{$pool.features[?(@.name == 'Buzz')].value[0]}}' == 'yes'"
    FizzBuzzChosen: "'{{$aliases.FizzChosen}}' && '{{$aliases.BuzzChosen}}'"
```

`ConditionalTests`

Facultatif. Une liste de conditions et les cas de test correspondants qui sont exécutés lorsque chaque condition est satisfaite. Chaque condition peut comporter plusieurs scénarios de test ; toutefois, vous ne pouvez attribuer un scénario de test donné qu'à une seule condition.

Par défaut, IDT exécute tout scénario de test qui n'est pas affecté à une condition de cette liste. Si vous ne spécifiez pas cette section, IDT exécute tous les groupes de tests de la suite de tests.

Chaque élément de la `ConditionalTests` liste inclut les paramètres suivants :

`Condition`

Expression de contexte qui correspond à une valeur booléenne. Si la valeur évaluée est vraie, IDT exécute les scénarios de test spécifiés dans le `Tests` paramètre.

`Tests`

La liste des descripteurs de test.

Chaque descripteur de test utilise l'identifiant du groupe de test et un ou plusieurs identifiants de cas de test pour identifier les tests individuels à exécuter à partir d'un groupe de test spécifique. Le descripteur de test utilise le format suivant :

```
GroupId: group-id
CaseIds: [test-id, test-id] # optional
```

Example

Exemple

L'exemple suivant utilise des expressions contextuelles génériques que vous pouvez définir comme tellesAliases.

```
ConditionalTests:
    - Condition: "{{$aliases.Condition1}}"
      Tests:
          - GroupId: A
          - GroupId: B
    - Condition: "{{$aliases.Condition2}}"
      Tests:
          - GroupId: D
    - Condition: "{{$aliases.Condition1}} || {{$aliases.Condition2}}"
      Tests:
          - GroupId: C
```

Sur la base des conditions définies, IDT sélectionne les groupes de test comme suit :

- Si `Condition1` c'est vrai, IDT exécute les tests dans les groupes de tests A, B et C.
- Si `Condition2` c'est vrai, IDT exécute les tests dans les groupes de test C et D.

`Order`

Facultatif. Ordre dans lequel les tests doivent être exécutés. Vous spécifiez l'ordre des tests au niveau du groupe de test. Si vous ne spécifiez pas cette section, IDT exécute tous les groupes de test applicables dans un ordre aléatoire. La valeur de `Order` est une liste de listes de descripteurs de groupes. Tout groupe de test non répertorié peut être exécuté en `Order` parallèle avec n'importe quel autre groupe de test répertorié.

Chaque liste de descripteurs de groupe contient un ou plusieurs descripteurs de groupe et identifie l'ordre dans lequel exécuter les groupes spécifiés dans chaque descripteur. Vous pouvez utiliser les formats suivants pour définir des descripteurs de groupes individuels :

- *`group-id`*: l'ID de groupe d'un groupe de test existant.

- `[group-id, group-id]`—Liste des groupes de tests qui peuvent être exécutés dans n'importe quel ordre les uns par rapport aux autres.
- `"*"`—Wildcard. Cela équivaut à la liste de tous les groupes de test qui ne sont pas déjà spécifiés dans la liste actuelle des descripteurs de groupes.

La valeur pour `Order` doit également répondre aux exigences suivantes :

- Les identifiants de groupes de test que vous spécifiez dans un descripteur de groupe doivent exister dans votre suite de tests.
- Chaque liste de descripteurs de groupes doit inclure au moins un groupe de test.
- Chaque liste de descripteurs de groupes doit contenir des identifiants de groupe uniques. Vous ne pouvez pas répéter un identifiant de groupe de test dans des descripteurs de groupe individuels.
- Une liste de descripteurs de groupes peut comporter au maximum un descripteur de groupe générique. Le descripteur de groupe générique doit être le premier ou le dernier élément de la liste.

Example

Exemple

Pour une suite de tests contenant les groupes de tests A, B, C, D et E, la liste d'exemples suivante montre différentes manières de spécifier qu'IDT doit d'abord exécuter le groupe de test A, puis exécuter le groupe de test B, puis exécuter les groupes de test C, D et E dans n'importe quel ordre.

- ```
  Order:
      - - A
        - B
        - [C, D, E]
  ```
- ```
  Order:
      - - A
        - B
        - "*"
  ```
- ```
  Order:
      - - A
        - B
  ```
  ```

```
    - - B
      - C

    - - B
      - D

    - - B
      - E
```

`Features`

Facultatif. Liste des fonctionnalités du produit que vous souhaitez qu'IDT ajoute au `awsiotdevicetester_report.xml` fichier. Si vous ne spécifiez pas cette section, IDT n'ajoutera aucune fonctionnalité du produit au rapport.

Les fonctionnalités d'un produit sont des informations définies par l'utilisateur concernant des critères spécifiques auxquels un appareil peut répondre. Par exemple, la fonctionnalité du produit MQTT peut indiquer que l'appareil publie correctement les messages MQTT. Dans`awsiotdevicetester_report.xml`, les fonctionnalités du produit sont définies sous la forme de `supportednot-supported`, ou d'une valeur personnalisée définie par l'utilisateur, selon que les tests spécifiés ont été réussis ou non.

Chaque élément de la `Features` liste comprend les paramètres suivants :

`Name`

Le nom de la fonctionnalité.

`Value`

Facultatif. La valeur personnalisée que vous souhaitez utiliser dans le rapport à la place de`supported`. Si cette valeur n'est pas spécifiée, l'IDT basé sur le paramètre définit la valeur de la fonction sur `supported` ou en `not-supported` fonction des résultats des tests. Si vous testez la même fonctionnalité dans des conditions différentes, vous pouvez utiliser une valeur personnalisée pour chaque instance de cette fonctionnalité dans la `Features` liste, et IDT concatène les valeurs des entités pour les conditions prises en charge. Pour plus d'informations, veuillez consulter la rubrique

`Condition`

Expression de contexte qui correspond à une valeur booléenne. Si la valeur évaluée est vraie, IDT ajoute la fonctionnalité au rapport de test une fois l'exécution de la suite de tests terminée. Si la valeur évaluée est fausse, le test n'est pas inclus dans le rapport.

`Tests`

Facultatif. La liste des descripteurs de test. Tous les tests spécifiés dans cette liste doivent réussir pour que la fonctionnalité soit prise en charge.

Chaque descripteur de test de cette liste utilise l'identifiant du groupe de test et un ou plusieurs identifiants de cas de test pour identifier les tests individuels à exécuter à partir d'un groupe de test spécifique. Le descripteur de test utilise le format suivant :

```
GroupId: group-id
CaseIds: [test-id, test-id] # optional
```

Vous devez spécifier l'une `Tests` ou l'autre des fonctionnalités de la `Features` liste ou `OneOfTests` pour chacune d'entre elles.

`OneOfTests`

Facultatif. La liste des descripteurs de test. Au moins l'un des tests spécifiés dans cette liste doit réussir pour que la fonctionnalité soit prise en charge.

Chaque descripteur de test de cette liste utilise l'identifiant du groupe de test et un ou plusieurs identifiants de cas de test pour identifier les tests individuels à exécuter à partir d'un groupe de test spécifique. Le descripteur de test utilise le format suivant :

```
GroupId: group-id
CaseIds: [test-id, test-id] # optional
```

Vous devez spécifier l'une `Tests` ou l'autre des fonctionnalités de la `Features` liste ou `OneOfTests` pour chacune d'entre elles.

`IsRequired`

La valeur booléenne qui définit si la fonctionnalité est requise dans le rapport de test. La valeur par défaut est `false`.

Contexte de l'orchestrateur de tests

Le contexte de l'orchestrateur de test est un document JSON en lecture seule qui contient les données mises à la disposition de l'orchestrateur de test pendant l'exécution. Le contexte de l'orchestrateur de test n'est accessible que depuis l'orchestrateur de test et contient des informations

qui déterminent le flux de test. Par exemple, vous pouvez utiliser les informations configurées par les testeurs dans le `userdata.json` fichier pour déterminer si un test spécifique est requis pour être exécuté.

Le contexte de l'orchestrateur de test utilise le format suivant :

```
{
    "pool": {
        <device-json-pool-element>
    },
    "userData": {
        <userdata-json-content>
    },
    "config": {
        <config-json-content>
    }
}
```

`pool`

Informations sur le pool de périphériques sélectionné pour le test. Pour un pool de périphériques sélectionné, ces informations sont extraites de l'élément de tableau de pool de périphériques de niveau supérieur correspondant défini dans le `device.json` fichier.

`userData`

Informations contenues dans le `userdata.json` fichier.

`config`

Informations contenues dans le `config.json` fichier.

Vous pouvez interroger le contexte à l'aide de la notation JSONPath. La syntaxe des requêtes JSONPath dans les définitions d'état est. `{{query}}` Lorsque vous accédez aux données depuis le contexte de l'orchestrateur de test, assurez-vous que chaque valeur correspond à une chaîne, un nombre ou un booléen.

Pour plus d'informations sur l'utilisation de la notation JSONPath pour accéder aux données depuis le contexte, consultez. Utiliser le contexte IDT

Configuration de la machine d'état IDT

> **Important**
>
> À partir de IDT v4.5.2, cette machine à états est obsolète. Nous vous recommandons vivement d'utiliser le nouvel orchestrateur de test. Pour de plus amples informations, veuillez consulter Configuration de l'orchestrateur de test IDT.

Une machine à états est une construction qui contrôle le flux d'exécution de la suite de tests. Il détermine l'état de départ d'une suite de tests, gère les transitions d'état en fonction de règles définies par l'utilisateur et continue de passer par ces états jusqu'à ce qu'il atteigne l'état final.

Si votre suite de tests n'inclut pas de machine à états définie par l'utilisateur, IDT générera une machine à états pour vous. La machine à états par défaut exécute les fonctions suivantes :

- Permet aux testeurs de sélectionner et d'exécuter des groupes de tests spécifiques, au lieu de recourir à la suite de tests complète.
- Si aucun groupe de test spécifique n'est sélectionné, exécute chaque groupe de test de la suite de tests dans un ordre aléatoire.
- Génère des rapports et imprime un résumé de console présentant les résultats des tests pour chaque groupe de test et chaque cas de test.

La machine d'état d'une suite de tests IDT doit répondre aux critères suivants :

- Chaque état correspond à une action que doit effectuer IDT, par exemple exécuter un groupe de test ou produire un fichier de rapport.
- Le passage à un état exécute l'action associée à cet état.
- Chaque état définit la règle de transition pour l'état suivant.
- L'état final doit être l'un `Succeed` ou l'autre`Fail`.

Format de la machine à états

Vous pouvez utiliser le modèle suivant pour configurer votre propre *<custom-test-suite-folder>*`/suite/state_machine.json` fichier :

```
{
```

```
  "Comment": "<description>",
  "StartAt": "<state-name>",
  "States": {
    "<state-name>": {
      "Type": "<state-type>",
      // Additional state configuration
    }

    // Required states
    "Succeed": {
      "Type": "Succeed"
    },
    "Fail": {
      "Type": "Fail"
    }
  }
}
```

Tous les champs qui contiennent des valeurs sont requis, comme indiqué ici :

`Comment`

Description de la machine à états.

`StartAt`

Nom de l'état dans lequel IDT commence à exécuter la suite de tests. La valeur de `StartAt` doit être définie sur l'un des états répertoriés dans l'`States`objet.

`States`

Objet qui fait correspondre les noms d'état définis par l'utilisateur à des états IDT valides. Chaque État. *`state-name`*l'objet contient la définition d'un état valide mappé à. *`state-name`*

L'`States`objet doit inclure les `Fail` états `Succeed` et. Pour plus d'informations sur les états valides, consultezÉtats valides et définitions d'états.

États valides et définitions d'états

Cette section décrit les définitions d'état de tous les états valides qui peuvent être utilisés dans la machine d'état IDT. Certains des états suivants prennent en charge les configurations au niveau du scénario de test. Toutefois, nous vous recommandons de configurer les règles de transition d'état au niveau du groupe de test plutôt qu'au niveau du cas de test, sauf si cela est absolument nécessaire.

Définitions des États

- RunTask
- Choice
- Parallèle
- AddProductFeatures
- Rapport
- LogMessage
- SelectGroup
- Fail
- Succeed

RunTask

L'`RunTask`État exécute des scénarios de test à partir d'un groupe de tests défini dans la suite de tests.

```
{
    "Type": "RunTask",
    "Next": "<state-name>",
    "TestGroup": "<group-id>",
    "TestCases": [
        "<test-id>"
    ],
    "ResultVar": "<result-name>"
}
```

Tous les champs qui contiennent des valeurs sont requis, comme indiqué ici :

`Next`

Nom de l'état vers lequel passer après l'exécution des actions dans l'état actuel.

`TestGroup`

Facultatif. ID du groupe de test à exécuter. Si cette valeur n'est pas spécifiée, IDT exécute le groupe de test sélectionné par le testeur.

`TestCases`

Facultatif. Un tableau d'identifiants de cas de test provenant du groupe spécifié dans`TestGroup`. Sur la base des valeurs de `TestGroup` et`TestCases`, IDT détermine le comportement d'exécution du test comme suit :

- Lorsque `TestGroup` les deux `TestCases` sont spécifiés, IDT exécute les cas de test spécifiés à partir du groupe de test.
- Lorsqu'ils `TestCases` sont spécifiés mais `TestGroup` non spécifiés, IDT exécute les cas de test spécifiés.
- Lorsqu'il `TestGroup` est spécifié, mais `TestCases` non spécifié, IDT exécute tous les cas de test au sein du groupe de test spécifié.
- Lorsque ni l'`TestGroup`un ni l'autre n'`TestCases`est spécifié, IDT exécute tous les cas de test à partir du groupe de test sélectionné par le lanceur de tests dans la CLI IDT. Pour permettre la sélection de groupes pour les testeurs, vous devez inclure à la fois les `Choice` états `RunTask` et les états dans votre `statemachine.json` fichier. Pour un exemple de la façon dont cela fonctionne, voir Exemple de machine à états : exécuter des groupes de test sélectionnés par l'utilisateur.

  Pour plus d'informations sur l'activation des commandes IDT CLI pour les testeurs, consultezthe section called “Activer les commandes IDT CLI”.

`ResultVar`

Nom de la variable de contexte à définir avec les résultats du test. Ne spécifiez pas cette valeur si vous n'avez pas indiqué de valeur pour`TestGroup`. IDT définit la valeur de la variable que vous définissez dans `true` ou `ResultVar` en `false` fonction des éléments suivants :

- Si le nom de la variable est au format*`text`*`_`*`text`*`_passed`, la valeur est définie de manière à indiquer si tous les tests du premier groupe de tests ont été réussis ou ignorés.
- Dans tous les autres cas, la valeur est définie selon que tous les tests de tous les groupes de tests ont été réussis ou ignorés.

Généralement, vous utilisez `RunTask` state pour spécifier un identifiant de groupe de test sans spécifier d'identifiant de cas de test individuel, de sorte qu'IDT exécute tous les cas de test dans le groupe de test spécifié. Tous les cas de test exécutés par cet état sont exécutés en parallèle, dans un ordre aléatoire. Toutefois, si tous les scénarios de test nécessitent l'exécution d'un périphérique et qu'un seul périphérique est disponible, les scénarios de test seront exécutés de manière séquentielle.

Gestion des erreurs

Si l'un des groupes de test ou des ID de cas de test spécifiés n'est pas valide, cet état génère l'erreur d'`RunTaskError`exécution. Si l'état rencontre une erreur d'exécution, il définit également la `hasExecutionError` variable dans le contexte de la machine à états sur`true`.

Choice

L'`Choice`état vous permet de définir dynamiquement l'état suivant vers lequel passer en fonction des conditions définies par l'utilisateur.

```
{
    "Type": "Choice",
    "Default": "<state-name>",
    "FallthroughOnError": true | false,
    "Choices": [
        {
            "Expression": "<expression>",
            "Next": "<state-name>"
        }
    ]
}
```

Tous les champs qui contiennent des valeurs sont requis, comme indiqué ici :

`Default`

État par défaut vers lequel passer si aucune des expressions définies dans ne `Choices` peut être évaluée`true`.

`FallthroughOnError`

Facultatif. Spécifie le comportement lorsque l'état rencontre une erreur lors de l'évaluation des expressions. Définissez cette `true` valeur si vous souhaitez ignorer une expression si l'évaluation entraîne une erreur. Si aucune expression ne correspond, la machine à états passe à l'`Default`état. Si la `FallthroughOnError` valeur n'est pas spécifiée, elle est définie par défaut sur`false`.

`Choices`

Tableau d'expressions et d'états permettant de déterminer l'état vers lequel passer après avoir exécuté les actions dans l'état actuel.

`Choices.Expression`

Chaîne d'expression dont l'évaluation correspond à une valeur booléenne. Si l'expression est évaluée à`true`, la machine à états passe à l'état défini dans`Choices.Next`. Les chaînes d'expression récupèrent les valeurs du contexte de la machine à états, puis exécutent des opérations sur celles-ci pour obtenir une valeur booléenne. Pour plus d'informations sur l'accès au contexte de la machine à états, consultezContexte de la machine à états.

`Choices.Next`

Nom de l'état vers lequel passer si l'expression définie dans `Choices.Expression` correspond à`true`.

Gestion des erreurs

L'`Choice`état peut nécessiter la gestion des erreurs dans les cas suivants :

- Certaines variables des expressions de choix n'existent pas dans le contexte de la machine à états.
- Le résultat d'une expression n'est pas une valeur booléenne.
- Le résultat d'une recherche JSON n'est pas une chaîne, un nombre ou un booléen.

Vous ne pouvez pas utiliser un `Catch` bloc pour gérer les erreurs dans cet état. Si vous souhaitez arrêter l'exécution de la machine d'état lorsqu'elle rencontre une erreur, vous devez `FallthroughOnError` définir sur`false`. Toutefois, nous vous recommandons de configurer l'une des `FallthroughOnError` options suivantes et`true`, en fonction de votre cas d'utilisation, d'effectuer l'une des opérations suivantes :

- Si une variable à laquelle vous accédez ne devrait pas exister dans certains cas, utilisez la valeur de `Default` et des `Choices` blocs supplémentaires pour spécifier l'état suivant.
- Si une variable à laquelle vous accédez doit toujours exister, définissez son `Default` état sur`Fail`.

Parallèle

L'`Parallel`état vous permet de définir et d'exécuter de nouvelles machines à états en parallèle les unes avec les autres.

```
{
    "Type": "Parallel",
    "Next": "<state-name>",
    "Branches": [
        <state-machine-definition>
    ]
}
```

Tous les champs qui contiennent des valeurs sont requis, comme indiqué ici :

`Next`

Nom de l'état vers lequel passer après l'exécution des actions dans l'état actuel.

`Branches`

Un tableau de définitions de machines à états à exécuter. Chaque définition de machine à états doit contenir ses propres `Fail` états `StartAtSucceed`, et. Les définitions de machines à états de ce tableau ne peuvent pas faire référence à des états en dehors de leur propre définition.

> **Note**
>
> Comme chaque machine d'état de branche partage le même contexte de machine d'état, le fait de définir des variables dans une branche puis de lire ces variables dans une autre branche peut entraîner un comportement inattendu.

L'`Parallel`état ne passe à l'état suivant qu'après avoir exécuté toutes les machines d'état de branche. Chaque état nécessitant un appareil attendra de fonctionner jusqu'à ce que l'appareil soit disponible. Si plusieurs appareils sont disponibles, cet état exécute des scénarios de test à partir de plusieurs groupes en parallèle. Si un nombre suffisant de périphériques ne sont pas disponibles, les scénarios de test s'exécuteront de manière séquentielle. Comme les scénarios de test sont exécutés dans un ordre aléatoire lorsqu'ils sont exécutés en parallèle, différents appareils peuvent être utilisés pour exécuter des tests à partir du même groupe de test.

Gestion des erreurs

Assurez-vous que la machine d'état de branche et la machine d'état parent passent à l'`Fail`état pour gérer les erreurs d'exécution.

Comme les machines d'état de branche ne transmettent pas d'erreurs d'exécution à la machine d'état parent, vous ne pouvez pas utiliser de `Catch` bloc pour gérer les erreurs d'exécution dans les machines d'état de branche. Utilisez plutôt la `hasExecutionErrors` valeur dans le contexte de la machine à états partagés. Pour un exemple de la façon dont cela fonctionne, voirExemple de machine à états : exécuter deux groupes de tests en parallèle.

AddProductFeatures

L'`AddProductFeatures`état vous permet d'ajouter des fonctionnalités du produit au `awsiotdevicetester_report.xml` fichier généré par IDT.

Les fonctionnalités d'un produit sont des informations définies par l'utilisateur concernant des critères spécifiques auxquels un appareil peut répondre. Par exemple, la fonctionnalité `MQTT` du produit peut indiquer que l'appareil publie correctement les messages MQTT. Dans le rapport, les fonctionnalités du produit sont définies sous `supported` la forme d'une valeur ou d'une valeur personnalisée, en fonction de la réussite des tests spécifiés. `not-supported`

> **Note**
>
> L'`AddProductFeatures`État ne produit pas de rapports par lui-même. Cet état doit passer à l'`Report`état permettant de générer des rapports.

```
{
    "Type": "Parallel",
    "Next": "<state-name>",
    "Features": [
        {
            "Feature": "<feature-name>",
            "Groups": [
                "<group-id>"
            ],
            "OneOfGroups": [
                "<group-id>"
            ],
            "TestCases": [
                "<test-id>"
            ],
            "IsRequired": true | false,
```

```
            "ExecutionMethods": [
                "<execution-method>"
            ]
        }
    ]
}
```

Tous les champs qui contiennent des valeurs sont requis, comme indiqué ici :

`Next`

Nom de l'état vers lequel passer après l'exécution des actions dans l'état actuel.

`Features`

Un ensemble de fonctionnalités du produit à afficher dans le `awsiotdevicetester_report.xml` fichier.

`Feature`

Le nom de la fonctionnalité

`FeatureValue`

Facultatif. La valeur personnalisée à utiliser dans le rapport à la place de`supported`. Si cette valeur n'est pas spécifiée, la valeur de la fonction est définie sur `supported` ou en fonction des résultats des tests`not-supported`.

Si vous utilisez une valeur personnalisée pour`FeatureValue`, vous pouvez tester la même fonctionnalité dans des conditions différentes, et IDT concatène les valeurs des caractéristiques pour les conditions prises en charge. Par exemple, l'extrait suivant montre l'`MyFeature`entité avec deux valeurs de fonction distinctes :

```
...
{
    "Feature": "MyFeature",
    "FeatureValue": "first-feature-supported",
    "Groups": ["first-feature-group"]
},
{
    "Feature": "MyFeature",
    "FeatureValue": "second-feature-supported",
    "Groups": ["second-feature-group"]
```

```
},
...
```

Si les deux groupes de test réussissent, la valeur de la fonctionnalité est définie sur`first-feature-supported, second-feature-supported`.

`Groups`

Facultatif. Un tableau d'identifiants de groupes de test. Tous les tests au sein de chaque groupe de test spécifié doivent réussir pour que la fonctionnalité soit prise en charge.

`OneOfGroups`

Facultatif. Un tableau d'identifiants de groupes de test. Tous les tests au sein d'au moins un des groupes de tests spécifiés doivent réussir pour que la fonctionnalité soit prise en charge.

`TestCases`

Facultatif. Un tableau d'identifiants de cas de test. Si vous spécifiez cette valeur, les règles suivantes s'appliquent :

- Tous les scénarios de test spécifiés doivent être réussis pour que la fonctionnalité soit prise en charge.
- `Groups`ne doit contenir qu'un seul identifiant de groupe de test.
- `OneOfGroups`ne doit pas être spécifiée.

`IsRequired`

Facultatif. Définissez sur `false` pour marquer cette fonctionnalité en tant que fonctionnalité facultative dans le rapport. La valeur par défaut est `true`.

`ExecutionMethods`

Facultatif. Tableau de méthodes d'exécution correspondant à la `protocol` valeur spécifiée dans le `device.json` fichier. Si cette valeur est spécifiée, les testeurs doivent spécifier une `protocol` valeur correspondant à l'une des valeurs de ce tableau pour inclure la fonctionnalité dans le rapport. Si cette valeur n'est pas spécifiée, la fonctionnalité sera toujours incluse dans le rapport.

Pour utiliser l'`AddProductFeatures`état, vous devez définir la valeur de `ResultVar` in the `RunTask` state sur l'une des valeurs suivantes :

- Si vous avez spécifié des identifiants de cas de test individuels, définissez le `ResultVar` paramètre sur*group-id_test-id*`_passed`.
- Si vous n'avez pas spécifié d'ID de cas de test individuels, définissez cette `ResultVar` option sur*group-id*`_passed`.

L'`AddProductFeatures`État vérifie les résultats des tests de la manière suivante :

- Si vous n'avez spécifié aucun ID de cas de test, le résultat de chaque groupe de test est déterminé à partir de la valeur de la *group-id*`_passed` variable dans le contexte de la machine à états.
- Si vous avez spécifié des ID de cas de test, le résultat de chacun des tests est déterminé à partir de la valeur de la *group-id_test-id*`_passed` variable dans le contexte de la machine à états.

Gestion des erreurs

Si un identifiant de groupe fourni dans cet état n'est pas un identifiant de groupe valide, cet état entraîne une erreur d'`AddProductFeaturesError`exécution. Si l'état rencontre une erreur d'exécution, il définit également la `hasExecutionErrors` variable dans le contexte de la machine à états sur`true`.

Rapport

L'`Report`état génère les `awsiotdevicetester_report.xml` fichiers *suite-name*`_Report.xml` et. Cet état diffuse également le rapport vers la console.

```
{
    "Type": "Report",
    "Next": "<state-name>"
}
```

Tous les champs qui contiennent des valeurs sont requis, comme indiqué ici :

`Next`

Nom de l'état vers lequel passer après l'exécution des actions dans l'état actuel.

Vous devez toujours passer à l'`Report`état vers la fin du flux d'exécution des tests afin que les testeurs puissent voir les résultats des tests. Généralement, l'état suivant après cet état est`Succeed`.

Gestion des erreurs

Si cet état rencontre des problèmes lors de la génération des rapports, il émet l'erreur d'`ReportError`exécution.

LogMessage

L'`LogMessage`état génère le `test_manager.log` fichier et transmet le message du journal à la console.

```
{
    "Type": "LogMessage",
    "Next": "<state-name>"
    "Level": "info | warn | error"
    "Message": "<message>"
}
```

Tous les champs qui contiennent des valeurs sont requis, comme indiqué ici :

`Next`

Nom de l'état vers lequel passer après l'exécution des actions dans l'état actuel.

`Level`

Le niveau d'erreur auquel le message de journal doit être créé. Si vous spécifiez un niveau non valide, cet état génère un message d'erreur et le supprime.

`Message`

Le message à enregistrer.

SelectGroup

L'`SelectGroup`état met à jour le contexte de la machine à états pour indiquer quels groupes sont sélectionnés. Les valeurs définies par cet état sont utilisées par tous `Choice` les états suivants.

```
{
    "Type": "SelectGroup",
    "Next": "<state-name>"
    "TestGroups": [
        <group-id>"
    ]
}
```

Tous les champs qui contiennent des valeurs sont requis, comme indiqué ici :

`Next`

Nom de l'état vers lequel passer après l'exécution des actions dans l'état actuel.

`TestGroups`

Un ensemble de groupes de test qui seront marqués comme sélectionnés. Pour chaque ID de groupe de test de ce tableau, la *`group-id`*`_selected` variable est définie sur `true` dans le contexte. Assurez-vous de fournir des ID de groupe de test valides, car IDT ne valide pas l'existence des groupes spécifiés.

Fail

L'`Fail`état indique que la machine à états ne s'est pas exécutée correctement. Il s'agit d'un état final pour la machine à états, et chaque définition de machine à états doit inclure cet état.

```
{
    "Type": "Fail"
}
```

Succeed

L'`Succeed`état indique que la machine à états s'est exécutée correctement. Il s'agit d'un état final pour la machine à états, et chaque définition de machine à états doit inclure cet état.

```
{
    "Type": "Succeed"
}
```

Contexte de la machine à états

Le contexte de la machine à états est un document JSON en lecture seule qui contient les données mises à la disposition de la machine à états pendant l'exécution. Le contexte de la machine à états n'est accessible qu'à partir de la machine à états et contient des informations qui déterminent le flux de test. Par exemple, vous pouvez utiliser les informations configurées par les testeurs dans le `userdata.json` fichier pour déterminer si un test spécifique est requis pour être exécuté.

Le contexte de la machine à états utilise le format suivant :

```
{
    "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`

Informations sur le pool de périphériques sélectionné pour le test. Pour un pool de périphériques sélectionné, ces informations sont extraites de l'élément de tableau de pool de périphériques de niveau supérieur correspondant défini dans le `device.json` fichier.

`userData`

Informations contenues dans le `userdata.json` fichier.

`config`

Informations épinglez le `config.json` fichier.

`suiteFailed`

La valeur est définie sur le `false` démarrage de la machine à états. Si un groupe de test échoue dans un `RunTask` état, cette valeur est définie sur la durée restante `true` de l'exécution de la machine à états.

`specificTestGroups`

Si le testeur sélectionne des groupes de tests spécifiques à exécuter au lieu de l'ensemble de la suite de tests, cette clé est créée et contient la liste des identifiants de groupes de test spécifiques.

`specificTestCases`

Si le lanceur de tests sélectionne des cas de test spécifiques à exécuter au lieu de la suite de tests complète, cette clé est créée et contient la liste des identifiants de cas de test spécifiques.

`hasExecutionErrors`

Ne se ferme pas au démarrage de la machine à états. Si un état rencontre une erreur d'exécution, cette variable est créée et définie `true` pour la durée restante de l'exécution de la machine à états.

Vous pouvez interroger le contexte à l'aide de la notation JSONPath. La syntaxe des requêtes JSONPath dans les définitions d'état est. `{{$.query}}` Vous pouvez utiliser des requêtes JSONPath comme chaînes d'espace réservé dans certains états. IDT remplace les chaînes d'espace réservé par la valeur de la requête JSONPath évaluée à partir du contexte. Vous pouvez utiliser des espaces réservés pour les valeurs suivantes :

- La `TestCases` valeur en `RunTask` états.
- `Choice`État de `Expression` la valeur.

Lorsque vous accédez aux données depuis le contexte de la machine à états, assurez-vous que les conditions suivantes sont remplies :

- Vos chemins JSON doivent commencer par `$.`
- Chaque valeur doit être évaluée sous la forme d'une chaîne, d'un nombre ou d'un booléen.

Pour plus d'informations sur l'utilisation de la notation JSONPath pour accéder aux données depuis le contexte, consultez. Utiliser le contexte IDT

Erreurs d'exécution

Les erreurs d'exécution sont des erreurs dans la définition de la machine à états que la machine à états rencontre lors de l'exécution d'un état. IDT enregistre les informations relatives à chaque erreur dans le `test_manager.log` fichier et transmet le message de journal à la console.

Vous pouvez utiliser les méthodes suivantes pour gérer les erreurs d'exécution :

- Ajoutez un `Catch`bloc dans la définition de l'état.
- Vérifiez la valeur de la `hasExecutionErrors`valeur dans le contexte de la machine à états.

attraper

Pour l'utiliserCatch, ajoutez ce qui suit à la définition de votre état :

```
"Catch": [
    {
        "ErrorEquals": [
            "<error-type>"
        ]
        "Next": "<state-name>"
    }
]
```

Tous les champs qui contiennent des valeurs sont requis, comme indiqué ici :

`Catch.ErrorEquals`

Tableau des types d'erreurs à détecter. Si une erreur d'exécution correspond à l'une des valeurs spécifiées, la machine à états passe à l'état spécifié dans`Catch.Next`. Consultez la définition de chaque état pour obtenir des informations sur le type d'erreur qu'il produit.

`Catch.Next`

État suivant vers lequel passer si l'état actuel rencontre une erreur d'exécution correspondant à l'une des valeurs spécifiées dans`Catch.ErrorEquals`.

Les blocs de capture sont gérés de manière séquentielle jusqu'à ce qu'un d'entre eux corresponde. Si aucune erreur ne correspond à celles répertoriées dans les blocs Catch, les machines d'état continuent de s'exécuter. Les erreurs d'exécution étant le résultat de définitions d'état incorrectes, nous vous recommandons de passer à l'état Fail lorsqu'un état rencontre une erreur d'exécution.

a ExecutionError

Lorsque certains états rencontrent des erreurs d'exécution, en plus d'émettre l'erreur, ils définissent également la `hasExecutionError` valeur sur `true` dans le contexte de la machine à états. Vous pouvez utiliser cette valeur pour détecter lorsqu'une erreur se produit, puis utiliser un `Choice` état pour faire passer la machine à états à l'`Fail`état.

Cette méthode présente les caractéristiques suivantes.

- La machine à états ne démarre avec aucune valeur assignée à`hasExecutionError`, et cette valeur n'est pas disponible tant qu'un état particulier ne la définit pas. Cela signifie que vous

devez définir explicitement la valeur `FallthroughOnError` to `false` pour les `Choice` états qui accèdent à cette valeur afin d'empêcher l'arrêt de la machine à états si aucune erreur d'exécution ne se produit.

- Une fois défini sur`true`, il n'`hasExecutionError`est jamais défini sur false ni supprimé du contexte. Cela signifie que cette valeur n'est utile que la première fois qu'elle est définie sur`true`, et pour tous les états suivants, elle ne fournit pas de valeur significative.
- La `hasExecutionError` valeur est partagée avec toutes les machines d'état de branche de l'`Parallel`état, ce qui peut entraîner des résultats inattendus en fonction de l'ordre dans lequel elle est consultée.

En raison de ces caractéristiques, nous vous déconseillons d'utiliser cette méthode si vous pouvez utiliser un bloc Catch à la place.

Exemples de machines à états

Cette section fournit des exemples de configurations de machines à états.

Exemples

- Exemple de machine à états : exécuter un seul groupe de test
- Exemple de machine à états : exécuter des groupes de test sélectionnés par l'utilisateur
- Exemple de machine à états : exécuter un seul groupe de test avec les fonctionnalités du produit
- Exemple de machine à états : exécuter deux groupes de tests en parallèle

Exemple de machine à états : exécuter un seul groupe de test

Cette machine à états :

- Exécute le groupe de test avec un identifiant`GroupA`, qui doit être présent dans la suite dans un `group.json` fichier.
- Vérifie l'absence d'erreurs d'exécution et effectue la transition vers le `Fail` cas échéant.
- Génère un rapport et passe à `Succeed` s'il n'y a pas d'erreur, et `Fail` sinon.

```
{
    "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"
        }
    }
}
```

Exemple de machine à états : exécuter des groupes de test sélectionnés par l'utilisateur

Cette machine à états :

- Vérifie si le testeur a sélectionné des groupes de test spécifiques. La machine à états ne vérifie pas les cas de test spécifiques car les testeurs ne peuvent pas sélectionner de cas de test sans sélectionner également un groupe de test.
- Si des groupes de test sont sélectionnés :

- Exécute les scénarios de test au sein des groupes de test sélectionnés. Pour ce faire, la machine d'état ne spécifie pas explicitement de groupes de test ou de cas de test dans l'`RunTask`état.
- Génère un rapport après avoir exécuté tous les tests et sorties.

- Si les groupes de test ne sont pas sélectionnés :
  - Exécute des tests dans le groupe de test`GroupA`.
  - Génère des rapports et des sorties.

```
{
    "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"
        }
    }
}
```

Exemple de machine à états : exécuter un seul groupe de test avec les fonctionnalités du produit

Cette machine à états :

- Exécute le groupe de test`GroupA`.
- Vérifie l'absence d'erreurs d'exécution et effectue la transition vers le `Fail` cas échéant.
- Ajoute la `FeatureThatDependsOnGroupA` fonctionnalité au `awsiotdevicetester_report.xml` fichier :
  - En `GroupA` cas de réussite, la fonctionnalité est réglée sur`supported`.
  - La fonctionnalité n'est pas marquée comme facultative dans le rapport.
- Génère un rapport et passe à `Succeed` s'il n'y a pas d'erreur, et `Fail` sinon

```
{
    "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"
        }
    }
}
```

Exemple de machine à états : exécuter deux groupes de tests en parallèle

Cette machine à états :

- Exécute les groupes `GroupA` de `GroupB` test et de test en parallèle. Les `ResultVar` variables stockées dans le contexte par les `RunTask` états dans les machines à états de branche sont disponibles pour l'`AddProductFeatures`état.
- Vérifie l'absence d'erreurs d'exécution et effectue la transition vers le `Fail` cas échéant. Cette machine d'état n'utilise pas de `Catch` bloc car cette méthode ne détecte pas les erreurs d'exécution dans les machines d'état de branche.
- Ajoute des fonctionnalités au `awsiotdevicetester_report.xml` fichier en fonction des groupes qui passent
  - En `GroupA` cas de réussite, la fonctionnalité est réglée sur`supported`.
  - La fonctionnalité n'est pas marquée comme facultative dans le rapport.
- Génère un rapport et passe à `Succeed` s'il n'y a pas d'erreur, et `Fail` sinon

Si deux appareils sont configurés dans le pool de périphériques, `GroupA` les deux `GroupB` peuvent fonctionner en même temps. Toutefois, si l'`GroupA`un ou l'autre `GroupB` contient plusieurs tests, les deux appareils peuvent être affectés à ces tests. Si un seul appareil est configuré, les groupes de test s'exécuteront de manière séquentielle.

```
{
    "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"
        }
    }
}
```

Créer un fichier exécutable pour le scénario de test IDT

Vous pouvez créer et placer un exécutable de scénario de test dans un dossier de suite de tests de la manière suivante :

- Pour les suites de tests qui utilisent des arguments ou des variables d'environnement provenant des `test.json` fichiers pour déterminer les tests à exécuter, vous pouvez créer un seul scénario de test exécutable pour l'ensemble de la suite de tests, ou un exécutable de test pour chaque groupe de tests de la suite de tests.
- Pour une suite de tests dans laquelle vous souhaitez exécuter des tests spécifiques en fonction de commandes spécifiées, vous devez créer un fichier exécutable pour chaque cas de test de la suite de tests.

En tant que rédacteur de tests, vous pouvez déterminer quelle approche convient à votre cas d'utilisation et structurer l'exécutable de votre scénario de test en conséquence. Assurez-vous de fournir le chemin exécutable du scénario de test correct dans chaque `test.json` fichier et que le fichier exécutable spécifié s'exécute correctement.

Lorsque tous les appareils sont prêts pour l'exécution d'un scénario de test, IDT lit les fichiers suivants :

- Le `test.json` scénario de test sélectionné détermine les processus à démarrer et les variables d'environnement à définir.

- Le `suite.json` for the test suite détermine les variables d'environnement à définir.

IDT lance le processus exécutable de test requis en fonction des commandes et des arguments spécifiés dans le `test.json` fichier, et transmet les variables d'environnement requises au processus.

Utiliser le SDK du client IDT

Les SDK du client IDT vous permettent de simplifier la façon dont vous écrivez la logique de test dans votre exécutable de test à l'aide de commandes d'API que vous pouvez utiliser pour interagir avec IDT et vos appareils testés. IDT fournit actuellement les SDK suivants :

- SDK client IDT pour Python
- SDK client IDT pour Go
- SDK client IDT pour Java

Ces SDK se trouvent dans le *`<device-tester-extract-location>`*`/sdks` dossier. Lorsque vous créez un nouvel exécutable de scénario de test, vous devez copier le SDK que vous souhaitez utiliser dans le dossier contenant votre exécutable de scénario de test et référencer le SDK dans votre code. Cette section fournit une brève description des commandes d'API disponibles que vous pouvez utiliser dans les exécutables de vos scénarios de test.

Dans cette section

- Interaction avec les appareils
- Interaction IDT
- Interaction avec l'hôte

Interaction avec les appareils

Les commandes suivantes vous permettent de communiquer avec l'appareil testé sans avoir à implémenter de fonctions supplémentaires d'interaction avec l'appareil et de gestion de la connectivité.

`ExecuteOnDevice`

Permet aux suites de tests d'exécuter des commandes shell sur un appareil prenant en charge les connexions SSH ou Docker shell.

`CopyToDevice`

Permet aux suites de tests de copier un fichier local depuis la machine hôte qui exécute IDT vers un emplacement spécifié sur un appareil prenant en charge les connexions SSH ou Docker shell.

`ReadFromDevice`

Permet aux suites de tests de lire à partir du port série des appareils prenant en charge les connexions UART.

> **Note**
>
> IDT ne gérant pas les connexions directes aux appareils établies à l'aide des informations d'accès aux appareils issues du contexte, nous vous recommandons d'utiliser ces commandes API d'interaction avec les appareils dans les exécutables de vos scénarios de test. Toutefois, si ces commandes ne répondent pas aux exigences de votre scénario de test, vous pouvez récupérer les informations d'accès à l'appareil à partir du contexte IDT et les utiliser pour établir une connexion directe avec l'appareil à partir de la suite de tests.
> Pour établir une connexion directe, récupérez les informations dans les `resource.devices.connectivity` champs `device.connectivity` et pour votre appareil testé et pour les périphériques ressources, respectivement. Pour plus d'informations sur l'utilisation du contexte IDT, consultezUtiliser le contexte IDT.

Interaction IDT

Les commandes suivantes permettent à vos suites de tests de communiquer avec IDT.

`PollForNotifications`

Permet aux suites de tests de vérifier les notifications provenant d'IDT.

`GetContextValue` et **`GetContextString`**

Permet aux suites de tests de récupérer des valeurs à partir du contexte IDT. Pour de plus amples informations, veuillez consulter Utiliser le contexte IDT.

`SendResult`

Permet aux suites de tests de communiquer les résultats des scénarios de test à IDT. Cette commande doit être appelée à la fin de chaque scénario de test dans une suite de tests.

Interaction avec l'hôte

La commande suivante permet à vos suites de tests de communiquer avec la machine hôte.

`PollForNotifications`

Permet aux suites de tests de vérifier les notifications provenant d'IDT.

`GetContextValue` et **`GetContextString`**

Permet aux suites de tests de récupérer des valeurs à partir du contexte IDT. Pour de plus amples informations, veuillez consulter Utiliser le contexte IDT.

`ExecuteOnHost`

Permet aux suites de tests d'exécuter des commandes sur la machine locale et permet à IDT de gérer le cycle de vie des exécutables des scénarios de test.

Activer les commandes IDT CLI

La `run-suite` commande IDT CLI fournit plusieurs options qui permettent au lanceur de tests de personnaliser l'exécution des tests. Pour permettre aux testeurs d'utiliser ces options pour exécuter votre suite de tests personnalisée, vous implémentez la prise en charge de la CLI IDT. Si vous n'implémentez pas le support, les testeurs pourront toujours exécuter des tests, mais certaines options de la CLI ne fonctionneront pas correctement. Pour offrir une expérience client optimale, nous vous recommandons de mettre en œuvre la prise en charge des arguments suivants pour la `run-suite` commande dans la CLI IDT :

`timeout-multiplier`

Spécifie une valeur supérieure à 1,0 qui sera appliquée à tous les délais d'expiration lors de l'exécution des tests.

Les testeurs peuvent utiliser cet argument pour augmenter le délai d'expiration des scénarios de test qu'ils souhaitent exécuter. Lorsqu'un lanceur de tests spécifie cet argument dans sa `run-suite` commande, IDT l'utilise pour calculer la valeur de la variable d'environnement IDT_TEST_TIMEOUT et définit le champ dans le contexte IDT. `config.timeoutMultiplier` Pour étayer cet argument, vous devez procéder comme suit :

- Au lieu d'utiliser directement la valeur de délai d'attente du `test.json` fichier, lisez la variable d'environnement IDT_TEST_TIMEOUT pour obtenir la valeur de délai d'expiration correctement calculée.

- Récupérez la `config.timeoutMultiplier` valeur dans le contexte IDT et appliquez-la à des délais d'expiration prolongés.

Pour plus d'informations sur la fermeture anticipée en raison d'événements liés au délai imparti, consultez. Spécifier le comportement de sortie

`stop-on-first-failure`

Spécifie qu'IDT doit arrêter d'exécuter tous les tests en cas d'échec.

Lorsqu'un lanceur de tests spécifie cet argument dans sa `run-suite` commande, IDT arrête d'exécuter les tests dès qu'il rencontre un échec. Toutefois, si les scénarios de test sont exécutés en parallèle, cela peut entraîner des résultats inattendus. Pour mettre en œuvre le support, assurez-vous que si IDT rencontre cet événement, votre logique de test indique à tous les scénarios de test en cours d'exécution de s'arrêter, de nettoyer les ressources temporaires et de communiquer un résultat de test à IDT. Pour plus d'informations sur la gestion anticipée des défaillances, consultezSpécifier le comportement de sortie.

`group-id` et **`test-id`**

Spécifie qu'IDT ne doit exécuter que les groupes de tests ou les cas de test sélectionnés.

Les testeurs peuvent utiliser ces arguments avec leur `run-suite` commande pour spécifier le comportement d'exécution du test suivant :

- Exécutez tous les tests au sein des groupes de tests spécifiés.
- Exécutez une sélection de tests au sein d'un groupe de tests spécifié.

Pour prendre en charge ces arguments, la machine à états de votre suite de tests doit inclure un ensemble spécifique d'`Choice`états `RunTask` et dans votre machine à états. Si vous n'utilisez pas de machine à états personnalisée, la machine à états IDT par défaut inclut les états requis pour vous et vous n'avez pas besoin de prendre d'autres mesures. Toutefois, si vous utilisez une machine à états personnalisée, utilisez-la Exemple de machine à états : exécuter des groupes de test sélectionnés par l'utilisateur comme exemple pour ajouter les états requis dans votre machine à états.

Pour plus d'informations sur les commandes IDT CLI, consultezDéboguer et exécuter des suites de tests personnalisées.

Rédiger des journaux d'événements

Pendant le test, vous envoyez des données à la console `stdout` et vous `stderr` devez y écrire des journaux d'événements et des messages d'erreur. Pour plus d'informations sur le format des messages de console, consultezFormat des messages de console.

Lorsque l'IDT a terminé d'exécuter la suite de tests, ces informations sont également disponibles dans le `test_manager.log` fichier situé dans le *<devicetester-extract-location>*`/results/`*<execution-id>*`/logs` dossier.

Vous pouvez configurer chaque scénario de test pour écrire les journaux de son exécution, y compris les journaux du périphérique testé, dans le *<group-id>*`_`*<test-id>* fichier situé dans le *<device-tester-extract-location>*`/results/`*execution-id*`/logs` dossier. Pour ce faire, récupérez le chemin d'accès au fichier journal à partir du contexte IDT contenant la `testData.logFilePath` requête, créez un fichier sur ce chemin et inscrivez le contenu souhaité. IDT met automatiquement à jour le chemin en fonction du scénario de test en cours d'exécution. Si vous choisissez de ne pas créer le fichier journal pour un scénario de test, aucun fichier n'est généré pour ce scénario de test.

Vous pouvez également configurer votre exécutable texte pour créer des fichiers journaux supplémentaires, selon les besoins, dans le *<device-tester-extract-location>*`/logs` dossier. Nous vous recommandons de spécifier des préfixes uniques pour les noms de fichiers journaux afin que vos fichiers ne soient pas remplacés.

Signaler les résultats à IDT

IDT écrit les résultats des tests dans les fichiers `awsiotdevicetester_report.xml` et les *suite-name*`_report.xml` fichiers. Ces fichiers de rapport se trouvent dans*<device-tester-extract-location>*`/results/`*<execution-id>*`/`. Les deux rapports capturent les résultats de l'exécution de la suite de tests. Pour plus d'informations sur les schémas utilisés par IDT pour ces rapports, voir Consulter les résultats et les journaux des tests IDT

Pour renseigner le contenu du *suite-name*`_report.xml` fichier, vous devez utiliser la `SendResult` commande pour communiquer les résultats des tests à IDT avant la fin de l'exécution du test. Si IDT ne trouve pas les résultats d'un test, il émet une erreur pour le scénario de test. L'extrait Python suivant montre les commandes permettant d'envoyer un résultat de test à IDT :

```
request-variable = SendResultRequest(TestResult(result))
client.send_result(request-variable)
```

Si vous ne communiquez pas les résultats via l'API, IDT recherche les résultats des tests dans le dossier des artefacts de test. Le chemin d'accès à ce dossier est stocké dans le `testData.testArtifactsPath` fichier dans le contexte IDT. Dans ce dossier, IDT utilise le premier fichier XML trié par ordre alphabétique qu'il trouve comme résultat du test.

Si votre logique de test produit des résultats XML JUnit, vous pouvez écrire les résultats du test dans un fichier XML dans le dossier des artefacts pour fournir directement les résultats à IDT au lieu de les analyser puis d'utiliser l'API pour les soumettre à IDT.

Si vous utilisez cette méthode, assurez-vous que votre logique de test résume correctement les résultats du test et formatez votre fichier de résultats dans le même format que le *`suite-name`*`_report.xml` fichier. IDT n'effectue aucune validation des données que vous fournissez, sauf dans les cas suivants :

- IDT ignore toutes les propriétés de la `testsuites` balise. Au lieu de cela, il calcule les propriétés des balises à partir des résultats d'autres groupes de test rapportés.
- Au moins une `testsuite` balise doit figurer à l'intérieur`testsuites`.

Étant donné qu'IDT utilise le même dossier d'artefacts pour tous les scénarios de test et ne supprime pas les fichiers de résultats entre les tests, cette méthode peut également entraîner des rapports erronés si IDT lit le mauvais fichier. Nous vous recommandons d'utiliser le même nom pour le fichier de résultats XML généré dans tous les scénarios de test afin de remplacer les résultats de chaque scénario de test et de vous assurer que les résultats corrects sont disponibles pour IDT. Bien que vous puissiez utiliser une approche mixte pour créer des rapports dans votre suite de tests, c'est-à-dire utiliser un fichier de résultats XML pour certains cas de test et soumettre les résultats via l'API pour d'autres, nous ne recommandons pas cette approche.

Spécifier le comportement de sortie

Configurez votre exécutable texte pour qu'il se ferme toujours avec un code de sortie de 0, même si un scénario de test indique un échec ou un résultat d'erreur. Utilisez des codes de sortie différents de zéro uniquement pour indiquer qu'un scénario de test n'a pas été exécuté ou si l'exécutable du scénario de test n'a pas pu communiquer de résultats à IDT. Lorsque IDT reçoit un code de sortie différent de zéro, cela indique que le scénario de test a rencontré une erreur qui l'a empêché de s'exécuter.

IDT peut demander ou s'attendre à ce qu'un scénario de test s'arrête avant sa fin dans les événements suivants. Utilisez ces informations pour configurer le fichier exécutable de votre scénario de test afin de détecter chacun de ces événements dans le scénario de test :

Expiration

Se produit lorsqu'un scénario de test s'exécute pendant une durée supérieure à la valeur de délai spécifiée dans le `test.json` fichier. Si le lanceur de test a utilisé l'`timeout-multiplier`argument pour spécifier un multiplicateur de délai d'attente, IDT calcule la valeur du délai d'expiration avec le multiplicateur.

Pour détecter cet événement, utilisez la variable d'environnement IDT_TEST_TIMEOUT. Lorsqu'un lanceur de tests lance un test, IDT définit la valeur de la variable d'environnement IDT_TEST_TIMEOUT sur la valeur du délai d'attente calculée (en secondes) et transmet la variable à l'exécutable du scénario de test. Vous pouvez lire la valeur de la variable pour définir un temporisateur approprié.

Interrompre

Survient lorsque le lanceur de test interrompt l'IDT. Par exemple, en appuyant surCtrl+C.

Étant donné que les terminaux propagent les signaux à tous les processus enfants, vous pouvez simplement configurer un gestionnaire de signaux dans vos scénarios de test pour détecter les signaux d'interruption.

Vous pouvez également interroger régulièrement l'API pour vérifier la valeur du `CancellationRequested` booléen dans la réponse de l'`PollForNotifications`API. Lorsque IDT reçoit un signal d'interruption, il définit la valeur du `CancellationRequested` booléen sur. `true`

Arrêt dès le premier échec

Se produit lorsqu'un scénario de test exécuté en parallèle avec le scénario de test en cours échoue et que le lanceur de test a utilisé l'`stop-on-first-failure`argument pour spécifier qu'IDT doit s'arrêter en cas de défaillance.

Pour détecter cet événement, vous pouvez interroger régulièrement l'API afin de vérifier la valeur du `CancellationRequested` booléen dans la réponse de l'`PollForNotifications`API. Lorsqu'IDT rencontre une défaillance et est configuré pour s'arrêter lors du premier échec, il définit la valeur du `CancellationRequested` booléen sur. `true`

Lorsque l'un de ces événements se produit, IDT attend 5 minutes que les scénarios de test en cours soient terminés. Si tous les scénarios de test en cours ne se terminent pas dans les 5 minutes, IDT force l'arrêt de chacun de leurs processus. Si IDT n'a pas reçu les résultats des tests avant la fin des

processus, il marquera les cas de test comme ayant expiré. Il est recommandé de veiller à ce que vos scénarios de test exécutent les actions suivantes lorsqu'ils rencontrent l'un des événements :

1. Arrêtez d'exécuter la logique de test normale.
2. Nettoyez toutes les ressources temporaires, telles que les artefacts de test présents sur l'appareil testé.
3. Signalez un résultat de test à IDT, tel qu'un échec ou une erreur.
4. Sortir.

Utiliser le contexte IDT

Lorsque IDT exécute une suite de tests, celle-ci peut accéder à un ensemble de données qui peuvent être utilisées pour déterminer le mode d'exécution de chaque test. Ces données sont appelées contexte IDT. Par exemple, la configuration des données utilisateur fournie par les testeurs dans un `userdata.json` fichier est mise à la disposition des suites de tests dans le contexte IDT.

Le contexte IDT peut être considéré comme un document JSON en lecture seule. Les suites de tests peuvent récupérer des données et écrire des données dans le contexte à l'aide de types de données JSON standard tels que des objets, des tableaux, des nombres, etc.

Schéma de contexte

Le contexte IDT utilise le format suivant :

```
{
    "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`

Informations contenues dans le `config.json`fichier. Le `config` champ contient également les champs supplémentaires suivants :

`config.timeoutMultiplier`

Le multiplicateur pour toute valeur de délai d'attente utilisée par la suite de tests. Cette valeur est spécifiée par le lanceur de tests à partir de la CLI IDT. La valeur par défaut est 1.

`config.idRootPath`

Cette valeur est un espace réservé pour la valeur du chemin absolu d'IDT lors de la configuration du `userdata.json` fichier. Ceci est utilisé par les commandes build et flash.

`device`

Informations sur le périphérique sélectionné pour le test. Ces informations sont équivalentes à l'élément du `devices` tableau dans le `device.json`fichier du périphérique sélectionné.

`devicePool`

Informations sur le pool de périphériques sélectionné pour le test. Ces informations sont équivalentes à l'élément de tableau de pool de périphériques de niveau supérieur défini dans le `device.json` fichier pour le pool de périphériques sélectionné.

`resource`

Informations sur les périphériques de ressources contenues dans le `resource.json` fichier.

`resource.devices`

Ces informations sont équivalentes au `devices` tableau défini dans le `resource.json` fichier. Chaque `devices` élément inclut le champ supplémentaire suivant :

`resource.device.name`

Nom du périphérique ressource. Cette valeur est définie sur la `requiredResource.name` valeur du `test.json` fichier.

`testData.awsCredentials`

Les AWS informations d'identification utilisées par le test pour se connecter au AWS cloud. Ces informations sont extraites du `config.json` fichier.

`testData.logFilePath`

Le chemin d'accès au fichier journal dans lequel le scénario de test écrit les messages de journal. La suite de tests crée ce fichier s'il n'existe pas.

`userData`

Informations fournies par le testeur dans le `userdata.json`fichier.

Accédez aux données dans le contexte

Vous pouvez interroger le contexte à l'aide de la notation JSONPath à partir de vos fichiers de configuration et de votre exécutable texte avec les API `GetContextValue` and`GetContextString`. La syntaxe des chaînes JSONPath permettant d'accéder au contexte IDT varie comme suit :

- Dans `suite.json` et`test.json`, tu utilises`{{`*`query`*`}}`. En d'autres termes, n'utilisez pas l'élément racine `$.` pour démarrer votre expression.
- Dans`statemachine.json`, tu utilises`{{$.`*`query`*`}}`.
- Dans les commandes d'API, vous utilisez *`query`* ou`{{$.`*`query`*`}}`, selon la commande. Pour plus d'informations, consultez la documentation intégrée dans les SDK.

Le tableau suivant décrit les opérateurs d'une expression foobar JSONPath typique :

Opérateur	Description
`$`	L'élément racine. Comme la valeur de contexte de premier niveau pour IDT est un objet, vous l'utiliserez généralement `$.` pour démarrer vos requêtes.
`.childName`	Accède à l'élément enfant dont le nom `childName` provient d'un objet. S'il est appliqué à un tableau, il produit un nouveau tableau avec cet opérateur appliqué à chaque élément. Le nom de l'élément distingue les majuscules et minuscules. Par exemple, la requête pour accéder à la `awsRegion` valeur de l'`config`objet est`$.config.awsRegion` .
`[start:end]`	Filtre les éléments d'un tableau, en récupérant les éléments en commençant par l'`start`index et en remontant jusqu'à l'`end`index, dans les deux cas inclus.
`[index1, index2, ... , indexN]`	Filtre les éléments d'un tableau, en récupéran t les éléments uniquement à partir des indices spécifiés.
`[?(expr)]`	Filtre les éléments d'un tableau à l'aide de l'`expr`expression. Cette expression doit être évaluée à une valeur booléenne.

Pour créer des expressions de filtre, utilisez la syntaxe suivante :

```
<jsonpath> | <value> operator <jsonpath> | <value>
```

Dans cette syntaxe :

- `jsonpath`est un JSONPath qui utilise la syntaxe JSON standard.

- `value`est une valeur personnalisée qui utilise la syntaxe JSON standard.
- `operator`est l'un des opérateurs suivants :
  - <(Inférieur à)
  - <=(Inférieur ou égal à)
  - ==(Égal à)

    Si le JSONPath ou la valeur de votre expression est un tableau, un booléen ou une valeur d'objet, il s'agit du seul opérateur binaire pris en charge que vous pouvez utiliser.
  - >=(Supérieur ou égal à)
  - >(Supérieur à)
  - =~(Correspondance d'expressions régulières). Pour utiliser cet opérateur dans une expression de filtre, le JSONPath ou la valeur du côté gauche de votre expression doit être une chaîne et le côté droit doit être une valeur de modèle conforme à la syntaxe RE2.

Vous pouvez utiliser des requêtes JSONPath sous la forme {{*`query`*}} comme chaînes d'espace réservé dans les `environmentVariables` champs `args` et des `test.json` fichiers et dans les `environmentVariables` champs des fichiers. `suite.json` IDT effectue une recherche contextuelle et remplit les champs avec la valeur évaluée de la requête. Par exemple, dans le `suite.json` fichier, vous pouvez utiliser des chaînes d'espace réservé pour spécifier des valeurs de variables d'environnement qui changent avec chaque scénario de test et IDT remplira les variables d'environnement avec la valeur correcte pour chaque cas de test. Toutefois, lorsque vous utilisez des chaînes d'espace réservé dans des `suite.json` fichiers `test.json` et, les considérations suivantes s'appliquent à vos requêtes :

- Vous devez écrire toutes les occurrences de la `devicePool` clé dans votre requête en minuscules. C'est-à-dire, utilisez `devicepool` plutôt.
- Pour les tableaux, vous ne pouvez utiliser que des tableaux de chaînes. De plus, les tableaux utilisent un format non standard. `item1, item2,...,itemN` Si le tableau ne contient qu'un seul élément, il est sérialisé en tant que tel`item`, ce qui le rend impossible à distinguer d'un champ de chaîne.
- Vous ne pouvez pas utiliser d'espaces réservés pour récupérer des objets depuis le contexte.

En raison de ces considérations, nous vous recommandons, dans la mesure du possible, d'utiliser l'API pour accéder au contexte de votre logique de test au lieu d'utiliser des chaînes de caractères dans `suite.json` les fichiers `test.json` et. Cependant, dans certains cas, il peut être plus

pratique d'utiliser des espaces réservés JSONPath pour récupérer des chaînes uniques à définir comme variables d'environnement.

Configuration des paramètres pour les testeurs

Pour exécuter des suites de tests personnalisées, les testeurs doivent configurer leurs paramètres en fonction de la suite de tests qu'ils souhaitent exécuter. Les paramètres sont spécifiés en fonction des modèles de fichiers de configuration situés dans le *<device-tester-extract-location>*`/configs/` dossier. Si nécessaire, les testeurs doivent également configurer des AWS informations d'identification qu'IDT utilisera pour se connecter au AWS cloud.

En tant que rédacteur de tests, vous devrez configurer ces fichiers pour déboguer votre suite de tests. Vous devez fournir des instructions aux testeurs afin qu'ils puissent configurer les paramètres suivants selon les besoins pour exécuter vos suites de tests.

Configurer device.json

Le `device.json` fichier contient des informations sur les appareils sur lesquels les tests sont exécutés (par exemple, adresse IP, informations de connexion, système d'exploitation et architecture du processeur).

Les testeurs peuvent fournir ces informations à l'aide du `device.json` fichier modèle suivant situé dans le *<device-tester-extract-location>*`/configs/` dossier.

```
[
    {
        "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>",
                }
            }
        ]
    }
]
```

Tous les champs qui contiennent des valeurs sont requis, comme indiqué ici :

`id`

ID alphanumérique défini par l'utilisateur qui identifie de façon unique un ensemble d'appareils appelé un groupe d'appareils. Le matériel doit être identique pour les appareils d'un même groupe. Lorsque vous exécutez une suite de tests, les appareils du groupe sont utilisés pour paralléliser la charge de travail. Plusieurs appareils sont utilisés pour exécuter différents tests.

`sku`

Valeur alphanumérique qui identifie de façon unique l'appareil que vous testez. Le SKU est utilisé pour suivre les appareils qualifiés.

> **Note**
>
> Si vous souhaitez mettre votre carte en vente dans le catalogue des appareils AWS partenaires, le SKU que vous spécifiez ici doit correspondre au SKU que vous avez utilisé lors du processus de mise en vente.

`features`

Facultatif. Un tableau contenant les fonctions prises en charge de l'appareil. Les fonctionnalités de l'appareil sont des valeurs définies par l'utilisateur que vous configurez dans votre suite de tests. Vous devez fournir à vos testeurs des informations sur les noms et les valeurs des fonctionnalités à inclure dans le `device.json` fichier. Par exemple, si vous souhaitez tester un périphérique qui fonctionne comme un serveur MQTT pour d'autres appareils, vous pouvez configurer votre logique de test pour valider les niveaux pris en charge spécifiques pour une fonctionnalité nomméeMQTT_QoS. Les testeurs fournissent le nom de cette fonctionnalité et définissent la valeur de la fonctionnalité en fonction des niveaux de QoS pris en charge par leur appareil. Vous pouvez récupérer les informations fournies depuis le contexte IDT avec la `devicePool.features` requête ou depuis le contexte de la machine à états avec la `pool.features` requête.

`features.name`

Le nom de la fonctionnalité.

`features.value`

Les valeurs des fonctionnalités prises en charge.

`features.configs`

Paramètres de configuration, si nécessaire, pour la fonctionnalité.

`features.config.name`

Nom du paramètre de configuration.

`features.config.value`

Les valeurs de réglage prises en charge.

devices

Un ensemble d'appareils du pool à tester. Au moins un appareil est requis.

devices.id

Un identificateur unique défini par l'utilisateur pour l'appareil testé.

devices.pairedResource

Identifiant unique défini par l'utilisateur pour un périphérique ressource. Cette valeur est requise lorsque vous testez des appareils à l'aide du protocole de `no-op` connectivité.

connectivity.protocol

Le protocole de communication utilisé pour communiquer avec cet appareil. Chaque appareil d'un pool doit utiliser le même protocole.

Actuellement, les seules valeurs prises en charge sont `ssh` et `uart` pour les appareils physiques, `docker` pour les conteneurs Docker et `no-op` pour les appareils qui n'ont pas de connexion directe avec la machine hôte IDT mais qui ont besoin d'un périphérique ressource comme intergiciel physique pour communiquer avec la machine hôte.

Pour les appareils non opérationnels, vous configurez l'ID du périphérique ressource dans`devices.pairedResource`. Vous devez également spécifier cet ID dans le `resource.json` fichier. L'appareil jumelé doit être un appareil physiquement jumelé à l'appareil testé. Une fois qu'IDT a identifié le périphérique ressource jumelé et s'y est connecté, IDT ne se connecte pas aux autres périphériques ressources conformément aux fonctionnalités décrites dans le `test.json` fichier.

connectivity.ip

L'adresse IP de l'appareil testé.

Cette propriété s'applique uniquement si `connectivity.protocol` est défini sur `ssh`.

connectivity.port

Facultatif. Numéro de port à utiliser pour les connexions SSH.

La valeur par défaut est 22.

Cette propriété s'applique uniquement si `connectivity.protocol` est défini sur `ssh`.

`connectivity.publicKeyPath`

Facultatif. Le chemin complet vers la clé publique utilisée pour authentifier les connexions au périphérique testé. Lorsque vous spécifiez le`publicKeyPath`, IDT valide la clé publique de l'appareil lorsqu'il établit une connexion SSH avec le périphérique testé. Si cette valeur n'est pas spécifiée, IDT crée une connexion SSH, mais ne valide pas la clé publique de l'appareil.

Nous vous recommandons vivement de spécifier le chemin d'accès à la clé publique et d'utiliser une méthode sécurisée pour récupérer cette clé publique. Pour les clients SSH standard basés sur une ligne de commande, la clé publique est fournie dans le `known_hosts` fichier. Si vous spécifiez un fichier de clé publique distinct, ce fichier doit utiliser le même format que le `known_hosts` fichier, c'est-à-dire`ip-address key-type public-key`.

`connectivity.auth`

Informations d'authentification pour la connexion.

Cette propriété s'applique uniquement si `connectivity.protocol` est défini sur `ssh`.

`connectivity.auth.method`

Méthode d'authentification utilisée pour accéder à un appareil sur le protocole de connectivité donné.

Les valeurs prises en charge sont :

- `pki`
- `password`

`connectivity.auth.credentials`

Informations d'identification utilisées pour l'authentification.

`connectivity.auth.credentials.password`

Mot de passe utilisé pour se connecter à l'appareil à tester.

Cette valeur s'applique uniquement si `connectivity.auth.method` est défini sur `password`.

`connectivity.auth.credentials.privKeyPath`

Chemin complet de la clé privée utilisée pour se connecter à l'appareil testé.

Cette valeur s'applique uniquement si `connectivity.auth.method` est défini sur `pki`.

`connectivity.auth.credentials.user`

Nom d'utilisateur pour la connexion à l'appareil testé.

`connectivity.serialPort`

Facultatif. Port série auquel le périphérique est connecté.

Cette propriété s'applique uniquement si `connectivity.protocol` est défini sur `uart`.

`connectivity.containerId`

ID de conteneur ou nom du conteneur Docker en cours de test.

Cette propriété s'applique uniquement si `connectivity.protocol` est défini sur `docker`.

`connectivity.containerUser`

Facultatif. Le nom de l'utilisateur à l'intérieur du conteneur. La valeur par défaut est l'utilisateur indiqué dans le Dockerfile.

La valeur par défaut est 22.

Cette propriété s'applique uniquement si `connectivity.protocol` est défini sur `docker`.

> **Note**
>
> Pour vérifier si les testeurs configurent la mauvaise connexion au périphérique pour un test, vous pouvez la récupérer dans le contexte `pool.Devices[0].Connectivity.Protocol` de la machine à états et la comparer à la valeur attendue dans un `Choice` état. Si un protocole incorrect est utilisé, imprimez un message en utilisant l'`LogMessage`état et passez à l'`Fail`état.
> Vous pouvez également utiliser un code de gestion des erreurs pour signaler un échec de test pour des types de périphériques incorrects.

(Facultatif) Configurer userdata.json

Le `userdata.json` fichier contient toutes les informations supplémentaires requises par une suite de tests mais qui ne sont pas spécifiées dans le `device.json` fichier. Le format de ce fichier

dépend du `userdata_scheme.json`fichier défini dans la suite de tests. Si vous êtes rédacteur de tests, assurez-vous de fournir ces informations aux utilisateurs qui exécuteront les suites de tests que vous écrivez.

(Facultatif) Configurer resource.json

Le `resource.json` fichier contient des informations sur les périphériques qui seront utilisés comme périphériques de ressources. Les périphériques ressources sont des appareils nécessaires pour tester certaines fonctionnalités d'un périphérique testé. Par exemple, pour tester la capacité Bluetooth d'un appareil, vous pouvez utiliser un périphérique ressource pour vérifier si votre appareil peut s'y connecter correctement. Les périphériques de ressources sont facultatifs et vous pouvez avoir besoin d'autant de périphériques de ressources que nécessaire. En tant que rédacteur de test, vous utilisez le fichier test.json pour définir les fonctionnalités du périphérique de ressources requises pour un test. Les testeurs utilisent ensuite le `resource.json` fichier pour fournir un pool de périphériques de ressources dotés des fonctionnalités requises. Assurez-vous de fournir ces informations aux utilisateurs qui exécuteront les suites de tests que vous écrivez.

Les testeurs peuvent fournir ces informations à l'aide du `resource.json` fichier modèle suivant situé dans le *<device-tester-extract-location>*`/configs/` dossier.

```
[
    {
        "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>",
            }
        }
    ]
}
]
```

Tous les champs qui contiennent des valeurs sont requis, comme indiqué ici :

`id`

ID alphanumérique défini par l'utilisateur qui identifie de façon unique un ensemble d'appareils appelé un groupe d'appareils. Le matériel doit être identique pour les appareils d'un même groupe. Lorsque vous exécutez une suite de tests, les appareils du groupe sont utilisés pour paralléliser la charge de travail. Plusieurs appareils sont utilisés pour exécuter différents tests.

`features`

Facultatif. Un tableau contenant les fonctions prises en charge de l'appareil. Les informations requises dans ce champ sont définies dans les fichiers test.json de la suite de tests et déterminent les tests à exécuter et la manière de les exécuter. Si la suite de tests ne nécessite aucune fonctionnalité, ce champ n'est pas obligatoire.

`features.name`

Le nom de la fonctionnalité.

`features.version`

La version fonctionnelle.

`features.jobSlots`

Paramètre pour indiquer le nombre de tests pouvant utiliser simultanément l'appareil. La valeur par défaut est 1.

`devices`

Un ensemble d'appareils du pool à tester. Au moins un appareil est requis.

`devices.id`

Un identificateur unique défini par l'utilisateur pour l'appareil testé.

`connectivity.protocol`

Le protocole de communication utilisé pour communiquer avec cet appareil. Chaque appareil d'un pool doit utiliser le même protocole.

Actuellement, les seules valeurs prises en charge sont `ssh` et `uart` pour les appareils physiques, ainsi que `docker` pour les conteneurs Docker.

`connectivity.ip`

L'adresse IP de l'appareil testé.

Cette propriété s'applique uniquement si `connectivity.protocol` est défini sur `ssh`.

`connectivity.port`

Facultatif. Numéro de port à utiliser pour les connexions SSH.

La valeur par défaut est 22.

Cette propriété s'applique uniquement si `connectivity.protocol` est défini sur `ssh`.

`connectivity.publicKeyPath`

Facultatif. Le chemin complet vers la clé publique utilisée pour authentifier les connexions au périphérique testé. Lorsque vous spécifiez le`publicKeyPath`, IDT valide la clé publique de l'appareil lorsqu'il établit une connexion SSH avec le périphérique testé. Si cette valeur n'est pas spécifiée, IDT crée une connexion SSH, mais ne valide pas la clé publique de l'appareil.

Nous vous recommandons vivement de spécifier le chemin d'accès à la clé publique et d'utiliser une méthode sécurisée pour récupérer cette clé publique. Pour les clients SSH

standard basés sur une ligne de commande, la clé publique est fournie dans le `known_hosts` fichier. Si vous spécifiez un fichier de clé publique distinct, ce fichier doit utiliser le même format que le `known_hosts` fichier, c'est-à-dire`ip-address key-type public-key`.

`connectivity.auth`

Informations d'authentification pour la connexion.

Cette propriété s'applique uniquement si `connectivity.protocol` est défini sur `ssh`.

`connectivity.auth.method`

Méthode d'authentification utilisée pour accéder à un appareil sur le protocole de connectivité donné.

Les valeurs prises en charge sont :

- `pki`
- `password`

`connectivity.auth.credentials`

Informations d'identification utilisées pour l'authentification.

`connectivity.auth.credentials.password`

Mot de passe utilisé pour se connecter à l'appareil à tester.

Cette valeur s'applique uniquement si `connectivity.auth.method` est défini sur `password`.

`connectivity.auth.credentials.privKeyPath`

Chemin complet de la clé privée utilisée pour se connecter à l'appareil testé.

Cette valeur s'applique uniquement si `connectivity.auth.method` est défini sur `pki`.

`connectivity.auth.credentials.user`

Nom d'utilisateur pour la connexion à l'appareil testé.

`connectivity.serialPort`

Facultatif. Port série auquel le périphérique est connecté.

Cette propriété s'applique uniquement si `connectivity.protocol` est défini sur `uart`.

`connectivity.containerId`

ID de conteneur ou nom du conteneur Docker en cours de test.

Cette propriété s'applique uniquement si `connectivity.protocol` est défini sur `docker`.

`connectivity.containerUser`

Facultatif. Le nom de l'utilisateur à l'intérieur du conteneur. La valeur par défaut est l'utilisateur indiqué dans le Dockerfile.

La valeur par défaut est 22.

Cette propriété s'applique uniquement si `connectivity.protocol` est défini sur `docker`.

(Facultatif) Configurer config.json

Le `config.json` fichier contient des informations de configuration pour IDT. En règle générale, les testeurs n'ont pas besoin de modifier ce fichier, sauf pour fournir leurs informations AWS d'identification utilisateur pour IDT et, éventuellement, pour une AWS région. Si des AWS informations d'identification avec les autorisations requises sont fournies, AWS IoT Device Tester collecte et soumet les statistiques d'utilisation à AWS. Il s'agit d'une fonctionnalité opt-in utilisée pour améliorer la fonctionnalité IDT. Pour de plus amples informations, veuillez consulter Soumettre les statistiques d'utilisation de l'IDT.

Les testeurs peuvent configurer leurs AWS informations d'identification de l'une des manières suivantes :

- Fichier d'informations d'identification

  IDT utilise le même fichier d'informations d'identification que l' AWS CLI. Pour de plus amples informations, veuillez consulter Fichiers de configuration et d'informations d'identification.

  L'emplacement du fichier d'informations d'identification varie en fonction du système d'exploitation que vous utilisez :

  - macOS, Linux : `~/.aws/credentials`
  - Windows: `C:\Users\`*`UserName`*`\.aws\credentials`
- Variables d'environnement

Les variables d'environnement sont des variables gérées par le système d'exploitation et utilisées par les commandes du système. Les variables définies au cours d'une session SSH ne sont pas disponibles après la fermeture de cette session. IDT peut utiliser les variables d'`AWS_SECRET_ACCESS_KEY`environnement `AWS_ACCESS_KEY_ID` et pour stocker les informations d'identification AWS

Pour définir ces variables sous Linux, macOS ou Unix, utilisez export:

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

Pour définir ces variables sous Windows, utilisez set :

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

Pour configurer les AWS informations d'identification pour IDT, les testeurs modifient la `auth` section du `config.json` fichier situé dans le *<device-tester-extract-location>*`/configs/` dossier.

```
{
    "log": {
        "location": "logs"
    },
    "configFiles": {
        "root": "configs",
        "device": "configs/device.json"
    },
    "testPath": "tests",
    "reportPath": "results",
    "awsRegion": "<region>",
    "auth": {
        "method": "file | environment",
        "credentials": {
            "profile": "<profile-name>"
        }
    }
}
]
```

Tous les champs qui contiennent des valeurs sont requis, comme indiqué ici :

> **Note**
>
> Tous les chemins de ce fichier sont définis par rapport à*<device-tester-extract-location>*.

`log.location`

Le chemin d'accès au dossier des journaux dans le*<device-tester-extract-location>*.

`configFiles.root`

Le chemin d'accès au dossier contenant les fichiers de configuration.

`configFiles.device`

Le chemin d'accès au `device.json` fichier.

`testPath`

Le chemin d'accès au dossier contenant les suites de tests.

`reportPath`

Le chemin d'accès au dossier qui contiendra les résultats des tests une fois qu'IDT aura exécuté une suite de tests.

`awsRegion`

Facultatif. La AWS région que les suites de tests utiliseront. Si ce paramètre n'est pas défini, les suites de tests utiliseront la région par défaut spécifiée dans chaque suite de tests.

`auth.method`

Méthode utilisée par IDT pour récupérer les AWS informations d'identification. Les valeurs prises en charge sont `file` la récupération des informations d'identification à partir d'un fichier d'informations d'identification et la récupération `environment` des informations d'identification à l'aide de variables d'environnement.

`auth.credentials.profile`

Le profil d'informations d'identification à utiliser à partir du fichier d'informations d'identification. Cette propriété s'applique uniquement si `auth.method` est défini sur `file`.

Déboguer et exécuter des suites de tests personnalisées

Une fois la configuration requise définie, IDT peut exécuter votre suite de tests. Le temps d'exécution de la suite de tests complète dépend du matériel et de la composition de la suite de tests. À titre de référence, il faut environ 30 minutes pour terminer la suite complète de tests de qualification FreeRTOS sur un Raspberry Pi 3B.

Lorsque vous écrivez votre suite de tests, vous pouvez utiliser IDT pour exécuter la suite de tests en mode débogage afin de vérifier votre code avant de l'exécuter ou de le fournir aux testeurs.

Exécutez IDT en mode debug

Étant donné que les suites de tests dépendent de l'IDT pour interagir avec les appareils, fournir le contexte et recevoir les résultats, vous ne pouvez pas simplement déboguer vos suites de tests dans un IDE sans aucune interaction IDT. Pour ce faire, la CLI IDT fournit la `debug-test-suite` commande qui vous permet d'exécuter IDT en mode de débogage. Exécutez la commande suivante pour afficher les options disponibles pour `debug-test-suite` :

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

Lorsque vous exécutez IDT en mode débogage, IDT ne lance pas réellement la suite de tests ni n'exécute l'orchestrateur de test ; il interagit plutôt avec votre IDE pour répondre aux demandes émanant de la suite de tests exécutée dans l'IDE et imprime les journaux sur la console. L'IDT n'expire pas et attend de sortir jusqu'à ce qu'il soit interrompu manuellement. En mode débogage, IDT n'exécute pas non plus l'orchestrateur de test et ne génère aucun fichier de rapport. Pour déboguer votre suite de tests, vous devez utiliser votre IDE pour fournir certaines informations que IDT obtient généralement à partir des fichiers de configuration. Assurez-vous de fournir les informations suivantes :

- Variables d'environnement et arguments pour chaque test. IDT ne lira pas ces informations depuis `test.json` ou`suite.json`.
- Arguments pour sélectionner les périphériques de ressources. IDT ne lira pas ces informations depuis`test.json`.

Pour déboguer vos suites de tests, procédez comme suit :

1. Créez les fichiers de configuration des paramètres nécessaires à l'exécution de la suite de tests. Par exemple, si votre suite de tests nécessite le `device.json`ou`resource.json`, et`user data.json`, assurez-vous de tous les configurer selon vos besoins.

2. Exécutez la commande suivante pour placer IDT en mode de débogage et sélectionnez les appareils nécessaires pour exécuter le test.

   ```
   devicetester_[linux | mac | win_x86-64] debug-test-suite [options]
   ```

   Après avoir exécuté cette commande, IDT attend les demandes de la suite de tests, puis y répond. IDT génère également les variables d'environnement requises pour le traitement des dossiers pour le SDK client IDT.

3. Dans votre IDE, utilisez la `debug` configuration `run` or pour effectuer les opérations suivantes :

   a. Définissez les valeurs des variables d' IDT-generated environnement.

   b. Définissez la valeur de toutes les variables ou arguments d'environnement que vous avez spécifiés dans votre `suite.json` fichier `test.json` and.

   c. Définissez les points d'arrêt selon vos besoins.

4. Exécutez la suite de tests dans votre IDE.

   Vous pouvez déboguer et réexécuter la suite de tests autant de fois que nécessaire. Le délai d'expiration de l'IDT n'est pas dépassé en mode débogage.

5. Une fois le débogage terminé, interrompez IDT pour quitter le mode de débogage.

Commandes IDT CLI pour exécuter des tests

La section suivante décrit les commandes de la CLI IDT :

IDT v4.0.0

`help`

Répertorie les informations sur la commande spécifiée.

`list-groups`

Répertorie les groupes dans une suite de tests donnée.

`list-suites`

Répertorie les suites de tests disponibles.

`list-supported-products`

Répertorie les produits pris en charge pour votre version d'IDT, en l'occurrence les versions FreeRTOS, et les versions de la suite de tests de qualification FreeRTOS disponibles pour la version IDT actuelle.

`list-test-cases`

Répertorie les cas de tests d'un groupe de tests donné. L'option suivante est prise en charge :

- `group-id`. Le groupe de test à rechercher. Cette option est obligatoire et doit spécifier un groupe unique.

`run-suite`

Exécute une suite de tests sur un groupe d'appareils. Voici quelques options couramment utilisées :

- `suite-id`. Version de la suite de tests à exécuter. Si celle-ci n'est pas spécifiée, IDT utilise la dernière version dans le dossier `tests`.
- `group-id`. Les groupes de test à exécuter, sous forme de liste séparée par des virgules. Si cette option n'est pas spécifiée, IDT exécute tous les groupes de tests de la suite de tests.
- `test-id`. Les cas de test à exécuter, sous forme de liste séparée par des virgules. Lorsqu'il est spécifié, `group-id` doit spécifier un seul groupe.
- `pool-id`. Le pool d'appareils à tester. Les testeurs doivent spécifier un pool s'ils ont plusieurs pools d'appareils définis dans votre `device.json` fichier.
- `timeout-multiplier`. Configure IDT pour modifier le délai d'exécution du test spécifié dans le `test.json` fichier pour un test avec un multiplicateur défini par l'utilisateur.
- `stop-on-first-failure`. Configure IDT pour arrêter l'exécution lors du premier échec. Cette option doit être utilisée avec `group-id` pour déboguer les groupes de tests spécifiés.
- `userdata`. Définit le fichier contenant les informations de données utilisateur requises pour exécuter la suite de tests. Cela n'est obligatoire que s'`userdataRequired`il est défini sur true dans le `suite.json` fichier de la suite de tests.

Pour de plus amples informations sur les options `run-suite`, utilisez l' option `help` suivante :

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

`debug-test-suite`

Exécutez la suite de tests en mode debug. Pour de plus amples informations, veuillez consulter Exécutez IDT en mode debug.

Consulter les résultats et les journaux des tests IDT

Cette section décrit le format dans lequel IDT génère les journaux de console et les rapports de test.

Format des messages de console

AWS IoT Device Tester utilise un format standard pour imprimer des messages sur la console lorsqu'elle démarre une suite de tests. L'extrait suivant montre un exemple de message de console généré par IDT.

```
[INFO] [2000-01-02 03:04:05]: Using suite: MyTestSuite_1.0.0
 executionId=9a52f362-1227-11eb-86c9-8c8590419f30
```

La plupart des messages de console contiennent les champs suivants :

`time`

Un horodatage ISO 8601 complet pour l'événement enregistré.

`level`

Le niveau du message pour l'événement enregistré. Généralement, le niveau du message enregistré est l'un `info` des suivants `warn` : ou`error`. IDT émet un `panic` message `fatal` OR s'il rencontre un événement attendu qui entraîne sa fermeture anticipée.

`msg`

Le message enregistré.

`executionId`

Chaîne d'identification unique pour le processus IDT en cours. Cet identifiant est utilisé pour différencier les essais IDT individuels.

Les messages de console générés à partir d'une suite de tests fournissent des informations supplémentaires sur le périphérique testé, ainsi que sur la suite de tests, le groupe de test et les

scénarios de test exécutés par IDT. L'extrait suivant montre un exemple de message de console généré à partir d'une suite de tests.

```
[INFO] [2000-01-02 03:04:05]: Hello world! suiteId=MyTestSuitegroupId=myTestGroup
 testCaseId=myTestCase deviceId=my-
deviceexecutionId=9a52f362-1227-11eb-86c9-8c8590419f30
```

La partie spécifique à la suite de tests du message de console contient les champs suivants :

`suiteId`

Nom de la suite de tests en cours d'exécution.

`groupId`

ID du groupe de test en cours d'exécution.

`testCaseId`

ID du scénario de test en cours d'exécution.

`deviceId`

Identifiant de l'appareil testé utilisé par le scénario de test en cours.

Le résumé des tests contient des informations sur la suite de tests, les résultats des tests pour chaque groupe exécuté, ainsi que l'emplacement des journaux et des fichiers de rapport générés. L'exemple suivant montre un message récapitulatif du test.

```
========== 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 schéma de rapport

`awsiotdevicetester_report.xml`est un rapport signé qui contient les informations suivantes :

- La version IDT.
- La version de la suite de tests.
- Signature du rapport et clé utilisées pour signer le rapport.
- Le SKU de l'appareil et le nom du pool d'appareils spécifiés dans le `device.json` fichier.
- La version du produit et les fonctionnalités de l'appareil testées.
- Le récapitulatif des résultats des tests. Ces informations sont les mêmes que celles contenues dans le *`suite-name`*`_report.xml` fichier.

```
<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>
```

Le fichier `awsiotdevicetester_report.xml` contient une balise `<awsproduct>` qui contient des informations relatives au produit testé et les caractéristiques du produit qui ont été validées par une suite de tests.

Attributs utilisés dans la **`<awsproduct>`** balise

`name`

Nom du produit testé.

`version`

Version du produit testé.

`features`

Caractéristiques validées. Les fonctionnalités marquées comme `required` étant requises pour que la suite de tests valide le dispositif. L'extrait de code suivant montre comment ces informations apparaissent dans le fichier `awsiotdevicetester_report.xml`.

```
<feature name="ssh" value="supported" type="required"></feature>
```

Les fonctionnalités marquées comme ne `optional` sont pas requises pour la validation. Les extraits suivants illustrent des fonctions facultatives.

```
<feature name="hsi" value="supported" type="optional"></feature>
<feature name="mqtt" value="not-supported" type="optional"></feature>
```

Schéma de rapport de la suite de tests

Le rapport *suite-name*_Result.xml est au format JUnit XML. Vous pouvez intégrer des plateformes de déploiement/d'intégration continues tels que Jenkins, Bamboo, etc. Le rapport contient un résumé global des résultats des tests.

```
<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>
```

La section du rapport dans le awsiotdevicetester_report.xml ou *suite-name*_report.xml répertorie les tests effectués et les résultats.

La première balise XML <testsuites> contient le résumé de l'exécution des tests. Par exemple :

```
<testsuites name="MyTestSuite results" time="2299" tests="28" failures="0" errors="0"
 disabled="0">
```

Attributs utilisés dans la **<testsuites>** balise

name

Nom de la suite de tests.

time

Le temps, en secondes, nécessaire pour exécuter la suite de tests.

tests

Nombre de tests exécutés.

failures

Nombre de tests exécutés mais dont le résultat n'est pas probant.

errors

Nombre de tests qu'IDT n'a pas pu exécuter.

disabled

Cet attribut n'est pas utilisé et peut être ignoré.

En cas d'erreurs ou d'échecs de tests, vous pouvez identifier les tests concernés à l'aide des balises XML `<testsuites>`. Les balises XML `<testsuite>` au sein de la balise `<testsuites>` montrent le récapitulatif des résultats d'un groupe de tests. Par exemple :

```
<testsuite name="combination" package="" tests="1" failures="0" time="161" disabled="0"
 errors="0" skipped="0">
```

Le format est similaire à la balise `<testsuites>`, mais avec un attribut appelé `skipped` qui n'est pas utilisé et qui ne peut pas être ignoré. Chaque balise XML `<testsuite>` inclut des balises `<testcase>` pour chaque test exécuté pour un groupe de tests. Par exemple :

```
<testcase classname="Security Test" name="IP Change Tests" attempts="1"></testcase>
```

Attributs utilisés dans la **<testcase>** balise

name

Nom du test.

`attempts`

Nombre de fois où IDT a exécuté le test.

Lorsqu'un test échoue ou qu'une erreur se produit, les balises `<failure>` ou `<error>` sont ajoutées à la balise `<testcase>` avec des informations relatives au dépannage. Par exemple :

```
<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>
```

Soumettre les statistiques d'utilisation de l'IDT

Si vous fournissez des AWS informations d'identification avec les autorisations requises, AWS IoT Device Tester collecte et soumet les statistiques d'utilisation à AWS. Il s'agit d'une fonctionnalité opt-in utilisée pour améliorer la fonctionnalité IDT. IDT collecte des informations telles que les suivantes :

- L'ID de AWS compte utilisé pour exécuter IDT
- Les commandes IDT CLI utilisées pour exécuter des tests
- La suite de tests exécutée
- Les suites de tests dans le *`<device-tester-extract-location>`* dossier
- Le nombre d'appareils configurés dans le pool de périphériques
- Noms des scénarios de test et durées d'exécution
- Informations sur les résultats des tests, par exemple si les tests ont été réussis, ont échoué, ont rencontré des erreurs ou ont été ignorés
- Caractéristiques du produit testées
- Comportement de sortie IDT, tel que les sorties inattendues ou anticipées

Toutes les informations envoyées par IDT sont également enregistrées dans un `metrics.log` fichier du *`<device-tester-extract-location>`*`/results/`*`<execution-id>`*`/` dossier. Vous pouvez consulter le fichier journal pour voir les informations collectées lors d'un test. Ce fichier est généré uniquement si vous choisissez de collecter des statistiques d'utilisation.

Pour désactiver la collecte des métriques, il n'est pas nécessaire de prendre d'autres mesures. Ne stockez simplement pas vos AWS informations d'identification et, si vous en avez, ne configurez pas le `config.json` fichier pour y accéder. AWS

Inscrivez-vous pour un Compte AWS

Pour commencer AWS, vous avez besoin d'un Compte AWS. Pour plus d'informations sur la création d'un Compte AWS, voir Getting started with an Compte AWS dans le Guide de Gestion de compte AWS référence.

Pour activer l'accès, ajoutez des autorisations à vos utilisateurs, groupes ou rôles :

- Utilisateurs et groupes dans AWS IAM Identity Center :

  Créez un jeu d'autorisations. Suivez les instructions de la rubrique Création d'un jeu d'autorisations du Guide de l'utilisateur AWS IAM Identity Center .
- Utilisateurs gérés dans IAM par un fournisseur d'identité :

  Créez un rôle pour la fédération d'identité. Suivez les instructions de la rubrique Création d'un rôle pour un fournisseur d'identité tiers (fédération) dans le Guide de l'utilisateur IAM.
- Utilisateurs IAM :
  - Créez un rôle que votre utilisateur peut assumer. Suivez les instructions de la rubrique Création d'un rôle pour un utilisateur IAM dans le Guide de l'utilisateur IAM.
  - (Non recommandé) Attachez une politique directement à un utilisateur ou ajoutez un utilisateur à un groupe d'utilisateurs. Suivez les instructions de la rubrique Ajout d'autorisations à un utilisateur (console) du Guide de l'utilisateur IAM.

Fournir AWS informations d'identification à IDT

Pour autoriser IDT à accéder à vos AWS informations d'identification et à envoyer des statistiques AWS, procédez comme suit :

1. Stockez les AWS informations d'identification de votre utilisateur IAM sous forme de variables d'environnement ou dans un fichier d'informations d'identification :

   a. Pour utiliser des variables d'environnement, exécutez la commande suivante :

   ```
   AWS_ACCESS_KEY_ID=access-key
   AWS_SECRET_ACCESS_KEY=secret-access-key
   ```

b. Pour utiliser le fichier d'informations d'identification, ajoutez les informations suivantes au `.aws/credentials file:`

```
[profile-name]
aws_access_key_id=access-key
aws_secret_access_key=secret-access-key
```

2. Configurez la `auth` section du `config.json` fichier. Pour de plus amples informations, veuillez consulter (Facultatif) Configurer config.json.

Maintenir les versions des suites de tests

IDT pour FreeRTOS organise les ressources de test en suites de tests et en groupes de tests :

- Une suite de tests est un ensemble de groupes de tests utilisés pour vérifier qu'un appareil fonctionne avec des versions spécifiques de FreeRTOS.
- Un groupe de tests est l'ensemble des tests individuels liés à une fonctionnalité particulière, telle que la messagerie BLE et MQTT.

Depuis IDT v3.0.0, les suites de tests sont versionnées à l'aide d'un format `major.minor.patch` commençant par 1.0.0. Lorsque vous téléchargez IDT, le package inclut la version de suite de tests la plus récente.

Lorsque vous démarrez IDT dans l'interface de ligne de commande, IDT vérifie si une version de suite de tests plus récente est disponible. Si tel est le cas, vous êtes invité à effectuer une mise à jour vers la nouvelle version. Vous pouvez choisir de mettre à jour ou de poursuivre avec vos tests actuels.

> **Note**
>
> IDT prend en charge les trois versions de suite de tests les plus récentes pour la qualification. Pour de plus amples informations, veuillez consulter Comprendre la politique de support pour AWS IoT Device Tester.

Vous pouvez télécharger des suites de tests en utilisant la commande `upgrade-test-suite`. Vous pouvez également utiliser le paramètre optionnel `-upgrade-test-suite` *`flag`* lorsque vous

démarrez IDT, qui *flag* peut être « y » pour toujours télécharger la dernière version, ou « n » pour utiliser la version existante.

Vous pouvez également exécuter la `list-supported-versions` commande pour répertorier les versions de FreeRTOS et de suites de tests prises en charge par la version actuelle d'IDT.

De nouveaux tests peuvent introduire de nouveaux paramètres de configuration IDT. Si les paramètres sont facultatifs, IDT vous en avertit et continue à exécuter les tests. Si les paramètres sont requis, IDT vous en avertit et arrête l'exécution. Après avoir configuré les paramètres, vous pouvez continuer à exécuter les tests.

Résolution des erreurs

Chaque exécution de suite de tests a un ID d'exécution unique qui est utilisé pour créer un dossier nommé `results/`*`execution-id`* dans le répertoire `results`. Les journaux de groupes de tests individuels se trouvent dans le répertoire `results/`*`execution-id`*`/logs`. Utilisez la sortie de la console IDT pour FreeRTOS pour trouver l'identifiant d'exécution, l'identifiant du cas de test et l'identifiant du groupe de tests du scénario qui a échoué, puis ouvrez le fichier journal du scénario de test nommé. `results/`*`execution-id`*`/logs/`*`test_group_id`*`__`*`test_case_id`*`.log` Les informations dans ce fichier incluent, entre autres :

- Sortie complète des commandes de création et flash.
- Sortie de l'exécution des tests.
- IDT plus détaillé pour les sorties de console FreeRTOS.

Nous vous recommandons le flux de dépannage suivant :

1. Si le message d'erreur « n'*`user/role`*est pas autorisé à accéder à cette ressource » s'affiche, assurez-vous de configurer les autorisations comme indiqué dansCréez et configurez un AWS compte.
2. Lisez la sortie de la console pour trouver des informations, telles que l'UUID d'exécution et les tâches en cours d'exécution.
3. Recherchez dans le fichier `FRQ_Report.xml` les déclarations d'erreurs de chaque test. Ce répertoire contient des fichiers journaux d'exécution de chaque groupe de tests.
4. Consultez les fichiers journaux ci-dessous`/results/`*`execution-id`*`/logs`.
5. Vérifiez que les éléments suivants ne présentant pas de problème :

- La configuration des appareils, comme les fichiers de configuration JSON dans le dossier `/configs/`.
- Interface de l'appareil Vérifiez dans les journaux afin de déterminer quelle interface échoue.
- Outils de l'appareil. Assurez-vous que les chaînes d'outils de génération et de flashage de l'appareil sont installées et configurées correctement.
- Pour FRQ 1.x.x, assurez-vous qu'une version propre et clonée du code source de FreeRTOS est disponible. Les versions de FreeRTOS sont étiquetées en fonction de la version de FreeRTOS. Pour cloner une version spécifique du code, utilisez les commandes suivantes :

```
git clone --branch version-number https://github.com/aws/amazon-freertos.git
cd amazon-freertos
git submodule update --checkout --init --recursive
```

Résoudre les problèmes de configuration de l'appareil

Lorsque vous utilisez IDT pour FreeRTOS, vous devez mettre en place les bons fichiers de configuration avant d'exécuter le binaire. Si vous obtenez des erreurs d'analyse et de configuration, votre première étape devrait être de localiser et d'utiliser un modèle de configuration approprié pour votre environnement. Ces modèles sont situés dans le répertoire *IDT_ROOT*`/configs`.

Si le problème persiste, consultez les processus de débogage suivants.

Quels fichiers examiner ?

Commencez par lire la sortie de la console pour trouver des informations, telles que l'UUID d'exécution, qui est référencé en tant que `execution-id` dans cette documentation.

Ensuite, examinez le fichier `FRQ_Report.xml` dans le répertoire `/results/`*execution-id*. Ce fichier contient tous les scénarios de tests qui ont été exécutés et des extraits d'erreur pour chaque échec. Pour obtenir tous les journaux d'exécution, recherchez le fichier `/results/`*execution-id*`/logs/`*test_group_id*`__`*test_case_id*`.log` pour chaque cas de test.

Codes d'erreur IDT

Le tableau suivant explique les codes d'erreur générés par IDT pour FreeRTOS :

Code d'erreur	Nom du code d'erreur	Cause première possible	Résolution des problèmes
201	InvalidInputError	Les champs dans `device.json` , `config.json` ou `userdata.json` sont manquants ou dans un format incorrect.	Assurez-vous que les champs obligatoires ne sont pas manquants et qu'ils sont au format requis dans les fichiers répertoriés. Pour de plus amples informations, veuillez consulter Premier test de votre carte microcontrôleur.
202	ValidationError	Les champs de `device.json` , `config.json` ou `userdata.json` contiennent des valeurs non valides.	Vérifiez le message d'erreur situé à droite du code d'erreur dans le rapport : • AWS Région non valide - Spécifiez une AWS région valide dans votre `config.json` fichier. Pour plus d'informations sur AWS les régions, consultez Régions et points de terminaison. • AWS Informations d'identification non valides - Définissez des AWS informations d'identification valides sur votre

Code d'erreur	Nom du code d'erreur	Cause première possible	Résolution des problèmes
			machine de test (via des variables d'environnement ou le fichier d'informations d'identification). Vérifiez que le champ d'authentification est configuré correctement. Pour de plus amples informations, veuillez consulter Créez et configurez un AWS compte.

Code d'erreur	Nom du code d'erreur	Cause première possible	Résolution des problèmes
203	CopySourceCodeError	Impossible de copier le code source de FreeRTOS dans le répertoire spécifié.	Vérifiez les éléments suivants : • Vérifiez qu'un `sourcePath` valide est spécifié dans votre fichier `userdata.json` . • Supprimez le `build` dossier situé dans le répertoire du code source de FreeRTOS, s'il existe. Pour de plus amples informations, veuillez consulter Configurer les paramètres de création, de flash et de test. • Windows impose une limite de caractères pour les noms de chemin de fichier. Un chemin de fichier long provoquera une erreur.

Code d'erreur	Nom du code d'erreur	Cause première possible	Résolution des problèmes
204	BuildSourceError	Impossible de compiler le code source de FreeRTOS.	Vérifiez les éléments suivants : • Vérifiez que les informations sous `buildTool` dans votre fichier `userdata.json` sont correctes. • Si vous utilisez `cmake` comme outil de génération, assurez-vous que `{{enableTests}}` est spécifié dans la commande `buildTool` . Pour de plus amples informations, veuillez consulter Configurer les paramètres de création, de flash et de test. • Si vous avez extrait l'IDT pour FreeRTOS vers un chemin de fichier de votre système contenant des espaces, `C:\Users\My`

Code d'erreur	Nom du code d'erreur	Cause première possible	Résolution des problèmes
			`Name\Desktop\` par exemple, vous aurez peut-être besoin de guillemets supplémentaires dans vos commandes de compilation pour vous assurer que les chemins sont correctement analysés. La même chose peut être nécessaire pour vos commandes flash.
205	FlashOrRunTestError	IDT FreeRTOS n'est pas en mesure de flasher ou d'exécuter FreeRTOS sur votre DUT.	Vérifiez que les informations sous `flashTool` dans votre fichier `userdata.json` sont correctes. Pour de plus amples informations, veuillez consulter Configurer les paramètres de création, de flash et de test.

Code d'erreur	Nom du code d'erreur	Cause première possible	Résolution des problèmes
206	StartEchoServerError	IDT FreeRTOS n'est pas en mesure de démarrer le serveur Echo pour les WiFi tests de sockets sécurisés.	Vérifiez que les ports configurés sous `echoServerConfiguration` dans votre fichier `userdata.json` ne sont pas utilisés ou bloqués par le pare-feu ou les paramètres réseau.

Erreurs d'analyse du fichier de configuration de débogage

Parfois, une faute de frappe dans une configuration JSON peut entraîner des erreurs d'analyse. La plupart du temps, le problème est le résultat d'une virgule, d'une apostrophe ou d'un crochet manquant dans le fichier JSON. IDT pour FreeRTOS effectue la validation JSON et imprime les informations de débogage. Il imprime la ligne dans laquelle l'erreur s'est produite, le numéro de ligne et le numéro de colonne de l'erreur de syntaxe. Ces informations devraient suffire pour vous aider à résoudre l'erreur, mais si vous rencontrez des difficultés pour localiser cette dernière, vous pouvez effectuer la validation manuellement dans un environnement de développement intégré, un éditeur de texte tel qu'Atom ou Sublime, ou via un outil en ligne comme JSONLint.

Les résultats des tests de débogage et les erreurs d'analyse

Lors de l'exécution d'un groupe de test à partir de FreeRTOS-Libraries-Integration-TestsFullPKCS11_CoreFullTransportInterfaceTLS, FullPKCS11_, FullPKCS11_, FullPKCS11_ _ECCOnboard_ECC, Onboard_RSA FullPKCS11_ _RSA ou OtaCorePreProvisioned, IDT pour FreeRTOS analyse les résultats du test depuis le périphérique de test avec la connexion série. PreProvisioned Parfois, des sorties série supplémentaires sur le périphérique peuvent interférer avec l'analyse des résultats du test.

Dans le cas mentionné ci-dessus, d'étranges raisons d'échec du scénario de test, telles que des chaînes provenant de sorties de périphériques indépendantes, sont produites. Le fichier journal du

cas de test IDT pour FreeRTOS (qui inclut toutes les sorties série reçues par IDT pour FreeRTOS pendant le test) peut afficher les informations suivantes :

```
<unrelated device output>
TEST(Full_PKCS11_Capabilities, PKCS11_Capabilities)<unrelated device output>
<unrelated device output>
 PASS
```

Dans l'exemple ci-dessus, la sortie du périphérique indépendante empêche IDT pour FreeRTOS de détecter le résultat du test qui est PASS.

Vérifiez les points suivants pour garantir des tests optimaux.

- Assurez-vous que les macros de journalisation utilisées sur l'appareil sont compatibles avec les threads. Voir Implémentation des macros de journalisation de la bibliothèque pour plus d'informations.
- Assurez-vous que les sorties de la connexion série sont minimales pendant les tests. Les sorties d'autres appareils peuvent poser problème même si vos macros de journalisation sont correctement sécurisées par thread, car les résultats des tests seront générés lors d'appels séparés pendant les tests.

Un journal des cas de test IDT pour FreeRTOS devrait idéalement afficher une sortie ininterrompue des résultats de test, comme ci-dessous :

```
---------STARTING TESTS---------
TEST(Full_OTA_PAL, otaPal_CloseFile_ValidSignature) PASS
TEST(Full_OTA_PAL, otaPal_CloseFile_InvalidSignatureBlockWritten) PASS
-----------------------
2 Tests 0 Failures 0 Ignored
```

Défaillances du contrôle d'intégrité du débogage

Si vous utilisez la version FRQ 1.x.x de FreeRTOS, les contrôles d'intégrité suivants s'appliquent.

Lorsque vous exécutez le groupe de test FreeRTOSIntegrity et que vous rencontrez des échecs, assurez-vous d'abord de n'avoir modifié aucun des fichiers du répertoire. *freertos* Si ce n'est pas

le cas et que vous rencontrez toujours des problèmes, assurez-vous d'utiliser la bonne branche. Si vous exécutez la `list-supported-products` commande IDT, vous pouvez trouver la branche balisée du *`freertos`* dépôt que vous devez utiliser.

Si vous avez cloné la bonne branche balisée du `freertos` dépôt et que vous rencontrez toujours des problèmes, assurez-vous d'avoir également exécuté la `submodule update` commande. Le flux de travail de clonage pour le `freertos` dépôt est le suivant.

```
git clone --branch version-number https://github.com/aws/amazon-freertos.git
cd amazon-freertos
git submodule update --checkout —init —recursive
```

La liste des fichiers recherchés par le vérificateur d'intégrité se trouve dans le `checksums.json` fichier de votre *`freertos`* répertoire. Pour qualifier un port FreeRTOS sans aucune modification des fichiers et de la structure des dossiers, assurez-vous qu'aucun des fichiers répertoriés dans les sections « » et `exhaustive` `minimal` « » `checksums.json` du fichier n'a été modifié. Pour exécuter avec un SDK configuré, vérifiez qu'aucun des fichiers de la section « `minimal` » n'a été modifié.

Si vous exécutez IDT avec un SDK et que vous avez modifié certains fichiers de votre *`freertos`* répertoire, assurez-vous de configurer correctement votre SDK dans votre fichier. `userdata` Dans le cas contraire, le vérificateur d'intégrité vérifiera tous les fichiers du *`freertos`* répertoire.

Défaillances du groupe de FullWiFi test de débogage

Si vous utilisez FRQ 1.x.x et que vous rencontrez des échecs dans le groupe de FullWiFi test et que le test « `AFQP_WiFiConnectMultipleAP` » échoue, cela peut être dû au fait que les deux points d'accès ne se trouvent pas dans le même sous-réseau que l'ordinateur hôte exécutant IDT. Assurez-vous que les deux points d'accès se trouvent dans le même sous-réseau que l'ordinateur hôte exécutant IDT.

Déboguer les erreurs « paramètre requis manquant »

Étant donné que de nouvelles fonctionnalités sont ajoutées à IDT pour FreeRTOS, des modifications peuvent être apportées aux fichiers de configuration. L'utilisation d'un ancien fichier de configuration peut corrompre votre configuration. Si tel est le cas, le fichier *`test_group_id__test_case_id`*`.log` sous le répertoire `results/`*`execution-id`*`/logs` répertorie explicitement tous les paramètres manquants. IDT pour FreeRTOS valide vos schémas de fichiers de configuration JSON pour s'assurer que la dernière version prise en charge a été utilisée.

Déboguer les erreurs « le test n'a pas pu démarrer »

Vous pouvez voir des erreurs qui pointent sur des défaillances lors du démarrage du test. Comme il existe plusieurs causes possibles, vérifiez bien les points suivants :

- Assurez-vous que le nom du groupe que vous avez inclus dans votre commande d'exécution existe réellement. Celui-ci est référencé directement à partir du fichier `device.json`.
- Assurez-vous que le ou les appareils du groupe ont des paramètres de configuration corrects.

Déboguer les erreurs « Impossible de trouver le début des résultats du test »

Des erreurs peuvent s'afficher lorsque IDT tente d'analyser les résultats produits par le périphérique testé. Il existe plusieurs causes possibles, alors vérifiez l'exactitude des points suivants :

- Assurez-vous que le périphérique testé dispose d'une connexion stable avec votre machine hôte. Vous pouvez consulter le fichier journal pour un test qui indique ces erreurs afin de voir ce que reçoit IDT.
- Si vous utilisez FRQ 1.x.x et que le périphérique testé est connecté via un réseau lent ou une autre interface, ou si vous ne voyez pas l'indicateur « ---------STARTING TESTS--------- » dans le journal d'un groupe de test FreeRTOS avec les autres sorties du groupe de test FreeRTOS, vous pouvez essayer d'augmenter la valeur de dans votre configuration de données utilisateur. `testStartDelayms` Pour de plus amples informations, veuillez consulter Configurer les paramètres de création, de flash et de test.

Déboguer une erreur « Échec du test : __ résultats attendus mais ___ »

Des erreurs peuvent s'afficher qui indiquent un échec lors du test. Le test attend un certain nombre de résultats et ne le voit pas pendant le test. Certains tests FreeRTOS sont exécutés avant qu'IDT ne voie le résultat de l'appareil. Si cette erreur s'affiche, vous pouvez essayer d'augmenter la valeur de `testStartDelayms` dans votre configuration de données utilisateur. Pour de plus amples informations, veuillez consulter Configurer les paramètres de création, de flash et de test.

Déboguer une erreur « _____ n'a pas été sélectionné en raison de contraintes » ConditionalTests

Cela signifie que vous effectuez un test sur un pool de périphériques incompatible avec le test. Cela peut se produire avec les tests OTA E2E. Par exemple, lors de l'exécution du groupe de `OTADataplaneMQTT` test et dans votre fichier de `device.json` configuration, vous avez choisi OTA

comme Non ou `OTADataPlaneProtocol` comme HTTP. Le groupe de test choisi pour exécuter doit correspondre à vos sélections `device.json` de capacités.

Déboguer un délai IDT lors de la surveillance de la sortie de l'appareil

L'IDT peut expirer pour un certain nombre de raisons. Si un délai d'attente survient pendant la phase de surveillance des sorties de l'appareil d'un test et que vous pouvez voir les résultats dans le journal des cas de test IDT, cela signifie que les résultats ont été mal analysés par IDT. L'une des raisons pourrait être l'entrelacement des messages du journal au milieu des résultats des tests. Si tel est le cas, veuillez consulter le guide de portage de FreeRTOS pour plus de détails sur la manière dont les journaux UNITY doivent être configurés.

Un délai d'attente pendant la surveillance des sorties de l'appareil peut également être dû au redémarrage du périphérique après un seul échec du test TLS. L'appareil exécute ensuite l'image flashée et provoque une boucle infinie qui apparaît dans les journaux. Dans ce cas, assurez-vous que votre appareil ne redémarre pas après un échec du test.

Déboguer une erreur « accès non autorisé à la ressource »

Le message d'erreur « n'*`user/role`*est pas autorisé à accéder à cette ressource » peut s'afficher dans la sortie du terminal ou dans le `test_manager.log` fichier ci-dessous/`results/`*`execution-id`*`/logs`. Pour résoudre ce problème, attachez la stratégie gérée AWS `IoTDeviceTesterForFreeRTOSFullAccess` à votre utilisateur de test. Pour de plus amples informations, veuillez consulter Créez et configurez un AWS compte.

Déboguer les erreurs de test du réseau

Pour les tests basés sur réseau, IDT lance un serveur echo qui se lie à un port non réservé sur la machine hôte. Si vous rencontrez des erreurs dues à des délais impartis ou à des connexions indisponibles lors des WiFi tests de sockets sécurisés, assurez-vous que votre réseau est configuré pour autoriser le trafic vers les ports configurés compris entre 1024 et 49151.

Le test des sockets sécurisées utilise les ports 33333 et 33334 par défaut. Les WiFi tests utilisent le port 33335 par défaut. Si ces trois ports sont utilisés ou bloqués par un pare-feu ou un réseau, vous pouvez choisir d'utiliser d'autres ports dans userdata.json pour les tests. Pour de plus amples informations, veuillez consulter Configurer les paramètres de création, de flash et de test. Vous pouvez utiliser les commandes suivantes pour vérifier si un port spécifique est en cours d'utilisation :

- Windows: `netsh advfirewall firewall show rule name=all | grep port`
- Linux : `sudo netstat -pan | grep port`

- macOS : `netstat -nat | grep port`

Défaillances de mise à jour OTA dues à une charge utile de version identique

Si les scénarios de test OTA échouent parce que la même version se trouve sur l'appareil après l'exécution d'un OTA, cela peut être dû au fait que votre système de compilation (par exemple cmake) n'a pas remarqué les modifications apportées par IDT au code source de FreeRTOS et n'a pas créé de binaire mis à jour. L'opération OTA est alors effectuée avec le même binaire que celui qui est actuellement sur le périphérique, ce qui entraîne l'échec du test. Pour résoudre les défaillances de mise à jour OTA, commencez par vous assurer que vous utilisez la dernière version prise en charge de votre système de build.

Échec du test OTA sur le cas de test **`PresignedUrlExpired`**

Pour que ce test aboutisse, le temps de mise à jour OTA doit être supérieur à 60 secondes. Dans le cas contraire, le message d'erreur suivant se trouve dans le journal : « Test takes less than 60 seconds (url expired time) to finish. Please reach out to us. » (Le test dure moins de 60 secondes (délai d'expiration de l'URL). Veuillez nous contacter).

Erreurs de port et d'interface du périphérique de débogage

Cette section contient des informations sur les interfaces utilisées par IDT pour se connecter à vos appareils.

Plateformes prises en charge

IDT prend en charge Windows, macOS et Linux. Ces trois plateformes ont différents schémas de dénomination pour les appareils :

- Linux : `/dev/tty*`
- macOS : `/dev/tty.*` ou `/dev/cu.*`
- Windows : COM*

Pour vérifier le port de votre appareil :

- Pour Linux/macOS, ouvrez un terminal et lancez-le`ls /dev/tty*`.
- Pour macOS, ouvrez un terminal et exécutez `ls /dev/tty.*` ou `ls /dev/cu.*`.
- Pour Windows, ouvrez le gestionnaire d'appareils et développez le groupe d'appareils en série.

Pour vérifier que l'appareil est connecté à un port :

- Pour Linux, assurez-vous que le package `udev` est installé, puis exécutez `udevadm info --name=PORT`. Cet utilitaire imprime les informations de pilote de l'appareil qui vous permettent de vérifier que vous utilisez le bon port.
- Pour macOS, ouvrez Launchpad et recherchez **`System Information`**.
- Pour Windows, ouvrez le gestionnaire d'appareils et développez le groupe d'appareils en série.

Interfaces du périphérique

Chaque appareil intégré est différent, ce qui signifie qu'ils peuvent avoir un ou plusieurs ports série. Il est courant pour les appareils d'avoir deux connexions lorsqu'ils sont connectés à une machine :

- Un port de données pour flasher l'appareil.
- Un port de lecture pour lire la sortie.

  Vous devez définir le bon port de lecture dans votre fichier `device.json`. Dans le cas contraire, la lecture de la sortie à partir de l'appareil peut échouer.

  Lorsqu'il existe plusieurs ports, assurez-vous d'utiliser le port de lecture de l'appareil dans votre fichier `device.json`. Par exemple, si vous branchez un appareil Espressif WRover et que les deux ports qui lui sont assignés sont `/dev/ttyUSB0` et `/dev/ttyUSB1`, utilisez `/dev/ttyUSB1` dans votre fichier `device.json`.

Pour Windows, suivez la même logique.

Lecture des données du périphérique

IDT pour FreeRTOS utilise la création de périphériques individuels et des outils flash pour spécifier la configuration des ports. Si vous testez votre appareil et que vous n'obtenez pas de sortie, essayez les paramètres par défaut suivants :

- Vitesse de transmission : 115200
- Bits de données : 8
- Parité : aucune
- Bits d'arrêt : 1
- Contrôle de flux : aucun

Ces paramètres sont gérés par IDT pour FreeRTOS. Vous n'avez pas à les définir. Cependant, vous pouvez utiliser la même méthode pour lire manuellement la sortie de l'appareil. Sur Linux ou macOS, vous pouvez le faire avec la commande `screen`. Sous Windows, vous pouvez utiliser un programme tel que TeraTerm.

```
Screen: screen /dev/cu.usbserial 115200
```

```
TeraTerm: Use the above-provided settings to set the fields explicitly in
the GUI.
```

Problèmes liés à la chaîne d'outils de développement

Cette section explique les problèmes qui peuvent survenir avec votre chaîne d'outils.

Code Composer Studio sur Ubuntu

Les dernières versions d'Ubuntu (17.10 et 18.04) incluent une version du package `glibc` qui n'est pas compatible avec Code Composer Studio 7.x. Nous vous recommandons d'installer Code Composer Studio 8.2 ou version ultérieure.

Voici les symptômes d'incompatibilité possibles :

- FreeRTOS ne parvient pas à se créer ou à flasher sur votre appareil.
- Le programme d'installation de Code Composer Studio peut rester bloqué.
- Aucune sortie de journal ne s'affiche dans la console lors du processus de génération ou flash.
- La commande de génération tente de se lancer en mode graphique même lorsqu'elle est appelée en mode sans tête.

Logging

Les journaux IDT pour FreeRTOS sont placés en un seul endroit. À partir du répertoire IDT racine, ces fichiers sont disponibles sous `results/`*`execution-id`*`/` :

- `FRQ_Report.xml`
- `awsiotdevicetester_report.xml`
- `logs/`*`test_group_id`*`__`*`test_case_id`*`.log`

`FRQ_Report.xml` et `logs/`*`test_group_id`*`__`*`test_case_id`*`.log` sont les journaux les plus importants à examiner. `FRQ_Report.xml` contient des informations sur les cas

de test qui ont échoué avec un message d'erreur spécifique. Vous pouvez ensuite utiliser `logs/test_group_id__test_case_id.log` pour approfondir le problème afin d'obtenir plus de contexte.

Erreurs de la console

Lorsqu'il AWS IoT Device Tester est exécuté, les défaillances sont signalées à la console par de brefs messages. Recherchez dans `results/execution-id/logs/test_group_id__test_case_id.log` pour en savoir plus sur l'erreur.

Erreurs de journal

Chaque exécution de suite de tests a un ID d'exécution unique qui est utilisé pour créer un dossier nommé `results/execution-id`. Les journaux des cas de test individuels figurent dans le répertoire `results/execution-id/logs`. Utilisez la sortie de la console IDT pour FreeRTOS pour trouver l'identifiant d'exécution, l'identifiant du cas de test et l'identifiant du groupe de test du scénario de test qui a échoué. Utilisez ensuite ces informations pour rechercher et ouvrir le fichier journal correspondant à ce scénario de test nommé `results/execution-id/logs/test_group_id__test_case_id.log` Les informations contenues dans ce fichier incluent la sortie complète de la commande flash et de la compilation, la sortie de l'exécution du test et la sortie de AWS IoT Device Tester console plus détaillée.

Problèmes liés au compartiment S3

Si vous appuyez CTRL+C pendant l'exécution d'IDT, IDT lancera un processus de nettoyage. Une partie de ce nettoyage consiste à supprimer les ressources Amazon S3 créées dans le cadre des tests IDT. Si le nettoyage ne peut pas être terminé, il se peut que vous rencontriez un problème lié à la création d'un trop grand nombre de compartiments Amazon S3. Cela signifie que la prochaine fois que vous exécuterez IDT, les tests commenceront à échouer.

Si vous appuyez CTRL+C pour arrêter IDT, vous devez le laisser terminer le processus de nettoyage pour éviter ce problème. Vous pouvez également supprimer de votre compte les compartiments Amazon S3 créés manuellement.

Résoudre les erreurs liées au délai d'expiration

Si vous constatez des erreurs de délai d'expiration lors de l'exécution d'une suite de tests, augmentez le délai d'expiration en spécifiant un facteur multiplicateur de délai d'expiration. Ce facteur est appliqué à la valeur de délai d'expiration par défaut. Toute valeur configurée pour cet indicateur

doit être supérieure ou égale à 1.0. Pour utiliser le multiplicateur de délai d'expiration, employez l'indicateur `--timeout-multiplier` lors de l'exécution de la suite de tests.

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
```

Fonctionnalité cellulaire et AWS frais

Lorsque la `Cellular` fonctionnalité est configurée `Yes` dans votre `device.JSON` fichier, vous FullSecureSockets utiliserez les instances T.micro EC2 pour exécuter les tests, ce qui peut entraîner des frais supplémentaires pour votre compte. AWS Pour plus d'informations, consultez Tarification Amazon EC2.

Politique de génération de rapports de qualification

Les rapports de qualification ne sont générés que par les versions AWS IoT Device Tester (IDT) compatibles avec les versions de FreeRTOS publiées au cours des deux dernières années. Si vous avez des questions concernant la politique d'assistance, veuillez contacter AWS Support.

Comprenez le AWS Politique gérée pour AWS IoT Device Tester

Une politique AWS gérée est une politique autonome créée et administrée par AWS. AWS les politiques gérées sont conçues pour fournir des autorisations pour de nombreux cas d'utilisation courants afin que vous puissiez commencer à attribuer des autorisations aux utilisateurs, aux groupes et aux rôles.

N'oubliez pas que les politiques AWS gérées peuvent ne pas accorder d'autorisations de moindre privilège pour vos cas d'utilisation spécifiques, car elles sont accessibles à tous les AWS clients. Nous vous recommandons de réduire encore les autorisations en définissant des politiques gérées par le client qui sont propres à vos cas d'utilisation.

Vous ne pouvez pas modifier les autorisations définies dans les politiques AWS gérées. Si les autorisations définies dans une politique AWS gérée sont AWS mises à jour, la mise à jour affecte toutes les identités principales (utilisateurs, groupes et rôles) auxquelles la politique est attachée. AWS est le plus susceptible de mettre à jour une politique AWS gérée lorsqu'une nouvelle politique Service AWS est lancée ou lorsque de nouvelles opérations d'API sont disponibles pour les services existants.

Pour plus d'informations, consultez Politiques gérées par AWS dans le Guide de l'utilisateur IAM.

Rubriques

- AWS politique gérée : AWS IoTDeviceTesterForFreeRTOSFullAccess
- Mises à jour de AWS stratégies gérées

AWS politique gérée : AWS IoTDeviceTesterForFreeRTOSFullAccess

La politique `AWSIoTDeviceTesterForFreeRTOSFullAccess` gérée contient les AWS IoT Device Tester autorisations suivantes pour la vérification des versions, les fonctionnalités de mise à jour automatique et la collecte de métriques.

Détails de l'autorisation

Cette politique inclut les autorisations suivantes :

- `iot-device-tester:SupportedVersion`

  Accorde AWS IoT Device Tester l'autorisation de récupérer la liste des produits pris en charge, des suites de tests et des versions IDT.
- `iot-device-tester:LatestIdt`

  Accorde AWS IoT Device Tester l'autorisation de récupérer la dernière version d'IDT disponible au téléchargement.
- `iot-device-tester:CheckVersion`

  Accorde AWS IoT Device Tester l'autorisation de vérifier la compatibilité des versions pour IDT, les suites de tests et les produits.
- `iot-device-tester:DownloadTestSuite`

  AWS IoT Device Tester Autorise le téléchargement des mises à jour de la suite de tests.

- `iot-device-tester:SendMetrics`

  Accorde AWS l'autorisation de collecter des statistiques relatives à l'utilisation AWS IoT Device Tester interne.

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"
                    ]
                }
            }
        }
    ]
}
```

Mises à jour de AWS stratégies gérées

Vous pouvez consulter les informations relatives aux mises à jour des politiques AWS gérées à AWS IoT Device Tester partir du moment où ce service a commencé à suivre ces modifications.

Version	Modifier	Description	Date
7 (Dernier)	Restructuration des `ec2:CreateTags` conditions.	Suppression de l'utilisation de`ForAnyValues` .	6/14/2023
6	`freertos:ListHardwarePlatforms` Supprimé de la politique.	Suppression des autorisations car cette action est obsolète depuis le 1er mars 2023.	6/2/2023
5	Des autorisations ont été ajoutées pour exécuter des tests sur le serveur Echo à l'aide d'EC2.	Cela permet de démarrer et d'arrêter une instance EC2 dans les AWS comptes des clients.	12/15/2020
4	Ajouté `iot:CancelJobExecution` .	Cette autorisation annule les jobs OTA.	7/17/2020

Version	Modifier	Description	Date
3	Les autorisations suivantes ont été ajoutées : • `iot-device-tester:DownloadTestSuite` , • `iot-device-tester:CheckVersion` , • `iot-device-tester:LatestIdt` , • `iot-device-tester:SupportedVersion` .	• `iot-device-tester:DownloadTestSuite` — AWS IoT Device Tester Autorise le téléchargement des mises à jour de la suite de tests, • `iot-device-tester:CheckVersion` — AWS IoT Device Tester Autorise à vérifier la compatibilité des versions pour IDT, les suites de tests et les produits, • `iot-device-tester:LatestIdt` — AWS IoT Device Tester Autorise à récupérer la dernière version d'IDT disponible en téléchargement, • `iot-device-tester:SupportedVersion` — Accorde AWS IoT Device Tester	3/23/2020

Version	Modifier	Description	Date
		l'autorisation de récupérer la liste des produits, suites de tests et versions IDT pris en charge.	
2	`iot-device-tester:SendMetrics` Autorisations ajoutées.	Accorde AWS l'autorisation de collecter des statistiques relatives à l'utilisation AWS IoT Device Tester interne.	2/18/2020
1	Première version.		2/12/2020

Comprendre la politique de support pour AWS IoT Device Tester

> **Important**
>
> Depuis octobre 2022, AWS IoT Device Tester AWS IoT FreeRTOS Qualification (FRQ) 1.0 ne génère pas de rapports de qualification signés. Vous ne pouvez pas qualifier les nouveaux appareils AWS IoT FreeRTOS à répertorier dans le catalogue des appareils partenaires dans AWS le cadre du programme de qualification des appareils utilisant AWS les versions IDT FRQ 1.0. Bien que vous ne puissiez pas qualifier les appareils FreeRTOS avec IDT FRQ 1.0, vous pouvez continuer à tester vos appareils FreeRTOS avec FRQ 1.0. Nous vous recommandons d'utiliser IDT FRQ 2.0 pour qualifier et répertorier les appareils FreeRTOS dans le catalogue d'appareils partenaires.AWS

AWS IoT Device Tester for FreeRTOS est un outil d'automatisation des tests permettant de valider le port FreeRTOS sur les appareils. En outre, vous pouvez qualifier vos appareils FreeRTOS et les répertorier dans AWS le catalogue des appareils partenaires. Le AWS IoT Device Tester for FreeRTOS prend en charge la validation et la qualification des bibliothèques FreeRTOS supportées à long terme (LTS) disponibles sur GitHub at FreeRTOS/FreeRTOS-LTS , et de la gamme principale de FreeRTOS disponible sur. FreeRTOS/FreeRTOS Nous vous recommandons d'utiliser les versions

les plus récentes de FreeRTOS et de FreeRTOS AWS IoT Device Tester pour valider et qualifier vos appareils.

En effet FreeRTOS-LTS, IDT prend en charge la validation et la qualification de la version FreeRTOS 202210 LTS. Cliquez ici pour plus d'informations sur les versions de FreeRTOS LTS et leur calendrier de maintenance. Une fois la période de support de ces versions LTS terminée, vous pouvez toujours poursuivre la validation, mais IDT ne générera pas de rapport vous permettant de soumettre votre appareil à des fins de qualification.

Pour les principaux FreeRTOS disponibles sur, nous prenons en charge la validation et la qualification de toutes les versions publiées FreeRTOS/FreeRTOS au cours des six derniers mois, ou des deux versions précédentes de FreeRTOS si elles sont publiées à plus de six mois d'intervalle. Cliquez ici pour connaître les versions actuellement prises en charge. Pour les versions non prises en charge de FreeRTOS, vous pouvez toujours poursuivre la validation, mais IDT ne générera pas de rapport qui vous permettra de soumettre votre appareil à des fins de qualification.

Consultez Versions prises en charge de AWS IoT Device Tester les dernières versions IDT et FreeRTOS prises en charge. Vous pouvez utiliser n'importe laquelle des versions prises en charge AWS IoT Device Tester de la version correspondante de FreeRTOS pour tester ou qualifier votre appareil. Si vous continuez à utiliser leVersions IDT non prises en charge pour FreeRTOS, vous ne recevrez pas les dernières corrections de bogues ou mises à jour.

Pour toute question concernant la politique d'assistance, contactez AWS le support client.

Sécurité dans AWS

La sécurité du cloud AWS est la priorité absolue. En tant que AWS client, vous bénéficiez d'un centre de données et d'une architecture réseau conçus pour répondre aux exigences des entreprises les plus sensibles en matière de sécurité.

La sécurité est une responsabilité partagée entre vous AWS et vous. Le modèle de responsabilité partagée décrit cette notion par les termes sécurité du cloud et sécurité dans le cloud :

- Sécurité du cloud : AWS est chargée de protéger l'infrastructure qui exécute les AWS services dans le AWS cloud. AWS vous fournit également des services que vous pouvez utiliser en toute sécurité. L'efficacité de notre sécurité est régulièrement testée et vérifiée par des auditeurs tiers dans le cadre des programmes de conformitéAWS. Pour en savoir plus sur les programmes de conformité qui s'appliquent à un AWS service, consultez la section AWS Services concernés par programme de conformité.
- Sécurité dans le cloud — Votre responsabilité est déterminée par le AWS service que vous utilisez. Vous êtes également responsable d'autres facteurs, y compris la sensibilité de vos données, les exigences de votre organisation, et la législation et la réglementation applicables.

Cette documentation vous aidera à comprendre comment appliquer le modèle de responsabilité partagée lors de son utilisation AWS. Les rubriques suivantes expliquent comment procéder à la configuration AWS pour atteindre vos objectifs de sécurité et de conformité. Vous apprendrez également à utiliser les AWS services qui peuvent vous aider à surveiller et à sécuriser vos AWS ressources.

Pour des informations plus détaillées sur AWS IoT la sécurité, voir Sécurité et identité pour AWS IoT.

Rubriques

- Identity and Access Management pour FreeRTOS
- Validation de la conformité
- Résilience dans AWS
- Sécurité de l'infrastructure dans FreeRTOS

Identity and Access Management pour FreeRTOS

Gestion des identités et des accès AWS (IAM) est un outil Service AWS qui permet à un administrateur de contrôler en toute sécurité l'accès aux AWS ressources. Les administrateurs IAM contrôlent qui peut être authentifié (connecté) et autorisé (autorisé) à utiliser les ressources FreeRTOS. IAM est un Service AWS outil que vous pouvez utiliser sans frais supplémentaires.

Rubriques

- Public ciblé
- Authentification par des identités
- Gestion de l'accès à l'aide de politiques
- Comment FreeRTOS fonctionne avec IAM
- Identity-based exemples de politiques pour FreeRTOS
- Résoudre les problèmes d'identité et d'accès à FreeRTOS

Public ciblé

La façon dont vous utilisez Gestion des identités et des accès AWS (IAM) varie en fonction de votre rôle :

- Utilisateur du service : demandez des autorisations à votre administrateur si vous ne pouvez pas accéder aux fonctionnalités (voir Résoudre les problèmes d'identité et d'accès à FreeRTOS)
- Administrateur du service : déterminez l'accès des utilisateurs et soumettez les demandes d'autorisation (voir Comment FreeRTOS fonctionne avec IAM)
- Administrateur IAM : rédigez des politiques pour gérer l'accès (voir Identity-based exemples de politiques pour FreeRTOS)

Authentification par des identités

L'authentification est la façon dont vous vous connectez à AWS l'aide de vos informations d'identification. Vous devez être authentifié en tant qu'utilisateur IAM ou en assumant un rôle IAM. Utilisateur racine d'un compte AWS

Vous pouvez vous connecter en tant qu'identité fédérée à l'aide d'informations d'identification provenant d'une source d'identité telle que AWS IAM Identity Center (IAM Identity Center), d'une authentification unique ou d'informations d'identification. Google/Facebook Pour plus d'informations

sur la connexion, consultez Connexion à votre Compte AWS dans le Guide de l'utilisateur Connexion à AWS .

Pour l'accès par programmation, AWS fournit un SDK et une CLI pour signer les demandes de manière cryptographique. Pour plus d'informations, consultez Signature AWS Version 4 pour les demandes d'API dans le Guide de l'utilisateur IAM.

Compte AWS utilisateur root

Lorsque vous créez un Compte AWS, vous commencez par une seule identité de connexion appelée utilisateur Compte AWS root qui dispose d'un accès complet à toutes Services AWS les ressources. Il est vivement déconseillé d'utiliser l'utilisateur racine pour vos tâches quotidiennes. Pour les tâches qui requièrent des informations d'identification de l'utilisateur racine, consultez Tâches qui requièrent les informations d'identification de l'utilisateur racine dans le Guide de l'utilisateur IAM.

Identité fédérée

Il est recommandé d'obliger les utilisateurs humains à utiliser la fédération avec un fournisseur d'identité pour accéder à Services AWS l'aide d'informations d'identification temporaires.

Une identité fédérée est un utilisateur provenant de l'annuaire de votre entreprise, de votre fournisseur d'identité Web ou Directory Service qui y accède à Services AWS l'aide d'informations d'identification provenant d'une source d'identité. Les identités fédérées assument des rôles qui fournissent des informations d'identification temporaires.

Pour une gestion des accès centralisée, nous vous recommandons d'utiliser AWS IAM Identity Center. Pour plus d'informations, consultez Qu'est-ce que IAM Identity Center ? dans le Guide de l'utilisateur AWS IAM Identity Center .

Utilisateurs et groupes IAM

Un utilisateur IAM est une identité qui dispose d'autorisations spécifiques pour une seule personne ou application. Nous vous recommandons d'utiliser ces informations d'identification temporaires au lieu des utilisateurs IAM avec des informations d'identification à long terme. Pour plus d'informations, voir Exiger des utilisateurs humains qu'ils utilisent la fédération avec un fournisseur d'identité pour accéder à AWS l'aide d'informations d'identification temporaires dans le guide de l'utilisateur IAM.

Les groupes IAM spécifient une collection d'utilisateurs IAM et permettent de gérer plus facilement les autorisations pour de grands ensembles d'utilisateurs. Pour plus d'informations, consultez Cas d'utilisation pour les utilisateurs IAM dans le Guide de l'utilisateur IAM.

Rôles IAM

Un rôle IAM est une identité dotée d'autorisations spécifiques qui fournit des informations d'identification temporaires. Vous pouvez assumer un rôle en passant d'un rôle d'utilisateur à un rôle IAM (console) ou en appelant une opération d' AWS API AWS CLI ou d'API. Pour plus d'informations, consultez Méthodes pour endosser un rôle dans le Guide de l'utilisateur IAM.

Les rôles IAM sont utiles pour l'accès des utilisateurs fédérés, les autorisations temporaires des utilisateurs IAM, les accès intercompte, les accès entre services et les applications exécutées sur Amazon EC2. Pour plus d'informations, consultez Accès intercompte aux ressources dans IAM dans le Guide de l'utilisateur IAM.

Gestion de l'accès à l'aide de politiques

Vous contrôlez l'accès en AWS créant des politiques et en les associant à AWS des identités ou à des ressources. Une politique définit les autorisations lorsqu'elles sont associées à une identité ou à une ressource. AWS évalue ces politiques lorsqu'un directeur fait une demande. La plupart des politiques sont stockées AWS sous forme de documents JSON. Pour plus d'informations les documents de politique JSON, consultez Vue d'ensemble des politiques JSON dans le Guide de l'utilisateur IAM.

À l'aide de politiques, les administrateurs précisent qui a accès à quoi en définissant quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

Par défaut, les utilisateurs et les rôles ne disposent d'aucune autorisation. Un administrateur IAM crée des politiques IAM et les ajoute aux rôles, que les utilisateurs peuvent ensuite assumer. Les politiques IAM définissent les autorisations quelle que soit la méthode que vous utilisez pour exécuter l'opération.

Identity-based politiques

Identity-based les politiques sont des documents de politique d'autorisation JSON que vous attachez à une identité (utilisateur, groupe ou rôle). Ces politiques contrôlent les actions que peuvent exécuter ces identités, sur quelles ressources et dans quelles conditions. Pour découvrir comment créer une politique basée sur l'identité, consultez Définition d'autorisations IAM personnalisées avec des politiques gérées par le client dans le Guide de l'utilisateur IAM.

Identity-based les politiques peuvent être des politiques intégrées (intégrées directement dans une seule identité) ou des politiques gérées (politiques autonomes associées à plusieurs identités).

Pour découvrir comment choisir entre des politiques gérées et en ligne, consultez Choix entre les politiques gérées et les politiques en ligne dans le Guide de l'utilisateur IAM.

Resource-based politiques

Resource-based les politiques sont des documents de politique JSON que vous attachez à une ressource. Les exemples incluent les politiques de confiance de rôle IAM et les stratégies de compartiment Amazon S3. Dans les services qui sont compatibles avec les politiques basées sur les ressources, les administrateurs de service peuvent les utiliser pour contrôler l'accès à une ressource spécifique. Vous devez spécifier un principal dans une politique basée sur les ressources.

Resource-based les politiques sont des politiques intégrées qui se trouvent dans ce service. Vous ne pouvez pas utiliser les politiques AWS gérées par IAM dans une stratégie basée sur les ressources.

Autres types de politique

AWS prend en charge des types de politiques supplémentaires qui peuvent définir les autorisations maximales accordées par les types de politiques les plus courants :

- Limites d'autorisations : une limite des autorisations définit le nombre maximum d'autorisations qu'une politique basée sur l'identité peut accorder à une entité IAM. Pour plus d'informations, consultez Limites d'autorisations pour des entités IAM dans le Guide de l'utilisateur IAM.
- Politiques de contrôle des services (SCP) : spécifient les autorisations maximales pour une organisation ou une unité organisationnelle dans AWS Organizations. Pour plus d'informations, consultez Politiques de contrôle de service dans le Guide de l'utilisateur AWS Organizations .
- Politiques de contrôle des ressources (RCP) : définissent les autorisations maximales disponibles pour les ressources de votre organisation. Pour plus d'informations, consultez Politiques de contrôle des ressources (RCP) dans le Guide de l'utilisateur AWS Organizations .
- Politiques de session : politiques avancées que vous passez en tant que paramètre lorsque vous créez par programmation une session temporaire pour un rôle ou un utilisateur fédéré. Pour plus d'informations, consultez Politiques de session dans le Guide de l'utilisateur IAM.

Plusieurs types de politique

Lorsque plusieurs types de politiques s'appliquent à la requête, les autorisations en résultant sont plus compliquées à comprendre. Pour savoir comment AWS déterminer s'il faut autoriser une demande lorsque plusieurs types de politiques sont impliqués, consultez la section Logique d'évaluation des politiques dans le guide de l'utilisateur IAM.

Comment FreeRTOS fonctionne avec IAM

Avant d'utiliser IAM pour gérer l'accès à FreeRTOS, découvrez quelles fonctionnalités IAM peuvent être utilisées avec FreeRTOS.

Fonctionnalités IAM que vous pouvez utiliser avec FreeRTOS

Fonctionnalité IAM	Support FreeRTOS
Identity-based politiques	Oui
Resource-based politiques	Non
Actions de politique	Oui
Ressources de politique	Oui
Clés de condition de politique (spécifiques au service)	Oui
ACL	Non
ABAC (identifications dans les politiques)	Partielle
Informations d'identification temporaires	Oui
Autorisations de principal	Oui
Rôles de service	Oui
Service-linked rôles	Non

Pour obtenir une vue d'ensemble de la façon dont FreeRTOS et les AWS autres services fonctionnent avec la plupart des fonctionnalités IAM, AWS consultez les services compatibles avec IAM dans le guide de l'utilisateur IAM.

Identity-based politiques pour FreeRTOS

Prend en charge les politiques basées sur l'identité : oui

Identity-based les politiques sont des documents de politique d'autorisation JSON que vous pouvez associer à une identité, telle qu'un utilisateur IAM, un groupe d'utilisateurs ou un rôle. Ces politiques contrôlent quel type d'actions des utilisateurs et des rôles peuvent exécuter, sur quelles ressources et dans quelles conditions. Pour découvrir comment créer une politique basée sur l'identité, consultez Définition d'autorisations IAM personnalisées avec des politiques gérées par le client dans le Guide de l'utilisateur IAM.

Avec les politiques IAM basées sur l'identité, vous pouvez spécifier des actions et ressources autorisées ou refusées, ainsi que les conditions dans lesquelles les actions sont autorisées ou refusées. Pour découvrir tous les éléments que vous utilisez dans une politique JSON, consultez Références des éléments de politique JSON IAM dans le Guide de l'utilisateur IAM.

Identity-based exemples de politiques pour FreeRTOS

Pour consulter des exemples de politiques FreeRTOS basées sur l'identité, consultez. Identity-based exemples de politiques pour FreeRTOS

Resource-based politiques au sein de FreeRTOS

Prend en charge les politiques basées sur les ressources : non

Resource-based les politiques sont des documents de politique JSON que vous attachez à une ressource. Par exemple, les politiques de confiance de rôle IAM et les politiques de compartiment Amazon S3 sont des politiques basées sur les ressources. Dans les services qui sont compatibles avec les politiques basées sur les ressources, les administrateurs de service peuvent les utiliser pour contrôler l'accès à une ressource spécifique. Pour la ressource dans laquelle se trouve la politique, cette dernière définit quel type d'actions un principal spécifié peut effectuer sur cette ressource et dans quelles conditions. Vous devez spécifier un principal dans une politique basée sur les ressources. Les principaux peuvent inclure des comptes, des utilisateurs, des rôles, des utilisateurs fédérés ou. Services AWS

Pour permettre un accès intercompte, vous pouvez spécifier un compte entier ou des entités IAM dans un autre compte en tant que principal dans une politique basée sur les ressources. Pour plus d'informations, consultez Accès intercompte aux ressources dans IAM dans le Guide de l'utilisateur IAM.

Actions politiques pour FreeRTOS

Prend en charge les actions de politique : oui

Les administrateurs peuvent utiliser les politiques AWS JSON pour spécifier qui a accès à quoi. C'est-à-dire, quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

L'élément `Action` d'une politique JSON décrit les actions que vous pouvez utiliser pour autoriser ou refuser l'accès à une politique. Intégration d'actions dans une politique afin d'accorder l'autorisation d'exécuter les opérations associées.

Pour consulter la liste des actions FreeRTOS, voir Actions définies par FreeRTOS dans la référence d'autorisation de service.

Les actions politiques dans FreeRTOS utilisent le préfixe suivant avant l'action :

```
awes
```

Pour indiquer plusieurs actions dans une seule déclaration, séparez-les par des virgules.

```
"Action": [
      "awes:action1",
      "awes:action2"
         ]
```

Pour consulter des exemples de politiques FreeRTOS basées sur l'identité, consultez. Identity-based exemples de politiques pour FreeRTOS

Ressources relatives aux politiques pour FreeRTOS

Prend en charge les ressources de politique : oui

Les administrateurs peuvent utiliser les politiques AWS JSON pour spécifier qui a accès à quoi. C'est-à-dire, quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

L'élément de politique JSON `Resource` indique le ou les objets auxquels l'action s'applique. Il est recommandé de définir une ressource à l'aide de son Amazon Resource Name (ARN). Pour les actions qui ne sont pas compatibles avec les autorisations de niveau ressource, utilisez un caractère générique (*) afin d'indiquer que l'instruction s'applique à toutes les ressources.

```
"Resource": "*"
```

Pour consulter la liste des types de ressources FreeRTOS et de leurs ARN, voir Ressources définies par FreeRTOS dans la référence d'autorisation de service. Pour savoir avec quelles actions vous pouvez spécifier l'ARN de chaque ressource, voir Actions définies par FreeRTOS.

Pour consulter des exemples de politiques FreeRTOS basées sur l'identité, consultez. Identity-based exemples de politiques pour FreeRTOS

Clés de conditions de politique pour FreeRTOS

Prend en charge les clés de condition de politique spécifiques au service : oui

Les administrateurs peuvent utiliser les politiques AWS JSON pour spécifier qui a accès à quoi. C'est-à-dire, quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

L'élément `Condition` indique à quel moment les instructions s'exécutent en fonction de critères définis. Vous pouvez créer des expressions conditionnelles qui utilisent des opérateurs de condition, tels que les signes égal ou inférieur à, pour faire correspondre la condition de la politique aux valeurs de la demande. Pour voir toutes les clés de condition AWS globales, voir les clés de contexte de condition AWS globales dans le guide de l'utilisateur IAM.

Pour consulter la liste des clés de condition FreeRTOS, voir Clés de condition pour FreeRTOS dans la référence d'autorisation de service. Pour savoir avec quelles actions et ressources vous pouvez utiliser une clé de condition, voir Actions définies par FreeRTOS.

Pour consulter des exemples de politiques FreeRTOS basées sur l'identité, consultez. Identity-based exemples de politiques pour FreeRTOS

ACL dans FreeRTOS

Prend en charge les ACL : non

Les listes de contrôle d'accès (ACL) vérifie quels principaux (membres de compte, utilisateurs ou rôles) ont l'autorisation d'accéder à une ressource. Les listes de contrôle d'accès sont similaires aux politiques basées sur les ressources, bien qu'elles n'utilisent pas le format de document de politique JSON.

ABAC avec FreeRTOS

Prend en charge ABAC (identifications dans les politiques) : partiellement

Attribute-based le contrôle d'accès (ABAC) est une stratégie d'autorisation qui définit les autorisations en fonction d'attributs appelés balises. Vous pouvez associer des balises aux entités et aux AWS ressources IAM, puis concevoir des politiques ABAC pour autoriser les opérations lorsque la balise du principal correspond à la balise de la ressource.

Pour contrôler l'accès basé sur des étiquettes, vous devez fournir les informations d'étiquette dans l'élément de condition d'une politique utilisant les clés de condition `aws:ResourceTag/key-name`, `aws:RequestTag/key-name` ou `aws:TagKeys`.

Si un service prend en charge les trois clés de condition pour tous les types de ressources, alors la valeur pour ce service est Oui. Si un service prend en charge les trois clés de condition pour certains types de ressources uniquement, la valeur est Partielle.

Pour plus d'informations sur ABAC, consultez Définition d'autorisations avec l'autorisation ABAC dans le Guide de l'utilisateur IAM. Pour accéder à un didacticiel décrivant les étapes de configuration de l'ABAC, consultez Utilisation du contrôle d'accès par attributs (ABAC) dans le Guide de l'utilisateur IAM.

Utilisation d'informations d'identification temporaires avec FreeRTOS

Prend en charge les informations d'identification temporaires : oui

Les informations d'identification temporaires fournissent un accès à court terme aux AWS ressources et sont automatiquement créées lorsque vous utilisez la fédération ou que vous changez de rôle. AWS recommande de générer dynamiquement des informations d'identification temporaires au lieu d'utiliser des clés d'accès à long terme. Pour plus d'informations, consultez Informations d'identification de sécurité temporaires dans IAM et Services AWS compatibles avec IAM dans le Guide de l'utilisateur IAM.

Cross-service autorisations principales pour FreeRTOS

Prend en charge les sessions d'accès direct (FAS) : oui

Les sessions d'accès direct (FAS) utilisent les autorisations du principal appelant et Service AWS, combinées Service AWS à la demande d'envoi de demandes aux services en aval. Pour plus de détails sur la politique relative à la transmission de demandes FAS, consultez la section Sessions de transmission d'accès.

Rôles de service pour FreeRTOS

Prend en charge les rôles de service : oui

Un rôle de service est un rôle IAM qu'un service endosse pour accomplir des actions en votre nom. Un administrateur IAM peut créer, modifier et supprimer un rôle de service à partir d'IAM. Pour plus d'informations, consultez Création d'un rôle pour la délégation d'autorisations à un Service AWS dans le Guide de l'utilisateur IAM.

> **Warning**
>
> La modification des autorisations associées à un rôle de service peut interrompre les fonctionnalités de FreeRTOS. Modifiez les rôles de service uniquement lorsque FreeRTOS fournit des instructions à cet effet.

Service-linked rôles pour FreeRTOS

Prend en charge les rôles liés à un service : non

Un rôle lié à un service est un type de rôle de service lié à un. Service AWS Le service peut assumer le rôle d'effectuer une action en votre nom. Service-linked les rôles apparaissent dans votre fichier Compte AWS et appartiennent au service. Un administrateur IAM peut consulter, mais ne peut pas modifier, les autorisations concernant les rôles liés à un service.

Pour plus d'informations sur la création ou la gestion des rôles liés à un service, consultez Services AWS qui fonctionnent avec IAM. Recherchez un service dans le tableau qui inclut un `Yes` dans la colonne des Service-linked rôles. Choisissez le lien Oui pour consulter la documentation du rôle lié à ce service.

Identity-based exemples de politiques pour FreeRTOS

Par défaut, les utilisateurs et les rôles ne sont pas autorisés à créer ou à modifier des ressources FreeRTOS. Pour octroyer aux utilisateurs des autorisations d'effectuer des actions sur les ressources dont ils ont besoin, un administrateur IAM peut créer des politiques IAM.

Pour apprendre à créer une politique basée sur l'identité IAM à l'aide de ces exemples de documents de politique JSON, consultez Création de politiques IAM (console) dans le Guide de l'utilisateur IAM.

Pour plus de détails sur les actions et les types de ressources définis par FreeRTOS, y compris le format des ARN pour chacun des types de ressources, voir Actions, ressources et clés de condition pour FreeRTOS dans la référence d'autorisation de service.

Rubriques

- Bonnes pratiques en matière de politiques
- Utilisation de la console FreeRTOS
- Autorisation accordée aux utilisateurs pour afficher leurs propres autorisations

Bonnes pratiques en matière de politiques

Identity-based les politiques déterminent si quelqu'un peut créer, accéder ou supprimer des ressources FreeRTOS dans votre compte. Ces actions peuvent entraîner des frais pour votre Compte AWS. Lorsque vous créez ou modifiez des politiques basées sur l'identité, suivez ces instructions et recommandations :

- Commencez AWS par les politiques gérées et passez aux autorisations du moindre privilège : pour commencer à accorder des autorisations à vos utilisateurs et à vos charges de travail, utilisez les politiques AWS gérées qui accordent des autorisations pour de nombreux cas d'utilisation courants. Ils sont disponibles dans votre Compte AWS. Nous vous recommandons de réduire davantage les autorisations en définissant des politiques gérées par les AWS clients spécifiques à vos cas d'utilisation. Pour plus d'informations, consultez politiques gérées par AWS ou politiques gérées par AWS pour les activités professionnelles dans le Guide de l'utilisateur IAM.
- Accordez les autorisations de moindre privilège : lorsque vous définissez des autorisations avec des politiques IAM, accordez uniquement les autorisations nécessaires à l'exécution d'une seule tâche. Pour ce faire, vous définissez les actions qui peuvent être entreprises sur des ressources spécifiques dans des conditions spécifiques, également appelées autorisations de moindre privilège. Pour plus d'informations sur l'utilisation d'IAM pour appliquer des autorisations, consultez politiques et autorisations dans IAM dans le Guide de l'utilisateur IAM.
- Utilisez des conditions dans les politiques IAM pour restreindre davantage l'accès : vous pouvez ajouter une condition à vos politiques afin de limiter l'accès aux actions et aux ressources. Par exemple, vous pouvez écrire une condition de politique pour spécifier que toutes les demandes doivent être envoyées via SSL. Vous pouvez également utiliser des conditions pour accorder l'accès aux actions de service si elles sont utilisées par le biais d'un service spécifique Service AWS, tel que CloudFormation. Pour plus d'informations, consultez Conditions pour éléments de politique JSON IAM dans le Guide de l'utilisateur IAM.

- Utilisez l'Analyseur d'accès IAM pour valider vos politiques IAM afin de garantir des autorisations sécurisées et fonctionnelles : l'Analyseur d'accès IAM valide les politiques nouvelles et existantes de manière à ce que les politiques IAM respectent le langage de politique IAM (JSON) et les bonnes pratiques IAM. IAM Access Analyzer fournit plus de 100 vérifications de politiques et des recommandations exploitables pour vous aider à créer des politiques sécurisées et fonctionnelles. Pour plus d'informations, consultez Validation de politiques avec IAM Access Analyzer dans le Guide de l'utilisateur IAM.
- Exiger l'authentification multifactorielle (MFA) : si vous avez un scénario qui nécessite des utilisateurs IAM ou un utilisateur root, activez l'authentification MFA pour une sécurité accrue. Compte AWS Pour exiger la MFA lorsque des opérations d'API sont appelées, ajoutez des conditions MFA à vos politiques. Pour plus d'informations, consultez Sécurisation de l'accès aux API avec MFA dans le Guide de l'utilisateur IAM.

Pour plus d'informations sur les bonnes pratiques dans IAM, consultez Bonnes pratiques de sécurité dans IAM dans le Guide de l'utilisateur IAM.

Utilisation de la console FreeRTOS

Pour accéder à la console FreeRTOS, vous devez disposer d'un minimum d'autorisations. Ces autorisations doivent vous permettre de répertorier et de consulter les détails des ressources FreeRTOS de votre. Compte AWS Si vous créez une politique basée sur l'identité qui est plus restrictive que l'ensemble minimum d'autorisations requis, la console ne fonctionnera pas comme prévu pour les entités (utilisateurs ou rôles) tributaires de cette politique.

Il n'est pas nécessaire d'accorder des autorisations de console minimales aux utilisateurs qui appellent uniquement l'API AWS CLI ou l' AWS API. Autorisez plutôt l'accès à uniquement aux actions qui correspondent à l'opération d'API qu'ils tentent d'effectuer.

Pour garantir que les utilisateurs et les rôles peuvent toujours utiliser la console FreeRTOS, associez également le FreeRTOS ou la politique gérée aux `ConsoleAccess` entités `ReadOnly` AWS . Pour plus d'informations, consultez Ajout d'autorisations à un utilisateur dans le Guide de l'utilisateur IAM.

Autorisation accordée aux utilisateurs pour afficher leurs propres autorisations

Cet exemple montre comment créer une politique qui permet aux utilisateurs IAM d'afficher les politiques en ligne et gérées attachées à leur identité d'utilisateur. Cette politique inclut les autorisations permettant d'effectuer cette action sur la console ou par programmation à l'aide de l'API AWS CLI or 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": "*"
        }
    ]
}
```

Résoudre les problèmes d'identité et d'accès à FreeRTOS

Utilisez les informations suivantes pour vous aider à diagnostiquer et à résoudre les problèmes courants que vous pouvez rencontrer lorsque vous travaillez avec FreeRTOS et IAM.

Rubriques

- Je ne suis pas autorisé à effectuer une action dans FreeRTOS

- Je ne suis pas autorisé à effectuer iam : PassRole
- Autoriser les personnes extérieures à moi Compte AWS pour accéder à mes ressources FreeRTOS

Je ne suis pas autorisé à effectuer une action dans FreeRTOS

Si vous recevez une erreur qui indique que vous n'êtes pas autorisé à effectuer une action, vos politiques doivent être mises à jour afin de vous permettre d'effectuer l'action.

L'exemple d'erreur suivant se produit quand l'utilisateur IAM `mateojackson` tente d'utiliser la console pour afficher des informations détaillées sur une ressource *`my-example-widget`* fictive, mais ne dispose pas des autorisations `awes:`*`GetWidget`* fictives.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 awes:GetWidget on resource: my-example-widget
```

Dans ce cas, la politique qui s'applique à l'utilisateur `mateojackson` doit être mise à jour pour autoriser l'accès à la ressource *`my-example-widget`* à l'aide de l'action `awes:`*`GetWidget`*.

Si vous avez besoin d'aide, contactez votre AWS administrateur. Votre administrateur vous a fourni vos informations d'identification de connexion.

Je ne suis pas autorisé à effectuer iam : PassRole

Si vous recevez un message d'erreur indiquant que vous n'êtes pas autorisé à effectuer l'`iam:PassRole`action, vos politiques doivent être mises à jour pour vous permettre de transmettre un rôle à FreeRTOS.

Certains vous Services AWS permettent de transmettre un rôle existant à ce service au lieu de créer un nouveau rôle de service ou un rôle lié à un service. Pour ce faire, vous devez disposer des autorisations nécessaires pour transmettre le rôle au service.

L'exemple d'erreur suivant se produit lorsqu'un utilisateur IAM nommé `marymajor` essaie d'utiliser la console pour effectuer une action dans FreeRTOS. Toutefois, l'action nécessite que le service ait des autorisations accordées par un rôle de service. Mary n'est pas autorisée à transmettre le rôle au service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

Dans ce cas, les politiques de Mary doivent être mises à jour pour lui permettre d'exécuter l'action `iam:PassRole`.

Si vous avez besoin d'aide, contactez votre AWS administrateur. Votre administrateur vous a fourni vos informations d'identification de connexion.

Autoriser les personnes extérieures à moi Compte AWS pour accéder à mes ressources FreeRTOS

Vous pouvez créer un rôle que les utilisateurs provenant d'autres comptes ou les personnes extérieures à votre organisation pourront utiliser pour accéder à vos ressources. Vous pouvez spécifier qui est autorisé à assumer le rôle. Pour les services qui prennent en charge les politiques basées sur les ressources ou les listes de contrôle d'accès (ACL), vous pouvez utiliser ces politiques pour donner l'accès à vos ressources.

Pour plus d'informations, consultez les éléments suivants :

- Pour savoir si FreeRTOS prend en charge ces fonctionnalités, consultez. Comment FreeRTOS fonctionne avec IAM
- Pour savoir comment fournir l'accès à vos ressources sur celles Comptes AWS que vous possédez, consultez la section Fournir l'accès à un utilisateur IAM dans un autre utilisateur Compte AWS que vous possédez dans le Guide de l'utilisateur IAM.
- Pour savoir comment fournir l'accès à vos ressources à des tiers Comptes AWS, consultez la section Fournir un accès à des ressources Comptes AWS détenues par des tiers dans le guide de l'utilisateur IAM.
- Pour savoir comment fournir un accès par le biais de la fédération d'identité, consultez Fournir un accès à des utilisateurs authentifiés en externe (fédération d'identité) dans le Guide de l'utilisateur IAM.
- Pour en savoir plus sur la différence entre l'utilisation des rôles et des politiques basées sur les ressources pour l'accès intercompte, consultez Accès intercompte aux ressources dans IAM dans le Guide de l'utilisateur IAM.

Validation de la conformité

Pour savoir si un programme Services AWS de conformité Service AWS s'inscrit dans le champ d'application de programmes de conformité spécifiques, consultez Services AWS la section de

conformité et sélectionnez le programme de conformité qui vous intéresse. Pour des informations générales, voir Programmes de AWS conformité Programmes AWS de .

Vous pouvez télécharger des rapports d'audit tiers à l'aide de AWS Artifact. Pour plus d'informations, voir Téléchargement de rapports dans AWS Artifact .

Votre responsabilité en matière de conformité lors de l'utilisation Services AWS est déterminée par la sensibilité de vos données, les objectifs de conformité de votre entreprise et les lois et réglementations applicables. Pour plus d'informations sur votre responsabilité en matière de conformité lors de l'utilisation Services AWS, consultez AWS la documentation de sécurité.

Résilience dans AWS

L'infrastructure AWS mondiale est construite autour des AWS régions et des zones de disponibilité. AWS Les régions fournissent plusieurs zones de disponibilité physiquement séparées et isolées, reliées par un réseau à latence faible, à haut débit et hautement redondant. Avec les zones de disponibilité, vous pouvez concevoir et exploiter des applications et des bases de données qui basculent automatiquement d'une zone de disponibilité à l'autre sans interruption. Les zones de disponibilité sont plus hautement disponibles, tolérantes aux pannes et évolutives que les infrastructures traditionnelles à un ou plusieurs centres de données.

Pour plus d'informations sur AWS les régions et les zones de disponibilité, consultez la section Infrastructure AWS mondiale.

Sécurité de l'infrastructure dans FreeRTOS

AWS les services gérés sont protégés par les procédures de sécurité du réseau AWS mondial décrites dans le livre blanc Amazon Web Services : présentation des processus de sécurité.

Vous utilisez des appels d'API AWS publiés pour accéder aux AWS services via le réseau. Les clients doivent prendre en charge le protocole TLS (Transport Layer Security) 1.2 ou version ultérieure. Nous recommandons TLS 1.3 ou version ultérieure. Les clients doivent également prendre en charge les suites de chiffrement à parfaite confidentialité (PFS), telles que Ephemeral (DHE) ou Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Diffie-Hellman La plupart des systèmes modernes tels que Java 7 et les versions ultérieures prennent en charge ces modes.

En outre, les demandes doivent être signées à l'aide d'un ID de clé d'accès et d'une clé d'accès secrète associée à un principal IAM. Vous pouvez également utiliser AWS Security Token Service

(AWS STS) pour générer des informations d’identification de sécurité temporaires et signer les demandes.

Amazon-FreeRTOS Guide de migration du référentiel Github

Si vous avez un projet FreeRTOS existant basé sur le référentiel amazon-freertos désormais obsolète, procédez comme suit :

1. Restez à jour avec les derniers correctifs de sécurité accessibles au public. Consultez la page des bibliothèques FreeRTOS LTS pour les mises à jour, ou abonnez-vous au dépôt pour recevoir FreeRTOS-LTS GitHub les derniers correctifs LTS contenant des corrections de bogues critiques et de sécurité. Vous pouvez télécharger ou cloner les derniers correctifs FreeRTOS LTS requis directement à partir des référentiels individuels. GitHub
2. Envisagez de refactoriser l'implémentation de l'interface de transport réseau afin d'optimiser votre plate-forme matérielle. Les API abstraites telles que les sockets sécurisés et les API Wifi ne sont pas requises par la dernière bibliothèque CoreMQTT. Voir Interface de transport pour plus de détails.

Annexe

Le tableau suivant fournit des recommandations pour tous les projets de démonstration, les bibliothèques existantes et les API abstraites du Amazon-FreeRTOS référentiel.

Bibliothèques et démos migrées

Nom	Type	Recommandations	
Noyau HTTP	démos et bibliothèque	Clonez ou téléchargez la bibliothèque CoreHTTP directement depuis le dépôt CoreHTTP (sous-module si vous utilisez git) dans l'organisation FreeRTOS Github. Les démos CoreHTTP se trouvent dans la distribution FreeRTOS principale. Pour plus de détails, consultez la page CoreHTTP.	
Noyau MQTT	démos et bibliothèque	Clonez ou téléchargez la bibliothèque CoreMQTT directement depuis le dépôt CoreMQTT (sous-module	

Nom	Type	Recommandations
		si vous utilisez git) dans l'organisation FreeRTOS Github. Les démos de CoreMQTT se trouvent dans la distribution principale de FreeRTOS. Pour plus de détails, consultez la page CoreMQTT.
noyau MQTT-Agent	démos et bibliothèque	Clonez ou téléchargez la MQTT-Agent bibliothèque principale directement depuis le MQTT-Agent référentiel principal (sous-module si vous utilisez git) dans l'organisation FreeRTOS Github. Les principales MQTT-Agent démos se trouvent dans le MQTT-Agent-Demos référentiel principal. Pour plus de détails, reportez-vous à la MQTT-Agent page principale.
device_defender_pour_aws	démos et bibliothèque	La bibliothèque AWS IoT Device Defender se trouve dans son référentiel au AWS GitHub sein de l'organisation. Clonez-le ou téléchargez-le (sous-module si vous utilisez git) directement depuis le dépôt AWS IoT Device Defender. Les démos de AWS IoT Device Defender se trouvent dans la distribution principale de FreeRTOS. Pour plus de détails, consultez la page AWS IoT Device Defender.

Nom	Type	Recommandations	
device_shadow_for_aws	démos et bibliothèque	La bibliothèque AWS IoT Device Shadow se trouve dans son référentiel au AWS GitHub sein de l'organisation. Clonez-le ou téléchargez-le (sous-module si vous utilisez git) directement depuis le dépôt AWS IoT Device (Shadow). Les démos de AWS IoT Device Shadow se trouvent dans la distribution principale de FreeRTOS. Pour plus de détails, consultez la page AWS IoT Device Shadow.	
jobs_for_aws	démos et bibliothèque	La bibliothèque AWS IoT d'emplois se trouve dans son référentiel au sein de l'AWS GitHub organisation. Clonez-le ou téléchargez-le (sous-module si vous utilisez git) directement depuis le dépôt AWS IoT Jobs. Les démos de AWS IoT Jobs se trouvent dans la distribution principale de FreeRTOS. Pour plus de détails, consultez la page AWS IoT Offres d'emploi.	

Nom	Type	Recommandations
OTA	démos et bibliothèque	La bibliothèque de mises à jour AWS IoT Over-The-Air (OTA) se trouve dans son référentiel au AWS GitHub sein de l'organisation. Clonez-le ou téléchargez-le (sous-module si vous utilisez git) directement depuis le dépôt AWS IoT OTA. Les démos AWS IoT OTA se trouvent dans la distribution principale de FreeRTOS. Pour plus de détails, consultez la page AWS IoT OTA.
CLI et FreeRTOS_ Plus_CLI	démos et bibliothèque	Un exemple de CLI est en cours d'exécution sur WinSim. Reportez-vous à la page de l'interface de ligne de commande FreeRTOS Plus pour plus de détails. Les intégrations de référence FreeRTOS IoT présentée s sur les plateformes NXP i.MX RT1060 et STM32U5 fournissent également des exemples de CLI sur du matériel réel.
journalisation	macro	Il existe des implémentations de la macro de journalisation pour des plateformes matérielles spécifiqu es utilisées par certaines bibliothè ques FreeRTOS. Reportez-vous à la page de journalisation pour savoir comment implémenter la macro de journalisation. Reportez-vous à l'une des références IoT proposées par FreeRTOS pour un exemple d'exécution sur du matériel réel.

Nom	Type	Recommandations	
greengrass connectivity	démo	[Migration en cours] Ce projet de démonstration partait du principe que la connectivité au cloud était disponible avant la connexion à un appareil AWS IoT Greengrass. Un nouveau projet démontrant les capacités locales d'authentification et de découverte est en cours de développement. Attendez-vous à ce que le nouveau projet de démonstration soit publié prochainement dans l'organisation FreeRTOS Github.	

Bibliothèques et démos obsolètes

Nom	Type	Recommandations	
BLE	démos et bibliothèques	La bibliothèque FreeRTOS BLE implémente le protocole propriétaire MQTT et prend en charge la publication et l'abonnement à des sujets MQTT via Bluetooth Low Energy (BLE) via un périphérique proxy tel qu'un téléphone mobile. Cela n'est plus obligatoire. Utilisez votre propre stack BLE ou une option tierce telle que NimBLE pour optimiser au mieux votre projet.	
dev_mode_key_provisioning	démos	Les intégrations de référence FreeRTOS IoT présentées sur les plateformes NXP i.MX RT1060, STM32U5_ESP32-C3ou sur les plateformes fournissent des	

Nom	Type	Recommandations
		exemples de provisionnement crucial à l'aide d'une CLI.
posix	abstraction et démo	L'utilisation n'est pas recommandée.
provisionnement wifi	exemple	Cet exemple montre comment fournir des WiFi informations d'identif ication sur un appareil à l'aide de la bibliothèque Amazon-FreeRTOS BLE. Reportez-vous à la référence FreeRTOS Featured IoT sur la plateforme ESP32C3 pour un exemple de provisionnement via BLE. WiFi
API abstraites héritées	code	Il s'agit d'API qui ont été créées pour fournir une interface abstraite pour diverses piles de logiciels tiers, modules de connectivité et plateform es de microcontrôleurs de divers fournisseurs. Par exemple, il existe des interfaces pour WiFi l'abstraction, des sockets sécurisés, etc. Ils sont pris en charge dans le Amazon-Fr eeRTOS référentiel et se trouvent dans le dossier `/libraries/ abstractions/` . Ces API ne sont pas requises lors de l'utilisation des bibliothèques FreeRTOS LTS.

Les bibliothèques et les démos du tableau ci-dessus ne recevront pas de correctifs de sécurité ni de corrections de bogues.

Third-party bibliothèques

Lorsque des démos Amazon-FreeRTOS utilisent des bibliothèques tierces, nous vous recommandons de les sous-modules directement à partir de leurs référentiels tiers.

- Cmock : clonez-le (sous-module si vous utilisez git) directement depuis le dépôt Cmock.
- jsmn : non recommandé et n'est plus pris en charge.
- lwip : clonez-le (sous-module si vous utilisez git) directement depuis le dépôt lwip-tcpip.
- lwip_osal : reportez-vous aux intégrations de référence proposées par FreeRTOS sur i.MX RT1060 ou STM32U5 pour savoir comment implémenter lwip_osal sur votre matériel. platform/board
- mbedtls : clonez-le (sous-module si vous utilisez git) directement depuis le dépôt. Mbed-TLS La configuration et les utilitaires mbedtls peuvent être réutilisés ; faites une copie locale dans ce cas.
- pkcs11 : clonez-le (sous-module si vous utilisez git) directement depuis la bibliothèque CorePKCS11 ou depuis le dépôt OASIS PKCS 11.
- tinycbor : clonez-le (sous-module si vous utilisez git) directement depuis le dépôt tinycbor.
- tinycrypt : nous vous recommandons d'utiliser les accélérateurs cryptographiques de votre plateforme MCU, s'ils sont disponibles. Si vous souhaitez continuer à utiliser tinycrypt, clonez-le (sous-module si vous utilisez git) directement depuis le dépôt tinycrypt.
- tracealyzer_recorder : clonez-le (sous-module si vous utilisez git) directement depuis le référentiel d'enregistreurs de traces de Percepio.
- unity : clonez-le (sous-module si vous utilisez git) directement depuis le ThrowTheSwitch/ Unitydépôt.
- win_pcap : win_pcap n'est plus maintenu. Nous vous recommandons d'utiliser libslirp, libpcap (posix) ou npcap à la place.

Tests de portage et tests d'intégration

Tous les tests du `/tests` dossier requis pour valider l'intégration des bibliothèques FreeRTOS ont été migrés vers le référentiel. FreeRTOS-Libraries-Integration-Tests Ils peuvent être utilisés pour tester la mise en œuvre de PAL et l'intégration de bibliothèques. Les mêmes tests sont utilisés par AWS IoT Device Tester (IDT) pour le programme de qualification des AWS appareils pour FreeRTOS.

Documentation archivée FreeRTOS

Ce contenu est archivé et n'est plus mis à jour. Il peut faire référence à des informations obsolètes ou à des fonctionnalités obsolètes. Pour obtenir les derniers conseils sur FreeRTOS, veuillez consulter les informations les plus récentes contenues dans ce guide de l'utilisateur de FreeRTOS ou dans toute autre documentation pertinente. AWS Pour obtenir des informations actualisées, voirQu'est-ce que FreeRTOS ?.

Rubriques

- Archive du guide de l'utilisateur de FreeRTOS
- Contenu du précédent guide de l'utilisateur de FreeRTOS

Archive du guide de l'utilisateur de FreeRTOS

Ces versions précédentes du guide de l'utilisateur de FreeRTOS peuvent être utilisées avec les versions de FreeRTOS LTS (support à long terme).

- Guide de l'utilisateur de FreeRTOS pour la version 202210.00 de FreeRTOS
- Guide de l'utilisateur de FreeRTOS pour FreeRTOS version 202012.00

Contenu du précédent guide de l'utilisateur de FreeRTOS

Ce contenu est obsolète mais fourni ici à titre de référence.

Voir Commencez avec FreeRTOS les liens vers le contenu récent.

Commencez avec FreeRTOS

> **Important**
>
> Cette page fait référence au Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Ce didacticiel pour démarrer avec FreeRTOS explique comment télécharger et configurer FreeRTOS sur une machine hôte, puis comment compiler et exécuter une application de démonstration simple sur une carte microcontrôleur qualifiée.

Tout au long de ce didacticiel, nous partons du principe que vous connaissez bien la AWS IoT console. AWS IoT Si tel n'est pas votre cas, nous vous recommandons de suivre le didacticiel Mise en route avec AWS IoT avant de poursuivre.

Rubriques :

- Application de démonstration FreeRTOS
- Premiers pas
- Résolution des problèmes de mise en route
- Utilisation de CMake avec FreeRTOS
- Developer-mode provisionnement des clés
- Board-specific guides de démarrage
- Prochaines étapes avec FreeRTOS

Application de démonstration FreeRTOS

> **Important**
>
> Cette page fait référence au Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

L'application de démonstration présentée dans ce didacticiel est la démo de l'agent CoreMQTT définie dans le fichier. *freertos*`/demos/coreMQTT_Agent/mqtt_agent_task.c` Il utilise le bibliothèque CoreMQTT pour se connecter au AWS cloud, puis publier périodiquement des messages sur un sujet MQTT hébergé par le courtier AWS IoT MQTT.

Une seule application de démonstration FreeRTOS peut être exécutée à la fois. Lorsque vous créez un projet de démonstration FreeRTOS, la première démo activée dans le fichier d'en-tête est *freertos*`/vendors/`*vendor*`/boards/`*board*`/aws_demos/config_files/`

`aws_demo_config.h` l'application qui s'exécute. Pour ce didacticiel, vous n'avez pas besoin d'activer ni de désactiver les démonstrations. La démo de l'agent CoreMQTT est activée par défaut.

Pour plus d'informations sur les applications de démonstration incluses dans FreeRTOS, consultez. Démos de FreeRTOS

Premiers pas

> **Important**
>
> Cette page fait référence au Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Pour commencer à utiliser FreeRTOS AWS IoT avec, vous devez avoir AWS un compte, un utilisateur autorisé à y accéder AWS IoT et les services cloud FreeRTOS. Vous devez également télécharger FreeRTOS et configurer le projet de démonstration FreeRTOS de votre forum pour qu'il fonctionne avec. AWS IoT Les sections suivantes vous guident au cours de ces exigences.

> **Note**
>
> - Si vous utilisez l'Espressif ESP32-DevKitC, ou le ESP-WROVER-KIT ESP32-WROOM-32SE, ignorez ces étapes et passez à. Commencer à utiliser l'Espressif ESP32-DevKitC et le ESP-WROVER-KIT
> - Si vous utilisez le Nordic nRF52840-DK, ignorez ces étapes et passez àDébuter avec le Nordic n RF52840-DK.

1. Configuration de votre AWS compte et autorisations
2. Enregistrement de votre carte MCU auprès AWS IoT
3. Téléchargement de FreeRTOS
4. Configuration des démos de FreeRTOS

Configuration de votre AWS compte et autorisations

Inscrivez-vous pour un Compte AWS

Pour commencer AWS, vous avez besoin d'un Compte AWS. Pour plus d'informations sur la création d'un Compte AWS, voir Getting started with an Compte AWS dans le Guide de Gestion de compte AWS référence.

Pour activer l'accès, ajoutez des autorisations à vos utilisateurs, groupes ou rôles :

- Utilisateurs et groupes dans AWS IAM Identity Center :

  Créez un jeu d'autorisations. Suivez les instructions de la rubrique Création d'un jeu d'autorisations du Guide de l'utilisateur AWS IAM Identity Center .
- Utilisateurs gérés dans IAM par un fournisseur d'identité :

  Créez un rôle pour la fédération d'identité. Suivez les instructions de la rubrique Création d'un rôle pour un fournisseur d'identité tiers (fédération) dans le Guide de l'utilisateur IAM.
- Utilisateurs IAM :
  - Créez un rôle que votre utilisateur peut assumer. Suivez les instructions de la rubrique Création d'un rôle pour un utilisateur IAM dans le Guide de l'utilisateur IAM.
  - (Non recommandé) Attachez une politique directement à un utilisateur ou ajoutez un utilisateur à un groupe d'utilisateurs. Suivez les instructions de la rubrique Ajout d'autorisations à un utilisateur (console) du Guide de l'utilisateur IAM.

Enregistrement de votre carte MCU auprès AWS IoT

Votre forum doit être enregistré AWS IoT pour pouvoir communiquer avec le AWS Cloud. Pour enregistrer votre forum auprès de AWS IoT celui-ci, vous devez avoir :

Une AWS IoT politique

La AWS IoT politique accorde à votre appareil l'autorisation d'accéder aux AWS IoT ressources. Il est stocké dans le AWS Cloud.

N'importe quelle AWS IoT chose

AWS IoT Tout ce qui vous permet de gérer vos appareils dans AWS IoT. Il est stocké dans le AWS Cloud.

Une clé privée et X.509 un certificat

La clé privée et le certificat permettent à votre appareil de s'authentifier. AWS IoT

Pour enregistrer votre tableau, suivez les procédures ci-dessous.

Pour créer un AWS IoT policy

1. Pour créer une politique IAM, vous devez connaître votre AWS région et votre numéro de AWS compte.

   Pour trouver votre numéro de AWS compte, ouvrez la console AWS de gestion, recherchez et développez le menu situé sous le nom de votre compte dans le coin supérieur droit, puis sélectionnez Mon compte. Votre ID de compte s'affiche sous Paramètres du compte.

   Pour trouver la AWS région associée à votre AWS compte, utilisez le AWS Command Line Interface. Pour l'installer AWS CLI, suivez les instructions du guide de l'AWS Command Line Interface utilisateur. Après avoir installé le AWS CLI, ouvrez une fenêtre d'invite de commande et entrez la commande suivante :

   ```
   aws iot describe-endpoint --endpoint-type=iot:Data-ATS
   ```

   La sortie doit se présenter comme suit :

   ```
   {
       "endpointAddress": "xxxxxxxxxxxxxx-ats.iot.us-west-2.amazonaws.com"
   }
   ```

   Dans cet exemple, la région est `us-west-2`.

   > **Note**
   >
   > Nous vous recommandons d'utiliser les points de terminaison ATS comme indiqué dans l'exemple.

2. Accédez à la console AWS IoT.
3. Dans le volet de navigation, choisissez successivement Sécurisé, Stratégies et Créer.
4. Entrez un nom pour identifier votre stratégie.

5. Dans la section Ajouter des instructions, choisissez Mode avancé. Copiez et collez le code JSON suivant dans la fenêtre de l'éditeur de stratégie. Remplacez *aws-region* et *aws-account* par votre AWS région et votre numéro de compte.

   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:*"
        }
    ]
}
```

   Cette stratégie accorde les autorisations suivantes :

   `iot:Connect`

   Accorde à votre appareil l'autorisation de se connecter au courtier de AWS IoT messages avec n'importe quel identifiant client.

`iot:Publish`

Accorde à votre appareil l'autorisation de publier un message MQTT sur n'importe quelle rubrique MQTT.

`iot:Subscribe`

Accorde à votre appareil l'autorisation de s'abonner au filtre de n'importe quelle rubrique MQTT.

`iot:Receive`

Accorde à votre appareil l'autorisation de recevoir des messages de l'agent de messages AWS IoT .

6. Choisissez Créer.

Pour créer un objet IoT, une clé privée et un certificat pour votre appareil

1. Accédez à la console AWS IoT.
2. Dans le volet de navigation, choisissez Gérer, puis Objets.
3. Si vous n'avez pas d'objets IoT enregistrés dans votre compte, la page Vous n'avez pas encore d'objets s'affiche. Si vous voyez cette page, choisissez Enregistrer un objet. Sinon, cliquez sur Create.
4. Sur la page Création d' AWS IoT objets, choisissez Créer un objet unique.
5. Sur la page Ajouter l'appareil au registre d'objets, saisissez un nom pour votre objet, puis choisissez Suivant.
6. Sur la page Ajouter un certificat pour votre objet, sous Création de One-click certificat, sélectionnez Créer un certificat.
7. Téléchargez votre clé privée et votre certificat en choisissant les liens Télécharger de chacun d'eux.
8. Choisissez Activer pour activer votre certificat. Les certificats doivent être activés avant utilisation.
9. Choisissez Joindre une politique pour associer une politique à votre certificat qui accorde à votre appareil l'accès aux AWS IoT opérations.
10. Choisissez la stratégie que vous venez de créer, puis choisissez Enregistrer l'objet.

Une fois que votre forum est enregistré AWS IoT, vous pouvez continuer surTéléchargement de FreeRTOS.

Téléchargement de FreeRTOS

Vous pouvez télécharger FreeRTOS depuis le dépôt FreeRTOS. GitHub

Après avoir téléchargé FreeRTOS, vous pouvez continuer. Configuration des démos de FreeRTOS

Configuration des démos de FreeRTOS

Vous devez modifier certains fichiers de configuration dans votre répertoire FreeRTOS avant de pouvoir compiler et exécuter des démos sur votre carte.

Pour configurer votre AWS IoT point de terminaison

Vous devez fournir votre point de terminaison à FreeRTOS afin que l'application exécutée sur AWS IoT votre carte puisse envoyer des demandes au point de terminaison approprié.

1. Accédez à la console AWS IoT.
2. Dans le panneau de navigation de gauche, choisissez Paramètres.

   Votre AWS IoT point de terminaison est affiché dans Terminal de données de l'appareil. Elle doit ressembler à *1234567890123*`-ats.iot.`*us-east-1*`.amazonaws.com`. Prenez note de ce point de terminaison.
3. Dans le volet de navigation, choisissez Gérer, puis Objets.

   Votre appareil doit porter un nom d' AWS IoT objet. Notez ce nom.
4. Ouvrir `demos/include/aws_clientcredential.h`.
5. Spécifiez des valeurs pour les constantes suivantes :

   - `#define clientcredentialMQTT_BROKER_ENDPOINT "`*Your AWS IoT endpoint*`";`
   - `#define clientcredentialIOT_THING_NAME "`*The AWS IoT thing name of your board*`"`

Pour configurer votre Wi-Fi

Si votre forum se connecte à Internet via une Wi-Fi connexion, vous devez fournir à FreeRTOS les informations d'identification pour vous connecter au réseau. Wi-Fi Si votre tableau n'est pas compatible Wi-Fi, vous pouvez ignorer ces étapes.

1. `demos/include/aws_clientcredential.h`.
2. Spécifiez des valeurs pour les constantes `#define` suivantes :

   - `#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`*

     Les types de sécurité valides sont :

     - `eWiFiSecurityOpen` (Ouvrir, aucune sécurité)
     - `eWiFiSecurityWEP` (Sécurité WEP)
     - `eWiFiSecurityWPA` (Sécurité WPA)
     - `eWiFiSecurityWPA2` (Sécurité WPA2)

Pour formater votre AWS IoT informations d'identification

FreeRTOS doit disposer AWS IoT du certificat et des clés privées associés à votre objet enregistré et de ses politiques d'autorisation pour pouvoir communiquer AWS IoT avec succès au nom de votre appareil.

> **Note**
>
> Pour configurer vos AWS IoT informations d'identification, vous devez disposer de la clé privée et du certificat que vous avez téléchargés depuis la AWS IoT console lors de l'enregistrement de votre appareil. Après avoir enregistré votre appareil en tant qu' AWS IoT objet, vous pouvez récupérer les certificats de l'appareil depuis la AWS IoT console, mais vous ne pouvez pas récupérer les clés privées.

FreeRTOS est un projet en langage C, et le certificat et la clé privée doivent être spécialement formatés pour être ajoutés au projet.

1. Dans une fenêtre du navigateur, ouvrez `tools/certificate_configuration/CertificateConfigurator.html`.

2. Sous Certificate PEM file (Fichier PEM de certificat), choisissez le *ID*`-certificate.pem.crt` que vous avez téléchargé à partir de la console AWS IoT .
3. Sous Fichier PEM de clé privée, choisissez le *ID*`-private.pem.key` que vous avez téléchargé à partir de la console AWS IoT .
4. Choisissez Generate and save aws_clientcredential_keys.h (Générer et enregistrer aws_clientcredential_keys.h) et enregistrez le fichier dans `demos/include`. Cela remplace le fichier existant dans le répertoire.

> **Note**
>
> Le certificat et la clé privée sont codés en dur à des fins de démonstration uniquement. Production-level les applications doivent stocker ces fichiers dans un emplacement sécurisé.

Après avoir configuré FreeRTOS, vous pouvez passer au guide de démarrage de votre carte pour configurer le matériel de votre plate-forme et son environnement de développement logiciel, puis compiler et exécuter la démo sur votre carte. Pour obtenir des instructions spécifiques à la carte, consultez Board-specific guides de démarrage. L'application de démonstration utilisée dans le didacticiel de mise en route est la démo d'authentification mutuelle CoreMQTT, qui se trouve à l'adresse. `demos/coreMQTT/mqtt_demo_mutual_auth.c`

Résolution des problèmes de mise en route

> **Important**
>
> Cette page fait référence au Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Les rubriques suivantes peuvent vous aider à résoudre les problèmes que vous rencontrez lors de la prise en main de FreeRTOS :

Rubriques

- Conseils généraux pour résoudre les problèmes de mise en route

- Installation d'un émulateur de terminal

Pour obtenir des conseils de résolution de problèmes spécifiques aux cartes, consultez le Commencez avec FreeRTOS manuel de votre carte.

Conseils généraux pour résoudre les problèmes de mise en route

Aucun message n'apparaît dans la AWS IoT console une fois que j'ai lancé le projet de démonstration Hello World. Que puis-je faire ?

Essayez les éléments suivants :

1. Ouvrez une fenêtre de terminal pour afficher la sortie de journalisation de l'exemple. Cela peut vous aider à déterminer ce qui ne va pas.
2. Vérifiez que vos informations d'identification réseau sont valides.

Les journaux affichés dans mon terminal lors de l'exécution d'une démo sont tronqués. Comment puis-je augmenter leur longueur ?

Augmentez la valeur `configLOGGING_MAX_MESSAGE_LENGTH` de 255 dans le `FreeRTOSconfig.h` fichier de la démo que vous exécutez :

```
#define configLOGGING_MAX_MESSAGE_LENGTH    255
```

Installation d'un émulateur de terminal

Un émulateur de terminal peut vous aider à diagnostiquer les problèmes ou à vérifier que le code de votre appareil s'exécute correctement. Il existe une variété d'émulateurs de terminal disponibles pour Windows, macOS et Linux.

Vous devez connecter votre carte à votre ordinateur avant d'essayer d'établir une connexion série à la carte avec un émulateur de terminal.

Utilisez les paramètres suivants pour configurer votre émulateur de terminal.

Paramètre du terminal	Value
Fréquence BAUD	115200

Paramètre du terminal	Value
Données	8 bits
Parité	Aucune
Arrêt	1 bit
Contrôle de flux	Aucune

Identification du port série de votre carte

Si vous ne connaissez pas le port série de votre carte, vous pouvez exécuter l'une des commandes suivantes à partir de la ligne de commande ou du terminal pour renvoyer les ports série de tous les appareils connectés à l'ordinateur hôte :

Windows

```
chgport
```

Linux

```
ls /dev/tty*
```

macOS

```
ls /dev/cu.*
```

Utilisation de CMake avec FreeRTOS

> **Important**
>
> Cette page fait référence au Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Vous pouvez utiliser CMake pour générer des fichiers de construction de projet à partir du code source de l'application FreeRTOS, ainsi que pour créer et exécuter le code source.

Vous pouvez également utiliser un IDE pour modifier, déboguer, compiler, flasher et exécuter du code sur FreeRTOS-qualified des appareils. Chaque manuel de démarrage propre à une carte inclut des instructions de définition de l'IDE pour une plateforme spécifique. Si vous préférez travailler sans IDE, vous pouvez utiliser d'autres outils d'édition et de débogage tiers pour développer et déboguer votre code, puis utiliser CMake pour générer et exécuter les applications.

Les cartes suivantes prennent en charge CMake :

- Espressif ESP32-DevKitC
- Espressif ESP-WROVER-KIT
- Infineon XMC4800 IoT Connectivity Kit
- Kit de démarrage Marvell MW320 AWS IoT
- Kit de démarrage Marvell MW322 AWS IoT
- Microchip Curiosity PIC32MZEF Bundle
- Kit de développement Nordic nRF52840 DK
- STMicroelectronicsSTM32L4 Discovery Kit IoT Node
- Instruments texans CC3220SF-LAUNCHXL
- Simulateur Microsoft Windows

Consultez les rubriques ci-dessous pour plus d'informations sur l'utilisation de CMake avec FreeRTOS.

Rubriques

- Conditions préalables
- Développement d'applications FreeRTOS avec des éditeurs de code et des outils de débogage tiers
- Création de FreeRTOS avec CMake

Conditions préalables

Vérifiez que votre machine hôte satisfait les prérequis suivants avant de continuer :

- La chaîne de compilation de votre appareil doit prendre en charge le système d'exploitation de la machine. CMake prend en charge toutes les versions de Windows, macOS et Linux.

  Le sous-système Windows pour Linux (WSL) n'est pas pris en charge. Utilisez CMake natif sur les machines Windows.

- Vous devez disposer de la version 3.13 (ou ultérieure) de CMake.

  Vous pouvez télécharger la distribution binaire de CMake depuis CMake.org.

  > **Note**
  >
  > Si vous téléchargez la distribution binaire de CMake, assurez-vous d'ajouter le fichier exécutable CMake l'exécutable à la variable d'environnement PATH avant d'utiliser CMake à partir de la ligne de commande.

  Vous pouvez également télécharger et installer CMake à l'aide d'un gestionnaire de package, tel quehomebrew sur macOS et scoop ou chocolatey sur Windows.

  > **Note**
  >
  > Les versions de package CMake dans les gestionnaires de package pour de nombreuses distributions Linux sont obsolètes. Si le gestionnaire de package de votre distribution ne fournit pas la version la plus récente de CMake, vous pouvez essayer d'autres gestionnaires de packages, comme `linuxbrew` ou `nix`.

- Votre système de build natif doit être compatible.

  CMake peut cibler de nombreux systèmes de build natifs, y compris GNU Make ou Ninja. Make et Ninja peuvent être installés à l'aide de gestionnaires de package sous Linux, macOS et Windows. Si vous utilisez Make sous Windows, vous pouvez installer une version autonome de Equation, ou vous pouvez installer MinGW, qui comprend Make.

  > **Note**
  >
  > Dans MinGW, le fichier exécutable de Make se nomme `mingw32-make.exe`, et non pas `make.exe`.

Nous vous recommandons d'utiliser Ninja, car il est plus rapide que Make et fournit également une prise en charge native pour tous les systèmes d'exploitation de bureau.

Développement d'applications FreeRTOS avec des éditeurs de code et des outils de débogage tiers

Vous pouvez utiliser un éditeur de code et une extension de débogage ou un outil de débogage tiers pour développer des applications pour FreeRTOS.

Si, par exemple, vous utilisez Visual Studio Code comme éditeur de code, vous pouvez installer l'extension Cortex-DebugVS Code en tant que débogueur. Lorsque vous avez fini de développer votre application, vous pouvez appeler l'outil de ligne de commande CMake pour générer votre projet à partir de VS Code. Pour plus d'informations sur l'utilisation de CMake pour créer des applications FreeRTOS, consultez. Création de FreeRTOS avec CMake

Pour le débogage, vous pouvez fournir un code VS avec la configuration de débogage similaire à ce qui suit :

```
"configurations": [
    {
        "name": "Cortex Debug",
        "cwd": "${workspaceRoot}",
        "executable": "./build/st/stm32l475_discovery/aws_demos.elf",
        "request": "launch",
        "type": "cortex-debug",
        "servertype": "stutil"
    }
]
```

Création de FreeRTOS avec CMake

CMake cible votre système d'exploitation hôte en tant que système cible par défaut. Pour l'utiliser pour la compilation croisée, CMake a besoin d'un fichier de chaîne d'outils, qui spécifie le compilateur que vous souhaitez utiliser. Dans FreeRTOS, nous fournissons des fichiers de chaîne d'outils par défaut dans. *freertos*`/tools/cmake/toolchains` La manière de fournir ce fichier à CMake varie selon que vous utilisez l'interface utilisateur graphique ou l'interface de ligne de commande CMake. Pour plus d'informations, suivez les instructions Génération de fichiers de build (outil de ligne de commande CMake) ci-dessous. Pour plus d'informations sur la compilation croisée dans CMake, consultez le wiki officiel CrossCompilingde CMake.

Pour créer un CMake-based projet

1. Exécutez CMake pour générer les fichiers de build pour un système de build natif, tel que Make ou Ninja.

   Vous pouvez utiliser l'outil de ligne de commande CMake ou l'interface graphique utilisateur CMake pour générer les fichiers de build pour votre système de build natif.

   Pour plus d'informations sur la génération de fichiers de compilation FreeRTOS, reportez-vous aux sections et. Génération de fichiers de build (outil de ligne de commande CMake) Génération de fichiers de build (GUI CMake)
2. Appelez le système de build natif pour transformer le projet en fichier exécutable.

   Pour plus d'informations sur la création de fichiers de compilation FreeRTOS, consultez. Création de FreeRTOS à partir des fichiers de construction générés

Génération de fichiers de build (outil de ligne de commande CMake)

Vous pouvez utiliser l'outil de ligne de commande CMake (cmake) pour générer des fichiers de compilation pour FreeRTOS. Pour générer les fichiers de build, vous devez spécifier une carte cible, un compilateur et l'emplacement du code source et du répertoire de build.

Vous pouvez utiliser les options suivantes pour cmake :

- `-DVENDOR`— Spécifie le tableau cible.
- `-DCOMPILER`— Spécifie le compilateur.
- `-S`— Spécifie l'emplacement du code source.
- `-B`— Spécifie l'emplacement des fichiers de compilation générés.

> **Note**
>
> Le compilateur doit se trouver dans la variable `PATH` du système, ou bien vous devez spécifier l'emplacement du compilateur.

Par exemple, si le fournisseur est Texas Instruments, que la carte est une CC3220 Launchpad et que le compilateur est GCC pour ARM, vous pouvez exécuter la commande suivante pour générer les fichiers source depuis le répertoire actuel vers un répertoire nommé *build-directory* :

```
cmake -DVENDOR=ti -DBOARD=cc3220_launchpad -DCOMPILER=arm-ti -S . -B build-directory
```

> **Note**
>
> Si vous utilisez Windows, vous devez spécifier le système de build natif, car CMake utilise Visual Studio par défaut. Par exemple :
>
> ```
> cmake -DVENDOR=ti -DBOARD=cc3220_launchpad -DCOMPILER=arm-ti -S . -B build-directory -G Ninja
> ```
>
> Ou:
>
> ```
> cmake -DVENDOR=ti -DBOARD=cc3220_launchpad -DCOMPILER=arm-ti -S . -B build-directory -G "MinGW Makefiles"
> ```

Les expressions régulières `${VENDOR}.*` et `${BOARD}.*` sont utilisées pour rechercher une carte correspondante, afin que vous n'ayez pas à utiliser les noms complets du fournisseur et de la carte pour les options `VENDOR` et `BOARD`. Vous pouvez utiliser les noms partiels, à condition qu'il n'existe qu'une seule correspondance. Par exemple, les commandes suivantes génèrent les mêmes fichiers de build à partir de la même source :

```
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
```

Vous pouvez utiliser l' option `CMAKE_TOOLCHAIN_FILE` si vous souhaitez utiliser un fichier de chaîne d'outils qui ne figure pas dans le répertoire `cmake/toolchains` par défaut. Par exemple :

```
cmake -DBOARD=cc3220 -DCMAKE_TOOLCHAIN_FILE='/path/to/toolchain_file.cmake' -S . -B build-directory
```

Si le fichier de chaîne d'outils n'utilise pas de chemins absolus pour votre compilateur, et que vous n'avez pas ajouté votre compilateur à la variable d' environnement PATH, il est possible que CMake ne trouve pas son emplacement. Pour vous assurer que CMake trouve votre fichier de chaîne d'outils, vous pouvez utiliser l'option AFR_TOOLCHAIN_PATH. Cette option recherche le chemin du répertoire de la chaîne d'outils spécifiée et le sous-dossier dans bin. Par exemple :

```
cmake -DBOARD=cc3220 -DCMAKE_TOOLCHAIN_FILE='/path/to/toolchain_file.cmake' -
DAFR_TOOLCHAIN_PATH='/path/to/toolchain/' -S . -B build-directory
```

Pour activer le débogage, définissez le paramètre CMAKE_BUILD_TYPE sur debug. Lorsque cette option est activée, CMake ajoute des indicateurs de débogage aux options de compilation et construit FreeRTOS avec des symboles de débogage.

```
# Build with debug symbols
cmake -DBOARD=cc3220 -DCOMPILER=arm-ti -DCMAKE_BUILD_TYPE=debug -S . -B build-directory
```

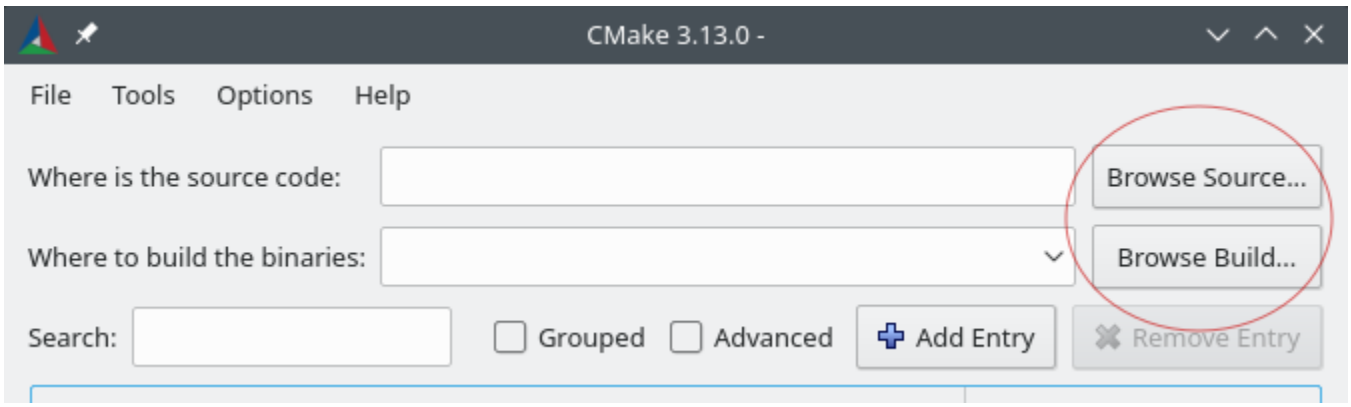
Vous pouvez également définir CMAKE_BUILD_TYPE sur release pour ajouter des indicateurs d'optimisation aux options de compilation.

Génération de fichiers de build (GUI CMake)

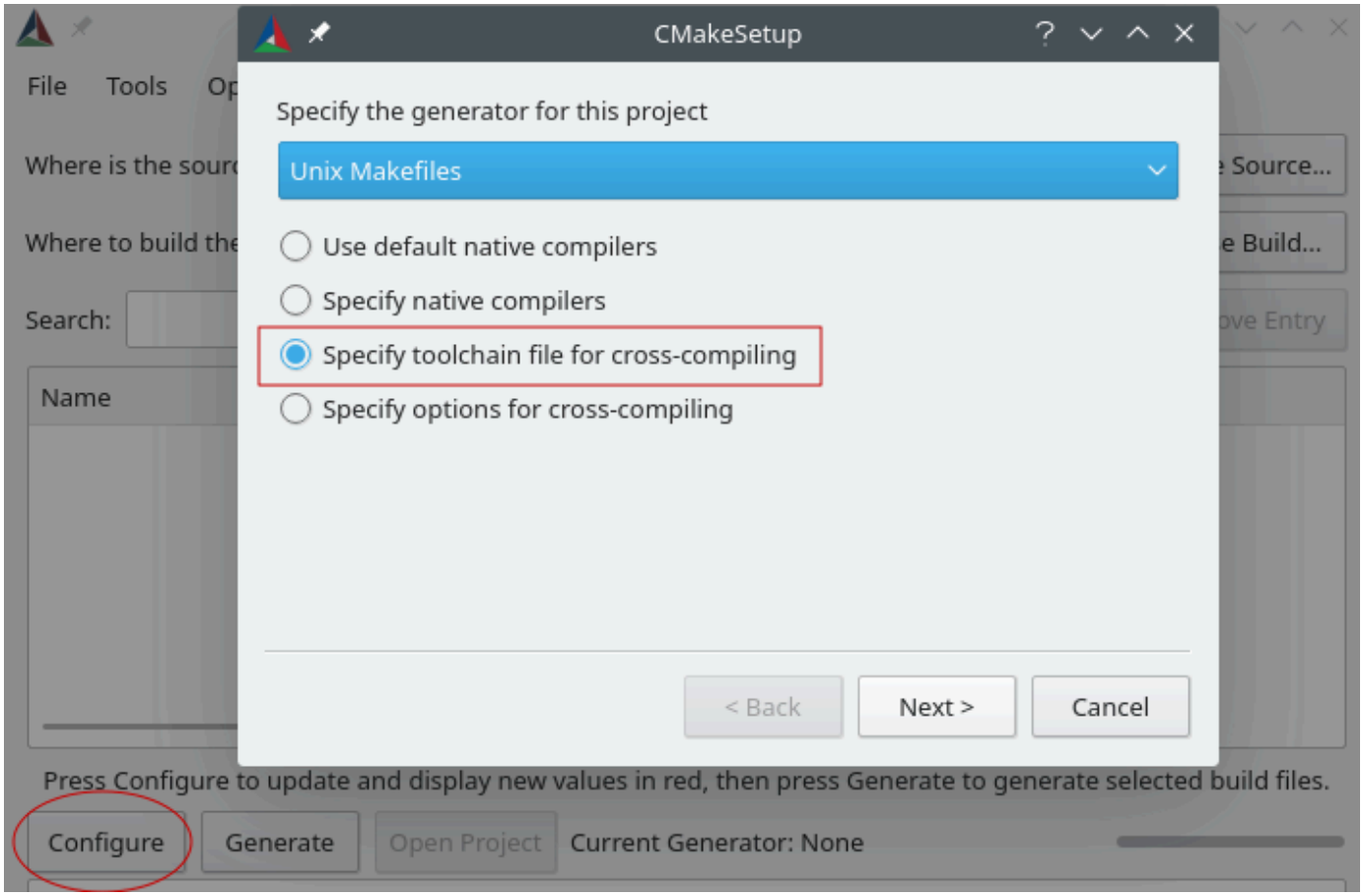
Vous pouvez utiliser l'interface graphique de CMake pour générer des fichiers de compilation FreeRTOS.

Pour générer des fichiers de build avec la GUI CMake

1. À partir de la ligne de commande, exécutez cmake-gui pour lancer la GUI.
2. Sélectionnez Browse Source (Sélectionner la source) et spécifiez l'entrée source, puis sélectionnez Browse Build (Sélectionner la build) et spécifiez la sortie de la build.



3. Choisissez Configure (Configurer), et sous Specify the build generator for this project (Spécifier le générateur de build pour ce projet), recherchez et choisissez le système de build que vous souhaitez utiliser pour générer les fichiers de build générés. Si vous ne voyez pas la fenêtre contextuelle, il se peut que vous réutilisiez un répertoire de build existant. Dans ce cas, supprimez le cache CMake en choisissant Delete Cache (Supprimer le cache) dans le menu File (Fichier).



4. Sélectionnez Specify toolchain file for cross-compiling (Spécifier le fichier de chaîne d'outils pour compilation croisée, puis sélectionnez Next (Suivant).
5. Choisissez le fichier de chaîne d'outils (par exemple, *freertos*`/tools/cmake/toolchains/arm-ti.cmake`), puis sélectionnez Terminer.

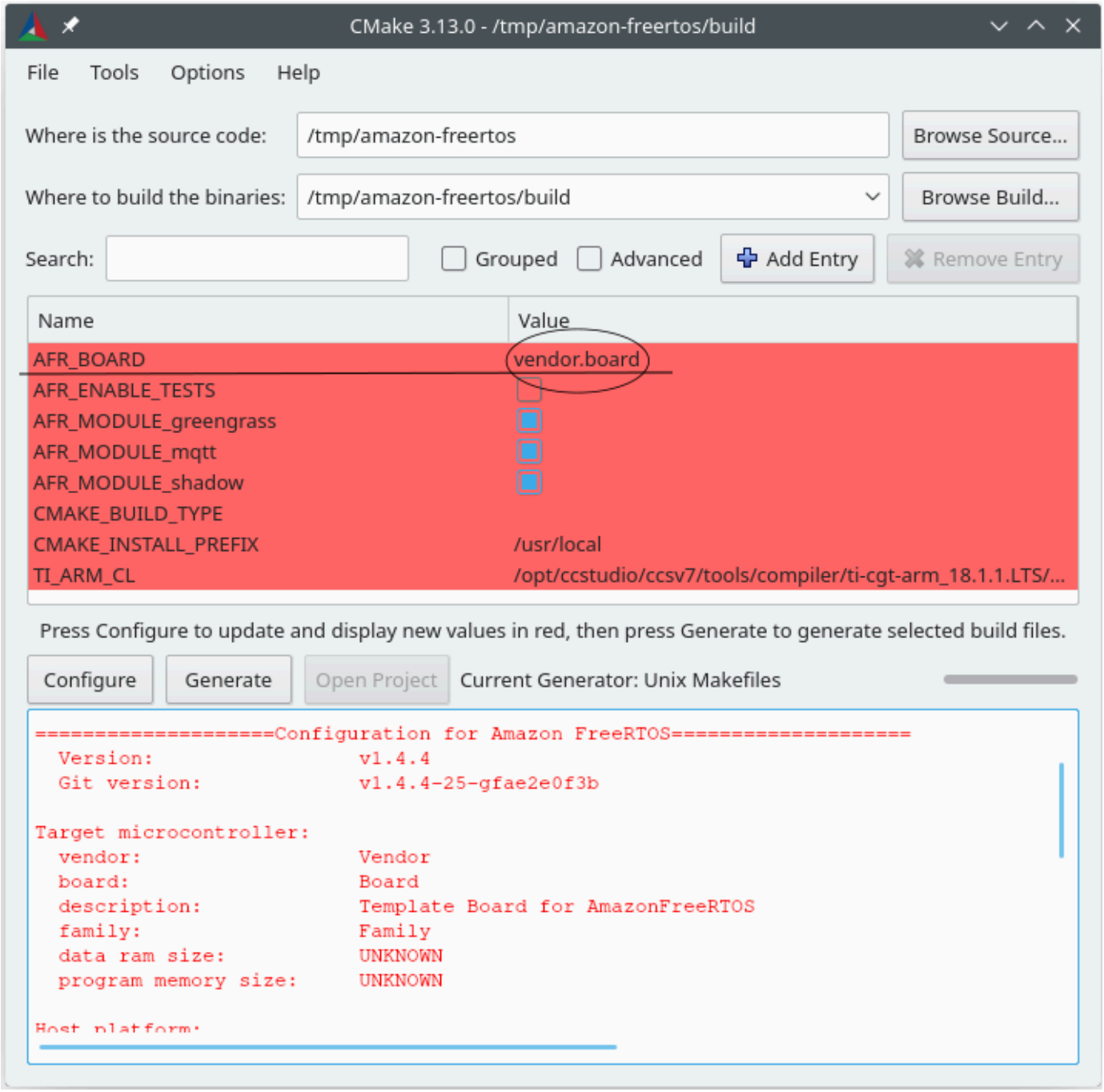
   La configuration par défaut de FreeRTOS est le tableau de modèles, qui ne fournit aucune cible de couche portable. Une fenêtre s'affiche alors avec le message Error in configuration process.

> **Note**
>
> Si vous voyez l'erreur suivante :

```
CMake Error at tools/cmake/toolchains/find_compiler.cmake:23 (message):
Compiler not found, you can specify search path with AFR_TOOLCHAIN_PATH.
```

Cela signifie que le compilateur ne se trouve pas dans votre variable d'environnement PATH. Vous pouvez définir la variable AFR_TOOLCHAIN_PATH dans l'interface utilisateur graphique pour indiquer à CMake où vous avez installé votre compilateur. Si vous ne voyez pas la variable AFR_TOOLCHAIN_PATH, choisissez Add Entry (Ajouter une entrée). Dans la fenêtre contextuelle, sous Name (Nom), tapez **AFR_TOOLCHAIN_PATH**. Sous Compiler Path (Chemin d'accès du compilateur), tapez le chemin d'accès à votre compilateur. Par exemple, `C:/toolchains/arm-none-eabi-gcc`.

6. La GUI devrait maintenant ressembler à ceci :



Sélectionnez AFR_BOARD, choisissez votre carte, puis sélectionnez à nouveau Configure (Configurer) .

7. Choisissez Générer. CMake génère les fichiers du système de génération (par exemple, makefiles ou ninja), et ces fichiers apparaissent dans le répertoire de génération que vous avez spécifié lors de la première étape. Suivez les instructions de la section suivante pour générer l'image binaire.

Création de FreeRTOS à partir des fichiers de construction générés

Génération avec un système de génération natif

Vous pouvez créer FreeRTOS avec un système de construction natif en appelant la commande build system depuis le répertoire des binaires de sortie.

Par exemple, si votre répertoire de sortie de fichier de build est `<build_dir>`, et que vous utilisez CMake en tant que système de build natif, exécutez les commandes suivantes :

```
cd <build_dir>
make -j4
```

Création d' avec CMake

Vous pouvez également utiliser l'outil de ligne de commande CMake pour créer FreeRTOS. CMake fournit une couche d'abstraction pour appeler les systèmes de build natifs. Par exemple :

```
cmake --build build_dir
```

Voici quelques autres utilisations courantes du mode de build de l'outil de ligne de commande CMake :

```
# 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
```

Pour plus d'informations sur le mode de build CMake, consultez la documentation CMake.

Developer-mode provisionnement des clés

> **Important**
>
> Cette page fait référence au Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez

déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Introduction

Cette section décrit deux options pour obtenir un certificat X.509 client fiable sur un appareil IoT à des fins de test en laboratoire. Selon les capacités de l'appareil, diverses opérations liées au provisionnement peuvent être prises en charge ou non, notamment la génération de clés ECDSA intégrées, l'importation de clés privées et l'enregistrement de certificats. X.509 En outre, différents cas d'utilisation nécessitent des niveaux de protection de clé différents, allant du stockage flash embarqué à l'utilisation de matériel de chiffrement dédié. Cette section fournit une logique pour travailler selon les capacités de chiffrement de votre appareil.

Option #1 : importation de clé privée depuis AWS IoT

Pour les tests en atelier, si votre appareil autorise l'importation de clés privées, suivez les instructions de la section Configuration des démos de FreeRTOS.

Option 2 : Génération de clés privées embarquées

Si votre appareil dispose d'un élément sécurisé, ou si vous préférez générer vos propres certificat et paire de clés d'appareil, suivez les instructions fournies ici.

Configuration initiale

Tout d'abord, effectuez les étapes indiquéesConfiguration des démos de FreeRTOS, mais sautez la dernière étape (c'est-à-dire, ne faites pas Pour formater vos AWS IoT informations d'identification). Comme résultat, le fichier `demos/include/aws_clientcredential.h` devrait avoir été mis à jour avec vos paramètres, mais pas le fichier `demos/include/aws_clientcredential_keys.h`.

Configuration du projet de démonstration

Ouvrez la démo d'authentification mutuelle CoreMQTT comme décrit dans le guide de votre tableau dans. Board-specific guides de démarrage Dans le projet, ouvrez le fichier `aws_dev_mode_key_provisioning.c` et modifiez la définition de `keyprovisioningFORCE_GENERATE_NEW_KEY_PAIR` (qui est définie sur zéro par défaut) à un :

```
#define keyprovisioningFORCE_GENERATE_NEW_KEY_PAIR 1
```

Ensuite, générez et exécutez le projet de démonstration, puis passez à l'étape suivante.

Extraction de clé publique

Étant donné que l'appareil n'a pas été approvisionné avec une clé privée ni un certificat client, la démo ne parviendra pas à s'authentifier. AWS IoT Cependant, la démo d'authentification mutuelle CoreMQTT commence par exécuter le provisionnement des clés en mode développeur, ce qui entraîne la création d'une clé privée si aucune n'était déjà présente. Vous devriez voir quelque chose comme ceci au début de la sortie de la console série.

```
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
```

Copiez les six lignes d'octets de la clé dans un fichier appelé `DevicePublicKeyAsciiHex.txt`. Ensuite, utilisez l'outil de ligne de commande « xxd » pour analyser les octets hexadécimaux en binaire :

```
xxd -r -ps DevicePublicKeyAsciiHex.txt DevicePublicKeyDer.bin
```

Utilisez « openssl » pour formater la clé publique de l'appareil codée en binaire (DER) au format PEM :

```
openssl ec -inform der -in DevicePublicKeyDer.bin -pubin -pubout -outform pem -out
 DevicePublicKey.pem
```

N'oubliez pas de désactiver le paramètre de génération de clé temporaire que vous avez activé ci-dessus. Sinon, l'appareil créera une autre paire de clés, et vous devrez répéter les étapes précédentes :

```
#define keyprovisioningFORCE_GENERATE_NEW_KEY_PAIR 0
```

Configuration de l'infrastructure à clés publiques

Suivez les instructions de la section Enregistrement de votre certificat CA pour créer une hiérarchie de certificats pour votre certificat d'atelier d'appareil. Arrêtez avant d'exécuter la

séquence décrite dans la section Création d'un certificat d'appareil à l'aide de votre certificat de CA.

Dans ce cas, l'appareil ne signera pas la demande de certificat (c'est-à-dire la demande de service de certificat ou CSR) car la logique de X.509 codage requise pour créer et signer une CSR a été exclue des projets de démonstration de FreeRTOS afin de réduire la taille de la ROM. Au lieu de cela, à des fins de test en atelier, créez une clé privée sur votre poste de travail et utilisez-la pour signer la CSR.

```
openssl genrsa -out tempCsrSigner.key 2048
openssl req -new -key tempCsrSigner.key -out deviceCert.csr
```

Une fois que votre autorité de certification a été créée et enregistrée auprès de celle-ci AWS IoT, utilisez la commande suivante pour émettre un certificat client basé sur le CSR de l'appareil qui a été signé à l'étape précédente :

```
openssl x509 -req -in deviceCert.csr -CA rootCA.pem -CAkey rootCA.key
 -CAcreateserial -out deviceCert.pem -days 500 -sha256 -force_pubkey
 DevicePublicKey.pem
```

Même si la CSR a été signée avec une clé privée temporaire, le certificat émis ne peut être utilisé qu'avec la clé privée réelle de l'appareil. Le même mécanisme peut être utilisé en production si vous stockez la clé de signataire de CSR dans un matériel distinct et que vous configurez votre autorité de certification pour qu'elle émette des certificats uniquement pour les demandes signées par cette clé spécifique. Cette clé devrait également rester sous le contrôle d'un administrateur désigné.

Importation de certificat

Une fois le certificat émis, l'étape suivante consiste à l'importer dans votre appareil. Vous devrez également importer votre certificat d'autorité de certification (CA), car il est nécessaire pour que la première authentification réussisse lors de l'utilisation AWS IoT de JITP. Dans le `aws_clientcredential_keys.h` fichier de votre projet, définissez la `keyCLIENT_CERTIFICATE_PEM` macro comme étant le contenu de l'appareil Cert.pem et définissez la `keyJITR_DEVICE_CERTIFICATE_AUTHORITY_PEM` macro comme le contenu de`rootCA.pem`.

Autorisation de l'appareil

Importez `deviceCert.pem` dans le AWS IoT registre comme décrit dans Utiliser votre propre certificat. Vous devez créer un nouvel AWS IoT objet, joindre le certificat EN ATTENTE et une politique à votre objet, puis marquer le certificat comme ACTIF. Toutes ces étapes peuvent être effectuées manuellement dans la AWS IoT console.

Une fois que le nouveau certificat client est ACTIF et associé à un objet et à une politique, exécutez à nouveau la démo d'authentification mutuelle CoreMQTT. Cette fois, la connexion au broker AWS IoT MQTT sera établie.

Board-specific guides de démarrage

> **Important**
>
> Cette page fait référence au Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Après avoir terminé lePremiers pas, consultez le guide de votre forum pour obtenir des instructions spécifiques pour démarrer avec FreeRTOS :

- Démarrez avec le kit de développement Cypress CYW943907AEVAL1F
- Démarrez avec le kit de développement Cypress CYW954907AEVAL1F
- Commencer à utiliser le kit Cypress CY8CKIT-064S0S2-4343W
- Démarrez avec l'Infineon XMC4800 IoT Connectivity Kit
- Commencer à utiliser le MW32x AWS IoT Kit de démarrage
- Commencer à utiliser le kit de développement MediaTek MT7697hx
- Démarrez avec la solution groupée Microchip Curiosity PIC32MZ EF
- Commencer à utiliser le Nuvoton NuMaker-IoT-M487
- Démarrez avec le NXP LPC54018 IoT Module
- Commencer à utiliser le Renesas Starter Kit+ pour RX65N-2MB
- Mise en route avec la carte STMicroelectronics STM32L4 Discovery Kit IoT Node

- Commencer à utiliser les Texas Instruments CC3220SF-LAUNCHXL
- Mise en route avec le simulateur d'appareils Windows
- Commencer à utiliser le kit IoT industriel Xilinx Avnet MicroZed

> **Note**
>
> Il n'est pas nécessaire de suivre les Premiers pas guides autonomes suivants pour démarrer avec FreeRTOS :
>
> - Démarrez avec l'élément sécurisé Microchip ATECC608A avec simulateur Windows
> - Commencer à utiliser l'Espressif ESP32-DevKitC et le ESP-WROVER-KIT
> - Débuter avec l'Espressif ESP32-WROOM-32SE
> - Débuter avec l'Espressif ESP32-S2
> - Démarrez avec le kit XMC4800 IoT Connectivity Kit et Infineon OPTIGA Trust X
> - Débuter avec le Nordic n RF52840-DK

Démarrez avec le kit de développement Cypress CYW943907AEVAL1F

> **Important**
>
> Cette intégration de référence est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Ce didacticiel fournit des instructions sur la mise en route du kit de développement Cypress CYW943907AEVAL1F. Si vous ne disposez pas du kit de développement Cypress CYW943907AEVAL1F, consultez le catalogue d'appareils des partenaires AWS pour en acheter un auprès de notre partenaire.

> **Note**
>
> Ce didacticiel vous explique comment configurer et exécuter la démo d'authentification mutuelle CoreMQTT. Le port FreeRTOS de cette carte ne prend actuellement pas en charge les démos du serveur TCP et du client.

Avant de commencer, vous devez configurer AWS IoT et télécharger FreeRTOS pour connecter votre appareil au Cloud. AWS Pour obtenir des instructions, consultez Premiers pas.

> **Important**
>
> - Dans cette rubrique, le chemin d'accès au répertoire de téléchargement de FreeRTOS est appelé. *`freertos`*
> - Les espaces dans le chemin d'accès *`freertos`* peuvent provoquer des échecs de construction. Lorsque vous clonez ou copiez le référentiel, assurez-vous que le chemin d'accès que vous créez ne contient pas d'espaces.
> - La longueur maximale d'un chemin sous Microsoft Windows est de 260 caractères. Les longs chemins de répertoire de téléchargement de FreeRTOS peuvent provoquer des échecs de compilation.
> - Le code source pouvant contenir des liens symboliques, si vous utilisez Windows pour extraire l'archive, vous devrez peut-être :
>   - Activez le mode développeur ou
>   - Utilisez une console élevée en tant qu'administrateur.
>
>   Ainsi, Windows peut créer correctement des liens symboliques lors de l'extraction de l'archive. Dans le cas contraire, les liens symboliques seront écrits sous forme de fichiers normaux contenant les chemins des liens symboliques sous forme de texte ou seront vides. Pour plus d'informations, consultez le billet de blog Symlinks in Windows 10 ! .
>
>   Si vous utilisez Git sous Windows, vous devez activer le mode développeur ou vous devez :
>   - Définissez `core.symlinks` ce paramètre sur true à l'aide de la commande suivante :
>
>     ```
>     git config --global core.symlinks true
>     ```

- Utilisez une console élevée en tant qu'administrateur chaque fois que vous utilisez une commande git qui écrit sur le système (par exemplegit pull,git clone, etgit submodule update --init --recursive).
- Comme indiqué dansTéléchargement de FreeRTOS, les ports FreeRTOS pour Cypress ne sont actuellement disponibles que sur. GitHub

Présentation de

Ce didacticiel comprend les instructions de mise en route suivantes :

1. Installation de logiciels sur la machine hôte pour développer et déboguer des applications intégrées pour votre carte de microcontrôleur.
2. Compilation croisée d'une application de démonstration FreeRTOS en une image binaire.
3. Chargement de l'image binaire de l'application dans votre carte et exécution de l'application.
4. Interaction avec l'application s'exécutant sur votre carte via une connexion série, à des fins de surveillance et de débogage.

Configuration de votre environnement de développement

Télécharger et installer le kit SDK WICED

Dans ce guide de démarrage, vous utilisez le SDK Cypress WICED Studio pour programmer votre carte avec la démo de FreeRTOS. Visitez le site web WICED Software pour télécharger le kit SDK WICED Studio depuis Cypress. Vous devez avoir un compte Cypress gratuit pour télécharger le logiciel. Le kit SDK WICED Studio est compatible avec les systèmes d'exploitation Windows, macOS et Linux.

> **Note**
>
> Certains systèmes d'exploitation nécessitent des étapes d'installation supplémentaires. Assurez-vous de lire et de suivre toutes les instructions d'installation du système d'exploitation et de la version de WICED Studio que vous installez.

Définir les variables d'environnement

Avant d'utiliser WICED Studio pour programmer votre carte, vous devez créer une variable d'environnement pour le répertoire d'installation du kit SDK WICED Studio SDK. Si WICED Studio est en cours d'exécution tandis que vous créez vos variables, vous devez redémarrer l'application après avoir défini vos variables.

> **Note**
>
> Le programme d'installation WICED Studio crée deux dossiers distincts nommés `WICED-Studio-`*`m`*`.`*`n`* sur votre machine, où `m` et `n` sont les numéros de version majeure et mineure, respectivement. Ce document assume un nom de dossier `WICED-Studio-6.2` mais veillez à utiliser le nom correct pour la version que vous installez. Lorsque vous définissez la variable d'environnement `WICED_STUDIO_SDK_PATH`, veillez à spécifier le chemin d'installation complet du kit SDK WICED Studio, et non pas le chemin d'installation de l'interface utilisateur WICED Studio. Dans Windows et macOS, le dossier `WICED-Studio-`*`m`*`.`*`n`* du kit SDK est créé dans le dossier `Documents` par défaut.

Pour créer les variables d'environnement sous Windows

1. Dans le Panneau de configuration, cliquez sur Système, Paramètres système avancés.
2. Sous l'onglet Avancé, choisissez Variables d'environnement.
3. Sous Variables d'utilisateur, choisissez Nouvelle.
4. Pour Nom de la variable :, tapez **`WICED_STUDIO_SDK_PATH`**. Pour Valeur de la variable, entrez le répertoire d'installation du kit SDK WICED Studio.

Pour créer la variable d'environnement sur Linux ou macOS

1. Ouvrez le fichier `/etc/profile` sur votre ordinateur, puis ajoutez la ligne suivante à la fin du fichier :

   ```
   export WICED_STUDIO_SDK_PATH=installation-path/WICED-Studio-6.2
   ```

2. Redémarrez l'ordinateur.
3. Ouvrez une fenêtre de terminal et exécutez les commandes suivantes :

```
cd freertos/vendors/cypress/WICED_SDK
```

```
perl platform_adjust_make.pl
```

```
chmod +x make
```

Établissement d'une connexion série

Pour établir une connexion série entre votre machine hôte et votre carte

1. Connectez la carte à votre ordinateur hôte à l'aide d'un Micro-B câble USB Standard-A .
2. Identifiez le numéro de port série USB pour la connexion à la carte sur votre ordinateur hôte.
3. Démarrez une fenêtre de terminal série et ouvrez une connexion avec les paramètres suivants :
   - Vitesse de transmission : 115200
   - Données : 8 bits
   - Parité : aucune
   - Bits d'arrêt : 1
   - Contrôle de flux : aucun

Pour plus d'informations sur l'installation d'une fenêtre de terminal et la configuration d'une connexion série, consultez Installation d'un émulateur de terminal.

Surveillance des messages MQTT dans le cloud

Avant de lancer le projet de démonstration FreeRTOS, vous pouvez configurer le client MQTT dans la console pour surveiller AWS IoT les messages que votre appareil envoie au Cloud. AWS

Pour vous abonner à la rubrique MQTT avec le AWS IoT Client MQTT

1. Connectez-vous à la console AWS IoT.
2. Dans le volet de navigation, choisissez Test, puis choisissez MQTT test client pour ouvrir le client MQTT.
3. Dans le champ Rubrique d'abonnement, saisissez ***your-thing-name*****/example/topic**, puis choisissez S'abonner à la rubrique.

Créez et exécutez le projet de démonstration FreeRTOS

Après avoir configuré une connexion série avec votre carte, vous pouvez créer le projet de démonstration FreeRTOS, flasher la démo sur votre carte, puis exécuter la démonstration.

Pour créer et exécuter le projet de démonstration FreeRTOS dans WICED Studio

1. Lancez WICED Studio.
2. Dans le menu File (Fichier), choisissez Import (Importer). Développez le dossier `General`, choisissez Projets existants dans l'espace de travail, puis choisissez Suivant.
3. Dans Select root directory (Sélectionner le répertoire racine), sélectionnez Browse..., (Parcourir...), accédez au chemin *`freertos`*`/projects/cypress/CYW943907AEVAL1F/wicedstudio`, puis sélectionnez OK.
4. Sous Projects (Projets), cochez la case correspondant uniquement au projet aws_demo. Choisissez Terminer pour importer le projet. Le projet cible aws_demo doit apparaître dans la fenêtre Créer la cible.
5. Développez le menu WICED Platform (Plateforme WICED) et choisissez WICED Filters off (Filtres WICED désactivés).
6. Dans la fenêtre Créer le filtre, développez aws_demo, cliquez avec le bouton droit de la souris sur le fichier `demo.aws_demo`, puis choisissez Créer la cible afin de créer la démonstration et de la télécharger sur votre carte. Normalement, la démonstration s'exécute automatiquement une fois qu'elle a été créée et téléchargée sur votre carte.

Résolution des problèmes

- Si vous utilisez Windows, il se peut que vous receviez le message d'erreur suivant lorsque vous développez et exécutez le projet de démonstration :

  ```
  : recipe for target 'download_dct' failed
  make.exe[1]: *** [download_dct] Error 1
  ```

  Pour résoudre ce problème, utilisez la procédure suivante :

  1. Accédez à *`WICED-Studio-SDK-PATH`*`\WICED-Studio-6.2\43xxx_Wi-Fi\tools\OpenOCD\Win32` et double-cliquez sur `openocd-all-brcm-libftdi.exe`.
  2. Accédez à *`WICED-Studio-SDK-PATH`*`\WICED-Studio-6.2\43xxx_Wi-Fi\tools\drivers\CYW9WCD1EVAL1` et double-cliquez sur `InstallDriver.exe`.

- Si vous utilisez Linux or macOS, il se peut que vous receviez le message d'erreur suivant lorsque vous développez et exécutez le projet de démonstration :

  ```
  make[1]: *** [download_dct] Error 127
  ```

  Pour résoudre cette erreur, cliquez sur la commande suivante pour ouvrir le package libusb-dev :

  ```
  sudo apt-get install libusb-dev
  ```

Pour obtenir des informations générales sur la résolution des problèmes liés à la prise en main de FreeRTOS, consultez. Résolution des problèmes de mise en route

Démarrez avec le kit de développement Cypress CYW954907AEVAL1F

> **Important**
>
> Cette intégration de référence est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Ce didacticiel fournit des instructions concernant la mise en route du kit de développement Cypress CYW954907AEVAL1F. Si vous ne disposez pas du kit de développement Cypress CYW954907AEVAL1F, consultez le catalogue d'appareils des partenaires AWS pour en acheter un auprès de notre partenaire.

> **Note**
>
> Ce didacticiel vous explique comment configurer et exécuter la démo d'authentification mutuelle CoreMQTT. Le port FreeRTOS de cette carte ne prend actuellement pas en charge les démos du serveur TCP et du client.

Avant de commencer, vous devez configurer AWS IoT et télécharger FreeRTOS pour connecter votre appareil au Cloud. AWS Pour obtenir des instructions, consultez Premiers pas. Dans ce didacticiel, le chemin d'accès au répertoire de téléchargement de FreeRTOS est appelé. *`freertos`*

⚠ Important

- Dans cette rubrique, le chemin d'accès au répertoire de téléchargement de FreeRTOS est appelé. *`freertos`*
- Les espaces dans le chemin d'accès *`freertos`* peuvent provoquer des échecs de construction. Lorsque vous clonez ou copiez le référentiel, assurez-vous que le chemin d'accès que vous créez ne contient pas d'espaces.
- La longueur maximale d'un chemin sous Microsoft Windows est de 260 caractères. Les longs chemins de répertoire de téléchargement de FreeRTOS peuvent provoquer des échecs de compilation.
- Le code source pouvant contenir des liens symboliques, si vous utilisez Windows pour extraire l'archive, vous devrez peut-être :
  - Activez le mode développeur ou
  - Utilisez une console élevée en tant qu'administrateur.

  Ainsi, Windows peut créer correctement des liens symboliques lors de l'extraction de l'archive. Dans le cas contraire, les liens symboliques seront écrits sous forme de fichiers normaux contenant les chemins des liens symboliques sous forme de texte ou seront vides. Pour plus d'informations, consultez le billet de blog Symlinks in Windows 10 ! .

  Si vous utilisez Git sous Windows, vous devez activer le mode développeur ou vous devez :
  - Définissez `core.symlinks` ce paramètre sur true à l'aide de la commande suivante :

    ```
    git config --global core.symlinks true
    ```
  - Utilisez une console élevée en tant qu'administrateur chaque fois que vous utilisez une commande git qui écrit sur le système (par exemplegit pull,git clone, etgit submodule update --init --recursive).
- Comme indiqué dansTéléchargement de FreeRTOS, les ports FreeRTOS pour Cypress ne sont actuellement disponibles que sur. GitHub

Présentation de

Ce didacticiel comprend les instructions de mise en route suivantes :

1. Installation de logiciels sur la machine hôte pour développer et déboguer des applications intégrées pour votre carte de microcontrôleur.
2. Compilation croisée d'une application de démonstration FreeRTOS en une image binaire.
3. Chargement de l'image binaire de l'application dans votre carte et exécution de l'application.
4. Interaction avec l'application s'exécutant sur votre carte via une connexion série, à des fins de surveillance et de débogage.

Configuration de votre environnement de développement

Télécharger et installer le kit SDK WICED

Dans ce guide de démarrage, vous utilisez le SDK Cypress WICED Studio pour programmer votre carte avec la démo de FreeRTOS. Visitez le site web WICED Software pour télécharger le kit SDK WICED Studio depuis Cypress. Vous devez avoir un compte Cypress gratuit pour télécharger le logiciel. Le kit SDK WICED Studio est compatible avec les systèmes d'exploitation Windows, macOS et Linux.

> **Note**
>
> Certains systèmes d'exploitation nécessitent des étapes d'installation supplémentaires. Assurez-vous de lire et de suivre toutes les instructions d'installation du système d'exploitation et de la version de WICED Studio que vous installez.

Définir les variables d'environnement

Avant d'utiliser WICED Studio pour programmer votre carte, vous devez créer une variable d'environnement pour le répertoire d'installation du kit SDK WICED Studio SDK. Si WICED Studio est en cours d'exécution tandis que vous créez vos variables, vous devez redémarrer l'application après avoir défini vos variables.

> **Note**
>
> Le programme d'installation WICED Studio crée deux dossiers distincts nommés `WICED-Studio-`*`m`*`.`*`n`* sur votre machine, où `m` et `n` sont les numéros de version majeure et mineure, respectivement. Ce document assume un nom de dossier `WICED-Studio-6.2` mais veillez à utiliser le nom correct pour la version que vous installez. Lorsque vous définissez la variable

d'environnement `WICED_STUDIO_SDK_PATH`, veillez à spécifier le chemin d'installation complet du kit SDK WICED Studio, et non pas le chemin d'installation de l'interface utilisateur WICED Studio. Dans Windows et macOS, le dossier `WICED-Studio-`*`m.n`* du kit SDK est créé dans le dossier `Documents` par défaut.

Pour créer les variables d'environnement sous Windows

1. Dans le Panneau de configuration, cliquez sur Système, Paramètres système avancés.
2. Sous l'onglet Avancé, choisissez Variables d'environnement.
3. Sous Variables d'utilisateur, choisissez Nouvelle.
4. Pour Nom de la variable :, tapez **`WICED_STUDIO_SDK_PATH`**. Pour Valeur de la variable, entrez le répertoire d'installation du kit SDK WICED Studio.

Pour créer la variable d'environnement sur Linux ou macOS

1. Ouvrez le fichier `/etc/profile` sur votre ordinateur, puis ajoutez la ligne suivante à la fin du fichier :

   ```
   export WICED_STUDIO_SDK_PATH=installation-path/WICED-Studio-6.2
   ```

2. Redémarrez l'ordinateur.
3. Ouvrez une fenêtre de terminal et exécutez les commandes suivantes :

   ```
   cd freertos/vendors/cypress/WICED_SDK
   ```

   ```
   perl platform_adjust_make.pl
   ```

   ```
   chmod +x make
   ```

Établissement d'une connexion série

Pour établir une connexion série entre votre machine hôte et votre carte

1. Connectez la carte à votre ordinateur hôte à l'aide d'un Micro-B câble USB Standard-A .
2. Identifiez le numéro de port série USB pour la connexion à la carte sur votre ordinateur hôte.

3. Démarrez une fenêtre de terminal série et ouvrez une connexion avec les paramètres suivants :

   - Vitesse de transmission : 115200
   - Données : 8 bits
   - Parité : aucune
   - Bits d'arrêt : 1
   - Contrôle de flux : aucun

Pour plus d'informations sur l'installation d'une fenêtre de terminal et la configuration d'une connexion série, consultez Installation d'un émulateur de terminal.

Surveillance des messages MQTT dans le cloud

Avant de lancer le projet de démonstration FreeRTOS, vous pouvez configurer le client MQTT dans la console pour surveiller AWS IoT les messages que votre appareil envoie au Cloud. AWS

Pour vous abonner à la rubrique MQTT avec AWS IoT Client MQTT

1. Connectez-vous à la console AWS IoT.
2. Dans le volet de navigation, choisissez Test, puis choisissez MQTT test client pour ouvrir le client MQTT.
3. Dans le champ Rubrique d'abonnement, saisissez ***your-thing-name*****/example/topic**, puis choisissez S'abonner à la rubrique.

Créez et exécutez le projet de démonstration FreeRTOS

Après avoir configuré une connexion série avec votre carte, vous pouvez créer le projet de démonstration FreeRTOS, flasher la démo sur votre carte, puis exécuter la démonstration.

Pour créer et exécuter le projet de démonstration FreeRTOS dans WICED Studio

1. Lancez WICED Studio.
2. Dans le menu File (Fichier), choisissez Import (Importer). Développez le dossier `General`, choisissez Projets existants dans l'espace de travail, puis choisissez Suivant.
3. Dans Select root directory (Sélectionner le répertoire racine), sélectionnez Browse..., (Parcourir...), accédez au chemin *`freertos`*`/projects/cypress/CYW954907AEVAL1F/wicedstudio`, puis sélectionnez OK.

4. Sous Projects (Projets), cochez la case correspondant uniquement au projet aws_demo. Choisissez Terminer pour importer le projet. Le projet cible aws_demo doit apparaître dans la fenêtre Créer la cible.
5. Développez le menu WICED Platform (Plateforme WICED) et choisissez WICED Filters off (Filtres WICED désactivés).
6. Dans la fenêtre Créer le filtre, développez aws_demo, cliquez avec le bouton droit de la souris sur le fichier `demo.aws_demo`, puis choisissez Créer la cible afin de créer la démonstration et de la télécharger sur votre carte. Normalement, la démonstration s'exécute automatiquement une fois qu'elle a été créée et téléchargée sur votre carte.

Résolution des problèmes

- Si vous utilisez Windows, il se peut que vous receviez le message d'erreur suivant lorsque vous développez et exécutez le projet de démonstration :

  ```
  : recipe for target 'download_dct' failed
  make.exe[1]: *** [download_dct] Error 1
  ```

  Pour résoudre ce problème, utilisez la procédure suivante :

  1. Accédez à *WICED-Studio-SDK-PATH*`\WICED-Studio-6.2\43xxx_Wi-Fi\tools\OpenOCD\Win32` et double-cliquez sur `openocd-all-brcm-libftdi.exe`.
  2. Accédez à *WICED-Studio-SDK-PATH*`\WICED-Studio-6.2\43xxx_Wi-Fi\tools\drivers\CYW9WCD1EVAL1` et double-cliquez sur `InstallDriver.exe`.
- Si vous utilisez Linux or macOS, il se peut que vous receviez le message d'erreur suivant lorsque vous développez et exécutez le projet de démonstration :

  ```
  make[1]: *** [download_dct] Error 127
  ```

  Pour résoudre cette erreur, cliquez sur la commande suivante pour ouvrir le package libusb-dev :

  ```
  sudo apt-get install libusb-dev
  ```

Pour obtenir des informations générales sur la résolution des problèmes liés à la prise en main de FreeRTOS, consultez. Résolution des problèmes de mise en route

Commencer à utiliser le kit Cypress CY8CKIT-064S0S2-4343W

> **Important**
>
> Cette intégration de référence est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Ce didacticiel fournit des instructions pour démarrer avec le CY8CKIT-064S0S2-4343Wkit. Si vous n'en avez pas déjà un, vous pouvez utiliser ce lien pour acheter un kit. Vous pouvez également utiliser ce lien pour accéder au guide de l'utilisateur du kit.

Prise en main

Avant de commencer, vous devez configurer AWS IoT FreeRTOS pour connecter votre appareil au Cloud. AWS Pour obtenir des instructions, veuillez consulter Premiers pas. Une fois que vous aurez rempli les prérequis, vous aurez un package FreeRTOS avec des informations d'identification. AWS IoT Core

> **Note**
>
> Dans ce didacticiel, le chemin d'accès au répertoire de téléchargement de FreeRTOS créé dans la section « Premiers pas » est appelé. `freertos`

Configuration de l'environnement de développement

FreeRTOS fonctionne avec un flux de construction CMake ou Make. Vous pouvez l'utiliser ModusToolbox pour votre flux de génération Make. Vous pouvez utiliser l'IDE Eclipse fourni avec ModusToolbox ou un IDE partenaire tel que IAR EW-Arm, Arm MDK ou Microsoft Visual Studio Code. L'IDE Eclipse est compatible avec les systèmes d'exploitation Windows, macOS et Linux.

Avant de commencer, téléchargez et installez la dernière version du ModusToolbox logiciel. Pour plus d'informations, consultez le guide ModusToolbox d'installation.

Outils de mise à jour pour la ModusToolbox version 2.1 ou antérieure

Si vous utilisez l'IDE Eclipse ModusToolbox 2.1 pour programmer ce kit, vous devez mettre à jour Firmware-loader OpenOCD et ses outils.

Dans les étapes suivantes, le *ModusToolbox* chemin par défaut pour :

- Windows l'est`C:\Users\`*user_name*`\ModusToolbox`.
- Linux est *user_home*`/ModusToolbox` ou l'endroit où vous choisissez d'extraire le fichier d'archive.
- macOS se trouve dans le dossier Applications du volume que vous sélectionnez dans l'assistant.

Mettre à jour OpenOCD

Ce kit nécessite Cypress OpenOCD 4.0.0 ou version ultérieure pour effacer et programmer correctement la puce.

Pour mettre à jour Cypress OpenOCD

1. Accédez à la page de publication de Cypress OpenOCD.
2. Téléchargez le fichier d'archive correspondant à votre système d'exploitation (Windows/Mac/Linux).
3. Supprimez les fichiers existants dans*ModusToolbox*`/tools_2.x/openocd`.
4. Remplacez les fichiers *ModusToolbox*`/tools_2.x/openocd` par le contenu extrait de l'archive que vous avez téléchargée lors d'une étape précédente.

Mise à jour Firmware-loader

Ce kit nécessite Cypress Firmware-loader 3.0.0 ou version ultérieure.

Pour mettre à jour Cypress Firmware-loader

1. Accédez à la page de Firmware-loader publication de Cypress.
2. Téléchargez le fichier d'archive correspondant à votre système d'exploitation (Windows/Mac/Linux).
3. Supprimez les fichiers existants dans*ModusToolbox*`/tools_2.x/fw-loader`.
4. Remplacez les fichiers *ModusToolbox*`/tools_2.x/fw-loader` par le contenu extrait de l'archive que vous avez téléchargée lors d'une étape précédente.

Vous pouvez également utiliser CMake pour générer des fichiers de construction de projet à partir du code source de l'application FreeRTOS, créer le projet à l'aide de votre outil de génération préféré, puis programmer le kit à l'aide d'OpenOCD. Si vous préférez utiliser un outil graphique pour programmer avec le flux CMake, téléchargez et installez Cypress Programmer depuis la page Web de Cypress Programming Solutions. Pour de plus amples informations, veuillez consulter Utilisation de CMake avec FreeRTOS.

Configuration du matériel

Suivez ces étapes pour configurer le matériel du kit.

1. Approvisionnez votre kit

   Suivez les instructions du guide d'approvisionnement pour le CY8CKIT-064S0S2-4343W kit afin d'approvisionner votre kit en toute sécurité. AWS IoT

   Ce kit nécessite la CySecureTools version 3.1.0 ou une version ultérieure.
2. Configuration d'une connexion série

   a. Connectez le kit à votre ordinateur hôte.

   b. Le port série USB du kit est automatiquement répertorié sur l'ordinateur hôte. Identifiez le numéro de port. Sous Windows, vous pouvez l'identifier à l'aide du gestionnaire de périphériques sous Ports (COM et LPT).

   c. Démarrez une fenêtre de terminal série et ouvrez une connexion avec les paramètres suivants :

      - Vitesse de transmission : 115200
      - Données : 8 bits
      - Parité : aucune
      - Bits d'arrêt : 1
      - Contrôle de flux : aucun

Créez et exécutez le projet de démonstration FreeRTOS

Dans cette section, vous allez créer et exécuter la démo.

1. Assurez-vous de suivre les étapes décrites dans le guide de provisionnement du CY8CKIT-064S0S2-4343W kit.

2. Créez la démo de FreeRTOS.

   a. Ouvrez l'IDE Eclipse pour ModusToolbox et choisissez ou créez un espace de travail.

   b. Dans le menu File (Fichier), choisissez Import (Importer).

      Développez Général, choisissez Projet existant dans l'espace de travail, puis cliquez sur Suivant.

   c. Dans le répertoire racine, entrez *`freertos`*`/projects/cypress/CY8CKIT-064S0S2-4343W/mtb/aws_demos` puis sélectionnez le nom du projet`aws_demos`. Il doit être sélectionné par défaut.

   d. Choisissez Terminer pour importer le projet dans votre espace de travail.

   e. Créez l'application en effectuant l'une des opérations suivantes :

      - Dans le panneau rapide, sélectionnez Créer une application aws_demos.
      - Choisissez Project, puis Build All.

      Assurez-vous que le projet est compilé sans erreur.

3. Surveillance des messages MQTT sur le cloud

   Avant de lancer la démonstration, vous pouvez configurer le client MQTT dans la AWS IoT console pour surveiller les messages que votre appareil envoie au AWS Cloud. Pour vous abonner à la rubrique MQTT avec le client AWS IoT MQTT, procédez comme suit.

   a. Connectez-vous à la console AWS IoT.

   b. Dans le volet de navigation, choisissez Test, puis choisissez MQTT test client pour ouvrir le client MQTT.

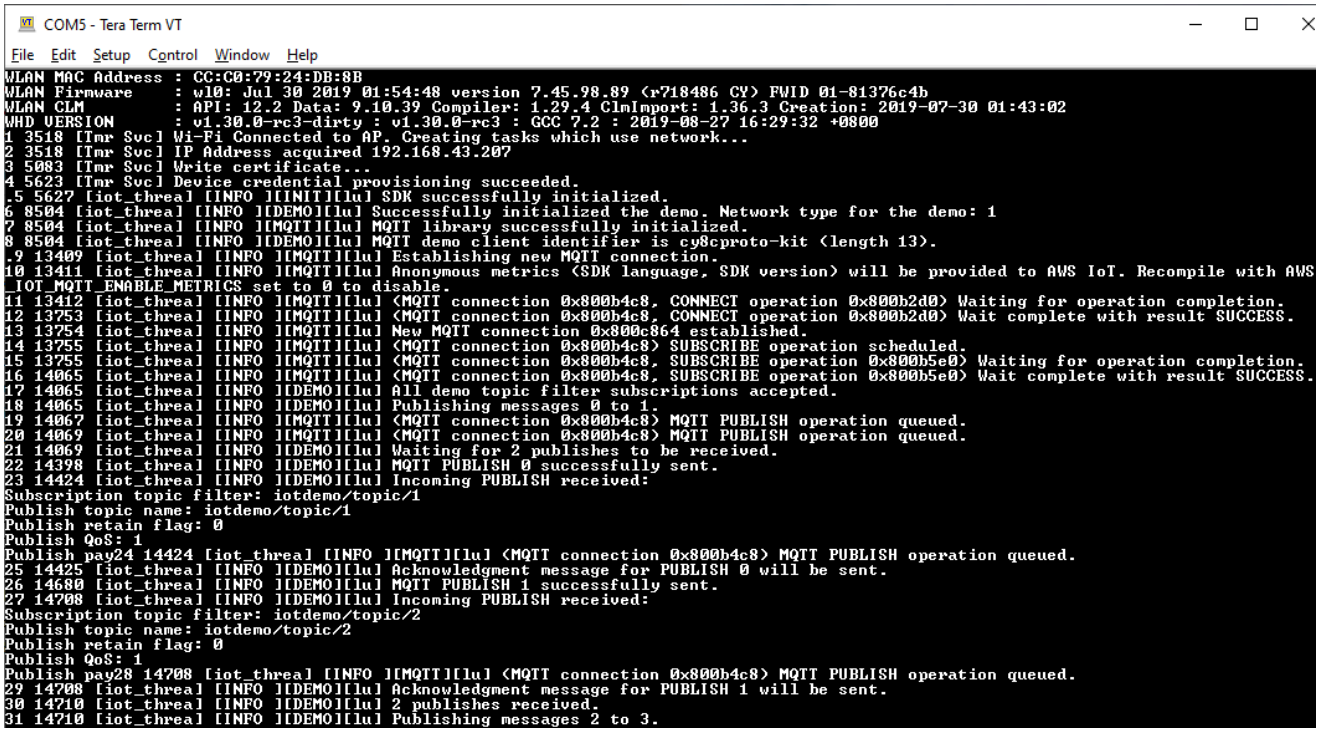
   c. Pour le sujet d'abonnement***`your-thing-name`*`/example/topic`**, entrez, puis choisissez S'abonner au sujet.

4. Exécutez le projet de démonstration FreeRTOS

   a. Sélectionnez le projet `aws_demos` dans l'espace de travail.

   b. Dans le panneau rapide, sélectionnez aws_demos Program (KitProg3). Cela programme la carte et l'application de démonstration démarre une fois la programmation terminée.

   c. Vous pouvez consulter l'état de l'application en cours d'exécution dans le terminal série. La figure suivante montre une partie de la sortie du terminal.



La démo MQTT publie des messages sur quatre sujets différents (`iotdemo/topic/`*n* où n = 1 à 4) et s'abonne à tous ces sujets pour recevoir les mêmes messages en retour. Lorsqu'un message est reçu, la démo publie un message d'accusé de réception sur le sujet`iotdemo/acknowledgements`. La liste suivante décrit les messages de débogage qui apparaissent dans la sortie du terminal, avec des références aux numéros de série des messages. Dans la sortie, les détails du pilote hôte WICED (WHD) sont d'abord imprimés sans numérotation de série.

1. 1 à 4 — L'appareil se connecte au point d'accès (AP) configuré et est approvisionné en se connectant au AWS serveur à l'aide du point de terminaison et des certificats configurés.
2. 5 à 13 — La bibliothèque CoreMQTT est initialisée et le périphérique établit une connexion MQTT.
3. 14 à 17 — L'appareil s'abonne à tous les sujets pour recevoir en retour les messages publiés.
4. 18 à 30 — L'appareil publie deux messages et attend de les recevoir en retour. Lorsque chaque message est reçu, l'appareil envoie un message d'accusé de réception.

Le même cycle de publication, de réception et de confirmation se poursuit jusqu'à ce que tous les messages soient publiés. Deux messages sont publiés par cycle jusqu'à ce que le nombre de cycles configuré soit terminé.

5. Utilisation de CMake avec FreeRTOS

   Vous pouvez également utiliser CMake pour créer et exécuter l'application de démonstration. Pour configurer CMake et un système de génération natif, voirConditions préalables.

   a. Utilisez la commande suivante pour générer les fichiers de compilation. Spécifiez le tableau cible à l'aide de l'`-DBOARD`option.

   ```
   cmake -DVENDOR=cypress -DBOARD=CY8CKIT_064S0S2_4343W -DCOMPILER=arm-gcc -
   S freertos -B build_dir
   ```

   Si vous utilisez Windows, vous devez spécifier le système de génération natif à l'aide de l'`-G`option car CMake utilise Visual Studio par défaut.

   Example

   ```
   cmake -DVENDOR=cypress -DBOARD=CY8CKIT_064S0S2_4343W -DCOMPILER=arm-gcc -
   S freertos -B build_dir -G Ninja
   ```

   Si `arm-none-eabi-gcc` n'est pas dans votre chemin d'accès shell, vous devez également définir la variable CMake `AFR_TOOLCHAIN_PATH`.

   Example

   ```
    -DAFR_TOOLCHAIN_PATH=/home/user/opt/gcc-arm-none-eabi/bin
   ```

   b. Utilisez la commande suivante pour créer le projet à l'aide de CMake.

   ```
   cmake --build build_dir
   ```

   c. Enfin, programmez les `cm4.hex` fichiers `cm0.hex` et générés sous à l'aide *`build_dir`* de Cypress Programmer.

Exécution d'autres démos

Les applications de démonstration suivantes ont été testées et vérifiées pour fonctionner avec la version actuelle. Vous trouverez ces démos dans le *freertos*/demos répertoire. Pour plus d'informations sur la façon d'exécuter ces démos, consultezDémos de FreeRTOS.

- Démonstration du Bluetooth Low Energy
- Over-the-Air Démo des mises à
- Démo du client Secure Sockets Echo
- AWS IoT Démo de Device Shadow

Débogage

Le KitProg 3 du kit prend en charge le débogage via le protocole SWD.

- Pour déboguer l'application FreeRTOS, sélectionnez le projet aws_demos dans l'espace de travail, puis sélectionnez aws_demos Debug (3) dans le Quick Panel. KitProg

Mises à jour OTA

Les microcontrôleurs PSoC 64 ont réussi tous les tests de qualification FreeRTOS requis. Cependant, la fonctionnalité OTA (over-the-air) optionnelle implémentée dans la bibliothèque de microprogrammes PSoC 64 Standard Secure est toujours en attente AWS d'évaluation. La fonctionnalité OTA telle qu'elle est implémentée passe actuellement tous les tests de qualification OTA à l'exception de aws_ota_test_case_rollback_if_unable_to_connect_after_update.py.

Lorsqu'une image OTA validée avec succès est appliquée à un appareil à l'aide du microcontrôleur pSOc64 Standard Secure et que le périphérique ne peut pas communiquer avec celui-ci AWS IoT Core, il ne peut pas revenir automatiquement à l'image originale dont le fonctionnement a été vérifié. AWS Cela peut rendre l'appareil inaccessible AWS IoT Core pour d'autres mises à jour. Cette fonctionnalité est toujours en cours de développement par l'équipe Cypress.

Pour plus d'informations, consultez la section Mises à jour OTA avec AWS et le CY8CKIT-064S0S2-4343W kit. Si vous avez d'autres questions ou si vous avez besoin d'une assistance technique, contactez la communauté des développeurs Cypress.

Démarrez avec l'élément sécurisé Microchip ATECC608A avec simulateur Windows

> **Important**
>
> Cette intégration de référence est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Ce didacticiel fournit des instructions pour commencer à utiliser l'élément sécurisé Microchip ATECC608A avec simulateur Windows.

Vous aurez besoin du matériel suivant :

- Élément sécurisé Microchip ATECC608A (clickboard)
- SAMD21 XPlained Pro
- Adaptateur mikroBUS Xplained Pro

Avant de commencer, vous devez configurer AWS IoT et télécharger FreeRTOS pour connecter votre appareil au Cloud. AWS Pour obtenir des instructions, consultez Premiers pas. Dans ce didacticiel, le chemin d'accès au répertoire de téléchargement de FreeRTOS est appelé. *`freertos`*

Présentation de

Ce didacticiel contient les étapes suivantes :

1. Connectez votre carte à un appareil hôte.
2. Installation de logiciels sur la machine hôte pour développer et déboguer des applications intégrées pour votre carte de microcontrôleur.
3. Cross-compile une application de démonstration FreeRTOS sur une image binaire.
4. Chargement de l'image binaire de l'application dans votre carte et exécution de l'application.

Configuration du matériel Microchip ATECC608A

Avant de pouvoir interagir avec votre périphérique Microchip ATECC608A, vous devez programmer la carte SAMD21.

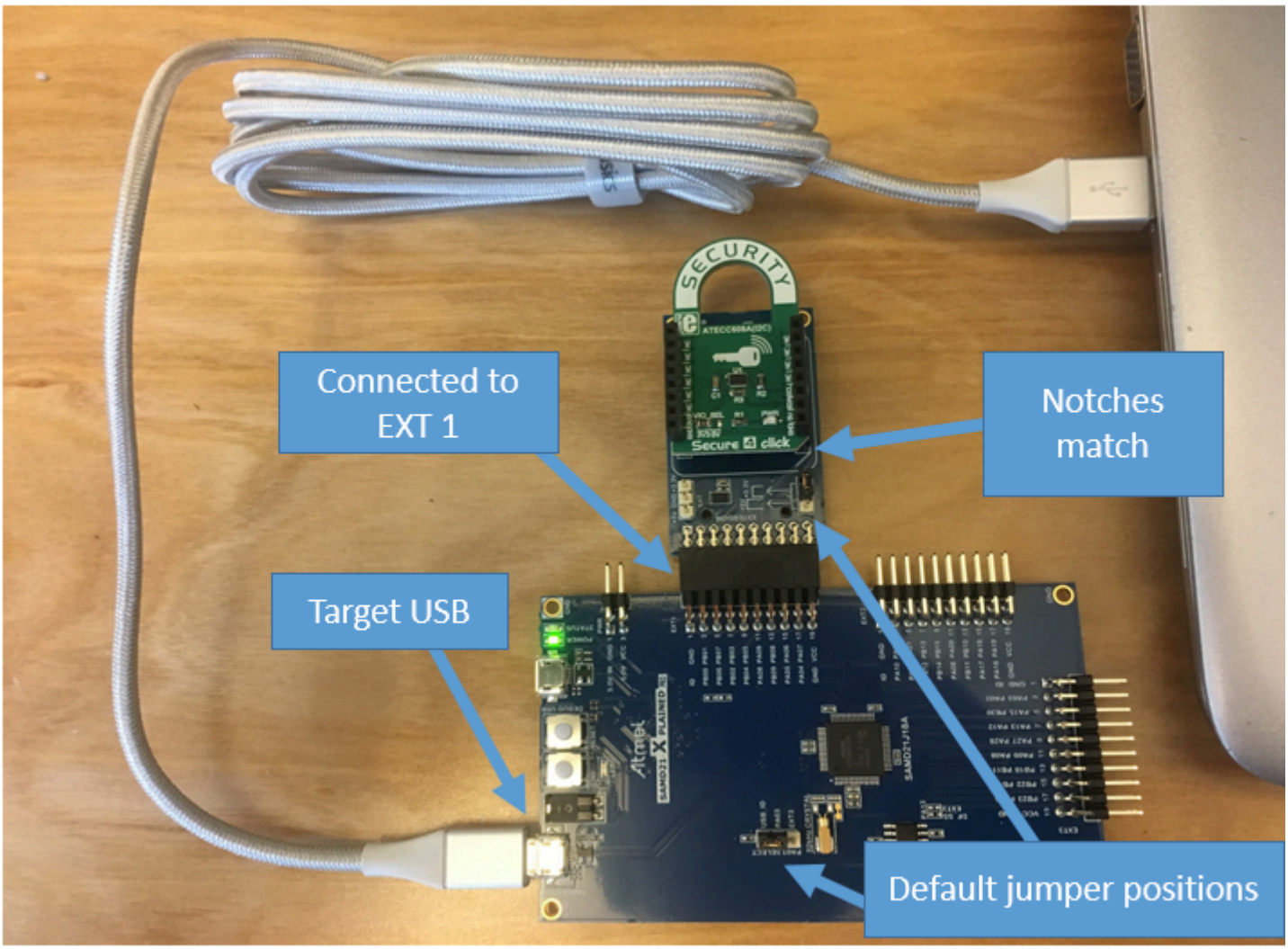
Pour configurer la carte SAMD21 XPlained Pro

1. Suivez le lien CryptoAuthSSH-XSTK (DM320109) - Dernière version du firmware pour télécharger un fichier .zip contenant des instructions (PDF) et un fichier binaire qui peut être programmé sur le D21.
2. Téléchargez et installez l'IDP Atmel Studio 7. Assurez-vous de sélectionner l'architecture du pilote SMART ARM MCU lors de l'installation.
3. Utilisez un câble USB 2.0 Micro B pour connecter le connecteur « Debug USB » à votre ordinateur et suivez les instructions du PDF. (Le connecteur « Debug USB » est le port USB le plus proche des broches et du voyant POWER.)

Pour connecter le matériel

1. Débranchez le câble USB micro du connecteur Debug USB.
2. Branchez l'adaptateur mikroBUS XPlained Pro sur la carte SAMD21 à l'emplacement EXT1.
3. Enfichez la carte ATECC608a Secure 4 Click sur la carte mikroBUSX XPlained Pro. Assurez-vous que le coin cranté du module Click board correspond à l'icône crantée de la carte.
4. Branchez le câble USB micro sur le port USB cible.

Votre configuration doit ressembler à ce qui suit.



Configurer votre environnement de développement.

Inscrivez-vous pour un Compte AWS

Pour commencer AWS, vous avez besoin d'un Compte AWS. Pour plus d'informations sur la création d'un Compte AWS, voir Getting started with an Compte AWS dans le Guide de Gestion de compte AWS référence.

Pour activer l'accès, ajoutez des autorisations à vos utilisateurs, groupes ou rôles :

- Utilisateurs et groupes dans AWS IAM Identity Center :

  Créez un jeu d'autorisations. Suivez les instructions de la rubrique Création d'un jeu d'autorisations du Guide de l'utilisateur AWS IAM Identity Center .
- Utilisateurs gérés dans IAM par un fournisseur d'identité :

Créez un rôle pour la fédération d'identité. Suivez les instructions de la rubrique Création d'un rôle pour un fournisseur d'identité tiers (fédération) dans le Guide de l'utilisateur IAM.

- Utilisateurs IAM :
  - Créez un rôle que votre utilisateur peut assumer. Suivez les instructions de la rubrique Création d'un rôle pour un utilisateur IAM dans le Guide de l'utilisateur IAM.
  - (Non recommandé) Attachez une politique directement à un utilisateur ou ajoutez un utilisateur à un groupe d'utilisateurs. Suivez les instructions de la rubrique Ajout d'autorisations à un utilisateur (console) du Guide de l'utilisateur IAM.

Configuration

1. Téléchargez le dépôt FreeRTOS depuis le référentiel FreeRTOS. GitHub

   Pour télécharger FreeRTOS depuis : GitHub

   1. Accédez au référentiel FreeRTOS GitHub .
   2. Choisissez Cloner ou télécharger.
   3. À partir de la ligne de commande sur votre ordinateur, clonez le référentiel dans un répertoire sur votre machine hôte.

      ```
      git clone https://github.com/aws/amazon-freertos.git -\-recurse-submodules
      ```

      Important

      - Dans cette rubrique, le chemin d'accès au répertoire de téléchargement de FreeRTOS est appelé. *freertos*
      - Les espaces dans le chemin d'accès *freertos* peuvent provoquer des échecs de construction. Lorsque vous clonez ou copiez le référentiel, assurez-vous que le chemin d'accès que vous créez ne contient pas d'espaces.
      - La longueur maximale d'un chemin sous Microsoft Windows est de 260 caractères. Les longs chemins de répertoire de téléchargement de FreeRTOS peuvent provoquer des échecs de compilation.
      - Le code source pouvant contenir des liens symboliques, si vous utilisez Windows pour extraire l'archive, vous devrez peut-être :
        - Activez le mode développeur ou

- Utilisez une console élevée en tant qu'administrateur.

Ainsi, Windows peut créer correctement des liens symboliques lors de l'extraction de l'archive. Dans le cas contraire, les liens symboliques seront écrits sous forme de fichiers normaux contenant les chemins des liens symboliques sous forme de texte ou seront vides. Pour plus d'informations, consultez le billet de blog Symlinks in Windows 10 ! .

Si vous utilisez Git sous Windows, vous devez activer le mode développeur ou vous devez :

- Définissez `core.symlinks` ce paramètre sur true à l'aide de la commande suivante :

```
git config -\-global core.symlinks true
```

- Utilisez une console élevée en tant qu'administrateur chaque fois que vous utilisez une commande git qui écrit sur le système (par exemplegit pull,git clone, etgit submodule update -\-init -\-recursive).

4. Dans le répertoire *`freertos`*, consultez la branche à utiliser.

2. Configurez votre environnement de développement.

   a. Installez la dernière version de WinPCap.

   b. Installez Microsoft Visual Studio.

      Les versions Visual Studio 2017 et 2019 fonctionnent. Toutes les éditions de ces versions de Visual Studio sont prises en charge (Community, Professionnel ou Entreprise).

      En plus de l'IDE, installez le composant Développement de bureau avec C++. Ensuite, sous Facultatif, installez le dernier SDK Windows 10.

   c. Assurez-vous que vous avez une connexion Ethernet câblée active.

Créez et exécutez le projet de démonstration FreeRTOS

> **Important**
>
> Le périphérique Microchip ATECC608A inclut une initialisation unique qui est verrouillée sur le périphérique la première fois qu'un projet est exécuté (pendant l'appel à `C_InitToken`).

Cependant, le projet de démonstration FreeRTOS et le projet de test ont des configurations différentes. Si le périphérique est verrouillé pendant les configurations du projet de démonstration, tous les tests du projet de test ne pourront pas aboutir.

Pour créer et exécuter le projet de démonstration FreeRTOS avec l'IDE Visual Studio

1. Chargez le projet dans Visual Studio.

   Dans le menu File (Fichier), choisissez Open (Ouvrir). Choisissez File/Solution, naviguez jusqu'au *freertos*`\projects\microchip\ecc608a_plus_winsim\visual_studio\aws_demos\aws_demos.sln` fichier, puis choisissez Ouvrir.

2. Reciblez le projet de démonstration.

   Le projet de démonstration dépend du kit SDK Windows, mais il n'a pas de version du kit SDK Windows spécifiée. Par défaut, l'IDE peut tenter de générer la démonstration avec une version du kit SDK qui n'est pas présente sur votre ordinateur. Pour définir la version du kit SDK Windows, cliquez avec le bouton droit de la souris sur aws_demos, puis choisissez Recibler les projets. Cette opération permet d'ouvrir la fenêtre Examiner les actions de la solution. Choisissez une version du kit SDK Windows qui est présente sur votre ordinateur (utilisez la valeur initiale dans la liste déroulante), puis choisissez OK.

3. Créez et exécutez le projet.

   Dans le menu Créer, choisissez Créer une solution et assurez-vous que la solution est générée sans erreur. Choisissez Dégoguer, Démarrer le débogage pour exécuter le projet. Lors de la première exécution, vous devez configurer l'interface de votre appareil et recompiler. Pour de plus amples informations, veuillez consulter Configurer votre interface réseau.

4. Allouez le composant Microchip ATECC608A.

   Microchip fournit plusieurs outils de script pour aider à configurer les composants ATECC608A. Naviguez jusqu'au *freertos*`\vendors\microchip\secure_elements\app\example_trust_chain_tool` README.md fichier et ouvrez-le.

   Suivez les instructions du fichier `README.md` pour mettre en service le périphérique. Voici les principales étapes à suivre :

   1. Créez et enregistrez une autorité de certification auprès de AWS.

2. Générez vos clés sur le composant Microchip ATECC608A et exportez la clé publique et le numéro de série du périphérique.
3. Générez un certificat pour l'appareil et enregistrez-le auprès de AWS.
4. Chargez le certificat CA et le certificat d'appareil sur le périphérique.

5. Créez et exécutez des exemples de FreeRTOS.

   Re-run le projet de démonstration à nouveau. Cette fois, vous devriez vous connecter.

Résolution des problèmes

Pour plus d'informations sur le dépannage, consultez Résolution des problèmes de mise en route.

Commencer à utiliser l'Espressif ESP32-DevKitC et le ESP-WROVER-KIT

> **Important**
>
> Cette intégration de référence est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

> **Note**
>
> Pour découvrir comment intégrer les bibliothèques modulaires et les démos FreeRTOS dans votre propre projet Espressif IDF, consultez notre article sur l'intégration de référence pour la plateforme. ESP32-C3

Suivez ce didacticiel pour démarrer avec l'Espressif ESP32-DevKitC équipé de ESP32-WROOM-32 ESP32-SOLO-1, ou de ESP-WROVER modules et du. ESP-WROVER-KIT-VB Pour en acheter un auprès de notre partenaire sur le catalogue des appareils AWS partenaires, cliquez sur les liens suivants :

- ESP32-WROOM-32 DevKitC
- ESP32-SOLO-1

- ESP32-WROVER-KIT

Ces versions des cartes de développement sont prises en charge sur FreeRTOS.

Pour plus d'informations sur les dernières versions de ces cartes, consultez la version ESP32-DevKitC V4 ou la ESP-WROVER-KIT v4.1 sur le site Web d'Espressif.

> **Note**
>
> Actuellement, le port FreeRTOS ESP32-WROVER-KIT pour DevKit ESP C ne prend pas en charge la fonctionnalité de multitraitement symétrique (SMP).

Présentation de

Ce didacticiel vous guide à travers les étapes suivantes :

1. Connexion de votre carte à un appareil hôte.
2. Installation de logiciels sur la machine hôte pour développer et déboguer des applications intégrées pour votre carte de microcontrôleur.
3. Compilation croisée d'une application de démonstration FreeRTOS en une image binaire.
4. Chargement de l'image binaire de l'application dans votre carte et exécution de l'application.
5. Interaction avec l'application s'exécutant sur votre carte via une connexion série, à des fins de surveillance et de débogage.

Conditions préalables

Avant de commencer à utiliser FreeRTOS sur votre tableau Espressif, vous devez configurer votre compte et vos autorisations. AWS

Inscrivez-vous pour un Compte AWS

Pour commencer AWS, vous avez besoin d'un Compte AWS. Pour plus d'informations sur la création d'un Compte AWS, voir Getting started with an Compte AWS dans le Guide de Gestion de compte AWS référence.

Pour activer l'accès, ajoutez des autorisations à vos utilisateurs, groupes ou rôles :

- Utilisateurs et groupes dans AWS IAM Identity Center :

Créez un jeu d'autorisations. Suivez les instructions de la rubrique Création d'un jeu d'autorisations du Guide de l'utilisateur AWS IAM Identity Center .

- Utilisateurs gérés dans IAM par un fournisseur d'identité :

  Créez un rôle pour la fédération d'identité. Suivez les instructions de la rubrique Création d'un rôle pour un fournisseur d'identité tiers (fédération) dans le Guide de l'utilisateur IAM.

- Utilisateurs IAM :
  - Créez un rôle que votre utilisateur peut assumer. Suivez les instructions de la rubrique Création d'un rôle pour un utilisateur IAM dans le Guide de l'utilisateur IAM.
  - (Non recommandé) Attachez une politique directement à un utilisateur ou ajoutez un utilisateur à un groupe d'utilisateurs. Suivez les instructions de la rubrique Ajout d'autorisations à un utilisateur (console) du Guide de l'utilisateur IAM.

Mise en route

> **Note**
>
> Les commandes Linux de ce didacticiel nécessitent que vous utilisiez le shell Bash.

1. Configurez le matériel Espressif.

   - Pour plus d'informations sur la configuration du matériel de la carte de ESP32-DevKitC développement, consultez le guide ESP32-DevKitC de démarrage de la version 4.
   - Pour plus d'informations sur la configuration du matériel de la carte de ESP-WROVER-KIT développement, consultez le ESP-WROVER-KIT V4.1 Guide de démarrage.

   > **Important**
   >
   > Lorsque vous atteignez la section Get Started des guides Espressif, arrêtez-vous, puis revenez aux instructions de cette page.

2. Téléchargez Amazon GitHubFreeRTOS depuis. (Pour obtenir des instructions, consultez le README.mdfichier.)
3. Configurez votre environnement de développement.

Pour communiquer avec votre tableau, vous devez installer une chaîne d'outils. Espressif fournit le logiciel ESP-IDF pour développer des logiciels pour leurs cartes. Comme ESP-IDF sa propre version du noyau FreeRTOS est intégrée en tant que composant, Amazon FreeRTOS inclut une version personnalisée de la ESP-IDF version 4.2 dans laquelle le noyau FreeRTOS a été supprimé. Cela permet de résoudre les problèmes liés aux fichiers dupliqués lors de la compilation. Pour utiliser la version personnalisée de la ESP-IDF version 4.2 incluse dans Amazon FreeRTOS, suivez les instructions ci-dessous correspondant au système d'exploitation de votre machine hôte.

Windows

1. Téléchargez ESP-IDF le programme d'installation universel en ligne pour Windows.
2. Exécutez le programme d'installation en ligne universel.
3. Lorsque vous arrivez à l'étape Télécharger ou utiliser ESP-IDF, sélectionnez Utiliser un ESP-IDF répertoire existant et définissez Choisir un ESP-IDF répertoire existant sur`freertos/vendors/espressif/esp-idf`.
4. Terminez l'installation.

macOS

1. Suivez les instructions de la configuration standard des prérequis de la chaîne d'outils (ESP-IDF v4.2) pour macOS.

   > **Important**
   >
   > Lorsque vous atteignez les instructions ESP-IDF « Obtenir » sous Prochaines étapes, arrêtez-vous, puis revenez aux instructions de cette page.

2. Ouvrez une fenêtre de ligne de commande.
3. Accédez au répertoire de téléchargement de FreeRTOS, puis exécutez le script suivant pour télécharger et installer la chaîne d'outils espressif pour votre plateforme.

   ```
   vendors/espressif/esp-idf/install.sh
   ```

4. Ajoutez les ESP-IDF outils de la chaîne d'outils au chemin de votre terminal à l'aide de la commande suivante.

```
source vendors/espressif/esp-idf/export.sh
```

Linux

1. Suivez les instructions de la configuration standard des prérequis de la chaîne d'outils (ESP-IDF v4.2) pour Linux.

   > **Important**
   >
   > Lorsque vous atteignez les instructions ESP-IDF « Obtenir » sous Prochaines étapes, arrêtez-vous, puis revenez aux instructions de cette page.

2. Ouvrez une fenêtre de ligne de commande.
3. Accédez au répertoire de téléchargement de FreeRTOS, puis exécutez le script suivant pour télécharger et installer la chaîne d'outils Espressif pour votre plateforme.

   ```
   vendors/espressif/esp-idf/install.sh
   ```

4. Ajoutez les ESP-IDF outils de la chaîne d'outils au chemin de votre terminal à l'aide de la commande suivante.

   ```
   source vendors/espressif/esp-idf/export.sh
   ```

4. Établissez une connexion série.

   a. Pour établir une connexion série entre votre machine hôte et le ESP32-DevKitC, vous devez installer les pilotes VCP CP210x USB to UART Bridge. Vous pouvez télécharger ces pilotes à partir de Silicon Labs.

      Pour établir une connexion série entre votre machine hôte et le ESP32-WROVER-KIT, vous devez installer le pilote de port COM virtuel FTDI. Vous pouvez télécharger ce pilote depuis FTDI.

   b. Suivez les étapes pour établir une connexion série avec l'ESP32.

   c. Une fois que vous avez établi une connexion série, notez le port série pour la connexion de votre carte. Vous en avez besoin pour flasher la démo.

Configuration des applications de démonstration FreeRTOS

Pour ce didacticiel, le fichier de configuration FreeRTOS se trouve à l'adresse. *freertos*/vendors/espressif/boards/*board-name*/aws_demos/config_files/FreeRTOSConfig.h (Par exemple, si cette option AFR_BOARD espressif.esp32_devkitc est sélectionnée, le fichier de configuration se trouve à l'adresse*freertos*/vendors/espressif/boards/esp32/aws_demos/config_files/FreeRTOSConfig.h.)

1. Si vous utilisez macOS ou Linux, ouvrez une invite du terminal. Si vous utilisez Windows, ouvrez l'application « ESP-IDF 4.x CMD » (si vous avez inclus cette option lors de l'installation de la ESP-IDF chaîne d'outils), ou l'application « Command Prompt » dans le cas contraire.
2. Pour vérifier que Python3 est installé, exécutez

   ```
   python --version
   ```

   La version installée s'affiche. Si Python 3.0.1 ou version ultérieure n'est pas installé, vous pouvez l'installer depuis le site Web de Python.
3. Vous avez besoin de l'interface de ligne de AWS commande (CLI) pour exécuter AWS IoT des commandes. Si vous utilisez Windows, utilisez la `easy_install awscli` commande pour installer la AWS CLI dans l'application « Command » ou « ESP-IDF 4.x CMD ».

   Si vous utilisez macOS ou Linux, reportez-vous à la section Installation de la AWS CLI.
4. Exécuter

   ```
   aws configure
   ```

   et configurez la AWS CLI avec votre ID de clé d' AWS accès, votre clé d'accès secrète et votre AWS région par défaut. Pour plus d'informations, consultez Configuration de l'interface CLI AWS (français non garanti).
5. Utilisez la commande suivante pour installer le AWS SDK pour Python (boto3) :

   - Sous Windows, dans l'application « Command » ou « ESP-IDF 4.x CMD », exécutez

     ```
     pip install boto3 --user
     ```

> **Note**
>
> Consultez la documentation de Boto3 pour plus de détails.

- Sur macOS ou Linux, exécutez

```
pip install tornado nose --user
```

puis exécutez

```
pip install boto3 --user
```

FreeRTOS inclut `SetupAWS.py` le script pour faciliter la configuration de votre carte Espressif à laquelle vous connecter. AWS IoT Pour configurer le script, ouvrez *`freertos`*`/tools/aws_config_quick_start/configure.json` et définissez les attributs suivants :

`afr_source_dir`

Chemin d'accès complet vers l'annuaire *`freertos`* sur votre ordinateur. Assurez-vous d'utiliser des barres obliques pour spécifier ce chemin.

`thing_name`

Le nom que vous souhaitez attribuer à l' AWS IoT élément qui représente votre tableau.

`wifi_ssid`

Le SSID de votre Wi-Fi réseau.

`wifi_password`

Le mot de passe de votre Wi-Fi réseau.

`wifi_security`

Type de sécurité de votre Wi-Fi réseau.

Les types de sécurité suivants sont valides :

- `eWiFiSecurityOpen` (Ouvrir, aucune sécurité)
- `eWiFiSecurityWEP` (Sécurité WEP)

- `eWiFiSecurityWPA` (Sécurité WPA)
- `eWiFiSecurityWPA2` (Sécurité WPA2)

6. Exécutez le script de configuration.

   a. Si vous utilisez macOS ou Linux, ouvrez une invite du terminal. Si vous utilisez Windows, ouvrez l'application « ESP-IDF 4.x CMD » ou « Command ».

   b. Accédez au *`freertos`*`/tools/aws_config_quick_start` répertoire et exécutez

   ```
   python SetupAWS.py setup
   ```

   Le script effectue les opérations suivantes :

   - Crée un objet, un certificat et une politique IoT.
   - Attache la politique IoT au certificat et le certificat à l' AWS IoT objet.
   - Remplit le `aws_clientcredential.h` fichier avec votre AWS IoT point de terminaison, votre Wi-Fi SSID et vos informations d'identification.
   - Formate votre certificat et votre clé privée et les écrit dans le fichier `aws_clientcredential_keys.h` d'en-tête.

   > **Note**
   >
   > Le certificat est codé en dur à des fins de démonstration uniquement. Production-level les applications doivent stocker ces fichiers dans un emplacement sécurisé.

   Pour plus d'informations`SetupAWS.py`, consultez le `README.md` dans le *`freertos`*`/tools/aws_config_quick_start` répertoire.

Surveillance des messages MQTT dans le cloud

Avant de lancer le projet de démonstration FreeRTOS, vous pouvez configurer le client MQTT dans la console pour surveiller AWS IoT les messages que votre appareil envoie au Cloud. AWS

Pour vous abonner à la rubrique MQTT avec le AWS IoT Client MQTT

1. Accédez à la console AWS IoT.

2. Dans le volet de navigation, choisissez Test, puis MQTT Test Client.
3. Dans le champ Rubrique d'abonnement, saisissez *your-thing-name*/example/topic, puis choisissez S'abonner à la rubrique.

Lorsque le projet de démonstration s'exécute avec succès sur votre appareil, le message « Hello World ! » s'affiche envoyé plusieurs fois au sujet auquel vous vous êtes abonné.

Créez, flashez et exécutez le projet de démonstration FreeRTOS à l'aide du script idf.py

Vous pouvez utiliser l'utilitaire IDF (`idf.py`) d'Espressif pour créer le projet et flasher les fichiers binaires sur votre appareil.

> **Note**
>
> Certaines configurations peuvent nécessiter que vous utilisiez l'option port `"-p port-name"` with `idf.py` pour spécifier le port correct, comme dans l'exemple suivant.
>
> ```
> idf.py -p /dev/cu.usbserial-00101301B flash
> ```

Créez et flashez des FreeRTOS sous Windows, Linux et macOS (v4.2) ESP-IDF

1. Naviguez jusqu'à la racine de votre répertoire de téléchargement de FreeRTOS.
2. Dans une fenêtre de ligne de commande, entrez la commande suivante pour ajouter les ESP-IDF outils au PATH de votre terminal.

   Windows (application « Command »)

   ```
   vendors\espressif\esp-idf\export.bat
   ```

   Windows (application « ESP-IDF 4.x CMD »)

   (Cela a déjà été fait lorsque vous avez ouvert l'application.)

   Linux/ macOS

   ```
   source vendors/espressif/esp-idf/export.sh
   ```

3. Configurez cmake dans le `build` répertoire et créez l'image du microprogramme avec la commande suivante.

```
idf.py -DVENDOR=espressif -DBOARD=esp32_wrover_kit -DCOMPILER=xtensa-esp32 build
```

Vous devriez voir une sortie semblable à la suivante.

```
   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'
```

S'il n'y a aucune erreur, la compilation générera les fichiers binaires .bin du microprogramme.

4. Effacez la mémoire flash de votre carte de développement à l'aide de la commande suivante.

```
idf.py erase_flash
```

5. Utilisez le `idf.py` script pour flasher le binaire de l'application sur votre tableau.

```
idf.py flash
```

6. Surveillez la sortie du port série de votre carte à l'aide de la commande suivante.

```
idf.py monitor
```

> **Note**
>
> Vous pouvez combiner ces commandes comme dans l'exemple suivant.
>
> ```
> idf.py erase_flash flash monitor
> ```

Pour certaines configurations de machine hôte, vous devez spécifier le port lorsque vous flashez la carte, comme dans l'exemple suivant.

```
idf.py erase_flash flash monitor -p /dev/ttyUSB1
```

Créez et flashez des FreeRTOS avec CMake

Outre le `idf.py` script fourni par le SDK IDF pour créer et exécuter votre code, vous pouvez également créer le projet avec CMake. Actuellement, il prend en charge les Makefiles Unix ou le système de construction Ninja.

Pour créer et flasher le projet

1. Dans une fenêtre de ligne de commande, naviguez jusqu'à la racine du répertoire de téléchargement de FreeRTOS.
2. Exécutez le script suivant pour ajouter les ESP-IDF outils au PATH de votre shell.

   Windows

   ```
   vendors\espressif\esp-idf\export.bat
   ```

   Linux/ macOS

   ```
   source vendors/espressif/esp-idf/export.sh
   ```
3. Entrez la commande suivante pour générer les fichiers de compilation.

Avec 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
```

Avec 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. Générez le projet.

   Avec Unix Makefiles

   ```
   make -C ./YOUR_BUILD_DIRECTORY -j8
   ```

   Avec Ninja

   ```
   ninja -C ./YOUR_BUILD_DIRECTORY -j8
   ```

5. Effacez le flash, puis faites clignoter le tableau.

   Avec Unix Makefiles

   ```
   make -C ./YOUR_BUILD_DIRECTORY erase_flash
   ```

   ```
   make -C ./YOUR_BUILD_DIRECTORY flash
   ```

   Avec Ninja

   ```
   ninja -C ./YOUR_BUILD_DIRECTORY erase_flash
   ```

   ```
   ninja -C ./YOUR_BUILD_DIRECTORY flash
   ```

Exécuter les démonstrations Bluetooth Low Energy

FreeRTOS prend en charge la connectivitéBibliothèque Bluetooth Low Energy.

Pour exécuter le projet de démonstration FreeRTOS via Bluetooth Low Energy, vous devez exécuter l'application de démonstration du SDK mobile FreeRTOS Bluetooth Low Energy sur un appareil mobile iOS ou Android.

Pour configurer l'application de démonstration du SDK mobile FreeRTOS Bluetooth Low Energy

1. Suivez les instructions de SDK mobiles pour appareils Bluetooth FreeRTOS pour télécharger et installer le kit SDK pour votre plateforme mobile sur votre ordinateur hôte.
2. Suivez les instructions de la rubrique Application de démonstration du SDK FreeRTOS Bluetooth Low Energy Mobile pour configurer la démonstration d'applications mobiles sur votre appareil mobile.

Pour savoir comment exécuter la démo MQTT via Bluetooth Low Energy sur votre carte mère, consultezMQTT sur Bluetooth Low Energy.

Pour obtenir des instructions sur la façon d'exécuter la démonstration de Wi-Fi provisionnement sur votre tableau, consultezWi-Fi approvisionnement.

Utilisation de FreeRTOS dans votre propre projet CMake pour ESP32

Si vous souhaitez utiliser FreeRTOS dans votre propre projet CMake, vous pouvez le configurer en tant que sous-répertoire et le créer avec votre application. Tout d'abord, procurez-vous une copie de FreeRTOS auprès de. GitHub Vous pouvez également le configurer en tant que sous-module Git à l'aide de la commande suivante afin de faciliter sa mise à jour à l'avenir.

```
git submodule add -b release https://github.com/aws/amazon-freertos.git freertos
```

Si une version ultérieure est publiée, vous pouvez mettre à jour votre copie locale à l'aide de ces commandes.

```
# 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"
```

Si la structure de répertoire de votre projet est la suivante :

```
- freertos (the copy that you obtained from GitHub or the AWS IoT console)
- src
  - main.c (your application code)
- CMakeLists.txt
```

Voici un exemple de `CMakeLists.txt` fichier de premier niveau qui peut être utilisé pour créer votre application avec FreeRTOS.

```
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)
```

Pour créer le projet, exécutez les commandes CMake suivantes. Assurez-vous que le compilateur ESP32 se trouve dans la variable d'environnement PATH.

```
cmake -S . -B build-directory -DCMAKE_TOOLCHAIN_FILE=freertos/tools/cmake/toolchains/
xtensa-esp32.cmake -GNinja
```

```
cmake --build build-directory
```

Pour flasher l'application sur votre carte, exécutez la commande suivante.

```
cmake --build build-directory --target flash
```

Utilisation de composants de FreeRTOS

Après l'exécution de CMake, vous trouverez tous les composants disponibles dans la sortie récapitulative. Cela devrait ressembler à l'exemple suivant.

```
====================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:
=============================================================================
```

Vous pouvez référencer n'importe quel composant de la `Modules to build` liste. Pour les lier à votre application, placez l'`AFR::`espace de noms devant le nom, par exemple, `AFR::core_mqtt``AFR::ota`, et ainsi de suite.

Ajoutez des composants personnalisés à l'aide de ESP-IDF

Vous pouvez ajouter d'autres composants lors de l'utilisation ESP-IDF. Par exemple, en supposant que vous souhaitez ajouter un composant appelé `example_component` et que votre projet ressemble à ceci :

```
- freertos
- components
  - example_component
    - include
      - example_component.h
    - src
```

```
      - example_component.c
    - CMakeLists.txt
- src
  - main.c
- CMakeLists.txt
```

Voici un exemple de `CMakeLists.txt` fichier pour votre composant.

```
add_library(example_component src/example_component.c)
target_include_directories(example_component PUBLIC include)
```

Ensuite, dans le `CMakeLists.txt` fichier de premier niveau, ajoutez le composant en insérant la ligne suivante juste après`add_subdirectory(freertos)`.

```
add_subdirectory(component/example_component)
```

Modifiez ensuite `target_link_libraries` pour inclure votre composant.

```
target_link_libraries(my_app PRIVATE AFR::core_mqtt PRIVATE example_component)
```

Ce composant est désormais automatiquement lié à votre code d'application par défaut. Vous pouvez désormais inclure ses fichiers d'en-tête et appeler les fonctions qu'il définit.

Remplacer les configurations pour FreeRTOS

Il n'existe actuellement aucune approche bien définie pour redéfinir les configurations en dehors de l'arborescence des sources FreeRTOS. Par défaut, CMake recherche les répertoires *`freertos`*`/vendors/espressif/boards/esp32/aws_demos/config_files/` et *`freertos`*`/demos/include/`. Cependant, vous pouvez utiliser une solution de contournement pour demander au compilateur de rechercher d'autres répertoires en premier. Par exemple, vous pouvez ajouter un autre dossier pour les configurations FreeRTOS.

```
- freertos
- freertos-configs
  - aws_clientcredential.h
  - aws_clientcredential_keys.h
  - iot_mqtt_agent_config.h
  - iot_config.h
```

```
- components
- src
- CMakeLists.txt
```

Les fichiers sous `freertos-configs` sont copiés à partir des répertoires *`freertos`*`/vendors/espressif/boards/esp32/aws_demos/config_files/` et *`freertos`*`/demos/include/`. Ensuite, dans votre `CMakeLists.txt` fichier de premier niveau, ajoutez cette ligne avant `add_subdirectory(freertos)` afin que le compilateur recherche d'abord ce répertoire.

```
include_directories(BEFORE freertos-configs)
```

Fournir votre propre sdkconfig pour ESP-IDF

Dans le cas où vous voulez fournir votre propre fichier `sdkconfig.default`, vous pouvez définir la variable CMake `IDF_SDKCONFIG_DEFAULTS` à partir de la ligne de commande :

```
cmake -S . -B build-directory -DIDF_SDKCONFIG_DEFAULTS=path_to_your_sdkconfig_defaults
 -DCMAKE_TOOLCHAIN_FILE=freertos/tools/cmake/toolchains/xtensa-esp32.cmake -GNinja
```

Si vous ne spécifiez pas d'emplacement pour votre propre `sdkconfig.default` fichier, FreeRTOS utilise le fichier par défaut situé dans. *`freertos`*`/vendors/espressif/boards/esp32/aws_demos/sdkconfig.defaults`

Pour plus d'informations, consultez la section Configuration du projet dans la référence de l'API Espressif et, si vous rencontrez des problèmes après avoir correctement compilé, consultez la section sur les options obsolètes et leurs remplacements sur cette page.

Résumé

Si vous gérez un projet incluant un composant appelé `example_component` et que vous voulez remplacer certaines configurations, voici un exemple complet du fichier `CMakeLists.txt` de niveau supérieur.

```
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)
```

Résolution des problèmes

- Si vous utilisez macOS et que le système d'exploitation ne vous reconnaît pas ESP-WROVER-KIT, assurez-vous que les pilotes D2XX ne sont pas installés. Pour les désinstaller, suivez les instructions fournies dans le manuel FTDI Drivers Installation Guide for macOS X.
- L'utilitaire de surveillance fourni par ESP-IDF (et invoqué à l'aide de make monitor) vous aide à décoder les adresses. Pour cette raison, cela peut vous aider à obtenir des traces significatives au cas où l'application cesserait de fonctionner. Pour plus d'informations, consultez la section Décodage automatique des adresses sur le site Web d'Espressif.
- Il est également possible d'activer GDBStub pour communiquer avec gdb sans nécessiter de matériel JTAG spécial. Pour plus d'informations, consultez Lancer GDB avec GDBStub sur le site Web d'Espressif.
- Pour plus d'informations sur la configuration d'un OpenOCD-based environnement si le débogage matériel JTAG est requis, consultez la section Débogage JTAG sur le site Web d'Espressif.
- S'il ne `pyserial` peut pas être installé `pip` sur macOS, téléchargez-le sur le site Web de pyserial.
- Si la carte se réinitialise continuellement, essayez d'effacer le flash en saisissant la commande suivante sur le terminal.

```
make erase_flash
```

- Si vous voyez des erreurs lorsque vous exécutez `idf_monitor.py`, utilisez Python 2.7.
- Les bibliothèques requises de ESP-IDF sont incluses dans FreeRTOS, il n'est donc pas nécessaire de les télécharger en externe. Si la variable d'`IDF_PATH`environnement est définie, nous vous recommandons de l'effacer avant de créer FreeRTOS.
- Sur Windows, la génération du projet peut prendre entre 3 et 4 minutes. Pour réduire le temps de construction, vous pouvez utiliser le `-j4` commutateur de la commande make.

```
make flash monitor -j4
```

- Si votre appareil ne parvient pas à se connecter AWS IoT, ouvrez le `aws_clientcredential.h` fichier et vérifiez que les variables de configuration sont correctement définies dans le fichier. `clientcredentialMQTT_BROKER_ENDPOINT[]`devrait ressembler à`1234567890123-ats.iot.us-east-1.amazonaws.com`.
- Si vous suivez les étapes décrites dans Utilisation de FreeRTOS dans votre propre projet CMake pour ESP32 et que vous constatez des erreurs de référence non définies dans l'éditeur de liens, c'est généralement en raison de l'absence de bibliothèques ou de démos dépendantes. Pour les ajouter, mettez à jour le fichier `CMakeLists.txt` (sous le répertoire racine) en utilisant la fonction CMake standard `target_link_libraries`.
- ESP-IDF La version 4.2 prend en charge l'utilisation du xtensa \ -esp32 \ -elf \ -gcc 8 \ .2 \ .0 \. chaîne d'outils. Si vous utilisez une version antérieure de la chaîne d'outils Xtensa, téléchargez la version requise.
- Si vous voyez un journal d'erreurs comme le suivant concernant les dépendances de python qui ne sont pas respectées pour la ESP-IDF version 4.2 :

```
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
```

Installez les dépendances Python sur votre plateforme à l'aide de la commande Python suivante :

```
root/vendors/espressif/esp-idf/requirements.txt
```

Pour plus d'informations sur la résolution des problèmes, consultezRésolution des problèmes de mise en route.

Débogage

Code de débogage sur Espressif ESP32-DevKitC et ESP-WROVER-KIT (v4.2) ESP-IDF

Cette section explique comment déboguer le matériel Espressif à l'aide de la version 4.2. ESP-IDF Vous avez besoin d'un câble JTAG vers USB. Nous utilisons un câble USB vers MPSSE (par exemple, le FTDI C232HM-DDHSL-0).

ESP-DevKitC Configuration du JTAG

Pour le C232HM-DDHSL-0 câble FTDI, il s'agit des connexions au 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             |
```

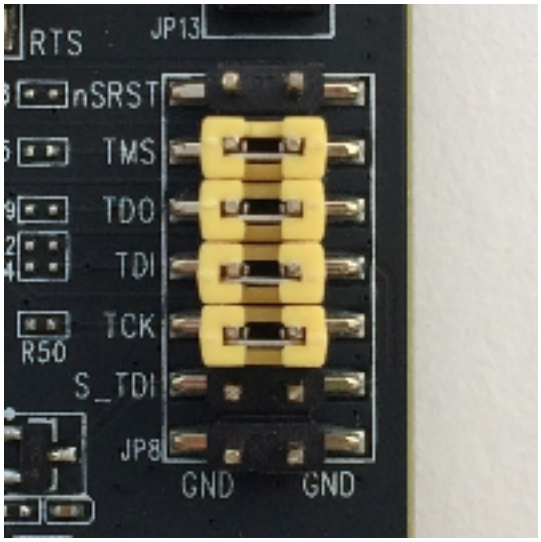
ESP-WROVER-KIT Configuration du JTAG

Pour le C232HM-DDHSL-0 câble FTDI, il s'agit des connexions au ESP32-WROVER-KIT.

```
| 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             |
```

Ces tableaux ont été développés à partir de la C232HM-DDHSL-0 fiche technique du FTDI. Pour plus d'informations, consultez la section « Connexion du câble C232HM MPSSE et détails mécaniques » de la fiche technique.

Pour activer le JTAG sur le ESP-WROVER-KIT, placez des jumpers sur les broches TMS, TDO, TDI, TCK et S_TDI, comme indiqué ici.



Débogage sous Windows (ESP-IDF v4.2)

Pour configurer le débogage sous Windows

1. Connectez le côté USB du FTDI C232HM-DDHSL-0 à votre ordinateur et l'autre côté comme décrit dansCode de débogage sur Espressif ESP32-DevKitC et ESP-WROVER-KIT (v4.2) ESP-IDF. Le C232HM-DDHSL-0 périphérique FTDI doit apparaître dans le Gestionnaire de périphériques sous Contrôleurs de bus série universels.
2. Dans la liste des périphériques de bus série universels, cliquez avec le bouton droit sur le C232HM-DDHSL-0périphérique, puis sélectionnez Propriétés.

> **Note**
>
> L'appareil peut être répertorié Port série USB.

Pour voir les propriétés de l'appareil, dans la fenêtre des propriétés, cliquez sur l'onglet Détails. Si le périphérique n'est pas répertorié, installez le pilote Windows pour FTDI C232HM-DDHSL-0.

3. Sous l'onglet Détails , choisissez Propriété, puis ID du matériel. Vous devriez voir quelque chose comme ça dans le champ Valeur.

   ```
   FTDIBUS\COMPORT&VID_0403&PID_6014
   ```

   Dans cet exemple, l'ID du fournisseur est 0403 et celui du produit est 6014.

   Vérifiez que ces ID correspondent à ceux de `projects/espressif/esp32/make/aws_demos/esp32_devkitj_v1.cfg`. Les identifiants sont spécifiés sur une ligne commençant par `ftdi_vid_pid` un identifiant de fournisseur et un identifiant de produit.

   ```
   ftdi_vid_pid 0x0403 0x6014
   ```

4. Téléchargez OpenOCD for Windows.
5. Décompressez le fichier sur `C:\` et ajoutez `C:\openocd-esp32\bin` à votre chemin système.
6. OpenOCD nécessite libusb, qui n'est pas installé par défaut sur Windows. Pour installer libusb :

   a. Téléchargez zadig.exe.

   b. Exécutez `zadig.exe`. Dans le menu Options, choisissez List All Devices (Afficher tous les périphériques).

   c. Dans le menu déroulant, choisissez C232HM-DDHSL-0.

   d. Dans le champ du pilote cible, à droite de la flèche verte, choisissez WinUSB.

   e. Pour la liste située sous le champ du pilote cible, cliquez sur la flèche, puis choisissez Installer le pilote. Choisissez Replace Driver (Remplacer le pilote).

7. Ouvrez une invite de commande, naviguez jusqu'à la racine de votre répertoire de téléchargement de FreeRTOS et exécutez la commande suivante.

   ```
   idf.py openocd
   ```

   Laissez l'invite de commande ouverte.

8. Ouvrez une nouvelle invite de commande, naviguez jusqu'à la racine de votre répertoire de téléchargement de FreeRTOS et exécutez

   ```
   idf.py flash monitor
   ```

9. Ouvrez une autre invite de commande, naviguez jusqu'à la racine de votre répertoire de téléchargement de FreeRTOS et attendez que la démo commence à s'exécuter sur votre forum. Quand c'est le cas, courez

   ```
   idf.py gdb
   ```

   Le programme doit s'arrêter dans la fonction `main`.

   > **Note**
   >
   > L'ESP32 prend en charge un maximum de deux points d'arrêt.

Débogage sur macOS (ESP-IDF v4.2)

1. Téléchargez le pilote FTDI pour macOS.
2. Téléchargez OpenOCD.
3. Extrayez le fichier .tar téléchargé et définissez le chemin d'accès dans `.bash_profile` sur `OCD_INSTALL_DIR/openocd-esp32/bin`.
4. Utilisez la commande suivante pour effectuer l'installation `libusb` sur macOS.

   ```
   brew install libusb
   ```

5. Utilisez la commande suivante pour décharger le pilote du port série.

   ```
   sudo kextunload -b com.FTDI.driver.FTDIUSBSerialDriver
   ```

6. Utilisez la commande suivante pour décharger le pilote du port série.

   ```
   sudo kextunload -b com.FTDI.driver.FTDIUSBSerialDriver
   ```

7. Si vous utilisez une version de macOS ultérieure à 10.9, utilisez la commande suivante pour décharger le pilote Apple FTDI.

   ```
   sudo kextunload -b com.apple.driver.AppleUSBFTDI
   ```

8. Utilisez la commande suivante pour obtenir l'ID produit et l'ID fournisseur du câble FTDI. Il répertorie les périphériques USB connectés.

```
system_profiler SPUSBDataType
```

Le résultat de `system_profiler` doit ressembler à ce qui suit.

```
   DEVICE:

   Product ID: product-ID
   Vendor ID: vendor-ID (Future Technology Devices International Limited)
```

9. Ouvrez le fichier `projects/espressif/esp32/make/aws_demos/esp32_devkitj_v1.cfg`. Les ID de fournisseur de produit de votre appareil sont spécifiés dans une ligne qui commence par `ftdi_vid_pid`. Modifiez les ID pour qu'ils correspondent à ceux de la sortie `system_profiler` de l'étape précédente.
10. Ouvrez une fenêtre de terminal, naviguez jusqu'à la racine de votre répertoire de téléchargement de FreeRTOS et utilisez la commande suivante pour exécuter OpenOCD.

    ```
    idf.py openocd
    ```

    Laissez cette fenêtre de terminal ouverte.
11. Ouvrez un nouveau terminal et utilisez la commande suivante pour charger le pilote du port série FTDI.

    ```
    sudo kextload -b com.FTDI.driver.FTDIUSBSerialDriver
    ```
12. Naviguez jusqu'à la racine de votre répertoire de téléchargement de FreeRTOS et lancez

    ```
    idf.py flash monitor
    ```
13. Ouvrez un autre nouveau terminal, naviguez jusqu'à la racine de votre répertoire de téléchargement de FreeRTOS et lancez

    ```
    idf.py gdb
    ```

    Le programme doit s'arrêter à `main`.

Débogage sous Linux (ESP-IDF v4.2)

1. Téléchargez OpenOCD. Extrayez le fichier tarball et suivez les instructions d'installation du fichier readme.
2. Utilisez la commande suivante pour installer libusb sous Linux.

```
sudo apt-get install libusb-1.0
```

3. Ouvrez un terminal et saisissez ls -l /dev/ttyUSB*pour afficher tous les périphériques USB reliés à votre ordinateur. Cela vous permet de vérifier si les ports USB de la carte sont reconnus par le système d'exploitation. Vous devriez voir une sortie semblable à la suivante.

```
   $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. Déconnectez-vous, puis connectez-vous et répétez l'alimentation de la carte pour que les modifications prennent effet. Dans une invite de terminal, affichez la liste des périphériques USB. Assurez-vous que le propriétaire du groupe est passé de `dialout` à`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
```

   L'interface `/dev/ttyUSB`n avec le nombre inférieur est utilisée pour la communication JTAG. L'autre interface est routée vers le port série (UART) de l'ESP32 et est utilisée pour télécharger du code vers la mémoire flash de l'ESP32.

5. Dans une fenêtre de terminal, naviguez jusqu'à la racine de votre répertoire de téléchargement de FreeRTOS et utilisez la commande suivante pour exécuter OpenOCD.

```
idf.py openocd
```

6. Ouvrez un autre terminal, naviguez jusqu'à la racine du répertoire de téléchargement de FreeRTOS et exécutez la commande suivante.

```
idf.py flash monitor
```

7. Ouvrez un autre terminal, naviguez à la racine du répertoire de téléchargement de FreeRTOS et exécutez la commande suivante :

   ```
   idf.py gdb
   ```

   Le programme doit s'arrêter à `main()`.

Débuter avec l'Espressif ESP32-WROOM-32SE

> **Important**
>
> Cette intégration de référence est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

> **Note**
>
> - Pour découvrir comment intégrer les bibliothèques modulaires et les démos FreeRTOS dans votre propre projet Espressif IDF, consultez notre article sur l'intégration de référence pour la plateforme. ESP32-C3
> - Actuellement, le port FreeRTOS ESP32-WROOM-32SE pour ne prend pas en charge la fonctionnalité de multitraitement symétrique (SMP).

Ce tutoriel explique comment démarrer avec l'Espressif ESP32-WROOM-32SE. Pour en acheter un auprès de notre partenaire sur le catalogue des appareils AWS partenaires, consultez ESP32-WROOM-32SE.

Présentation de

Ce didacticiel vous guide à travers les étapes suivantes :

1. Connectez votre carte à un appareil hôte.
2. Installation du logiciel sur votre machine hôte pour développer et déboguer les applications intégrées de votre carte de microcontrôleur.
3. Compilez de manière croisée une application de démonstration FreeRTOS en une image binaire.
4. Chargement de l'image binaire de l'application dans votre carte et exécution de l'application.
5. Surveillez et déboguez l'application en cours d'exécution à l'aide d'une connexion série.

Conditions préalables

Avant de commencer à utiliser FreeRTOS sur votre tableau Espressif, vous devez configurer votre compte et vos autorisations. AWS

Inscrivez-vous pour un Compte AWS

Pour commencer AWS, vous avez besoin d'un Compte AWS. Pour plus d'informations sur la création d'un Compte AWS, voir Getting started with an Compte AWS dans le Guide de Gestion de compte AWS référence.

Pour activer l'accès, ajoutez des autorisations à vos utilisateurs, groupes ou rôles :

- Utilisateurs et groupes dans AWS IAM Identity Center :

  Créez un jeu d'autorisations. Suivez les instructions de la rubrique Création d'un jeu d'autorisations du Guide de l'utilisateur AWS IAM Identity Center .
- Utilisateurs gérés dans IAM par un fournisseur d'identité :

  Créez un rôle pour la fédération d'identité. Suivez les instructions de la rubrique Création d'un rôle pour un fournisseur d'identité tiers (fédération) dans le Guide de l'utilisateur IAM.
- Utilisateurs IAM :
  - Créez un rôle que votre utilisateur peut assumer. Suivez les instructions de la rubrique Création d'un rôle pour un utilisateur IAM dans le Guide de l'utilisateur IAM.
  - (Non recommandé) Attachez une politique directement à un utilisateur ou ajoutez un utilisateur à un groupe d'utilisateurs. Suivez les instructions de la rubrique Ajout d'autorisations à un utilisateur (console) du Guide de l'utilisateur IAM.

Mise en route

> **Note**
>
> Les commandes Linux de ce didacticiel nécessitent que vous utilisiez le shell Bash.

1. Configurez le matériel Espressif.

   Pour plus d'informations sur la configuration du matériel de la carte de ESP32-WROOM-32SE développement, consultez le guide ESP32-DevKitC de démarrage de la version 4.

   > **Important**
   >
   > Lorsque vous atteignez la section Installation étape par étape du guide, suivez jusqu'à ce que vous ayez terminé l'étape 4 (Configuration des variables d'environnement). Arrêtez-vous après avoir terminé l'étape 4 et suivez les étapes restantes ici.

2. Téléchargez Amazon GitHubFreeRTOS depuis. (Pour obtenir des instructions, consultez le README.mdfichier.)
3. Configurez votre environnement de développement.

   Pour communiquer avec votre tableau, vous devez installer une chaîne d'outils. Espressif fournit le logiciel ESP-IDF pour développer des logiciels pour leurs cartes. Comme ESP-IDF sa propre version du noyau FreeRTOS est intégrée en tant que composant, Amazon FreeRTOS inclut une version personnalisée de la ESP-IDF version 4.2 dans laquelle le noyau FreeRTOS a été supprimé. Cela résout les problèmes liés aux fichiers dupliqués lors de la compilation. Pour utiliser la version personnalisée de la ESP-IDF version 4.2 incluse dans Amazon FreeRTOS, suivez les instructions ci-dessous correspondant au système d'exploitation de votre machine hôte.

   Windows

   1. Téléchargez ESP-IDF le programme d'installation universel en ligne pour Windows.
   2. Exécutez le programme d'installation en ligne universel.
   3. Lorsque vous arrivez à l'étape Télécharger ou utiliser ESP-IDF, sélectionnez Utiliser un ESP-IDF répertoire existant et définissez Choisir un ESP-IDF répertoire existant sur`freertos/vendors/espressif/esp-idf`.

4. Terminez l'installation.

macOS

1. Suivez les instructions de la configuration standard des prérequis de la chaîne d'outils (ESP-IDF v4.2) pour macOS.

   > **Important**
   >
   > Lorsque vous atteignez les instructions ESP-IDF « Obtenir » sous Prochaines étapes, arrêtez-vous, puis revenez aux instructions de cette page.

2. Ouvrez une fenêtre de ligne de commande.
3. Accédez au répertoire de téléchargement de FreeRTOS, puis exécutez le script suivant pour télécharger et installer la chaîne d'outils espressif pour votre plateforme.

   ```
   vendors/espressif/esp-idf/install.sh
   ```

4. Ajoutez les ESP-IDF outils de la chaîne d'outils au chemin de votre terminal à l'aide de la commande suivante.

   ```
   source vendors/espressif/esp-idf/export.sh
   ```

Linux

1. Suivez les instructions de la configuration standard des prérequis de la chaîne d'outils (ESP-IDF v4.2) pour Linux.

   > **Important**
   >
   > Lorsque vous atteignez les instructions ESP-IDF « Obtenir » sous Prochaines étapes, arrêtez-vous, puis revenez aux instructions de cette page.

2. Ouvrez une fenêtre de ligne de commande.
3. Accédez au répertoire de téléchargement de FreeRTOS, puis exécutez le script suivant pour télécharger et installer la chaîne d'outils Espressif pour votre plateforme.

```
vendors/espressif/esp-idf/install.sh
```

4. Ajoutez les ESP-IDF outils de la chaîne d'outils au chemin de votre terminal à l'aide de la commande suivante.

```
source vendors/espressif/esp-idf/export.sh
```

4. Établissez une connexion série.

   a. Pour établir une connexion série entre votre machine hôte et le ESP32-WROOM-32SE, installez les pilotes VCP CP210x USB to UART Bridge. Vous pouvez télécharger ces pilotes à partir de Silicon Labs.

   b. Suivez les étapes pour établir une connexion série avec ESP32.

   c. Une fois que vous avez établi une connexion série, notez le port série pour la connexion de votre carte. Vous en avez besoin pour flasher la démo.

Configuration des applications de démonstration FreeRTOS

Pour ce didacticiel, le fichier de configuration FreeRTOS se trouve à l'adresse. *freertos*/vendors/espressif/boards/*board-name*/aws_demos/config_files/FreeRTOSConfig.h (Par exemple, si cette option `AFR_BOARD espressif.esp32_devkitc` est sélectionnée, le fichier de configuration se trouve à l'adresse*freertos*/vendors/espressif/boards/esp32/aws_demos/config_files/FreeRTOSConfig.h.)

> **Important**
>
> L'appareil ATECC608A dispose d'une initialisation unique qui est verrouillée sur l'appareil lors de la première exécution d'un projet (lors de l'appel à). `C_InitToken` Cependant, le projet de démonstration FreeRTOS et le projet de test ont des configurations différentes. Si le périphérique est verrouillé pendant les configurations du projet de démonstration, tous les tests du projet de test ne réussiront pas.

1. Configurez le projet de démonstration FreeRTOS en suivant les étapes décrites dans. Configuration des démos de FreeRTOS Lorsque vous arrivez à la dernière étape Pour formater vos AWS IoT informations d'identification, arrêtez-vous et effectuez les étapes suivantes.

2. Microchip fournit plusieurs outils de script pour aider à configurer les composants ATECC608A. Accédez au répertoire *freertos*/vendors/microchip/example_trust_chain_tool et ouvrez le fichier README.md.
3. Pour approvisionner votre appareil, suivez les instructions du README.md fichier. Voici les principales étapes à suivre :

    1. Créez et enregistrez une autorité de certification auprès de AWS.
    2. Générez vos clés sur le composant ATECC608A et exportez la clé publique et le numéro de série du périphérique.
    3. Générez un certificat pour l'appareil et enregistrez-le auprès de AWS.
4. Chargez le certificat d'autorité de certification et le certificat de périphérique sur le périphérique en suivant les instructions de Developer-mode provisionnement des clés.

Surveillance des messages MQTT sur le AWS Cloud

Avant de lancer le projet de démonstration FreeRTOS, vous pouvez configurer le client MQTT dans la console pour surveiller AWS IoT les messages que votre appareil envoie au Cloud. AWS

Pour vous abonner à la rubrique MQTT avec le AWS IoT Client MQTT

1. Connectez-vous à la console AWS IoT.
2. Dans le volet de navigation, choisissez Test, puis MQTT Test Client.
3. Dans Sujet d'abonnement, entrez *your-thing-name*/example/topic puis choisissez S'abonner au sujet.

Créez, flashez et exécutez le projet de démonstration FreeRTOS à l'aide du script idf.py

Vous pouvez utiliser l'utilitaire IDF (idf.py) d'Espressif pour générer les fichiers de compilation, créer le binaire de l'application et flasher les fichiers binaires sur votre appareil.

> **Note**
>
> Certaines configurations peuvent nécessiter que vous utilisiez l'option de port « -p port-name » with idf.py pour spécifier le port correct, comme dans l'exemple suivant.
>
> ```
> idf.py -p /dev/cu.usbserial-00101301B flash
> ```

Créez et flashez des FreeRTOS sous Windows, Linux et macOS (v4.2) ESP-IDF

1. Accédez à la racine de votre répertoire de téléchargement de FreeRTOS.
2. Dans une fenêtre de ligne de commande, entrez la commande suivante pour ajouter les ESP-IDF outils au PATH de votre terminal :

   Windows (application « Command »)

   ```
   vendors\espressif\esp-idf\export.bat
   ```

   Windows (application « ESP-IDF 4.x CMD »)

   (Cela a déjà été fait lorsque vous avez ouvert l'application.)

   Linux/ macOS

   ```
   source vendors/espressif/esp-idf/export.sh
   ```

3. Configurez cmake dans le `build` répertoire et créez l'image du microprogramme à l'aide de la commande suivante.

   ```
   idf.py -DVENDOR=espressif -DBOARD=esp32_ecc608a_devkitc -DCOMPILER=xtensa-esp32
    build
   ```

   Vous devriez voir une sortie comme dans l'exemple suivant.

   ```
     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'
```

S'il n'y a aucune erreur, la compilation générera les fichiers binaires .bin du microprogramme.

4. Effacez la mémoire flash de votre carte de développement à l'aide de la commande suivante.

```
idf.py erase_flash
```

5. Utilisez le `idf.py` script pour flasher le binaire de l'application sur votre tableau.

```
idf.py flash
```

6. Surveillez la sortie du port série de votre carte à l'aide de la commande suivante.

```
idf.py monitor
```

> **Note**
>
> - Vous pouvez combiner ces commandes comme dans l'exemple suivant.
>
>   ```
>   idf.py erase_flash flash monitor
>   ```
>
> - Pour certaines configurations de machine hôte, vous devez spécifier le port lorsque vous flashez la carte, comme dans l'exemple suivant.
>
>   ```
>   idf.py erase_flash flash monitor -p /dev/ttyUSB1
>   ```

Créez et flashez des FreeRTOS avec CMake

Outre l'utilisation du `idf.py` script fourni par le SDK IDF pour créer et exécuter votre code, vous pouvez également créer le projet avec CMake. Actuellement, il supporte Unix Makefile et le système de construction Ninja.

Pour créer et flasher le projet

1. Dans une fenêtre de ligne de commande, naviguez jusqu'à la racine du répertoire de téléchargement de FreeRTOS.

2. Exécutez le script suivant pour ajouter les ESP-IDF outils au PATH de votre shell.

   Windows

   ```
   vendors\espressif\esp-idf\export.bat
   ```

   Linux/ macOS

   ```
   source vendors/espressif/esp-idf/export.sh
   ```

3. Entrez la commande suivante pour générer les fichiers de compilation.

   Avec 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
   ```

   Avec 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. Effacez le flash, puis faites clignoter le tableau.

   Avec Unix Makefiles

   ```
   make -C ./YOUR_BUILD_DIRECTORY erase_flash
   ```

   ```
   make -C ./YOUR_BUILD_DIRECTORY flash
   ```

   Avec Ninja

   ```
   ninja -C ./YOUR_BUILD_DIRECTORY erase_flash
   ```

   ```
   ninja -C ./YOUR_BUILD_DIRECTORY flash
   ```

Informations supplémentaires

Pour plus d'informations sur l'utilisation et le dépannage des cartes Espressif ESP32, consultez les rubriques suivantes :

- Utilisation de FreeRTOS dans votre propre projet CMake pour ESP32
- Résolution des problèmes
- Débogage

Débuter avec l'Espressif ESP32-S2

> **Important**
>
> Cette intégration de référence est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

> **Note**
>
> Pour découvrir comment intégrer les bibliothèques modulaires et les démos FreeRTOS dans votre propre projet Espressif IDF, consultez notre article sur l'intégration de référence pour la plateforme. ESP32-C3

Ce didacticiel explique comment démarrer avec le ESP32-S2 SoC Espressif et les cartes de ESP32-S2-Saola-1développement.

Présentation de

Ce didacticiel vous guide à travers les étapes suivantes :

1. Connectez votre carte à un appareil hôte.
2. Installation du logiciel sur votre machine hôte pour développer et déboguer les applications intégrées de votre carte de microcontrôleur.

3. Cross-compile une application de démonstration FreeRTOS sur une image binaire.
4. Chargement de l'image binaire de l'application dans votre carte et exécution de l'application.
5. Surveillance et débogage de l'application en cours d'exécution à l'aide d'une connexion série.

Conditions préalables

Avant de commencer à utiliser FreeRTOS sur votre tableau Espressif, vous devez configurer votre compte et vos autorisations. AWS

Inscrivez-vous pour un Compte AWS

Pour commencer AWS, vous avez besoin d'un Compte AWS. Pour plus d'informations sur la création d'un Compte AWS, voir Getting started with an Compte AWS dans le Guide de Gestion de compte AWS référence.

Pour activer l'accès, ajoutez des autorisations à vos utilisateurs, groupes ou rôles :

- Utilisateurs et groupes dans AWS IAM Identity Center :

  Créez un jeu d'autorisations. Suivez les instructions de la rubrique Création d'un jeu d'autorisations du Guide de l'utilisateur AWS IAM Identity Center .
- Utilisateurs gérés dans IAM par un fournisseur d'identité :

  Créez un rôle pour la fédération d'identité. Suivez les instructions de la rubrique Création d'un rôle pour un fournisseur d'identité tiers (fédération) dans le Guide de l'utilisateur IAM.
- Utilisateurs IAM :
  - Créez un rôle que votre utilisateur peut assumer. Suivez les instructions de la rubrique Création d'un rôle pour un utilisateur IAM dans le Guide de l'utilisateur IAM.
  - (Non recommandé) Attachez une politique directement à un utilisateur ou ajoutez un utilisateur à un groupe d'utilisateurs. Suivez les instructions de la rubrique Ajout d'autorisations à un utilisateur (console) du Guide de l'utilisateur IAM.

Mise en route

> **Note**
>
> Les commandes Linux de ce didacticiel nécessitent que vous utilisiez le shell Bash.

1. Configurez le matériel Espressif.

   Pour plus d'informations sur la configuration du matériel de la carte de ESP32-S2 développement, consultez le ESP32-S2-Saola-1 Guide de démarrage.

   > **Important**
   >
   > Lorsque vous atteignez la section Get Started des guides Espressif, arrêtez-vous, puis revenez aux instructions de cette page.

2. Téléchargez Amazon GitHubFreeRTOS depuis. (Pour obtenir des instructions, consultez le README.mdfichier.)
3. Configurez votre environnement de développement.

   Pour communiquer avec votre tableau, vous devez installer une chaîne d'outils. Espressif fournit le logiciel ESP-IDF pour développer des logiciels pour leurs cartes. Comme ESP-IDF sa propre version du noyau FreeRTOS est intégrée en tant que composant, Amazon FreeRTOS inclut une version personnalisée de la ESP-IDF version 4.2 dans laquelle le noyau FreeRTOS a été supprimé. Cela résout les problèmes liés aux fichiers dupliqués lors de la compilation. Pour utiliser la version personnalisée de la ESP-IDF version 4.2 incluse dans Amazon FreeRTOS, suivez les instructions ci-dessous correspondant au système d'exploitation de votre machine hôte.

   Windows

   1. Téléchargez ESP-IDF le programme d'installation universel en ligne pour Windows.
   2. Exécutez le programme d'installation en ligne universel.
   3. Lorsque vous arrivez à l'étape Télécharger ou utiliser ESP-IDF, sélectionnez Utiliser un ESP-IDF répertoire existant et définissez Choisir un ESP-IDF répertoire existant sur*freertos*/`vendors/espressif/esp-idf`.
   4. Terminez l'installation.

   macOS

   1. Suivez les instructions de la configuration standard des prérequis de la chaîne d'outils (ESP-IDF v4.2) pour macOS.

> **Important**
>
> Lorsque vous atteignez les instructions ESP-IDF « Obtenir » sous Prochaines étapes, arrêtez-vous, puis revenez aux instructions de cette page.

2. Ouvrez une fenêtre de ligne de commande.
3. Accédez au répertoire de téléchargement de FreeRTOS, puis exécutez le script suivant pour télécharger et installer la chaîne d'outils espressif pour votre plateforme.

```
vendors/espressif/esp-idf/install.sh
```

4. Ajoutez les ESP-IDF outils de la chaîne d'outils au chemin de votre terminal à l'aide de la commande suivante.

```
source vendors/espressif/esp-idf/export.sh
```

Linux

1. Suivez les instructions de la configuration standard des prérequis de la chaîne d'outils (ESP-IDF v4.2) pour Linux.

> **Important**
>
> Lorsque vous atteignez les instructions ESP-IDF « Obtenir » sous Prochaines étapes, arrêtez-vous, puis revenez aux instructions de cette page.

2. Ouvrez une fenêtre de ligne de commande.
3. Accédez au répertoire de téléchargement de FreeRTOS, puis exécutez le script suivant pour télécharger et installer la chaîne d'outils Espressif pour votre plateforme.

```
vendors/espressif/esp-idf/install.sh
```

4. Ajoutez les ESP-IDF outils de la chaîne d'outils au chemin de votre terminal à l'aide de la commande suivante.

```
source vendors/espressif/esp-idf/export.sh
```

4. Établissez une connexion série.

   a. Pour établir une connexion série entre votre machine hôte et le ESP32-DevKitC, installez les pilotes VCP CP210x USB to UART Bridge. Vous pouvez télécharger ces pilotes à partir de Silicon Labs.

   b. Suivez les étapes pour établir une connexion série avec ESP32.

   c. Une fois que vous avez établi une connexion série, notez le port série pour la connexion de votre carte. Vous en avez besoin pour flasher la démo.

Configuration des applications de démonstration FreeRTOS

Pour ce didacticiel, le fichier de configuration FreeRTOS se trouve à l'adresse. *freertos*/vendors/espressif/boards/*board-name*/aws_demos/config_files/FreeRTOSConfig.h (Par exemple, si cette option AFR_BOARD espressif.esp32_devkitc est sélectionnée, le fichier de configuration se trouve à l'adresse*freertos*/vendors/espressif/boards/esp32/aws_demos/config_files/FreeRTOSConfig.h.)

1. Si vous utilisez macOS ou Linux, ouvrez une invite du terminal. Si vous utilisez Windows, ouvrez l'application « ESP-IDF 4.x CMD » (si vous avez inclus cette option lors de l'installation de la ESP-IDF chaîne d'outils), ou l'application « Command Prompt » dans le cas contraire.
2. Pour vérifier que Python3 est installé, exécutez ce qui suit :

   ```
   python --version
   ```

   La version installée s'affiche. Si Python 3.0.1 ou version ultérieure n'est pas installé, vous pouvez l'installer depuis le site Web de Python.
3. Vous avez besoin de l'interface de ligne de AWS commande (CLI) pour exécuter AWS IoT des commandes. Si vous utilisez Windows, utilisez la `easy_install awscli` commande pour installer la AWS CLI dans l'application « Command » ou « ESP-IDF 4.x CMD ».

   Si vous utilisez macOS ou Linux, reportez-vous à la section Installation de la AWS CLI.
4. Exécuter

   ```
   aws configure
   ```

et configurez la AWS CLI avec votre ID de clé d' AWS accès, votre clé d'accès secrète et votre AWS région par défaut. Pour plus d'informations, consultez Configuration de l'interface CLI AWS (français non garanti).

5. Utilisez la commande suivante pour installer le AWS SDK pour Python (boto3) :

   - Sous Windows, dans l'application « Command » ou « ESP-IDF 4.x CMD », exécutez

     ```
     easy_install boto3
     ```

   - Sur macOS ou Linux, exécutez

     ```
     pip install tornado nose --user
     ```

     puis exécutez

     ```
     pip install boto3 --user
     ```

FreeRTOS inclut `SetupAWS.py` le script pour faciliter la configuration de votre carte Espressif à laquelle vous connecter. AWS IoT

Pour exécuter le script de configuration

1. Pour configurer le script, ouvrez *`freertos`*`/tools/aws_config_quick_start/configure.json` et définissez les attributs suivants :

   `afr_source_dir`

   Chemin d'accès complet vers l'annuaire *`freertos`* sur votre ordinateur. Assurez-vous d'utiliser des barres obliques pour spécifier ce chemin.

   `thing_name`

   Le nom que vous souhaitez attribuer à l' AWS IoT élément qui représente votre tableau.

   `wifi_ssid`

   Le SSID de votre Wi-Fi réseau.

`wifi_password`

Le mot de passe de votre Wi-Fi réseau.

`wifi_security`

Type de sécurité de votre Wi-Fi réseau. Les types de sécurité suivants sont valides :

- `eWiFiSecurityOpen` (Ouvrir, aucune sécurité)
- `eWiFiSecurityWEP` (Sécurité WEP)
- `eWiFiSecurityWPA` (Sécurité WPA)
- `eWiFiSecurityWPA2` (Sécurité WPA2)

2. Si vous utilisez macOS ou Linux, ouvrez une invite du terminal. Si vous utilisez Windows, ouvrez l'application « ESP-IDF 4.x CMD » ou « Command ».
3. Accédez au *`freertos`*`/tools/aws_config_quick_start` répertoire et exécutez

```
python SetupAWS.py setup
```

Le script effectue les opérations suivantes :

- Crée un AWS IoT objet, un certificat et une politique.
- Attache la AWS IoT politique au certificat et le certificat à l' AWS IoT objet.
- Remplit le `aws_clientcredential.h` fichier avec votre AWS IoT point de terminaison, votre Wi-Fi SSID et vos informations d'identification.
- Formate votre certificat et votre clé privée et les écrit dans le fichier `aws_clientcredential_keys.h` d'en-tête.

> **Note**
>
> Le certificat est codé en dur à des fins de démonstration uniquement. Production-level les applications doivent stocker ces fichiers dans un emplacement sécurisé.

Pour plus d'informations à ce sujet`SetupAWS.py`, consultez le `README.md` dans le *`freertos`*`/tools/aws_config_quick_start` répertoire.

Surveillance des messages MQTT sur le AWS Cloud

Avant de lancer le projet de démonstration FreeRTOS, vous pouvez configurer le client MQTT dans la console pour surveiller AWS IoT les messages que votre appareil envoie au Cloud. AWS

Pour vous abonner à la rubrique MQTT avec le AWS IoT Client MQTT

1. Connectez-vous à la console AWS IoT.
2. Dans le volet de navigation, choisissez Test, puis MQTT Test Client.
3. Dans le champ Rubrique d'abonnement, saisissez *your-thing-name*`/example/topic`, puis choisissez S'abonner à la rubrique.

Lorsque le projet de démonstration s'exécute avec succès sur votre appareil, le message « Hello World ! » s'affiche envoyé plusieurs fois au sujet auquel vous vous êtes abonné.

Créez, flashez et exécutez le projet de démonstration FreeRTOS à l'aide du script idf.py

Vous pouvez utiliser l'utilitaire IDF d'Espressif pour générer les fichiers de compilation, créer le binaire de l'application et flasher votre tableau.

Créez et flashez des FreeRTOS sous Windows, Linux et macOS (v4.2) ESP-IDF

Utilisez le `idf.py` script pour créer le projet et flasher les fichiers binaires sur votre appareil.

> **Note**
>
> Certaines configurations peuvent nécessiter que vous utilisiez l'option port `-p port-name` with `idf.py` pour spécifier le port correct, comme dans l'exemple suivant.
>
> ```
> idf.py -p /dev/cu.usbserial-00101301B flash
> ```

Pour créer et flasher le projet

1. Accédez à la racine du répertoire de téléchargement de FreeRTOS.
2. Dans une fenêtre de ligne de commande, entrez la commande suivante pour ajouter les ESP-IDF outils au PATH de votre terminal :

Windows (application « Command »)

```
vendors\espressif\esp-idf\export.bat
```

Windows (application « ESP-IDF 4.x CMD »)

(Cela a déjà été fait lorsque vous avez ouvert l'application.)

Linux/ macOS

```
source vendors/espressif/esp-idf/export.sh
```

3. Configurez cmake dans le `build` répertoire et créez l'image du microprogramme à l'aide de la commande suivante.

```
idf.py -DVENDOR=espressif -DBOARD=esp32s2_saola_1 -DCOMPILER=xtensa-esp32s2 build
```

Vous devriez voir une sortie comme dans l'exemple suivant.

```
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'
```

S'il n'y a aucune erreur, la compilation génère les fichiers binaires .bin du microprogramme.

4. Effacez la mémoire flash de votre carte de développement à l'aide de la commande suivante.

```
idf.py erase_flash
```

5. Utilisez le `idf.py` script pour flasher le binaire de l'application sur votre tableau.

```
idf.py flash
```

6. Surveillez la sortie du port série de votre carte à l'aide de la commande suivante.

```
idf.py monitor
```

> **Note**
>
> - Vous pouvez combiner ces commandes comme dans l'exemple suivant.
>
> ```
> idf.py erase_flash flash monitor
> ```
>
> - Pour certaines configurations de machine hôte, vous devez spécifier le port lorsque vous flashez la carte, comme dans l'exemple suivant.
>
> ```
> idf.py erase_flash flash monitor -p /dev/ttyUSB1
> ```

Créez et flashez des FreeRTOS avec CMake

Outre l'utilisation du `idf.py` script fourni par le SDK IDF pour créer et exécuter votre code, vous pouvez également créer le projet avec CMake. Actuellement, il supporte Unix Makefile et le système de construction Ninja.

Pour créer et flasher le projet

1. Dans une fenêtre de ligne de commande, naviguez jusqu'à la racine du répertoire de téléchargement de FreeRTOS.
2. Exécutez le script suivant pour ajouter les ESP-IDF outils au PATH de votre shell.

   - Windows

```
vendors\espressif\esp-idf\export.bat
```

- Linux/ macOS

```
source vendors/espressif/esp-idf/export.sh
```

3. Entrez la commande suivante pour générer les fichiers de compilation.

   - Avec 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
   ```

   - Avec 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. Générez le projet.

   - Avec Unix Makefiles

   ```
   make -C ./YOUR_BUILD_DIRECTORY -j8
   ```

   - Avec Ninja

   ```
   ninja -C ./YOUR_BUILD_DIRECTORY -j8
   ```

5. Effacez le flash, puis faites clignoter le tableau.

   - Avec Unix Makefiles

   ```
   make -C ./YOUR_BUILD_DIRECTORY erase_flash
   ```

   ```
   make -C ./YOUR_BUILD_DIRECTORY flash
   ```

   - Avec Ninja

   ```
   ninja -C ./YOUR_BUILD_DIRECTORY erase_flash
   ```

```
ninja -C ./YOUR_BUILD_DIRECTORY flash
```

Informations supplémentaires

Pour plus d'informations sur l'utilisation et le dépannage des cartes Espressif ESP32, consultez les rubriques suivantes :

- Utilisation de FreeRTOS dans votre propre projet CMake pour ESP32
- Résolution des problèmes
- Débogage

Démarrez avec l'Infineon XMC4800 IoT Connectivity Kit

> **Important**
>
> Cette intégration de référence est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Ce didacticiel fournit des instructions pour la mise en route avec l'Infineon XMC4800 IoT Connectivity Kit. Si vous ne possédez pas le kit de connectivité IoT Infineon XMC4800, consultez AWS le catalogue des appareils partenaires pour en acheter un auprès de notre partenaire.

Si vous souhaitez ouvrir une connexion série avec la carte pour consulter les informations de journalisation et de débogage, vous avez besoin d'un USB/Serial convertisseur 3,3 V, en plus du kit de connectivité IoT XMC4800. Le CP2104 est un USB/Serial convertisseur courant largement disponible sur des cartes telles que le CP2104 Friend d'Adafruit.

Avant de commencer, vous devez configurer AWS IoT et télécharger FreeRTOS pour connecter votre appareil au Cloud. AWS Pour obtenir des instructions, consultez Premiers pas. Dans ce didacticiel, le chemin d'accès au répertoire de téléchargement de FreeRTOS est appelé. *freertos*

Présentation de

Ce didacticiel comprend les instructions de mise en route suivantes :

1. Installation de logiciels sur la machine hôte pour développer et déboguer des applications intégrées pour votre carte de microcontrôleur.
2. Compilation croisée d'une application de démonstration FreeRTOS en une image binaire.
3. Chargement de l'image binaire de l'application dans votre carte et exécution de l'application.
4. Interaction avec l'application s'exécutant sur votre carte via une connexion série, à des fins de surveillance et de débogage.

Configurer votre environnement de développement.

FreeRTOS utilise l'environnement de développement DAVE d'Infineon pour programmer le XMC4800. Avant de commencer, vous devez télécharger et installer DAVE ainsi que certains J-Link pilotes pour communiquer avec le débogueur intégré.

Installer DAVE

1. Accédez à la page de téléchargement du logiciel DAVE d'Infineon.
2. Choisissez le package DAVE pour votre système d'exploitation et soumettez vos informations d'inscription. Après vous être enregistré auprès d'Infineon, vous devez recevoir un e-mail de confirmation avec un lien pour télécharger un fichier .zip.
3. Téléchargez le fichier .zip du package DAVE (`DAVE_version_os_date.zip`) et décompressez-le à l'emplacement où vous souhaitez installer DAVE (par exemple, `C:\DAVE4`).

   > **Note**
   >
   > Certains utilisateurs Windows ont signalé des problèmes dans l'utilisation de l'Explorateur Windows pour décompresser le fichier. Nous vous recommandons d'utiliser un programme tiers tel que 7-Zip.

4. Pour lancer DAVE, exécutez le fichier exécutable trouvé dans le dossier `DAVE_version_os_date.zip` décompressé.

Pour plus d'informations, consultez le document DAVE Quick Start Guide.

Installez les pilotes Segger J-Link

Pour communiquer avec la sonde de débogage intégrée de la carte EtherCAT XMC4800 Relax, vous avez besoin des pilotes inclus dans le pack logiciel et documentation. J-Link Vous pouvez télécharger le pack J-Link logiciel et documentation depuis la page de téléchargement du J-Link logiciel de Segger.

Établir une connexion série

La configuration d'une connexion série est facultative, mais recommandée. Une connexion série permet à votre carte d'envoyer les informations de journalisation et de débogage dans un formulaire que vous pouvez afficher sur votre machine de développement.

L'application de démonstration XMC4800 utilise une connexion série UART sur les broches P0.0 et les broches sont étiquetées sur la P0.1 sérigraphie de la carte XMC4800 Relax EtherCat. Pour configurer une connexion série :

1. Connectez la broche intitulée « RX< P0.0 » à la broche « TX » de votre USB/Serial convertisseur.
2. Connectez la broche intitulée « TX> P0.1 » à la broche « RX » de votre USB/Serial convertisseur.
3. Connectez la broche Ground de votre convertisseur série à l'une des broches étiquetées « GND » sur votre carte. Les appareils doivent partager une position commune.

Comme l'alimentation est fournie à partir du port de débogage USB, ne connectez pas la broche de voltage positif de l'adaptateur série à la carte.

> **Note**
>
> Certains câbles série utilisent un niveau de signalisation 5V. La carte XMC4800 et le module Wi-Fi Click nécessitent une tension de 3,3 V. N'utilisez pas le jumper IOREF de la carte pour modifier les signaux de la carte sur 5 V.

Avec le câble connecté, vous pouvez ouvrir une connexion série sur un émulateur de terminal tel que GNU Screen. La vitesse de transmission est définie sur 115200 par défaut avec des données 8 bits, aucune parité et 1 bit d'arrêt.

Surveillance des messages MQTT dans le cloud

Avant de lancer la démo de FreeRTOS, vous pouvez configurer le client MQTT dans la console pour surveiller AWS IoT les messages que votre appareil envoie au Cloud. AWS

Pour vous abonner à la rubrique MQTT avec le AWS IoT Client MQTT

1. Connectez-vous à la console AWS IoT.
2. Dans le volet de navigation, choisissez Test, puis choisissez MQTT test client pour ouvrir le client MQTT.
3. Dans le champ Rubrique d'abonnement, saisissez ***your-thing-name*/example/topic**, puis choisissez S'abonner à la rubrique.

Lorsque le projet de démonstration s'exécute avec succès sur votre appareil, vous voyez « Hello World ! » envoyé plusieurs fois au sujet auquel vous vous êtes abonné.

Créez et exécutez le projet de démonstration FreeRTOS

Importez la démo de FreeRTOS dans DAVE

1. Démarrez DAVE.
2. Dans DAVE, choisissez File (Fichier), Import (Importer). Dans la fenêtre Import (Importer), développez le dossier Infineon, choisissez DAVE Project, puis Next.
3. Dans la fenêtre Import DAVE Projects, choisissez Select Root Directory, puis Browse, et, enfin, le projet de démonstration XMC4800.

   Dans le répertoire où vous avez décompressé votre téléchargement de FreeRTOS, se trouve le projet de démonstration. `projects/infineon/xmc4800_iotkit/dave4/aws_demos`

   Assurez-vous que la case Copy Projects Into Workspace est décochée.
4. Choisissez Finish (Terminer).

   Le projet `aws_demos` doit être importé dans votre espace de travail et activé.
5. Dans le menu Project (Projet), choisissez Build Active Project (Générer le projet actif).

   Assurez-vous que le projet est généré sans erreur.

Exécutez le projet de démonstration FreeRTOS

1. Utilisez un câble USB pour connecter votre kit XMC4800 IoT Connectivity Kit à votre ordinateur. La carte comporte deux connecteurs microUSB. Utilisez celui étiqueté « X101", où Debug apparaît en regard de celui-ci sur le skillscreen de la carte.
2. Dans le menu Project (Projet), choisissez Rebuild Active Project pour reconstruire `aws_demos` et s'assurer que les modifications de configuration sont collectées.
3. Dans l'Explorateur de projets`aws_demos`, cliquez avec le bouton droit de la souris, choisissez Debug As, puis choisissez DAVE C/C ++ Application.
4. Double-click Débogage GDB SEGGER pour créer une confirmation de J-Link débogage. Choisissez Debug.
5. Lorsque le débogueur s'arrête au point d'arrêt dans `main()`, dans le menu Run (Exécuter), choisissez Resume (Reprendre).

Dans la AWS IoT console, le client MQTT des étapes 4 et 5 doit afficher les messages MQTT envoyés par votre appareil. Si vous utilisez la connexion série, vous voyez quelque chose comme ce qui suit sur la sortie UART :

```
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----
```

Créez la démo de FreeRTOS avec CMake

Si vous préférez ne pas utiliser d'IDE pour le développement de FreeRTOS, vous pouvez également utiliser CMake pour créer et exécuter les applications de démonstration ou les applications que vous avez développées à l'aide d'éditeurs de code et d'outils de débogage tiers.

> **Note**
>
> Cette section couvre l'utilisation de CMake sous Windows avec MingW comme système de génération natif. Pour de plus amples informations sur l'utilisation de CMake avec d'autres systèmes d'exploitation et options, veuillez consulter Utilisation de CMake avec FreeRTOS. (MinGW est un environnement de développement minimaliste pour les applications Microsoft Windows natives.)

Pour créer la démo de FreeRTOS avec CMake

1. Configurez la chaîne d'outils GNU Arm Embedded.

   a. Téléchargez une version Windows de la chaîne d'outils à partir de la page de téléchargement de la chaîne d'outils Arm Embedded.

> **Note**
>
> Nous vous recommandons de télécharger une version autre que « 8-2018-q4-majeure », en raison d'un bogue signalé avec l'utilitaire « objcopy » dans cette version.

b. Ouvrez le programme d'installation de la chaîne d'outils téléchargé et suivez les instructions de l'assistant d'installation pour installer la chaîne d'outils.

> **Important**
>
> Sur la dernière page de l'assistant d'installation, sélectionnez Ajouter un chemin d'accès à la variable d'environnement pour ajouter le chemin d'accès de la chaîne d'outils à la variable d'environnement du chemin d'accès système.

2. Installez CMake et MingW.

   Pour de plus amples informations, veuillez consulter Prérequis CMake.

3. Créez un dossier contenant les fichiers de compilation générés (*build-folder*).

4. Remplacez les répertoires par votre répertoire de téléchargement FreeRTOS *freertos* () et utilisez la commande suivante pour générer les fichiers de compilation :

```
cmake -DVENDOR=infineon -DBOARD=xmc4800_iotkit -DCOMPILER=arm-gcc -S . -B build-folder -G "MinGW Makefiles" -DAFR_ENABLE_TESTS=0
```

5. Remplacez les répertoires par le répertoire build (*build-folder*) et utilisez la commande suivante pour créer le binaire :

```
cmake --build . --parallel 8
```

   Cette commande crée la sortie binaire `aws_demos.hex` dans le répertoire de génération.

6. Flashez et exécutez l'image avec JLINK.

   a. Dans le répertoire de construction (*build-folder*), utilisez les commandes suivantes pour créer un script flash :

```
echo loadfile aws_demos.hex > flash.jlink
```

```
echo r >> flash.jlink
```

```
echo g >> flash.jlink
```

```
echo q >> flash.jlink
```

b. Flashez l'image à l'aide de l'exécutable JLNIK.

```
JLINK_PATH\JLink.exe  -device XMC4800-2048 -if SWD -speed auto -CommanderScript
 flash.jlink
```

Les journaux d'application doivent être visibles via la connexion en série que vous avez créée avec la carte.

Résolution des problèmes

Si ce n'est pas déjà fait, assurez-vous de configurer AWS IoT et de télécharger FreeRTOS pour connecter votre appareil au Cloud. AWS Pour obtenir des instructions, consultez Premiers pas.

Pour obtenir des informations générales sur la résolution des problèmes liés à la prise en main de FreeRTOS, consultez. Résolution des problèmes de mise en route

Démarrez avec le kit XMC4800 IoT Connectivity Kit et Infineon OPTIGA Trust X

> **Important**
>
> Cette intégration de référence est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Ce didacticiel fournit des instructions pour la mise en route avec l'élément sécurisé OPTIGA Trust X Secure Element et le kit de connectivité IoT XMC4800 Connectivity Kit. Par rapport au didacticiel Démarrez avec l'Infineon XMC4800 IoT Connectivity Kit, ce guide vous montre comment fournir des informations d'identification sécurisées à l'aide d'un élément sécurisé Infineon OPTIGA Trust X.

Vous aurez besoin du matériel suivant :

1. Host MCU - Kit de connectivité IoT Infineon XMC4800, consultez AWS le catalogue des appareils partenaires pour en acheter un auprès de notre partenaire.
2. Pack d'extension de sécurité :
   - Élément sécurisé - Infineon OPTIGA Trust X.

     Consultez le catalogue des appareils AWS partenaires pour les acheter auprès de notre partenaire.
   - Carte de personnalisation - Carte de personnalisation Infineon OPTIGA.
   - Carte d'adaptation - Adaptateur Infineon Mylo T.

Pour suivre ces étapes, vous devez ouvrir une connexion série avec la carte pour afficher les informations de journalisation et de débogage. (L'une des étapes vous oblige à copier une clé publique à partir de la sortie de débogage série de la carte et à la coller dans un fichier.) Pour ce faire, vous avez besoin d'un USB/Serial convertisseur 3,3 V en plus du kit de connectivité IoT XMC4800. Le EL-PN-47310126 USB/Serial convertisseur JBtek est connu pour fonctionner pour cette démo. Vous avez également besoin de trois fils de liaison mâle-mâle (pour la réception (RX), la transmission (TX) et la mise à la terre (GND)) pour connecter le câble série à la carte adaptateur Infineon T. Mylo

Avant de commencer, vous devez configurer AWS IoT et télécharger FreeRTOS pour connecter votre appareil au Cloud. AWS Pour obtenir des instructions, veuillez consulter Option 2 : Génération de clés privées embarquées. Dans ce didacticiel, le chemin d'accès au répertoire de téléchargement de FreeRTOS est appelé. *freertos*

Présentation de

Ce didacticiel contient les étapes suivantes :

1. Installation du logiciel sur la machine hôte pour développer et déboguer les applications intégrées de votre carte de microcontrôleur.
2. Cross-compile une application de démonstration FreeRTOS sur une image binaire.
3. Chargement de l'image binaire de l'application dans votre carte et exécution de l'application.
4. À des fins de surveillance et de débogage, l'interaction avec l'application s'exécutant sur votre carte via une connexion série.

Configurer votre environnement de développement.

FreeRTOS utilise l'environnement de développement DAVE d'Infineon pour programmer le XMC4800. Avant de commencer, téléchargez et installez DAVE ainsi que certains J-Link pilotes pour communiquer avec le débogueur intégré.

Installer DAVE

1. Accédez à la page de téléchargement du logiciel DAVE d'Infineon.
2. Choisissez le package DAVE pour votre système d'exploitation et soumettez vos informations d'inscription. Après vous être enregistré, vous devez recevoir un e-mail de confirmation avec un lien pour télécharger un fichier .zip.
3. Téléchargez le fichier .zip du package DAVE (`DAVE_version_os_date.zip`) et décompressez-le à l'emplacement où vous souhaitez installer DAVE (par exemple, `C:\DAVE4`).

   > **Note**
   >
   > Certains utilisateurs Windows ont signalé des problèmes dans l'utilisation de l'Explorateur Windows pour décompresser le fichier. Nous vous recommandons d'utiliser un programme tiers tel que 7-Zip.

4. Pour lancer DAVE, exécutez le fichier exécutable trouvé dans le dossier `DAVE_version_os_date.zip` décompressé.

Pour plus d'informations, consultez le document DAVE Quick Start Guide.

Installez les pilotes Segger J-Link

Pour communiquer avec la sonde de débogage intégrée au kit de connectivité IoT XMC4800, vous avez besoin des pilotes inclus dans J-Link le pack logiciel et documentation. Vous pouvez télécharger le pack J-Link logiciel et documentation depuis la page de téléchargement du J-Link logiciel de Segger.

Établir une connexion série

Connectez le câble du USB/Serial convertisseur à l'adaptateur Infineon Shield2Go. Cela permet à votre carte d'envoyer les informations de journalisation et de débogage dans un formulaire que vous pouvez afficher sur votre machine de développement. Pour configurer une connexion série :

1. Connectez la broche RX à la broche TX de votre USB/Serial convertisseur.

2. Connectez la broche TX à la broche RX de votre USB/Serial convertisseur.
3. Connectez la broche Ground de votre convertisseur série à l'une des broches GND sur votre carte. Les appareils doivent partager une position commune.

Comme l'alimentation est fournie à partir du port de débogage USB, ne connectez pas la broche de voltage positif de l'adaptateur série à la carte.

> **Note**
>
> Certains câbles série utilisent un niveau de signalisation 5V. La carte XMC4800 et le module Wi-Fi Click nécessitent une tension de 3,3 V. N'utilisez pas le jumper IOREF de la carte pour modifier les signaux de la carte sur 5 V.

Avec le câble connecté, vous pouvez ouvrir une connexion série sur un émulateur de terminal tel que GNU Screen. La vitesse de transmission est définie sur 115200 par défaut avec des données 8 bits, aucune parité et 1 bit d'arrêt.

Surveillance des messages MQTT dans le cloud

Avant de lancer le projet de démonstration FreeRTOS, vous pouvez configurer le client MQTT dans la console pour surveiller AWS IoT les messages que votre appareil envoie au Cloud. AWS

Pour vous abonner à la rubrique MQTT avec AWS IoT Client MQTT

1. Connectez-vous à la console AWS IoT.
2. Dans le volet de navigation, choisissez Test, puis choisissez MQTT test client pour ouvrir le client MQTT.
3. Dans le champ Rubrique d'abonnement, saisissez ***your-thing-name*****/example/topic**, puis choisissez S'abonner à la rubrique.

Lorsque le projet de démonstration s'exécute avec succès sur votre appareil, le message « Hello World ! » s'affiche envoyé plusieurs fois au sujet auquel vous vous êtes abonné.

Créez et exécutez le projet de démonstration FreeRTOS

Importez la démo de FreeRTOS dans DAVE

1. Démarrez DAVE.

2. Dans DAVE, choisissez File (Fichier), puis Import (Importer). Développez le dossier Infineon, choisissez DAVE Project (Projet DAVE), puis choisissez Next (Suivant).
3. Dans la fenêtre Import DAVE Projects, choisissez Select Root Directory, puis Browse, et, enfin, le projet de démonstration XMC4800.

   Dans le répertoire où vous avez décompressé votre téléchargement de FreeRTOS, se trouve le projet de démonstration. `projects/infineon/xmc4800_plus_optiga_trust_x/dave4/aws_demos/dave4`

   Assurez-vous que la case Copy Projects Into Workspace (Copier les projets dans l'espace de travail) est décochée.
4. Choisissez Finish (Terminer).

   Le projet `aws_demos` doit être importé dans votre espace de travail et activé.
5. Dans le menu Project (Projet), choisissez Build Active Project (Générer le projet actif).

   Assurez-vous que le projet est généré sans erreur.

Exécutez le projet de démonstration FreeRTOS

1. Dans le menu Project (Projet), choisissez Rebuild Active Project (Reconstruire le projet actif) pour reconstruire `aws_demos` et confirmer que les modifications de configuration sont collectées.
2. Dans l'Explorateur de projets`aws_demos`, cliquez avec le bouton droit de la souris, choisissez Debug As, puis choisissez DAVE C/C ++ Application.
3. Double-click Débogage GDB SEGGER pour créer une confirmation de J-Link débogage. Choisissez Debug.
4. Lorsque le débogueur s'arrête au point d'arrêt dans `main()`, dans le menu Run (Exécuter), choisissez Resume (Reprendre).

À ce stade, passez à l'étape d'extraction de clé publique dans Option 2 : Génération de clés privées embarquées. Une fois toutes les étapes terminées, accédez à la AWS IoT console. Le client MQTT que vous avez configuré précédemment doit afficher les messages MQTT envoyés par votre appareil. Via la connexion série de l'appareil, vous devriez voir quelque chose comme ce qui suit dans la sortie UART :

```
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----
```

Créez la démo de FreeRTOS avec CMake

Cette section couvre l'utilisation de CMake sous Windows avec MingW comme système de génération natif. Pour de plus amples informations sur l'utilisation de CMake avec d'autres systèmes

d'exploitation et options, veuillez consulter Utilisation de CMake avec FreeRTOS. (MinGW est un environnement de développement minimaliste pour les applications Microsoft Windows natives.)

Si vous préférez ne pas utiliser d'IDE pour le développement de FreeRTOS, vous pouvez utiliser CMake pour créer et exécuter les applications de démonstration ou les applications que vous avez développées à l'aide d'éditeurs de code et d'outils de débogage tiers.

Pour créer la démo de FreeRTOS avec CMake

1. Configurez la chaîne d'outils GNU Arm Embedded.

   a. Téléchargez une version Windows de la chaîne d'outils à partir de la page de téléchargement de la chaîne d'outils Arm Embedded.

   > **Note**
   >
   > En raison d'un bogue signalé dans l'utilitaire « objcopy », nous vous recommandons de télécharger une version autre que la version « 8-2018-q4-major ».

   b. Ouvrez le programme d'installation de la chaîne d'outils téléchargé et suivez les instructions de l'assistant.

   c. Sur la dernière page de l'assistant d'installation, sélectionnez Ajouter un chemin d'accès à la variable d'environnement pour ajouter le chemin d'accès de la chaîne d'outils à la variable d'environnement du chemin d'accès système.

2. Installez CMake et MingW.

   Pour de plus amples informations, veuillez consulter Prérequis CMake.

3. Créez un dossier contenant les fichiers de compilation générés (*`build-folder`*).

4. Remplacez les répertoires par votre répertoire de téléchargement FreeRTOS *`freertos`* () et utilisez la commande suivante pour générer les fichiers de compilation :

   ```
   cmake -DVENDOR=infineon -DBOARD=xmc4800_plus_optiga_trust_x -DCOMPILER=arm-gcc -S .
    -B build-folder -G "MinGW Makefiles" -DAFR_ENABLE_TESTS=0
   ```

5. Remplacez les répertoires par le répertoire build (*`build-folder`*) et utilisez la commande suivante pour créer le binaire :

   ```
   cmake --build . --parallel 8
   ```

Cette commande crée la sortie binaire `aws_demos.hex` dans le répertoire de génération.

6. Flashez et exécutez l'image avec JLINK.

   a. Dans le répertoire de construction (*`build-folder`*), utilisez les commandes suivantes pour créer un script flash :

   ```
   echo loadfile aws_demos.hex > flash.jlink
   echo r >> flash.jlink
   echo g >> flash.jlink
   echo q >> flash.jlink
   ```

   b. Flashez l'image à l'aide de l'exécutable JLNIK.

   ```
   JLINK_PATH\JLink.exe  -device XMC4800-2048 -if SWD -speed auto -CommanderScript
    flash.jlink
   ```

   Les journaux d'application doivent être visibles via la connexion en série que vous avez créée avec la carte. Passez à l'étape d'extraction de clé publique Option 2 : Génération de clés privées embarquées. Une fois toutes les étapes terminées, accédez à la AWS IoT console. Le client MQTT que vous avez configuré précédemment doit afficher les messages MQTT envoyés par votre appareil.

Résolution des problèmes

Pour plus d'informations sur le dépannage, consultez Résolution des problèmes de mise en route.

Commencer à utiliser le MW32x AWS IoT Kit de démarrage

> **Important**
>
> Cette intégration de référence est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Le AWS IoT Starter Kit est un kit de développement basé sur le 88MW320/88MW322 dernier microcontrôleur Cortex M4 intégré de NXP, qui intègre le 802. 11b/g/n Wi-Fi sur une seule puce de microcontrôleur. Le kit de développement est certifié FCC. Pour plus d'informations, consultez le catalogue des appareils AWS partenaires pour en acheter un auprès de notre partenaire. Les 88MW320/88MW322 modules sont également certifiés FCC et disponibles pour la personnalisation et la vente en volume.

Ce guide de démarrage explique comment compiler votre application avec le SDK sur un ordinateur hôte, puis charger le fichier binaire généré sur la carte à l'aide des outils fournis avec le SDK. Lorsque l'application commence à s'exécuter sur la carte mère, vous pouvez la déboguer ou interagir avec elle depuis la console série de votre ordinateur hôte.

Ubuntu 16.04 est la plate-forme hôte prise en charge pour le développement et le débogage. Vous pouvez peut-être utiliser d'autres plateformes, mais celles-ci ne sont pas officiellement prises en charge. Vous devez disposer des autorisations nécessaires pour installer le logiciel sur la plate-forme hôte. Les outils externes suivants sont nécessaires pour créer le SDK :

- Plateforme hôte Ubuntu 16.04
- Chaîne d'outils ARM version 4_9_2015q3
- IDE Eclipse 4.9.0

La chaîne d'outils ARM est requise pour la compilation croisée de votre application et du SDK. Le SDK tire parti des dernières versions de la chaîne d'outils pour optimiser l'empreinte de l'image et intégrer davantage de fonctionnalités dans un espace réduit. Ce guide part du principe que vous utilisez la version 4_9_2015q3 de la chaîne d'outils. L'utilisation d'anciennes versions de la chaîne d'outils n'est pas recommandée. Le kit de développement est pré-flashé avec le firmware du projet Wireless Microcontroller Demo.

Rubriques

- Configuration du matériel
- Configuration de l'environnement de développement
- Créez et exécutez le projet de démonstration FreeRTOS
- Débogage
- Résolution des problèmes

Configuration du matériel

Connectez la carte MW32x à votre ordinateur portable à l'aide d'un câble mini-USB vers USB. Connectez le câble mini-USB au seul connecteur mini-USB présent sur la carte. Vous n'avez pas besoin de changer de pull.

Si la carte est connectée à un ordinateur portable ou de bureau, vous n'avez pas besoin d'une alimentation externe.

Cette connexion USB fournit les fonctionnalités suivantes :

- Accès au tableau par console. Un tty/com port virtuel est enregistré auprès de l'hôte de développement et peut être utilisé pour accéder à la console.
- Accès JTAG au tableau. Cela peut être utilisé pour charger ou décharger des images du microprogramme dans la RAM ou le flash de la carte, ou à des fins de débogage.

Configuration de l'environnement de développement

À des fins de développement, la configuration minimale requise est la chaîne d'outils ARM et les outils fournis avec le SDK. Les sections suivantes fournissent des détails sur la configuration de la chaîne d'outils ARM.

Chaîne d'outils GNU

Le SDK prend officiellement en charge la chaîne d'outils GCC Compiler. La chaîne d'outils de compilation croisée pour GNU ARM est disponible sur GNU Arm Embedded Toolchain 4.9-2015-q3-update.

Le système de compilation est configuré pour utiliser la chaîne d'outils GNU par défaut. Les Makefiles supposent que les binaires de la chaîne d'outils du compilateur GNU sont disponibles sur le PATH de l'utilisateur et peuvent être invoqués depuis les Makefiles. Les Makefiles supposent également que les noms de fichiers des binaires de la chaîne d'outils GNU sont préfixés par. `arm-none-eabi-`

La chaîne d'outils GCC peut être utilisée avec GDB pour déboguer avec OpenOCD (fourni avec le SDK). Cela fournit le logiciel qui s'interface avec JTAG.

Nous recommandons la version 4_9_2015q3 de la chaîne d'outils intégrée gcc-arm.

Procédure de configuration de la chaîne d'outils Linux

Suivez ces étapes pour configurer la chaîne d'outils GCC sous Linux.

1. Téléchargez l'archive de la chaîne d'outils disponible sur la mise à jour GNU Arm Embedded Toolchain 4.9-2015-q3-update. Le fichier est`gcc-arm-none-eabi-4_9-2015q3-20150921-linux.tar.bz2`.
2. Copiez le fichier dans le répertoire de votre choix. Assurez-vous qu'il n'y a pas d'espaces dans le nom du répertoire.
3. Utilisez la commande suivante pour décompresser le fichier.

   ```
   tar -vxf filename
   ```

4. Ajoutez le chemin de la chaîne d'outils installée au chemin du système. Par exemple, ajoutez la ligne suivante à la fin du `.profile` fichier situé dans le `/home/user-name` répertoire.

   ```
   PATH=$PATH:path to gcc-arm-none-eabit-4_9_2015_q3/bin
   ```

> **Note**
>
> Les nouvelles distributions d'Ubuntu peuvent être fournies avec une version Debian du compilateur croisé GCC. Si tel est le cas, vous devez supprimer le compilateur croisé natif et suivre la procédure de configuration ci-dessus.

Travailler avec un hôte de développement Linux

Toute distribution de bureau Linux moderne telle qu'Ubuntu ou Fedora peut être utilisée. Toutefois, nous vous recommandons de passer à la version la plus récente. Les étapes suivantes ont été vérifiées pour fonctionner sur Ubuntu 16.04 et supposons que vous utilisez cette version.

Installation de packages

Le SDK inclut un script permettant de configurer rapidement votre environnement de développement sur une machine Linux nouvellement installée. Le script tente de détecter automatiquement le type de machine et d'installer le logiciel approprié, notamment les bibliothèques C, USB, FTDI, ncurses, python et latex. Dans cette section, le nom générique du répertoire *amzsdk_bundle-x.y.z* indique le répertoire racine du AWS SDK. Le nom réel du répertoire peut être différent. Vous devez disposer des privilèges root.

- Accédez au *amzsdk_bundle-x.y.z/* répertoire et exécutez cette commande.

```
./lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/bin/installpkgs.sh
```

Éviter le sudo

Dans ce guide, l'`flashprog`opération utilise le `flashprog.py` script pour flasher le NAND de la carte, comme expliqué ci-dessous. De même, l'`ramload`opération utilise le `ramload.py` script pour copier l'image du microprogramme de l'hôte directement vers la RAM du microcontrôleur, sans faire clignoter la NAND.

Vous pouvez configurer votre hôte de développement Linux pour qu'il exécute les `ramload` opérations `flashprog` et sans avoir besoin de la `sudo` commande à chaque fois. Pour ce faire, exécutez la commande suivante.

```
./lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/bin/perm_fix.sh
```

> **Note**
>
> Vous devez configurer les autorisations de votre hôte de développement Linux de cette manière pour garantir une expérience fluide avec Eclipse IDE.

Configuration de la console série

Insérez le câble USB dans le port USB de l'hôte Linux. Cela déclenche la détection de l'appareil. Vous devriez voir des messages comme celui-ci dans le `/var/log/messages` fichier ou après l'exécution de la `dmesg` commande.

```
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
```

Vérifiez que deux périphériques ttyUSB ont été créés. Le deuxième ttyUSB est la console série. Dans l'exemple ci-dessus, cela s'appelle « ttyUSB1 ».

Dans ce guide, nous utilisons minicom pour voir la sortie de la console série. Vous pouvez également utiliser d'autres programmes série tels que`putty`. Exécutez la commande suivante pour exécuter minicom en mode configuration.

```
minicom -s
```

Dans minicom, accédez à Configuration du port série et capturez les paramètres suivants.

```
| 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
```

Vous pouvez enregistrer ces paramètres dans Minicom en vue d'une utilisation ultérieure. La fenêtre Minicom présente maintenant les messages de la console série.

Choisissez la fenêtre de la console série et appuyez sur la touche Entrée. Cela affiche un hachage (#) à l'écran.

> **Note**
>
> Les cartes de développement incluent un dispositif en silicone FTDI. Le périphérique FTDI expose deux interfaces USB pour l'hôte. La première interface est associée à la fonctionnalité JTAG du microcontrôleur et la seconde interface est associée au port UartX physique du microcontrôleur.

Installation d'OpenOCD

OpenOCD est un logiciel qui fournit le débogage, la programmation intégrée au système et les tests de numérisation des limites pour les appareils cibles intégrés.

OpenOCD version 0.9 est requis. Il est également requis pour les fonctionnalités d'Eclipse. Si une version antérieure, telle que la version 0.7, était installée sur votre hôte Linux, supprimez ce dépôt à l'aide de la commande appropriée pour la distribution Linux que vous utilisez actuellement.

Exécutez la commande Linux standard pour installer OpenOCD,

```
apt-get install openocd
```

Si la commande ci-dessus n'installe pas la version 0.9 ou ultérieure, utilisez la procédure suivante pour télécharger et compiler le code source openocd.

Pour installer OpenOCD

1. Exécutez la commande suivante pour installer libusb-1.0.

   ```
   sudo apt-get install libusb-1.0
   ```

2. Téléchargez le code source d'openocd 0.9.0 depuis. http://openocd.org/
3. Décompressez openocd et naviguez jusqu'au répertoire où vous l'avez extrait.
4. Configurez openocd avec la commande suivante.

   ```
   ./configure --enable-ftdi --enable-jlink
   ```

5. Exécutez l'utilitaire make pour compiler opencd.

   ```
   make install
   ```

Configuration d'Eclipse

> **Note**
>
> Cette section suppose que vous avez effectué les étapes décrites dans Éviter le sudo

Eclipse est l'IDE préféré pour le développement et le débogage d'applications. Il fournit un IDE riche et convivial avec un support de débogage intégré, y compris le débogage sensible aux threads. Cette section décrit la configuration Eclipse commune à tous les hôtes de développement pris en charge.

Pour configurer Eclipse

1. Téléchargez et installez le Java Run Time Environment (JRE).

   Eclipse nécessite que vous installiez le JRE. Nous vous recommandons de l'installer d'abord, bien qu'il puisse être installé après avoir installé Eclipse. La version de JRE (32/64 bit) doit correspondre à la version d'Eclipse (32/64 bit). Vous pouvez télécharger le JRE depuis Java SE Runtime Environment 8 Downloads sur le site Web d'Oracle.
2. Téléchargez et installez « Eclipse IDE pour les développeurs C/C ++ » depuis http://www.eclipse.org. La version 4.9.0 ou ultérieure d'Eclipse est prise en charge. L'installation nécessite uniquement l'extraction de l'archive téléchargée. Vous exécutez le fichier exécutable Eclipse spécifique à la plate-forme pour démarrer l'application.

Créez et exécutez le projet de démonstration FreeRTOS

Il existe deux manières d'exécuter le projet de démonstration FreeRTOS :

- Utilisez la ligne de commande.
- Utilisez l'IDE Eclipse.

Cette rubrique couvre les deux options.

Allouer

- Selon que vous utilisez l'application de test ou de démonstration, définissez les données d'approvisionnement dans l'un des fichiers suivants :

  - `./tests/common/include/aws_clientcredential.h`
  - `./demos/common/include/aws_clientcredential.h`

  Par exemple :

```
#define clientcredentialWIFI_SSID "Wi-Fi SSID"
#define clientcredentialWIFI_PASSWORD "Wi-Fi password"
#define clientcredentialWIFI_SECURITY "Wi-Fi security"
```

> **Note**
>
> Vous pouvez saisir les valeurs Wi-Fi de sécurité suivantes :
>
> - `eWiFiSecurityOpen`
> - `eWiFiSecurityWEP`
> - `eWiFiSecurityWPA`
> - `eWiFiSecurityWPA2`
>
> Le SSID et le mot de passe doivent être placés entre guillemets.

Créez et exécutez la démo de FreeRTOS à l'aide de la ligne de commande

1. Utilisez la commande suivante pour commencer à créer l'application de démonstration.

   ```
   cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S .  -Bbuild -
   DAFR_ENABLE_TESTS=0
   ```

   Assurez-vous d'obtenir le même résultat que celui indiqué dans l'exemple suivant.

```
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. Accédez au répertoire de construction.

```
cd build
```

3. Exécutez l'utilitaire make pour créer l'application.

```
make all -j4
```

Assurez-vous d'obtenir le même résultat que celui illustré dans la figure suivante :



4. Utilisez les commandes suivantes pour créer une application de test.

```
cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S .  -Bbuild -
DAFR_ENABLE_TESTS=1
cd build
make all -j4
```

   Exécutez la `cmake` commande chaque fois que vous basculez entre le `aws_demos project` et le`aws_tests project`.

5. Écrivez l'image du microprogramme sur le flash de la carte de développement. Le microprogramme s'exécutera une fois la carte de développement réinitialisée. Vous devez créer le SDK avant de flasher l'image sur le microcontrôleur.

   a. Avant de flasher l'image du microprogramme, préparez le flash de la carte de développement avec les composants communs Layout et Boot2. Utilisez les commandes suivantes.

   ```
   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
   ```

   La `flashprog` commande initie ce qui suit :

- Mise en page — L'utilitaire flashprog reçoit d'abord l'instruction d'écrire une mise en page dans le flash. La disposition est similaire aux informations de partition pour le flash. La mise en page par défaut se trouve à`/lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/OpenOCD/mw300/layout.txt`.
- Boot2 — Il s'agit du chargeur de démarrage utilisé par le WMSDK. La `flashprog` commande écrit également un bootloader dans le flash. Le bootloader a pour tâche de charger l'image du microcontrôleur du microcontrôleur après son flashage. Assurez-vous d'obtenir le même résultat que celui indiqué dans la figure ci-dessous.

```
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. Le microprogramme utilise le Wi-Fi chipset pour ses fonctionnalités, et le Wi-Fi chipset possède son propre microprogramme qui doit également être présent dans le flash. Vous utilisez l'`flashprog.py`utilitaire pour flasher le Wi-Fi microprogramme de la même manière que vous l'avez fait pour flasher le chargeur de démarrage Boot2 et le microprogramme du microcontrôleur. Utilisez les commandes suivantes pour flasher le Wi-Fi microprogramme.

```
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
```

Assurez-vous que le résultat de la commande est similaire à celui de la figure ci-dessous.

```
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. Utilisez les commandes suivantes pour flasher le microprogramme du microcontrôleur.

```
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. Réinitialisez la carte. Vous devriez voir les journaux de l'application de démonstration.

e. Pour exécuter l'application de test, flashez le `aws_tests.bin` fichier binaire situé dans le même répertoire.

```
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
```

La sortie de votre commande doit être similaire à celle illustrée dans la figure ci-dessous.

```
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. Après avoir flashé le firmware et réinitialisé la carte, l'application de démonstration devrait démarrer comme indiqué dans la figure ci-dessous.

```
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----
```

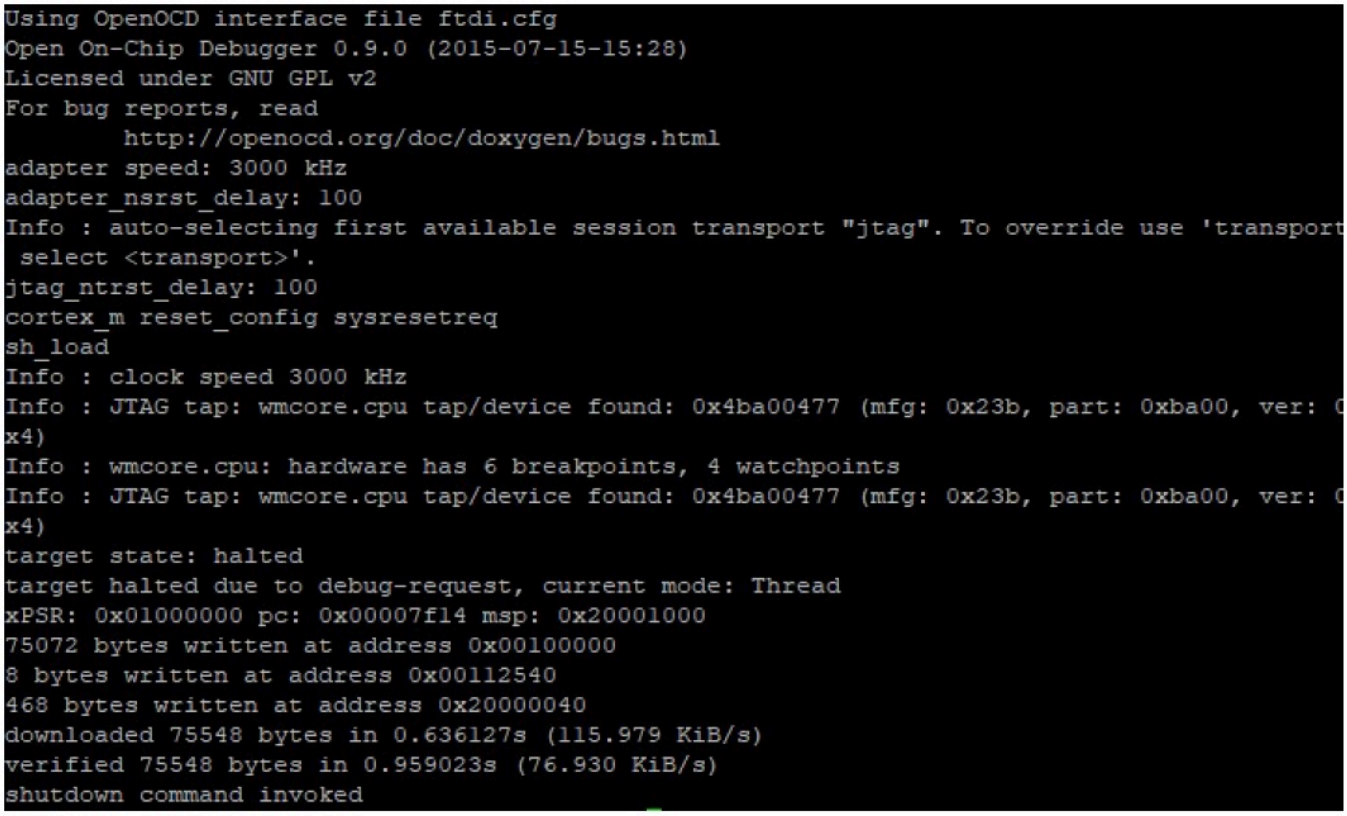
7. (Facultatif) Comme autre méthode pour tester votre image, utilisez l'utilitaire flashprog pour copier l'image du microcontrôleur depuis l'hôte directement dans la RAM du microcontrôleur. L'image n'est pas copiée dans le flash, elle sera donc perdue après le redémarrage du microcontrôleur.

   Le chargement de l'image du microprogramme dans la SRAM est une opération plus rapide car le fichier d'exécution est lancé immédiatement. Cette méthode est principalement utilisée pour le développement itératif.

   Utilisez les commandes suivantes pour charger le microprogramme dans la SRAM.

   ```
   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
   ```

   La sortie de commande est illustrée dans la figure ci-dessous.



Lorsque l'exécution de la commande est terminée, vous devriez voir les journaux de l'application de démonstration.

Créez et exécutez la démo FreeRTOS à l'aide de l'IDE Eclipse

1. Avant de configurer un espace de travail Eclipse, vous devez exécuter la `cmake` commande.

   Exécutez la commande suivante pour travailler avec le projet `aws_demos` Eclipse.

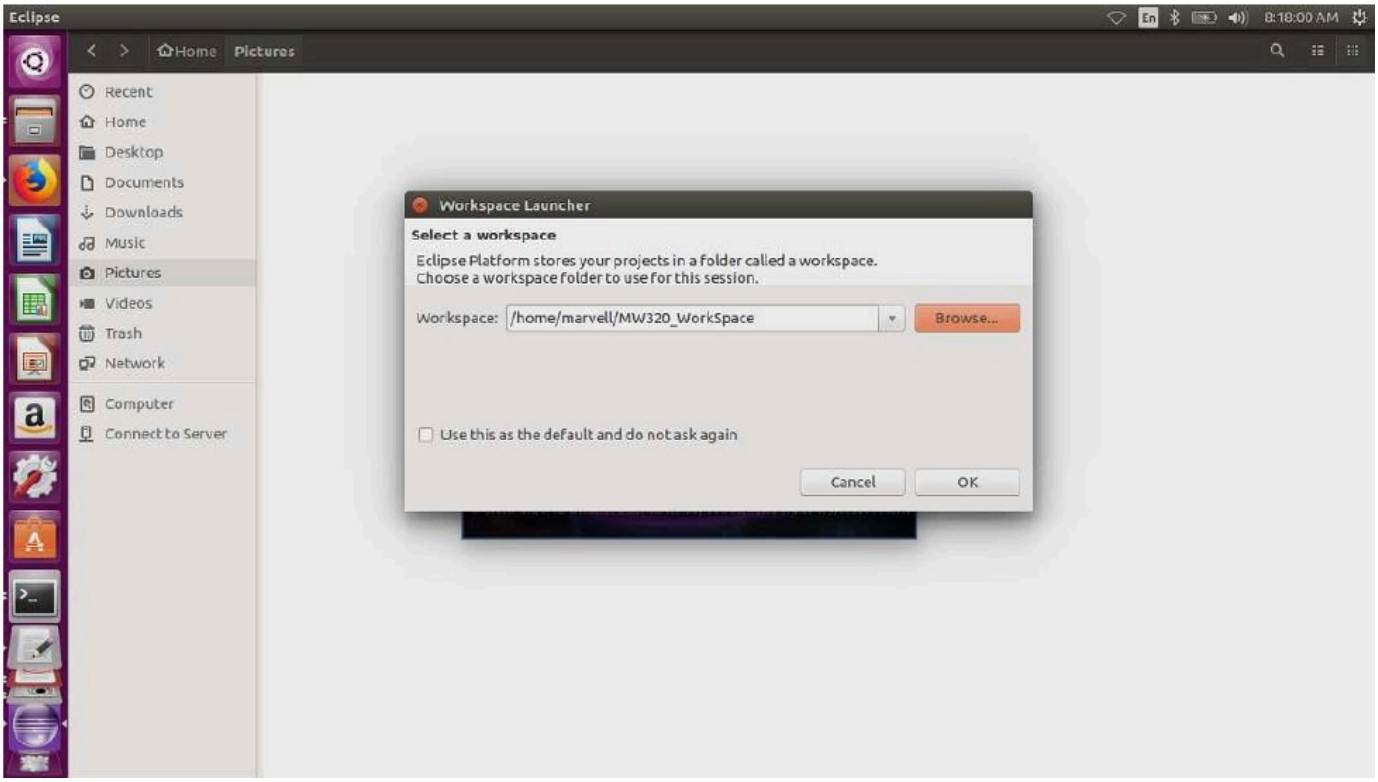
   ```
   cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S .  -Bbuild -
   DAFR_ENABLE_TESTS=0
   ```

   Exécutez la commande suivante pour travailler avec le projet `aws_tests` Eclipse.

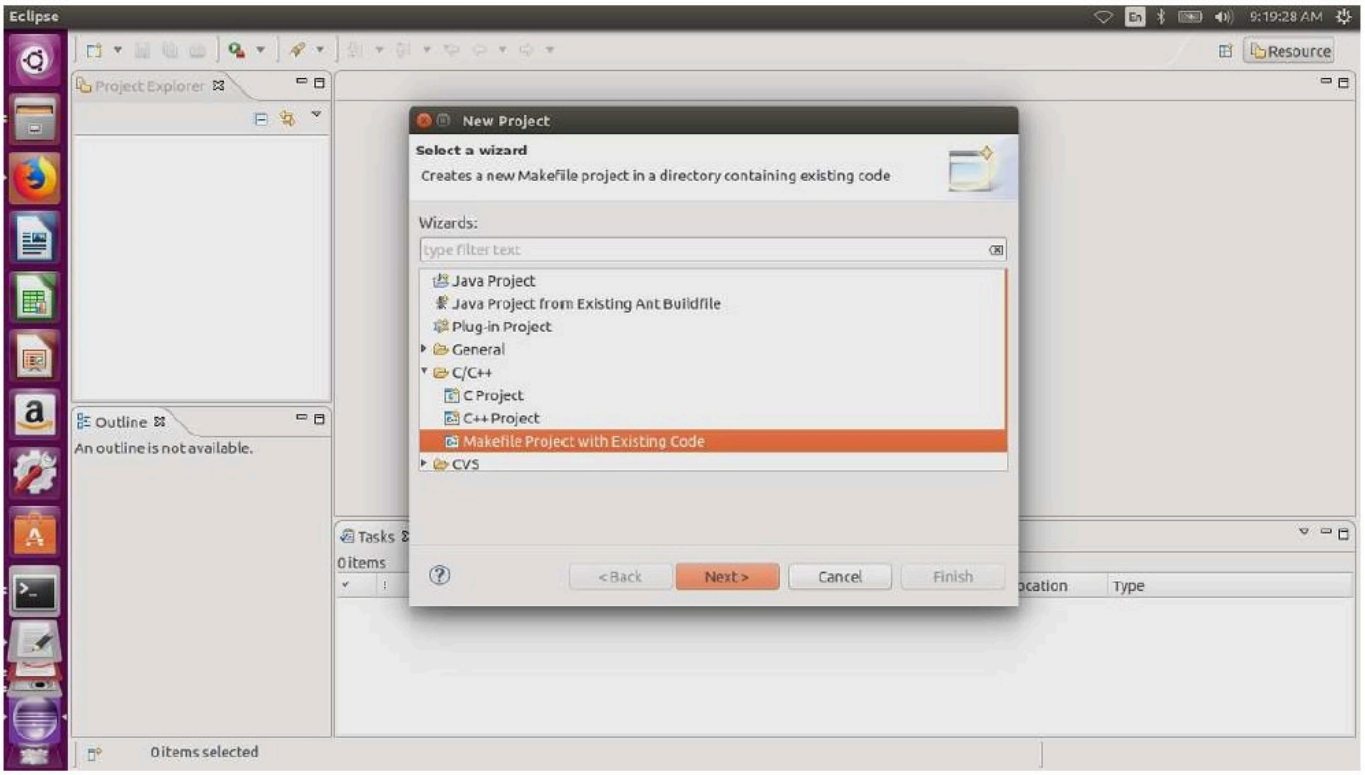
   ```
   cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S .  -Bbuild -
   DAFR_ENABLE_TESTS=1
   ```

> **Tip**
>
> Exécutez la `cmake` commande chaque fois que vous basculez entre le `aws_demos` projet et le `aws_tests` projet.

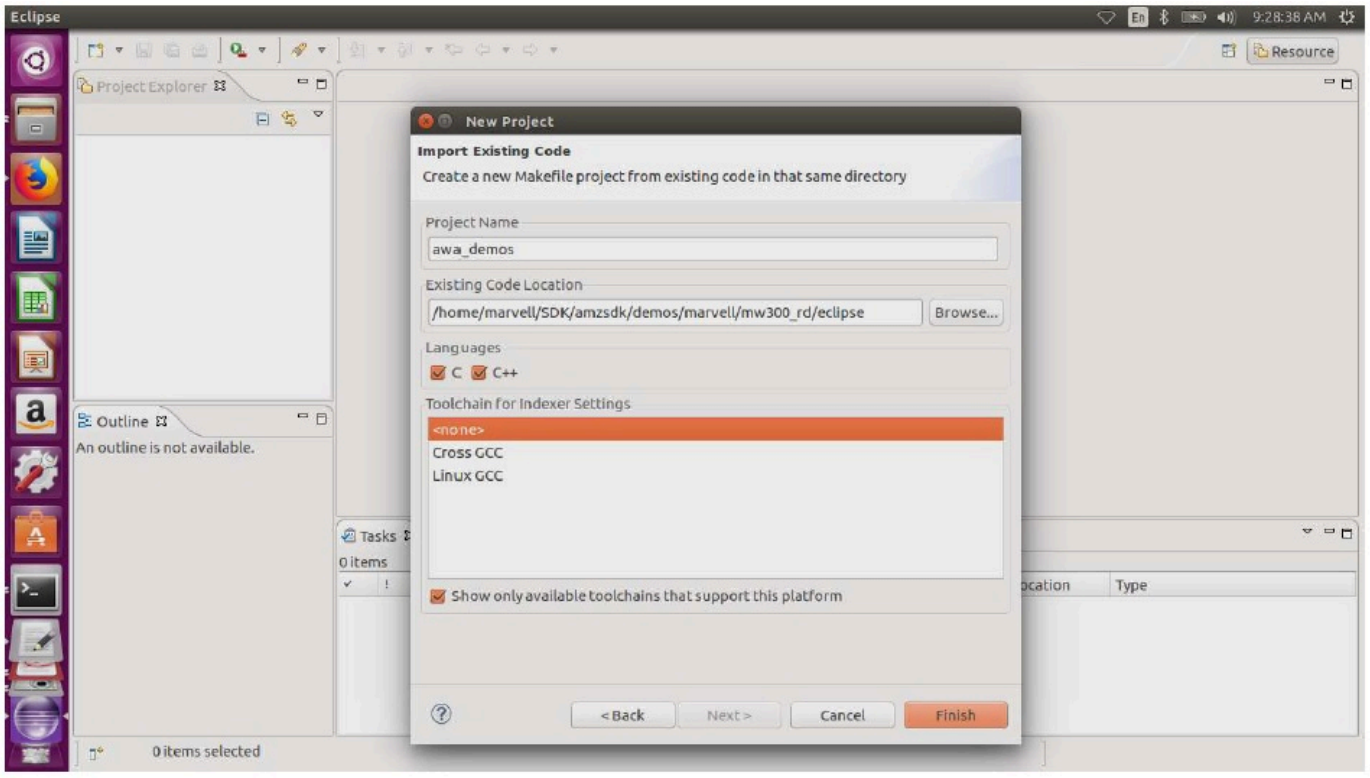
2. Ouvrez Eclipse et, lorsque vous y êtes invité, choisissez votre espace de travail Eclipse comme indiqué dans la figure ci-dessous.



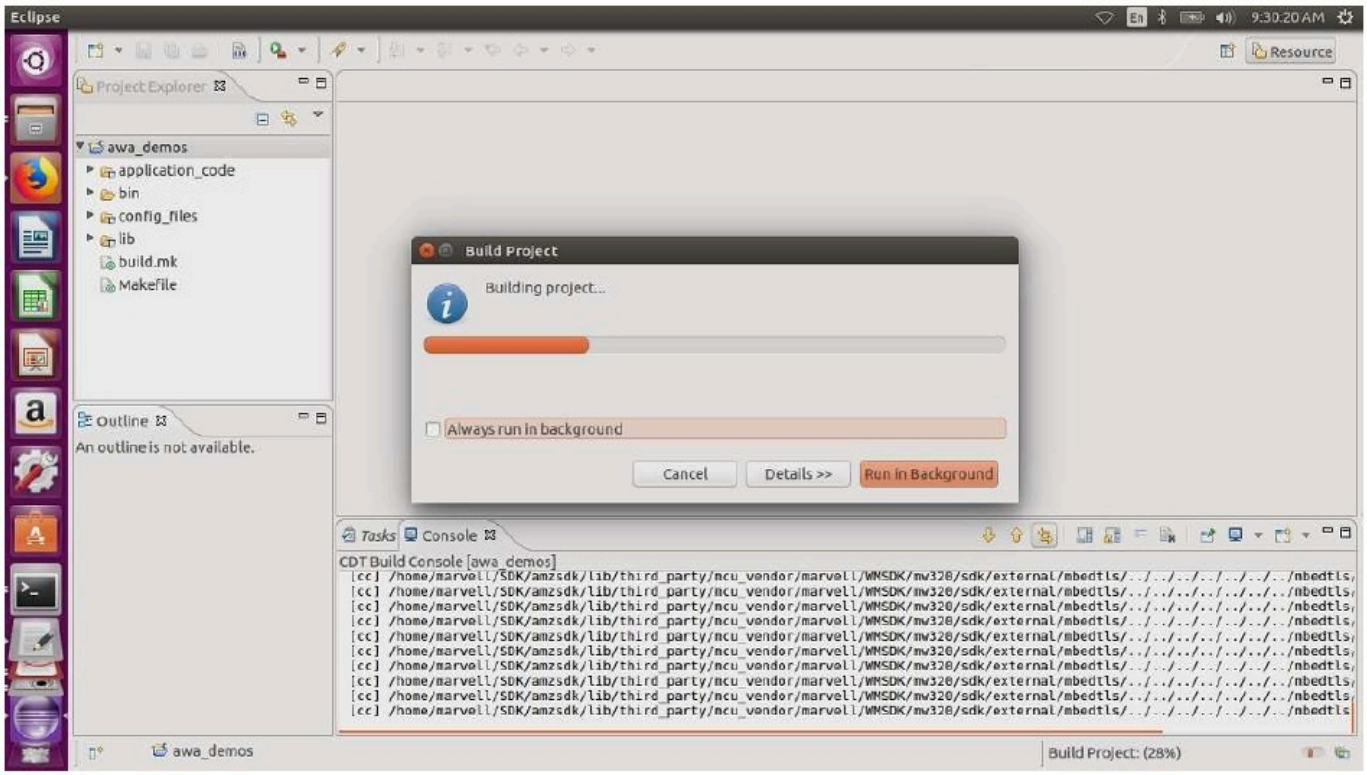
3. Choisissez l'option pour créer un projet Makefile : avec du code existant, comme indiqué dans la figure ci-dessous.



4. Choisissez Parcourir, spécifiez le répertoire du code existant, puis cliquez sur Terminer.



5. Dans le volet de navigation, choisissez aws_demos dans l'explorateur de projets. Right-click aws_demos pour ouvrir le menu, puis choisissez Build.



Si la compilation réussit, le `build/cmake/vendors/marvell/mw300_rd/aws_demos.bin` fichier est généré.

6. Utilisez les outils de ligne de commande pour flasher le fichier Layout (`layout.txt`), le binaire Boot2 (`boot2.bin`), le binaire du microprogramme du microprogramme du microcontrôleur (`aws_demos.bin`) et le Wi-Fi microprogramme.

   a. Avant de flasher l'image du microprogramme, préparez le flash de la carte de développement avec les composants communs, Layout et Boot2. Utilisez les commandes suivantes.

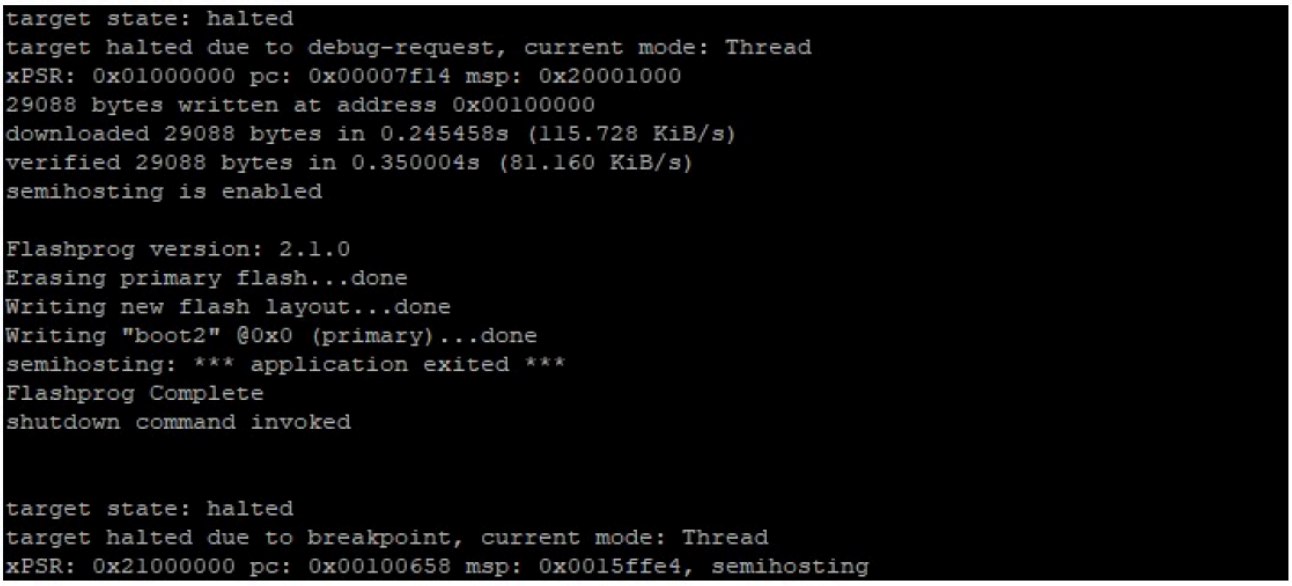
   ```
   cd amzsdk_bundle-x.y.z
   ./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py -l ./vendors/
   marvell/WMSDK/mw320/sdk/tools/OpenOCD/mw300/layout.txt --boot2 ./vendors/
   marvell/WMSDK/mw320/boot2/bin/boot2.bin
   ```

   La `flashprog` commande initie ce qui suit :

   - Mise en page — L'utilitaire flashprog reçoit d'abord l'instruction d'écrire une mise en page dans le flash. La disposition est similaire aux informations de partition pour le flash. La

mise en page par défaut se trouve à`/lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/OpenOCD/mw300/layout.txt`.

- Boot2 — Il s'agit du chargeur de démarrage utilisé par le WMSDK. La commande flashprog écrit également un chargeur de démarrage dans le flash. Le bootloader a pour tâche de charger l'image du firmware du microcontrôleur une fois celle-ci flashée. Assurez-vous d'obtenir le même résultat que celui indiqué dans la figure ci-dessous.



b. Le microprogramme utilise le Wi-Fi chipset pour ses fonctionnalités, et le Wi-Fi chipset possède son propre microprogramme qui doit également être présent dans le flash. Vous utilisez l'`flashprog.py`utilitaire pour flasher le Wi-Fi microprogramme de la même manière que vous l'avez fait pour flasher le chargeur de démarrage boot2 et le microprogramme du microcontrôleur. Utilisez les commandes suivantes pour flasher le Wi-Fi microprogramme.

```
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
```

Assurez-vous que le résultat de la commande est similaire à celui de la figure ci-dessous.

```
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. Utilisez les commandes suivantes pour flasher le microprogramme du microcontrôleur.

```
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. Réinitialisez la carte. Vous devriez voir les journaux de l'application de démonstration.

e. Pour exécuter l'application de test, flashez le `aws_tests.bin` fichier binaire situé dans le même répertoire.

```
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
```

La sortie de votre commande doit être similaire à celle illustrée dans la figure ci-dessous.

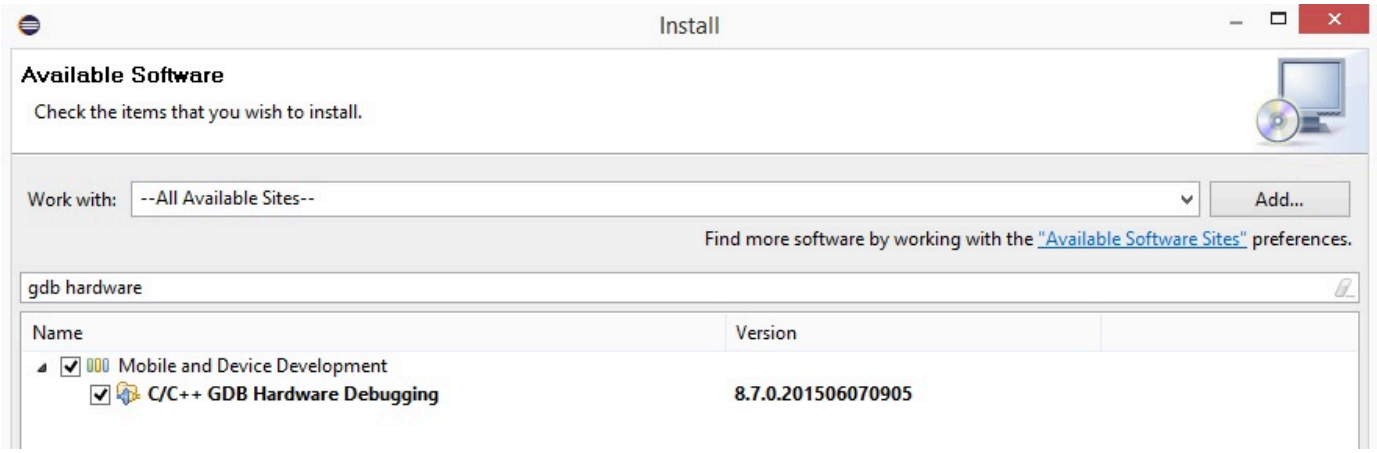
```
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00007f14 msp: 0x20001000
29088 bytes written at address 0x00100000
downloaded 29088 bytes in 0.245499s (115.708 KiB/s)
verified 29088 bytes in 0.350231s (81.107 KiB/s)
semihosting is enabled

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...
```

Débogage

- Démarrez Eclipse et choisissez Aide, puis choisissez Installer un nouveau logiciel. Dans le menu Travailler avec, sélectionnez Tous les sites disponibles. Entrez le texte du filtre`GDB Hardware`. Sélectionnez l'option C/C++ GDB Hardware Debugging et installez le plugin.



Résolution des problèmes

Problèmes liés au réseau

Vérifiez vos informations d'identification réseau. Voir « Provisionnement » dansCréez et exécutez le projet de démonstration FreeRTOS.

Activation de journaux supplémentaires

- Activez les journaux spécifiques au tableau.

  Activez les appels `wmstdio_init(UART0_ID, 0)` à la fonction `prvMiscInitialization` dans le `main.c` fichier pour les tests ou les démonstrations.

- Activation Wi-Fi des journaux

  Activez la macro `CONFIG_WLCMGR_DEBUG` dans le *`freertos`*`/vendors/marvell/WMSDK/mw320/sdk/src/incl/autoconf.h` fichier.

Utilisation de GDB

Nous vous recommandons d'utiliser les fichiers de gdb commandes `arm-none-eabi-gdb` et fournis avec le SDK. Accédez au répertoire .

```
cd freertos/lib/third_party/mcu_vendor/marvell/WMSDK/mw320
```

Exécutez la commande suivante (sur une seule ligne) pour vous connecter à GDB.

```
arm-none-eabi-gdb -x ./sdk/tools/OpenOCD/gdbinit ../../../../../../build/cmake/
vendors/marvell/mw300 _rd/aws_demos.axf
```

Commencer à utiliser le kit de développement MediaTek MT7697hx

> **Important**
>
> Cette intégration de référence est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Ce didacticiel fournit des instructions pour démarrer avec le kit de développement MediaTek MT7697hx. Si vous ne possédez pas le kit de développement MediaTek MT7697hx, consultez le catalogue des appareils AWS partenaires pour en acheter un auprès de notre partenaire.

Avant de commencer, vous devez configurer AWS IoT et télécharger FreeRTOS pour connecter votre appareil au Cloud. AWS Pour obtenir des instructions, consultez Premiers pas. Dans ce didacticiel, le chemin d'accès au répertoire de téléchargement de FreeRTOS est appelé. *freertos*

Présentation de

Ce didacticiel comprend les instructions de mise en route suivantes :

1. Installation de logiciels sur la machine hôte pour développer et déboguer des applications intégrées pour votre carte de microcontrôleur.
2. Compilation croisée d'une application de démonstration FreeRTOS en une image binaire.
3. Chargement de l'image binaire de l'application dans votre carte et exécution de l'application.
4. Interaction avec l'application s'exécutant sur votre carte via une connexion série, à des fins de surveillance et de débogage.

Configurer votre environnement de développement.

Avant de configurer votre environnement, connectez votre ordinateur au port USB du kit de développement MediaTek MT7697hx.

Télécharger et installer le kit MDK Keil

Vous pouvez utiliser le kit de développement de microcontrôleurs (MDK) GUI-based Keil pour configurer, créer et exécuter des projets FreeRTOS sur votre carte mère. Keil MDK inclut l'IDE µVision et le débogueur µVision.

> **Note**
>
> Le kit MDK Keil est pris en charge sur Windows 7, Windows 8 et Windows 10, machines 64 bits uniquement.

Pour télécharger et installer le kit MDK Keil

1. Accédez à la page de démarrage de Keil MDK, puis choisissez Télécharger. MDK-Core
2. Entrez et soumettez vos informations pour vous enregistrer auprès de Keil.
3. Right-click le fichier exécutable MDK et enregistrez le programme d'installation de Keil MDK sur votre ordinateur.
4. Ouvrez le programme d'installation du kit MDK Keil et suivez les étapes jusqu'à la fin. Assurez-vous d'installer le pack de MediaTek périphériques (série MT76x7).

Établir une connexion série

Connectez la carte à votre ordinateur hôte à l'aide d'un câble USB. Un port COM apparaît dans le Gestionnaire de périphériques Windows. Pour le débogage, vous pouvez ouvrir une session sur le port à l'aide d'un utilitaire de terminal tel que HyperTerminal ou TeraTerm.

Surveillance des messages MQTT dans le cloud

Avant de lancer le projet de démonstration FreeRTOS, vous pouvez configurer le client MQTT dans la console pour surveiller AWS IoT les messages que votre appareil envoie au Cloud. AWS

Pour vous abonner à la rubrique MQTT avec AWS IoT Client MQTT

1. Connectez-vous à la console AWS IoT.
2. Dans le volet de navigation, choisissez Test, puis choisissez MQTT test client pour ouvrir le client MQTT.
3. Dans le champ Rubrique d'abonnement, saisissez ***your-thing-name*****example/topic**, puis choisissez S'abonner à la rubrique.

Lorsque le projet de démonstration s'exécute avec succès sur votre appareil, le message « Hello World ! » s'affiche envoyé plusieurs fois au sujet auquel vous vous êtes abonné.

Créez et exécutez le projet de démonstration FreeRTOS avec Keil MDK

Pour créer le projet de démonstration FreeRTOS dans Keil µVision

1. Dans le menu Démarrer, ouvrez Keil µVision 5.
2. Ouvrez le fichier de projet `projects/mediatek/mt7697hx-dev-kit/uvision/aws_demos/aws_demos.uvprojx`.
3. Dans le menu, choisissez Project (Projet), puis Build target (Générer la cible).

   Une fois le code généré, vous voyez le fichier exécutable de la démonstration à l'adresse `projects/mediatek/mt7697hx-dev-kit/uvision/aws_demos/out/0bjects/aws_demo.axf`.

Pour exécuter le projet de démonstration FreeRTOS

1. Réglez le kit de développement MediaTek MT7697hx sur le mode PROGRAM.

   Pour placer le kit en mode PROGRAMME, maintenez enfoncé le bouton PROG. Alors que le bouton PROG est toujours enfoncé, maintenez enfoncé le bouton RESET, puis relâchez le bouton PROG.
2. Dans le menu, choisissez Flash, puis Configure Flash Tools (Configurer les outils flash).
3. Dans Options for Target '**`aws_demo`**', choisissez l'onglet Debug (Déboguer). Sélectionnez Utiliser, réglez le débogueur sur CMSIS-DAP Debugger, puis cliquez sur OK.
4. Dans le menu, choisissez Flash, puis Download (Télécharger).

   µVision vous informe lorsque le téléchargement est terminé.
5. Utilisez un utilitaire de terminal pour ouvrir la fenêtre de console série. Définissez le port série sur 115 200 bits/s, pas de parité, 8 bits et 1 bit d'arrêt.
6. Cliquez sur le bouton RESET de votre kit de développement MediaTek MT7697hx.

Résolution des problèmes

Débogage de projets FreeRTOS dans Keil µVision

Actuellement, vous devez modifier le MediaTek package inclus dans Keil µVision avant de pouvoir déboguer le projet de démonstration de FreeRTOS avec Keil µVision. MediaTek

Pour modifier le MediaTek package de débogage des projets FreeRTOS

1. Recherchez et ouvrez le fichier `Keil_v5\ARM\PACK\.Web\MediaTek.MTx.pdsc` dans votre dossier d'installation du kit MDK Keil.
2. Remplacez toutes les instances de `flag = Read32(0x20000000);` par `flag = Read32(0x0010FBFC);`.
3. Remplacez toutes les instances de `Write32(0x20000000, 0x76877697);` par `Write32(0x0010FBFC, 0x76877697);`.

Pour démarrer le débogage du projet

1. Dans le menu, choisissez Flash, puis Configure Flash Tools (Configurer les outils flash).
2. Choisissez l'onglet Target, puis sélectionnez Read/Write Memory Areas. Vérifiez qu'IRAM1 et IRAM2 sont sélectionnés.
3. Choisissez l'onglet Debug, puis CMSIS-DAP Debugger.
4. Ouvrez `vendors/mediatek/boards/mt7697hx-dev-kit/aws_demos/application_code/main.c` et définissez la macro `MTK_DEBUGGER` sur `1`.
5. Reconstruisez le projet de démonstration dans µVision.
6. Réglez le kit de développement MediaTek MT7697hx sur le mode PROGRAM.

   Pour placer le kit en mode PROGRAMME, maintenez enfoncé le bouton PROG. Alors que le bouton PROG est toujours enfoncé, maintenez enfoncé le bouton RESET, puis relâchez le bouton PROG.
7. Dans le menu, choisissez Flash, puis Download (Télécharger).

   µVision vous informe lorsque le téléchargement est terminé.
8. Appuyez sur le bouton RESET de votre kit de développement MediaTek MT7697hx.
9. Dans le menu µVision, choisissez Debug, puis Start/Stop Debug Session. La fenêtre Call Stack +Locals (Pile d'appel + Locales) s'ouvre lorsque vous démarrez le débogage de session.

10. Dans le menu, choisissez Debug (Déboguer), puis Stop (Arrêter) pour interrompre l'exécution du code. Le compteur du programme s'arrête à la ligne suivante :

```
{ volatile int wait_ice = 1 ; while ( wait_ice ) ; }
```

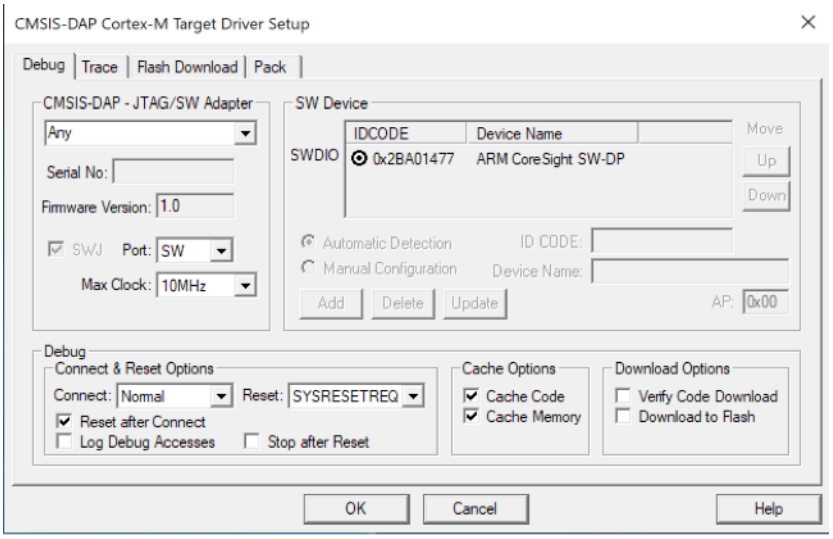
11. Dans la fenêtre Call Stack + Locals (Pile d'appels + Locales), changez la valeur de `wait_ice` en `0`.
12. Définissez des points d'arrêt dans le code source de votre projet, puis exécutez le code.

Dépannage des paramètres du débogueur IDE

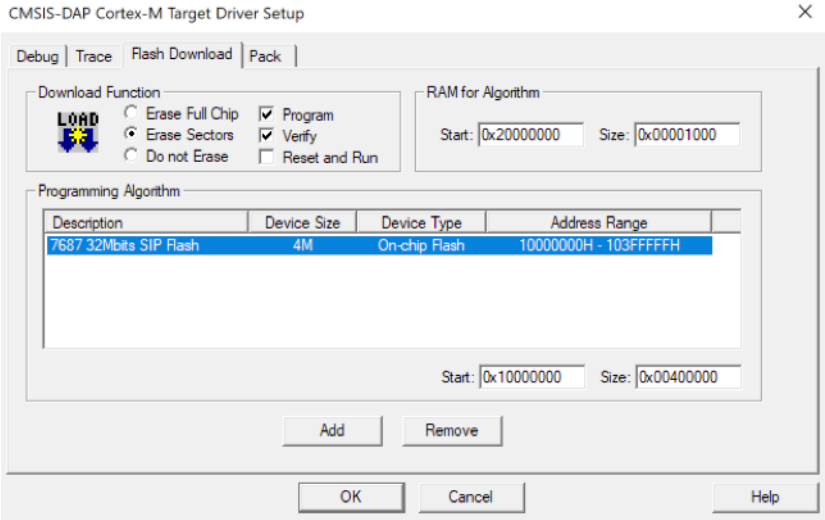
Si vous rencontrez des difficultés pour déboguer une application, les paramètres de votre débogueur sont peut-être incorrects.

Pour vérifier qu'ils sont corrects :

1. Ouvrez Keil µVision.
2. Right-click le `aws_demos` projet, choisissez Options, puis sous l'onglet Utilitaires, choisissez Paramètres, à côté de « -- Utiliser le pilote de débogage -- ».
3. Vérifiez que les paramètres sous l'onglet Debug (Débogage) s'affichent comme suit :



4. Vérifiez que les paramètres sous l'onglet Flash Download (Télécharger en mémoire flash s'affichent comme suit :



Pour obtenir des informations générales sur la résolution des problèmes liés à la prise en main de FreeRTOS, consultez. Résolution des problèmes de mise en route

Démarrez avec la solution groupée Microchip Curiosity PIC32MZ EF

Important

Cette intégration de référence est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Note

En accord avec Microchip, nous retirons le Curiosity PIC32MZEF (DM320104) de la branche principale du référentiel FreeRTOS Reference Integration et ne le proposerons plus dans les nouvelles versions. Microchip a publié un avis officiel indiquant que le PIC32MZEF (DM320104) n'est plus recommandé pour les nouveaux modèles. Les projets et le code source PIC32MZEF sont toujours accessibles via les balises de version précédentes. Microchip recommande aux clients d'utiliser la carte de PIC32MZ-EF-2.0 développement Curiosity (DM320209) pour les nouvelles conceptions. La plate-forme Pic32MZv1 se trouve toujours dans la version v202012.00 du référentiel FreeRTOS Reference Integration.

Cependant, la plateforme n'est plus prise en charge par la version v202107.00 du FreeRTOS Reference.

Ce didacticiel fournit des instructions pour démarrer avec le Microchip Curiosity PIC32MZ EF. Si vous ne possédez pas le pack Microchip Curiosity PIC32MZ EF, consultez le catalogue des appareils AWS partenaires pour en acheter un auprès de notre partenaire.

La solution groupée inclut les éléments suivants :

- Carte de développement Curiosity PIC32MZ EF
- MikroElectronika Tableau de clic USB UART
- MikroElectronika WiFi Tableau en 7 clics
- Carte fille PIC32 LAN8720 PHY

Vous avez également besoin des éléments suivants pour le débogage :

- Débogueur MPLAB Snap In-Circuit
- (Facultatif) PICkit 3 Programming Cable Kit

Avant de commencer, vous devez configurer AWS IoT et télécharger FreeRTOS pour connecter votre appareil au Cloud. AWS Pour obtenir des instructions, consultez Premiers pas.

⚠ Important

- Dans cette rubrique, le chemin d'accès au répertoire de téléchargement de FreeRTOS est appelé. *`freertos`*
- Les espaces dans le chemin d'accès *`freertos`* peuvent provoquer des échecs de construction. Lorsque vous clonez ou copiez le référentiel, assurez-vous que le chemin d'accès que vous créez ne contient pas d'espaces.
- La longueur maximale d'un chemin sous Microsoft Windows est de 260 caractères. Les longs chemins de répertoire de téléchargement de FreeRTOS peuvent provoquer des échecs de compilation.
- Le code source pouvant contenir des liens symboliques, si vous utilisez Windows pour extraire l'archive, vous devrez peut-être :

- Activez le mode développeur ou,
- Utilisez une console élevée en tant qu'administrateur.

Ainsi, Windows peut créer correctement des liens symboliques lors de l'extraction de l'archive. Dans le cas contraire, les liens symboliques seront écrits sous forme de fichiers normaux contenant les chemins des liens symboliques sous forme de texte ou seront vides. Pour plus d'informations, consultez le billet de blog Symlinks in Windows 10 ! .

Si vous utilisez Git sous Windows, vous devez activer le mode développeur ou vous devez :

- Définissez `core.symlinks` ce paramètre sur true à l'aide de la commande suivante :

```
git config --global core.symlinks true
```

- Utilisez une console élevée en tant qu'administrateur chaque fois que vous utilisez une commande git qui écrit sur le système (par exemplegit pull,git clone, etgit submodule update --init --recursive).

Présentation de

Ce didacticiel comprend les instructions de mise en route suivantes :

1. Connexion de votre carte à un appareil hôte.
2. Installation de logiciels sur la machine hôte pour développer et déboguer des applications intégrées pour votre carte de microcontrôleur.
3. Compilation croisée d'une application de démonstration FreeRTOS en une image binaire.
4. Chargement de l'image binaire de l'application dans votre carte et exécution de l'application.
5. Interaction avec l'application s'exécutant sur votre carte via une connexion série, à des fins de surveillance et de débogage.

Configurer le matériel Microchip Curiosity PIC32MZ EF

1. Connectez la carte MikroElectronika USB UART Click au connecteur MicroBus 1 du Microchip Curiosity PIC32MZ EF.
2. Connectez la carte fille PIC32 LAN8720 PHY au connecteur J18 de Microchip Curiosity PIC32MZ EF.

3. Connectez la carte MikroElectronika USB UART Click à votre ordinateur à l'aide d'un câble USB A vers USB mini-B.
4. Pour connecter votre carte à Internet, utilisez l'une des options suivantes :

   - Pour l'utiliser Wi-Fi, connectez la carte MikroElectronika Wi-Fi 7 clics au connecteur MicroBus 2 du Microchip Curiosity PIC32MZ EF. Consultez Configuration des démos de FreeRTOS.
   - Pour utiliser Ethernet afin de connecter la carte Microchip Curiosity PIC32MZ EF à Internet, connectez la carte fille PIC32 LAN8720 PHY au connecteur J18 sur la carte Microchip Curiosity PIC32MZ EF. Connectez une extrémité du câble Ethernet à la carte fille LAN8720 PHY. Connectez l'autre extrémité à votre routeur ou autre port Internet. Vous devez également définir la macro `PIC32_USE_ETHERNET` du préprocesseur.
5. Si ce n'est pas déjà fait, soudez le connecteur d'angle au connecteur ICSP de la carte Curiosity PIC32MZ EF.
6. Connectez une extrémité du câble ICSP du PICkit 3 Programming Cable Kit à la carte Microchip Curiosity PIC32MZ EF.

   Si vous n'avez pas le kit de câbles de programmation PickIt 3, vous pouvez utiliser des sauteurs de fil M-F Dupont pour câbler la connexion à la place. Notez que le cercle blanc indique la position de la broche 1.
7. Connectez l'autre extrémité du câble ICSP (ou cavaliers) au MPLAB Snap Debugger. La broche 1 du connecteur SIL Programming Connector à 8 broches est signalée par le triangle noir en bas à droite de la carte mère.

   Assurez-vous que l'ensemble du câblage vers la broche 1 sur la carte Curiosity PIC32MZ EF, signalée par le cercle blanc, est aligné avec la broche 1 du MPLAB Snap Debugger.

   Pour plus d'informations sur le MPLAB Snap Debugger, consultez la fiche d'information du MPLAB Snap Debugger. In-Circuit

Configuration du matériel Microchip Curiosity PIC32MZ EF à l'aide de PicKit On Board (PKOB)

Nous vous recommandons de suivre la procédure de configuration décrite dans la section précédente. Cependant, vous pouvez évaluer et exécuter des démos FreeRTOS avec un débogage de base à l'aide du PickIt On Board (PKOB) intégré en suivant ces étapes. programmer/debugger

1. Connectez la carte MikroElectronika USB UART Click au connecteur MicroBus 1 du Microchip Curiosity PIC32MZ EF.

2. Pour connecter votre carte à Internet, effectuez l'une des opérations suivantes :

   - Pour l'utiliser Wi-Fi, connectez la carte MikroElectronika Wi-Fi 7 clics au connecteur MicroBus 2 du Microchip Curiosity PIC32MZ EF. (Suivez les étapes « Pour configurer votre Wi-Fi » dansConfiguration des démos de FreeRTOS.
   - Pour utiliser Ethernet afin de connecter la carte Microchip Curiosity PIC32MZ EF à Internet, connectez la carte fille PIC32 LAN8720 PHY au connecteur J18 sur la carte Microchip Curiosity PIC32MZ EF. Connectez une extrémité du câble Ethernet à la carte fille LAN8720 PHY. Connectez l'autre extrémité à votre routeur ou autre port Internet. Vous devez également définir la macro `PIC32_USE_ETHERNET` du préprocesseur.

3. Connectez le port USB micro-B nommé « USB DEBUG » de la carte Microchip Curiosity PIC32MZ EF à votre ordinateur au moyen d'un câble USB type A vers USB micro-B.
4. Connectez la carte MikroElectronika USB UART Click à votre ordinateur à l'aide d'un câble USB A vers USB mini-B.

Configurer votre environnement de développement.

> **Note**
>
> Le projet FreeRTOS pour cet appareil est basé sur MPLAB Harmony v2. Pour créer le projet, vous devez utiliser des versions des outils MPLAB compatibles avec Harmony v2, comme la version 2.10 du compilateur MPLAB XC32 et les versions 2. X.X du configurateur MPLAB Harmony (MHC).

1. Installez Python version 3.x ou une version ultérieure.
2. Installez l'IDE MPLAB X :

> **Note**
>
> AWS FreeRTOS Reference Integrations v202007.00 n'est actuellement pris en charge que sur. MPLabv5.35 Les versions précédentes des intégrations de référence AWS FreeRTOS sont prises en charge sur. MPLabv5.40

MPLabv5.35 téléchargements

- Environnement de développement intégré MPLAB X pour Windows
- Environnement de développement intégré MPLAB X pour macOS
- Environnement de développement intégré MPLAB X pour Linux

Derniers téléchargements de MPlab () MPLabv5.40

- Environnement de développement intégré (IDE) MPLAB X pour Windows
- Environnement de développement intégré (IDE) MPLAB X pour macOS
- Environnement de développement intégré (IDE) MPLAB X pour Linux

3. Installez le compilateur MPLAB XC32 :

   - Compilateur XC32/32 MPLAB++ pour Windows
   - Compilateur XC32/32 MPLAB++ pour macOS
   - Compilateur MPLAB XC32/32 ++ pour Linux

4. Démarrez un émulateur de terminal UART et ouvrez une connexion avec les paramètres suivants :

   - Vitesse de transmission : 115200
   - Données : 8 bits
   - Parité : aucune
   - Bits d'arrêt : 1
   - Contrôle de flux : aucun

Surveillance des messages MQTT dans le cloud

Avant de lancer le projet de démonstration FreeRTOS, vous pouvez configurer le client MQTT dans la console pour surveiller AWS IoT les messages que votre appareil envoie au Cloud. AWS

Pour vous abonner à la rubrique MQTT avec le AWS IoT Client MQTT

1. Connectez-vous à la console AWS IoT.

2. Dans le volet de navigation, choisissez Test, puis choisissez MQTT test client pour ouvrir le client MQTT.
3. Dans le champ Rubrique d'abonnement, saisissez ***your-thing-name*****/example/topic**, puis choisissez S'abonner à la rubrique.

Lorsque le projet de démonstration s'exécute avec succès sur votre appareil, vous voyez « Hello World ! » envoyé plusieurs fois au sujet auquel vous vous êtes abonné.

Créez et exécutez le projet de démonstration FreeRTOS

Ouvrez la démo FreeRTOS dans l'IDE MPLAB

1. Ouvrez l'IDE MPLAB. Si vous avez plusieurs versions de compilateur installées, vous devez sélectionner le compilateur que vous souhaitez utiliser à partir de l'IDE.
2. Dans le menu Fichier, choisissez Nouveau projet.
3. Accédez à `projects/microchip/curiosity_pic32mzef/mplab/aws_demos` et ouvrez-le.
4. Choisissez Open Project (Ouvrir un projet).

> **Note**
>
> Lorsque vous ouvrez le projet pour la première fois, vous pouvez recevoir un message d'erreur sur le compilateur. Dans l'IDE, accédez à Outils, Options, Intégré, puis sélectionnez le compilateur que vous utilisez pour votre projet.

Pour utiliser Ethernet pour la connexion, vous devez définir la macro `PIC32_USE_ETHERNET` du préprocesseur.

Pour utiliser Ethernet pour se connecter à l'aide de l'IDE MPLAB

1. Dans l'IDE MPLAB, cliquez avec le bouton droit sur le projet et sélectionnez Propriétés.
2. Dans la boîte de dialogue Propriétés du projet, choisissez ***compiler-name***(Options globales) pour le développer, puis sélectionnez ***compiler-name***-gcc.
3. Pour les catégories Options, choisissez Prétraitement et messages, puis ajoutez la `PIC32_USE_ETHERNET` chaîne aux macros du préprocesseur.

Exécutez le projet de démonstration FreeRTOS

1. Générez une nouvelle build pour votre projet.
2. Sous l'onglet Projects (Projets), cliquez avec le bouton droit sur le dossier de niveau supérieur `aws_demos`, puis choisissez Debug (Déboguer).
3. Lorsque le débogueur s'arrête au point d'arrêt dans `main()`, dans le menu Run (Exécuter), choisissez Resume (Reprendre).

Créez la démo de FreeRTOS avec CMake

Si vous préférez ne pas utiliser d'IDE pour le développement de FreeRTOS, vous pouvez également utiliser CMake pour créer et exécuter les applications de démonstration ou les applications que vous avez développées à l'aide d'éditeurs de code et d'outils de débogage tiers.

Pour créer la démo de FreeRTOS avec CMake

1. Créez un répertoire pour contenir les fichiers de compilation générés, tels que*build-directory*.
2. Utilisez la commande suivante pour générer des fichiers de compilation à partir du code source.

```
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**
>
> Vous devez spécifier les chemins corrects vers les fichiers binaires d'Hexmate et de la chaîne d'outils, tels que les `C:\Program Files (x86)\Microchip\MPLABX \v5.35\mplab_platform\bin` chemins et. `C:\Program Files\Microchip \xc32\v2.40\bin`

3. Remplacez les répertoires par le répertoire build (*build-directory*), puis exécutez-les `make` à partir de ce répertoire.

Pour de plus amples informations, veuillez consulter Utilisation de CMake avec FreeRTOS.

Pour utiliser Ethernet pour la connexion, vous devez définir la macro `PIC32_USE_ETHERNET` du préprocesseur.

Résolution des problèmes

Pour plus d'informations sur le dépannage, consultez Résolution des problèmes de mise en route.

Débuter avec le Nordic n RF52840-DK

> **Important**
>
> Cette intégration de référence est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Ce didacticiel fournit des instructions pour démarrer avec le Nordic RF52840-DK n. Si vous ne possédez pas le Nordic nRF52840-DK, consultez le catalogue des appareils AWS partenaires pour en acheter un auprès de notre partenaire.

Avant de commencer, vous devez effectuer la procédure Configuration AWS IoT et Amazon Cognito pour FreeRTOS Bluetooth Low Energy.

Pour exécuter la démo FreeRTOS Bluetooth Low Energy, vous avez également besoin d'un appareil mobile iOS ou Android doté de fonctionnalités Bluetooth. Wi-Fi

> **Note**
>
> Si vous utilisez un appareil iOS, vous avez besoin de Xcode pour générer l'application de démonstration pour mobile. Si vous utilisez un appareil Android, vous pouvez utiliser Android Studio pour générer l'application de démonstration pour mobile.

Présentation de

Ce didacticiel comprend les instructions de mise en route suivantes :

1. Connexion de votre carte à un appareil hôte.
2. Installation de logiciels sur la machine hôte pour développer et déboguer des applications intégrées pour votre carte de microcontrôleur.

3. Compilation croisée d'une application de démonstration FreeRTOS en une image binaire.
4. Chargement de l'image binaire de l'application dans votre carte et exécution de l'application.
5. Interaction avec l'application s'exécutant sur votre carte via une connexion série, à des fins de surveillance et de débogage.

Configurer le matériel Nordic

Connectez l'ordinateur hôte au port USB J2, situé juste au-dessus du support de batterie de la carte Nordic nRF52840.

Pour plus d'informations sur la configuration du Nordic nRF52840-DK, consultez le guide de l'utilisateur du kit de développement nRF52840.

Configurer votre environnement de développement.

Télécharger et installer Segger Embedded Studio

FreeRTOS prend en charge Segger Embedded Studio en tant qu'environnement de développement pour le Nordic n. RF52840-DK

Pour configurer votre environnement, vous devez télécharger et installer Segger Embedded Studio sur votre ordinateur hôte.

Pour télécharger et installer Segger Embedded Studio

1. Accédez à la page Segger Embedded Studio Downloads, puis choisissez l'option Embedded Studio pour ARM pour votre système d'exploitation.
2. Exécutez le programme d'installation et suivez les instructions jusqu'à la fin.

Configurer l'application de démonstration du SDK FreeRTOS Bluetooth Low Energy Mobile

Pour exécuter le projet de démonstration FreeRTOS via Bluetooth Low Energy, vous devez exécuter l'application de démonstration FreeRTOS Bluetooth Low Energy Mobile SDK sur votre appareil mobile.

Pour configurer l'application de démonstration du SDK FreeRTOS Bluetooth Low Energy Mobile

1. Suivez les instructions de SDK mobiles pour appareils Bluetooth FreeRTOS pour télécharger et installer le kit SDK pour votre plateforme mobile sur votre ordinateur hôte.

2. Suivez les instructions de la rubrique Application de démonstration du SDK FreeRTOS Bluetooth Low Energy Mobile pour configurer la démonstration d'applications mobiles sur votre appareil mobile.

Établir une connexion série

Segger Embedded Studio inclut un émulateur de terminal que vous pouvez utiliser pour recevoir des messages de journal sur une connexion série à votre carte.

Pour établir une connexion série avec Segger Embedded Studio

1. Ouvrez Segger Embedded Studio.
2. Dans le menu supérieur, choisissez Target, Connect J-Link.
3. Dans le menu supérieur, choisissez Tools (Outils), Terminal Emulator (Émulateur de terminal), Properties (Propriétés) et définissez les propriétés selon les instructions contenues dans Installation d'un émulateur de terminal.
4. Dans le menu supérieur, choisissez Tools, Terminal Emulator, Connect ***port*** (115200, N,8,1).

> **Note**
>
> L'émulateur de terminal studio intégré Segger ne prend pas en charge une capacité d'entrée. Pour cela, utilisez un émulateur de terminal comme PuTTY, Tera Term ou GNU Screen. Configurez le terminal pour vous connecter à votre carte via une connexion série, comme indiqué dans Installation d'un émulateur de terminal.

Téléchargez et configurez FreeRTOS

Après avoir configuré votre matériel et votre environnement, vous pouvez télécharger FreeRTOS.

Télécharger FreeRTOS

Pour télécharger FreeRTOS for the Nordic RF52840-DK n, rendez-vous sur la page FreeRTOS GitHub et clonez le dépôt. Consultez le README.mdfichier pour obtenir des instructions.

⚠ Important

- Dans cette rubrique, le chemin d'accès au répertoire de téléchargement de FreeRTOS est appelé. *`freertos`*
- Les espaces dans le chemin d'accès *`freertos`* peuvent provoquer des échecs de construction. Lorsque vous clonez ou copiez le référentiel, assurez-vous que le chemin d'accès que vous créez ne contient pas d'espaces.
- La longueur maximale d'un chemin sous Microsoft Windows est de 260 caractères. Les longs chemins de répertoire de téléchargement de FreeRTOS peuvent provoquer des échecs de compilation.
- Le code source pouvant contenir des liens symboliques, si vous utilisez Windows pour extraire l'archive, vous devrez peut-être :
  - Activez le mode développeur ou
  - Utilisez une console élevée en tant qu'administrateur.

  Ainsi, Windows peut créer correctement des liens symboliques lors de l'extraction de l'archive. Dans le cas contraire, les liens symboliques seront écrits sous forme de fichiers normaux contenant les chemins des liens symboliques sous forme de texte ou seront vides. Pour plus d'informations, consultez le billet de blog Symlinks in Windows 10 ! .

  Si vous utilisez Git sous Windows, vous devez activer le mode développeur ou vous devez :
  - Définissez `core.symlinks` ce paramètre sur true à l'aide de la commande suivante :

    ```
    git config --global core.symlinks true
    ```
  - Utilisez une console élevée en tant qu'administrateur chaque fois que vous utilisez une commande git qui écrit sur le système (par exemplegit pull,git clone, etgit submodule update --init --recursive).

Configurer votre projet

Pour activer la démo, vous devez configurer votre projet pour qu'il fonctionne avec AWS IoT. Pour configurer votre projet de manière à ce qu'il fonctionne AWS IoT, votre appareil doit être enregistré en tant qu' AWS IoT objet. Vous devez avoir enregistré votre appareil lorsque vous Configuration AWS IoT et Amazon Cognito pour FreeRTOS Bluetooth Low Energy.

Pour configurer votre AWS IoT point de terminaison

1. Connectez-vous à la console AWS IoT.
2. Dans le panneau de navigation, sélectionnez Settings (Paramètres).

   Votre AWS IoT point de terminaison apparaît dans la zone de texte Terminal de données de l'appareil. Elle doit ressembler à *1234567890123*-ats.iot.*us-east-1*.amazonaws.com. Prenez note de ce point de terminaison.
3. Dans le volet de navigation, choisissez Gérer, puis Objets. Notez le nom de l' AWS IoT objet pour votre appareil.
4. Avec votre AWS IoT point de terminaison et le nom de votre AWS IoT objet à portée de main, ouvrez-le *freertos*/demos/include/aws_clientcredential.h dans votre IDE et spécifiez les valeurs #define des constantes suivantes :

   - clientcredentialMQTT_BROKER_ENDPOINT *Your AWS IoT endpoint*
   - clientcredentialIOT_THING_NAME *Your board's AWS IoT thing name*

Pour activer la démonstration

1. Vérifiez que la démonstration GATT Bluetooth Low Energy est activée. Accédez à vendors/nordic/boards/nrf52840-dk/aws_demos/config_files/iot_ble_config.h et ajoutez #define IOT_BLE_ADD_CUSTOM_SERVICES (1) à la liste d'instructions de définition.
2. Ouvrez vendors/nordic/boards/nrf52840-dk/aws_demos/config_files/aws_demo_config.h et définissez l'un ou l'autreCONFIG_OTA_HTTP_BLE_TRANSPORT_DEMO_ENABLED, CONFIG_OTA_MQTT_BLE_TRANSPORT_DEMO_ENABLED ou comme dans cet exemple.

```
/* 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. Puisque la puce Nordic est livrée avec très peu de RAM (250 Ko), il peut être nécessaire de changer la configuration BLE pour autoriser des entrées de table GATT plus importantes par rapport à la taille de chaque attribut. De cette façon, vous pouvez ajuster la quantité de mémoire obtenue par l'application. Pour ce faire, remplacez les définitions des attributs suivants dans le fichier *freertos*`/vendors/nordic/boards/nrf52840-dk/aws_demos/config_files/sdk_config.h` :

   - NRF_SDH_BLE_VS_UUID_COUNT

     Nombre d'UUID spécifiques au fournisseur. Augmentez ce nombre de 1 lorsque vous ajoutez un nouvel UUID spécifique au fournisseur.

   - NRF_SDH_BLE_GATTS_ATTR_TAB_SIZE

     Taille de la table attributaire en octets. La taille doit être un multiple de 4. Cette valeur indique la quantité de mémoire définie dédiée à la table attributaire (y compris la taille caractéristique). Elle varie donc d'un projet à l'autre. Si vous dépassez la taille de la table attributaire, vous recevrez une erreur NRF_ERROR_NO_MEM. Si vous modifiez le NRF_SDH_BLE_GATTS_ATTR_TAB_SIZE, vous devez généralement également reconfigurer les paramètres de la RAM.

   (Pour les tests, l'emplacement du fichier est *freertos*`/vendors/nordic/boards/nrf52840-dk/aws_tests/config_files/sdk_config.h`.)

Créez et exécutez le projet de démonstration FreeRTOS

Après avoir téléchargé FreeRTOS et configuré votre projet de démonstration, vous êtes prêt à créer et exécuter le projet de démonstration sur votre carte.

> **Important**
>
> Si c'est la première fois que vous exécutez la démonstration sur cette carte, vous devez flasher le programme d'amorçage sur la carte avant que la démonstration puisse être exécutée.

Pour générer et flasher le programme d'amorçage, suivez les étapes ci-dessous, mais au lieu d'utiliser le fichier de projet `projects/nordic/nrf52840-dk/ses/aws_demos/aws_demos.emProject`, utilisez `projects/nordic/nrf52840-dk/ses/aws_demos/bootloader/bootloader.emProject`.

Pour créer et exécuter la démo FreeRTOS Bluetooth Low Energy à partir de Segger Embedded Studio

1. Ouvrez Segger Embedded Studio. Dans le menu supérieur, choisissez File, puis Open Solution et accédez au fichier de projet `projects/nordic/nrf52840-dk/ses/aws_demos/aws_demos.emProject`.
2. Si vous utilisez l'émulateur de terminal Segger Embedded Studio, choisissez Tools (Outils) dans le menu supérieur, puis choisissez Terminal Emulator (Émulateur de terminal), Terminal Emulator (Émulateur de terminal) pour afficher les informations de votre connexion série.

   Si vous utilisez un autre outil de terminal, vous pouvez surveiller la sortie de cet outil à partir de votre connexion série.
3. Right-click le projet de `aws_demos` démonstration dans l'explorateur de projets, puis choisissez Build.

   > **Note**
   >
   > S'il s'agit de la première fois que vous utilisez Segger Embedded Studio, vous pouvez voir l'avertissement « No license for commercial use ». Segger Embedded Studio peut être utilisé gratuitement avec les appareils semi-conducteurs Nordic. Demandez une licence gratuite puis, lors de la configuration, choisissez Activer votre licence gratuite et suivez les instructions.
4. Choisissez Debug (Déboguer), puis Go (OK).

   Une fois la démonstration démarrée, elle attend d'être appairée avec un appareil mobile sur Bluetooth Low Energy.
5. Suivez les instructions de l'application de démonstration MQTT over Bluetooth Low Energy pour terminer la démonstration avec l'application de démonstration FreeRTOS Bluetooth Low Energy Mobile SDK en tant que proxy MQTT mobile.

Résolution des problèmes

Pour obtenir des informations générales sur la résolution des problèmes liés à la prise en main de FreeRTOS, consultez. Résolution des problèmes de mise en route

Commencer à utiliser le Nuvoton NuMaker-IoT-M487

> **Important**
>
> Cette intégration de référence est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Ce didacticiel fournit des instructions pour démarrer avec la carte de NuMaker-IoT-M487 développement Nuvoton. Le microcontrôleur de série comprend un Ethernet RJ45 intégré et Wi-Fi des modules. Si vous n'avez pas le Nuvoton NuMaker-IoT-M487, consultez le catalogue des appareils AWS partenaires pour en acheter un auprès de notre partenaire.

Avant de commencer, vous devez configurer AWS IoT et configurer votre logiciel FreeRTOS pour connecter votre carte de développement au Cloud. AWS Pour obtenir des instructions, veuillez consulter Premiers pas. Dans ce didacticiel, le chemin d'accès au répertoire de téléchargement de FreeRTOS est appelé. *freertos*

Présentation de

Ce didacticiel vous guide à travers les étapes suivantes :

1. Installation du logiciel sur voter machine hôte pour développer et déboguer des applications intégrées pour votre carte de microcontrôleur.
2. Cross-compile une application de démonstration FreeRTOS sur une image binaire.
3. Chargement de l'image binaire de l'application dans votre carte et exécution de l'application.

Configurer votre environnement de développement.

L'édition Keil MDK Nuvoton est conçue pour développer et déboguer des applications pour les cartes Nuvoton M487. La version Keil MDK v5 Essential, Plus ou Pro devrait également fonctionner pour

le microcontrôleur Nuvoton M487 (core). Cortex-M4 Vous pouvez télécharger l'édition Keil MDK Nuvoton avec une réduction de prix pour les microcontrôleurs de la série Nuvoton. Cortex-M4 Le kit de développement Keil MDK est pris en charge uniquement sur Windows.

Pour installer l'outil de développement pour NuMaker-IoT-M487

1. Téléchargez le kit de développement MDK Keil Nuvoton Edition sur le site web de Keil MDK.
2. Installez le kit de développement Keil MDK sur votre machine hôte à l'aide de votre licence. Le Keil MDK inclut l'IDE Keil µVision, une chaîne d'outils de compilation C/C ++ et le débogueur µVision.

   Si vous rencontrez des problèmes lors de l'installation, contactez Nuvoton pour obtenir de l'aide.
3. Installez le Nu-Link_ Keil_Driver_V 3.06.7215r (ou la dernière version), qui se trouve sur la page de l'outil de développement Nuvoton.

Créez et exécutez le projet de démonstration FreeRTOS

Pour créer le projet de démonstration FreeRTOS

1. Ouvrez l'IDE Keil µVision.
2. Dans le menu File (Fichier), choisissez Open (Ouvrir). Dans la boîte de dialogue Open file (Ouvrir un fichier) assurez-vous que le sélecteur de type de fichier est défini sur Project Files (Fichiers de projet).
3. Choisissez soit le projet de démonstration, Wi-Fi soit le projet de démonstration Ethernet à créer.

   - Pour ouvrir le projet de Wi-Fi démonstration, choisissez le projet cible `aws_demos.uvproj` dans le *`freertos`*`\projects\nuvoton\numaker_iot_m487_wifi\uvision\aws_demos` répertoire.
   - Pour ouvrir le projet de démonstration Ethernet, choisissez le projet cible `aws_demos_eth.uvproj` dans le répertoire *`freertos`*`\projects\nuvoton\numaker_iot_m487_wifi\uvision\aws_demos_eth`.
4. Pour vous assurer que vos paramètres sont corrects pour flasher la carte, cliquez avec le bouton droit sur le projet `aws_demo` dans l'IDE, puis choisissez Options. (Pour plus d'informations, consultez Résolution des problèmes.)
5. Dans l'onglet Utilitaires, vérifiez que l'option Utiliser le pilote cible pour la programmation Flash est sélectionnée et que Nuvoton Nu-Link Debugger est défini comme pilote cible.

6. Dans l'onglet Debug, à côté de Nuvoton Nu-Link Debugger, choisissez Settings.
7. Vérifiez que Chip Type (Type de puce) est défini sur M480.
8. Dans le volet de navigation Project (Projet) de l'IDE Keil µVision, choisissez le projet `aws_demos`. Dans le menu Project (Projet), choisissez Build Target (Générer la cible).

Vous pouvez utiliser le client MQTT de la AWS IoT console pour surveiller les messages que votre appareil envoie au AWS Cloud.

Pour vous abonner à la rubrique MQTT avec le AWS IoT Client MQTT

1. Connectez-vous à la console AWS IoT.
2. Dans le volet de navigation, choisissez Test, puis choisissez MQTT test client pour ouvrir le client MQTT.
3. Dans le champ Rubrique d'abonnement, saisissez ***your-thing-name*/example/topic**, puis choisissez S'abonner à la rubrique.

Pour exécuter le projet de démonstration FreeRTOS

1. Connectez votre Numaker-IoT-M487 carte à votre machine hôte (ordinateur).
2. Regénérez le projet.
3. Dans le menu Flash de l'IDE Keil µVision, choisissez Download (Télécharger).
4. Dans le menu Debug, choisissez Start/Stop Debug Session.
5. Lorsque le débogueur stoppe au point d'arrêt dans `main()`, ouvrez le menu Run (Exécuter), puis choisissez Run (F5) [Exécuter (F5)].

   Vous devriez voir les messages MQTT envoyés par votre appareil dans le client MQTT de la AWS IoT console.

Utilisation de CMake avec FreeRTOS

Vous pouvez également utiliser CMake pour créer et exécuter les applications de démonstration FreeRTOS ou les applications que vous avez développées à l'aide d'éditeurs de code et d'outils de débogage tiers.

Assurez-vous d'avoir installé le système de génération CMake. Suivez les instructions fournies dans Utilisation de CMake avec FreeRTOS, puis suivez la procédure décrite dans cette rubrique.

> **Note**
>
> Assurez-vous que le chemin d'accès à l'emplacement du compilateur (Keil) se trouve dans votre variable système Path, par exemple, `C:\Keil_v5\ARM\ARMCC\bin`.

Vous pouvez également utiliser le client MQTT dans la AWS IoT console pour surveiller les messages que votre appareil envoie au AWS Cloud.

Pour vous abonner à la rubrique MQTT avec le AWS IoT Client MQTT

1. Connectez-vous à la console AWS IoT.
2. Dans le volet de navigation, choisissez Test, puis choisissez MQTT test client pour ouvrir le client MQTT.
3. Dans le champ Rubrique d'abonnement, saisissez ***your-thing-name*****/example/topic**, puis choisissez S'abonner à la rubrique.

Pour générer des fichiers de génération à partir de fichiers source et exécuter le projet de démonstration

1. Sur votre machine hôte, ouvrez l'invite de commande et accédez au dossier *freertos*.
2. Créez un dossier qui contiendra les fichiers générés. Nous appellerons ce dossier le*BUILD_FOLDER*.
3. Générez les fichiers de compilation pour la démo Wi-Fi ou pour la démo Ethernet.
   - Pour Wi-Fi :

     Accédez au répertoire qui contient les fichiers sources du projet de démonstration FreeRTOS. Ensuite, générez les fichiers de build en exécutant la commande suivante.

     ```
     cmake -DVENDOR=nuvoton -DBOARD=numaker_iot_m487_wifi -DCOMPILER=arm-keil -S . -
     B BUILD_FOLDER -G Ninja
     ```

   - Pour la démonstration Ethernet :

     Accédez au répertoire qui contient les fichiers sources du projet de démonstration FreeRTOS. Ensuite, générez les fichiers de build en exécutant la commande suivante.

```
cmake -DVENDOR=nuvoton -DBOARD=numaker_iot_m487_wifi -DCOMPILER=arm-keil -
DAFR_ENABLE_ETH=1 -S . -B BUILD_FOLDER -G Ninja
```

4. Générez le fichier binaire à flasher sur la carte M487 en exécutant la commande suivante.

   ```
   cmake --build BUILD_FOLDER
   ```

   À ce stade, le fichier binaire `aws_demos.bin` doit se trouver dans le dossier *BUILD_FOLDER*`/vendors/Nuvoton/boards/numaker_iot_m487_wifi`.

5. Pour configurer la carte en mode clignotant, assurez-vous que le commutateur MSG (No.4 de l'ISW1 sur ICE) est activé. Lorsque vous branchez la carte, une fenêtre et une unité lui sont attribuées. (Consultez Résolution des problèmes.)

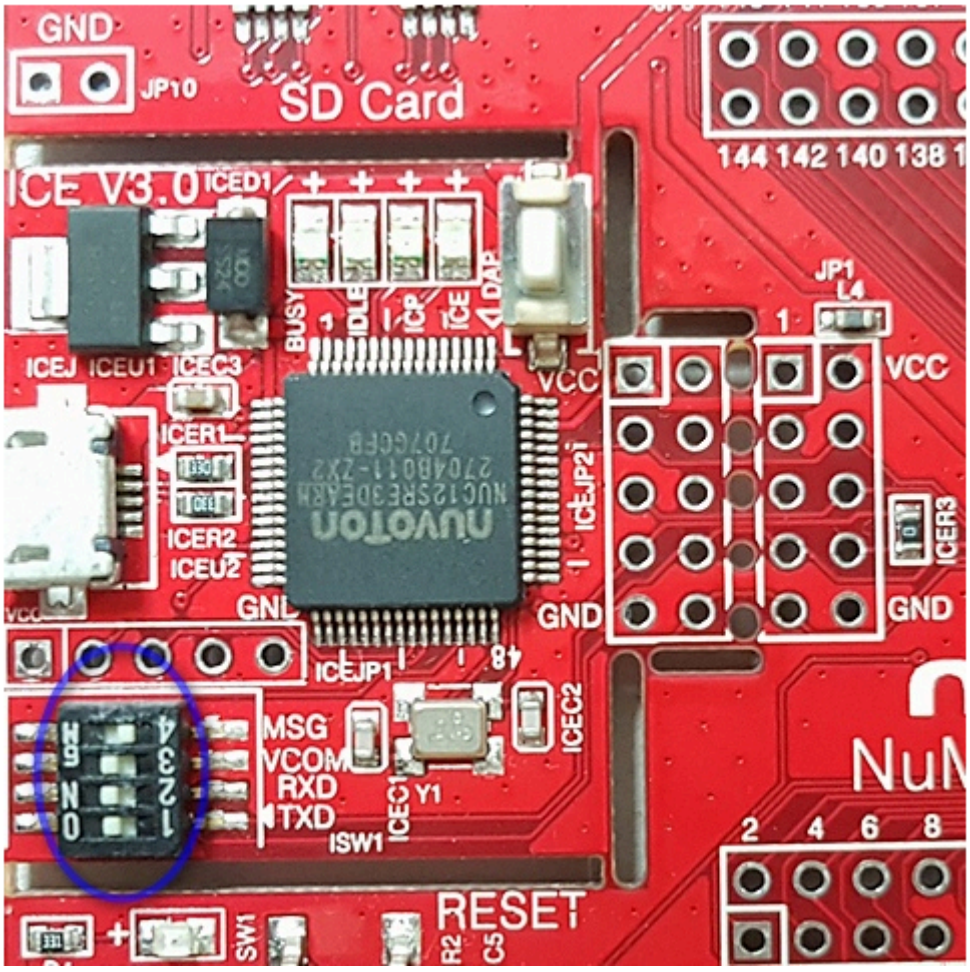
6. Ouvrez un émulateur de terminal pour afficher les messages via UART. Suivez les instructions décrites dans Installation d'un émulateur de terminal.

7. Exécutez le projet de démonstration en copiant le fichier binaire généré sur le périphérique.

   Si vous vous êtes abonné à la rubrique MQTT avec le client AWS IoT MQTT, les messages MQTT envoyés par votre appareil devraient s'afficher dans la console. AWS IoT

Résolution des problèmes

- Si votre Windows ne reconnaît pas le périphériqueVCOM, installez le pilote du port série NuMaker Windows à partir du lien Nu-Link USB Driver v1.6.
- Si vous connectez votre appareil au Keil MDK (IDE) via Nu-Link, assurez-vous que le commutateur MSG (No.4de l'ISW1 sur ICE) est éteint, comme indiqué.



Si vous rencontrez des problèmes lors de la configuration de votre environnement de développement ou de la connexion à votre carte, contactez Nuvoton.

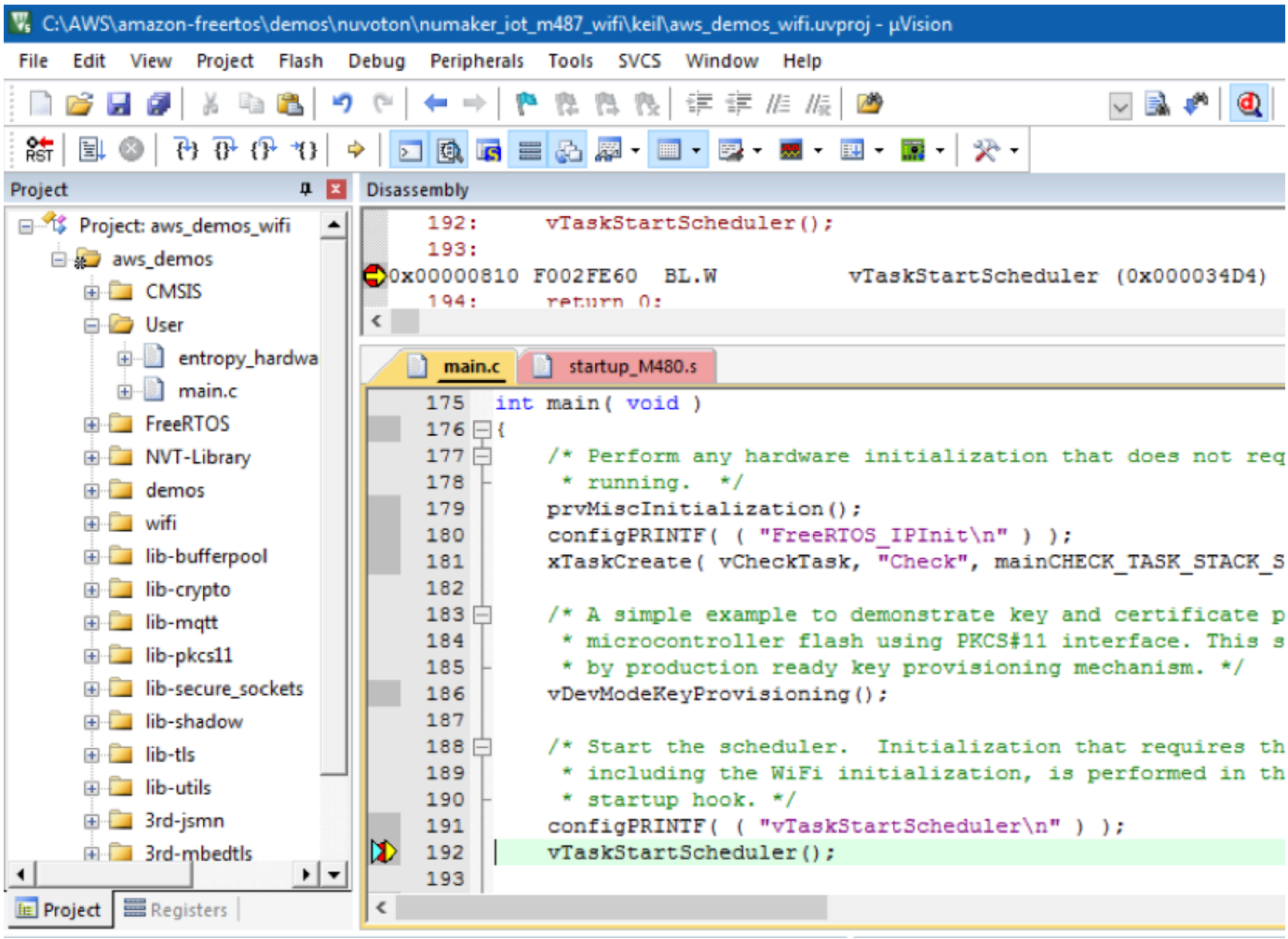
Débogage de projets FreeRTOS dans Keil µVision

Pour démarrer une session de débogage dans Keil µVision

1. Ouvrez Keil µVision.
2. Suivez les étapes pour intégrer le projet de démonstration FreeRTOS. Créez et exécutez le projet de démonstration FreeRTOS
3. Dans le menu Debug, choisissez Start/Stop Debug Session.

   La fenêtre Call Stack+Locals s'affiche lorsque vous démarrez une session de débogage. µVision flashe la démonstration sur la carte, exécute la démonstration et s'arrête au début de la fonction `main()`.

4. Définissez des points d'arrêt dans le code source de votre projet, puis exécutez le code. Le projet doit ressembler à ce qui suit :

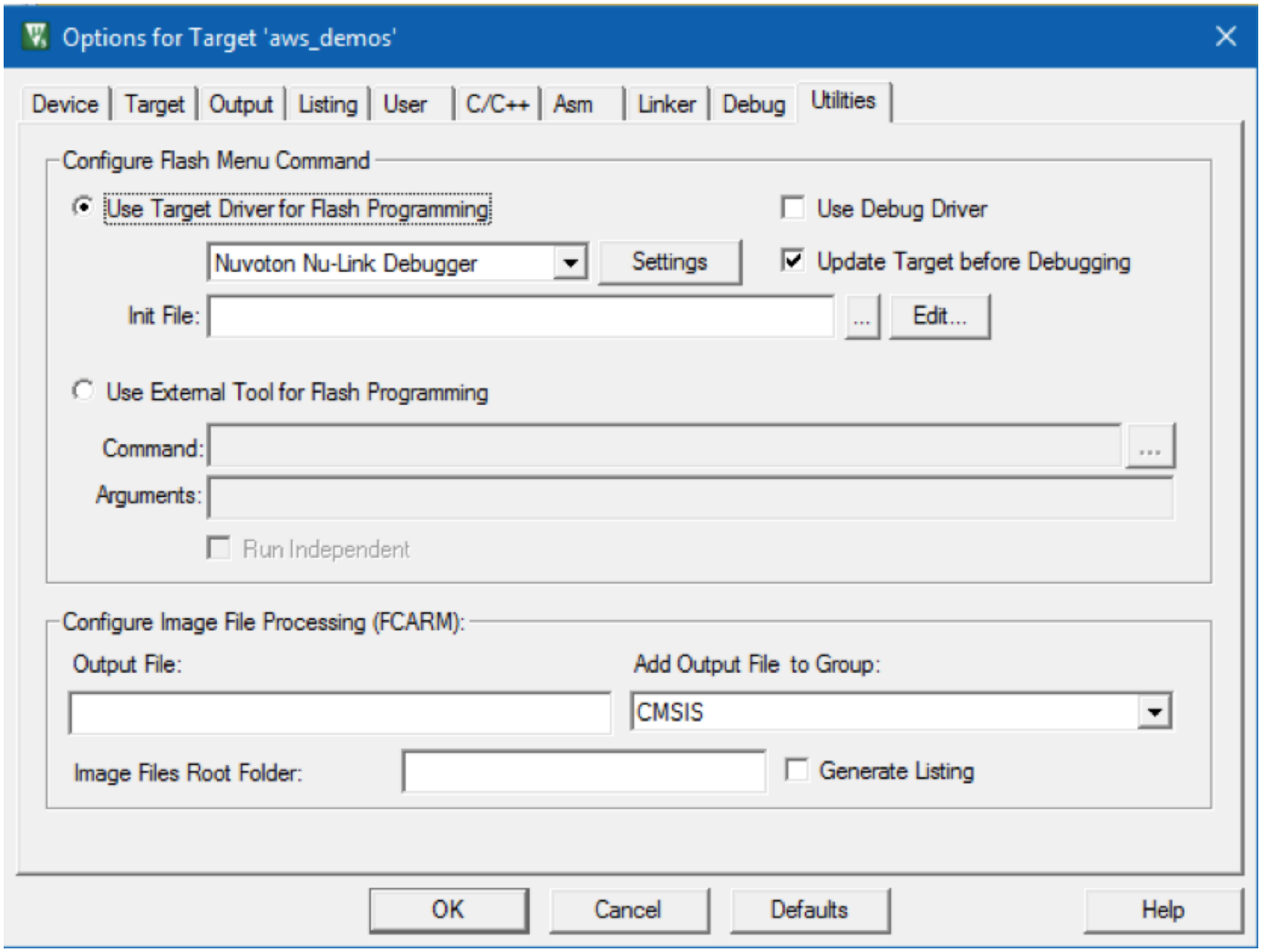


Dépannage des paramètres de débogage µVision

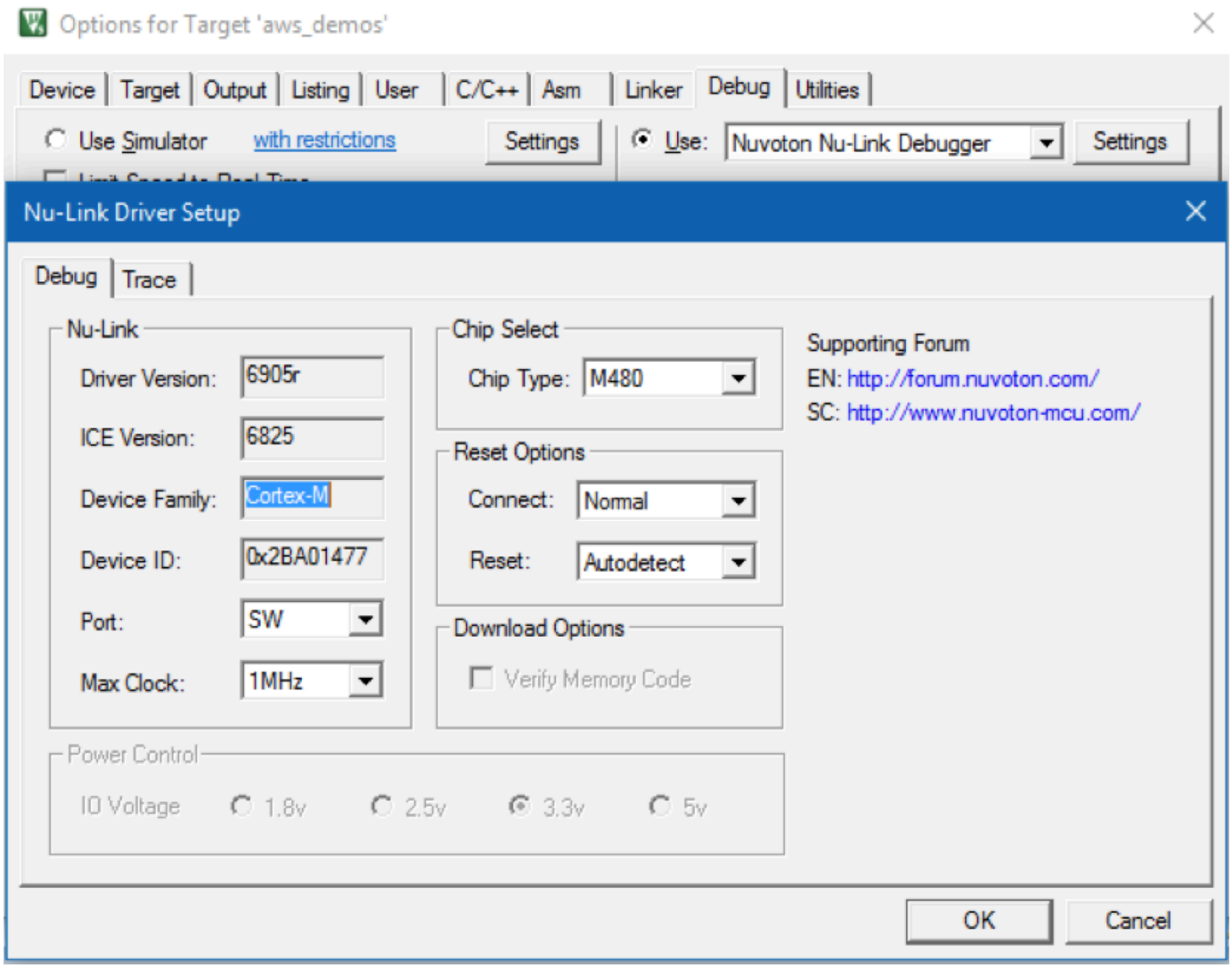
Si vous rencontrez des problèmes lors du débogage d'une application, vérifiez que vos paramètres de débogage sont définis correctement dans Keil µVision.

Pour vérifier que les paramètres de débogage µVision sont corrects

1. Ouvrez Keil µVision.
2. Right-click le `aws_demo` projet dans l'IDE, puis choisissez Options.
3. Dans l'onglet Utilitaires, vérifiez que l'option Utiliser le pilote cible pour la programmation Flash est sélectionnée et que Nuvoton Nu-Link Debugger est défini comme pilote cible.



4. Dans l'onglet Debug, à côté de Nuvoton Nu-Link Debugger, choisissez Settings.



5. Vérifiez que Chip Type (Type de puce) est défini sur M480.

Démarrez avec le NXP LPC54018 IoT Module

> **Important**
>
> Cette intégration de référence est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Ce didacticiel fournit des instructions sur la mise en route du module IoT NXP LPC54018. Si vous ne possédez pas de module IoT NXP LPC54018, consultez AWS le catalogue des appareils partenaires pour en acheter un auprès de notre partenaire. Utilisez un câble USB pour connecter votre module IoT NXP LPC54018 à votre ordinateur.

Avant de commencer, vous devez configurer AWS IoT et télécharger FreeRTOS pour connecter votre appareil au Cloud. AWS Pour obtenir des instructions, consultez Premiers pas. Dans ce didacticiel, le chemin d'accès au répertoire de téléchargement de FreeRTOS est appelé. *freertos*

Présentation de

Ce didacticiel comprend les instructions de mise en route suivantes :

1. Connexion de votre carte à un appareil hôte.
2. Installation de logiciels sur la machine hôte pour développer et déboguer des applications intégrées pour votre carte de microcontrôleur.
3. Compilation croisée d'une application de démonstration FreeRTOS en une image binaire.
4. Chargement de l'image binaire de l'application dans votre carte et exécution de l'application.

Configurer le matériel NXP

Pour configurer le NXP LPC54018

- Connectez votre ordinateur au port USB sur le NXP LPC54018.

Pour configurer le débogueur JTAG

Vous avez besoin d'un débogueur JTAG pour lancer et déboguer votre code exécuté sur la carte NXP LPC54018. FreeRTOS a été testé à l'aide d'un module IoT OM40006. Pour plus d'informations sur les débogueurs pris en charge, consultez le Manuel d'utilisation pour le module IoT NXP LPC54018 disponible à partir de la page du produit OM40007 LPC54018 IoT Module.

1. Si vous utilisez un débogueur OM40006 IoT Module, vous avez besoin d'un câble pour connecter le connecteur 20 broches entre le débogueur et le connecteur 10 broches du module NXP IoT.
2. Connectez NXP LPC54018 et le débogueur OM40006 IoT Module aux ports USB de votre ordinateur à l'aide de câbles mini-USB vers USB.

Configurer votre environnement de développement.

FreeRTOS prend en charge deux IDE pour le module IoT NXP LPC54018 : IAR Embedded Workbench et mCUXpresso.

Avant de commencer, installez l'un de ces environnements IDE.

Pour installer IAR Embedded Workbench for ARM

1. Accédez à IAR Embedded Workbench for ARM et téléchargez le logiciel.

   > **Note**
   >
   > IAR Embedded Workbench for ARM nécessite Microsoft Windows.

2. Lancez le programme d'installation et suivez les instructions.
3. Dans Assistant de licence, choisissez Register with IAR Systems to get an evaluation license (S'enregistrer auprès d'IAR Systems pour obtenir une licence d'évaluation).
4. Placez le chargeur de démarrage sur l'appareil avant de tenter d'exécuter toute démonstration.

Pour installer MCUXpresso depuis NXP

1. Téléchargez et exécutez le programme d'installation MCUXpresso à partir de NXP.

   > **Note**
   >
   > Les versions 10.3.x et ultérieures sont prises en charge.

2. Accédez à MCUXpresso SDK et choisissez Build your SDK (Créer votre kit SDK).

   > **Note**
   >
   > Les versions 2.5 et ultérieures sont prises en charge.

3. Choisissez Select Development Board (Sélectionner la carte de développement).
4. Sous Select Development Board (Sélectionner la carte de développement), dans le champ Search by Name (Recherche par nom), saisissez **`LPC54018-IoT-Module`**.
5. Sous Tableaux, sélectionnez LPC54018-IoT-Module.

6. Vérifiez les détails du matériel, puis choisissez Build MCUXepresso SDK.
7. Le kit SDK pour Windows utilisant l'IDE MCUXpresso est déjà créé. Choisissez Télécharger le rapport SDK. Si vous utilisez un autre système d'exploitation, sous Host OS (Système d'exploitation hôte), sélectionnez votre système d'exploitation, puis choisissez Download SDK (Télécharger le kit SDK).
8. Démarrez l'IDE MCUXpresso kits et choisissez l'onglet Installed SDKs (Kits SDK installés).
9. Faites glisser le kit SDK téléchargé et déposez-le dans la fenêtre Installed SDKs (Kits SDK installés).

Si vous rencontrez des problèmes lors de l'installation, consultez NXP Support (Support NXP) ou NXP Ressources Resources (Ressources pour développeurs NXP).

Surveillance des messages MQTT dans le cloud

Avant de lancer le projet de démonstration FreeRTOS, vous pouvez configurer le client MQTT dans la console pour surveiller AWS IoT les messages que votre appareil envoie au Cloud. AWS

Pour vous abonner à la rubrique MQTT avec le AWS IoT Client MQTT

1. Connectez-vous à la console AWS IoT.
2. Dans le volet de navigation, choisissez Test, puis choisissez MQTT test client pour ouvrir le client MQTT.
3. Dans le champ Rubrique d'abonnement, saisissez ***your-thing-name*****/example/topic**, puis choisissez S'abonner à la rubrique.

Lorsque le projet de démonstration s'exécute avec succès sur votre appareil, le message « Hello World ! » s'affiche envoyé plusieurs fois au sujet auquel vous vous êtes abonné.

Créez et exécutez le projet de démonstration FreeRTOS

Importez la démo de FreeRTOS dans votre IDE

Pour importer l'exemple de code FreeRTOS dans l'IDE IAR Embedded Workbench

1. Ouvrez IAR Embedded Workbench, et dans le menu File (Fichier), choisissez Open Workspace (Ouvrir l'espace de travail).
2. Dans la zone de texte search-directory, saisissez `projects/nxp/lpc54018iotmodule/iar/aws_demos`, puis choisissez aws_demos.eww.

3. Dans le menu Project (Projet), choisissez Rebuild All (Regénérer tout).

Pour importer l'exemple de code FreeRTOS dans l'IDE MCUXpresso

1. Ouvrez MCUXpresso, et dans le menu File (Fichier), choisissez Open Projects From File System (Ouvrir les projets depuis le système de fichiers).
2. Dans la zone de texte Directory, entrez `projects/nxp/lpc54018iotmodule/mcuxpresso/aws_demos`, puis choisissez Finish
3. Dans le menu Project (Projet), choisissez Build All (Générer tout).

Exécutez le projet de démonstration FreeRTOS

Pour exécuter le projet de démonstration FreeRTOS avec l'IDE IAR Embedded Workbench

1. Dans votre IDE, dans le menu Project (Projet), choisissez Build (Générer).
2. Dans le menu Project (Projet), choisissez Download et Debug (Télécharger et déboguer).
3. Dans le menu Debug (Déboguer), choisissez Start Debugging (Démarrer le débogage).
4. Lorsque le débogueur s'arrête au point d'arrêt dans `main`, dans le menu Debug (Débogage), choisissez Go (OK).

> **Note**
>
> Si une boîte de dialogue de sélection de J-Link périphériques s'ouvre, cliquez sur OK pour continuer. Dans la boîte de dialogue Paramètres du périphérique cible, choisissez Non spécifié Cortex-M4, puis cliquez sur OK. Vous n'avez besoin de le faire qu'une seule fois.

Pour exécuter le projet de démonstration FreeRTOS avec l'IDE mCUXpresso

1. Dans votre IDE, dans le menu Project (Projet), choisissez Build (Générer).
2. Si c'est la première fois que vous déboguez, choisissez le projet `aws_demos` et dans la barre d'outils Debug, cliquez sur le bouton bleu de débogage.
3. Toutes les sondes de débogage détectées sont affichées. Choisissez la sonde que vous souhaitez utiliser, puis choisissez OK pour démarrer le débogage.

> **Note**
>
> Lorsque le débogueur s'arrête au point d'arrêt dans `main()`, appuyez sur le bouton de redémarrage du débogage
>
> 
>
> une fois afin de réinitialiser la session de débogage. (Ceci est nécessaire en raison d'un bogue avec le débogueur MCUXpresso pour). NXP54018-IoT-Module

4. Lorsque le débogueur s'arrête au point d'arrêt dans `main()`, dans le menu Debug (Débogage), choisissez Go (OK).

Résolution des problèmes

Pour obtenir des informations générales sur la résolution des problèmes liés à la prise en main de FreeRTOS, consultez. Résolution des problèmes de mise en route

Commencer à utiliser le Renesas Starter Kit+ pour RX65N-2MB

> **Important**
>
> Cette intégration de référence est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Ce didacticiel fournit des instructions pour démarrer avec le Renesas Starter Kit+ pour. RX65N-2MB Si vous ne possédez pas le Renesas RSK+ pour RX65N-2MB, consultez le catalogue des appareils AWS partenaires et achetez-en un auprès de nos partenaires.

Avant de commencer, vous devez configurer AWS IoT et télécharger FreeRTOS pour connecter votre appareil au Cloud. AWS Pour obtenir des instructions, consultez Premiers pas. Dans ce didacticiel, le chemin d'accès au répertoire de téléchargement de FreeRTOS est appelé. *`freertos`*

Présentation de

Ce didacticiel comprend les instructions de mise en route suivantes :

1. Connexion de votre carte à un appareil hôte.
2. Installation de logiciels sur la machine hôte pour développer et déboguer des applications intégrées pour votre carte de microcontrôleur.
3. Compilation croisée d'une application de démonstration FreeRTOS en une image binaire.
4. Chargement de l’image binaire de l’application dans votre carte et exécution de l’application.

Configurer le matériel Renesas

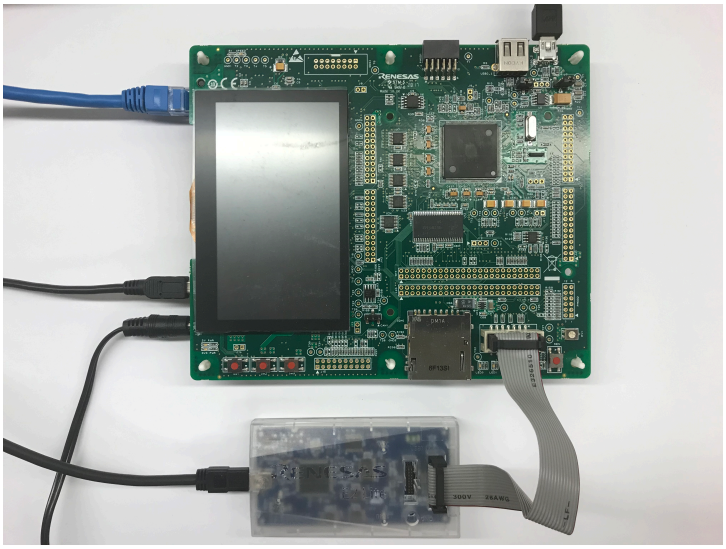
Pour configurer le RSK+ pour RX65N-2MB

1. Connectez l'adaptateur secteur +5V positif au connecteur PWR du RSK+ pour. RX65N-2MB
2. Connectez votre ordinateur au port USB2.0 FS du RSK+ pour. RX65N-2MB
3. Connectez votre ordinateur au USB-to-serial port du RSK+ pour. RX65N-2MB
4. Connectez un routeur ou un port Ethernet connecté à Internet au port Ethernet du RSK+ pour. RX65N-2MB

Pour configurer le module E2 Lite Debugger

1. Utilisez le câble plat à 14 broches pour connecter le module E2 Lite Debugger au port « E1/E2 Lite » du RSK+ pour. RX65N-2MB
2. Utilisez un câble USB pour connecter le module E2 Lite Debugger à votre machine hôte. Lorsque le module E2 Lite Debugger est connecté à la fois à la carte et à votre ordinateur, un voyant LED vert « ACT » clignote sur le débogueur.
3. Une fois le débogueur connecté à votre machine hôte et à RSK+ for RX65N-2MB, les pilotes du débogueur E2 Lite commencent à être installés.

   Notez que les privilèges d'administrateur sont requis pour installer les pilotes.



Configurer votre environnement de développement.

Pour configurer les configurations FreeRTOS pour le RSK+ RX65N-2MB pour, utilisez l'IDE et le compilateur Renesas e 2 studio. CC-RX

> **Note**
>
> L'IDE et le CC-RX compilateur Renesas e 2 studio ne sont pris en charge que sur les systèmes d'exploitation Windows 7, 8 et 10.

Pour télécharger et installer e2studio

1. Accédez à la page de téléchargement du programme d'installation du studio Renesas e 2 et téléchargez le programme d'installation hors ligne.
2. Vous êtes dirigé vers une page de connexion Renesas.

   Si vous avez un compte chez Renesas, entrez vos informations de connexion, puis choisissez Login.

   Si vous n'avez pas de compte, choisissez Register now (S'inscrire maintenant) et suivez les premières étapes d'inscription. Vous devez recevoir un e-mail contenant un lien d'activation de votre compte Renesas. Suivez ce lien pour finaliser votre inscription à Renesas, puis connectez-vous à Renesas.
3. Une fois connecté, téléchargez le programme d'installation e2studio sur votre ordinateur.
4. Ouvrez le programme d'installation et suivez les étapes jusqu'à la fin.

Pour plus d'informations, consultez le studio e 2 sur le site Web de Renesas.

Pour télécharger et installer le package de compilation RX Family C/C ++

1. Accédez à la page de téléchargement du package de compilation RX Family C/C ++, puis téléchargez le V3.00.00 package.
2. Ouvrez le fichier exécutable et installez le compilateur.

Pour plus d'informations, consultez le package de compilation C/C ++ pour la famille RX sur le site Web de Renesas.

> **Note**
>
> La version d'évaluation du compilateur est disponible gratuitement et valide pendant 60 jours. Le 61e jour, vous devez obtenir une clé de licence. Pour en savoir plus, consultez la page Evaluation Software Tools.

Créez et exécutez des exemples de FreeRTOS

Une fois que vous avez configuré le projet de démonstration, vous pouvez le créer et l'exécuter sur votre carte.

Créez la démo de FreeRTOS dans e^{2}studio

Pour importer et créer la démo dans e^{2}studio

1. Lancez e^{2}studio à partir du menu Démarrer.
2. Dans la fenêtre Select a directory as a workspace (Sélectionner un annuaire en tant qu'espace de travail), recherchez le dossier dans lequel vous souhaitez travailler, puis sélectionnez Launch (Lancer).
3. La première fois que vous ouvrez e^{2}studio, la fenêtre Toolchain Registry (Registre de la chaîne d'outils) s'ouvre. Choisissez Renesas Toolchains (Chaînes d'outils Renesas) et vérifiez que **`CC-RX v3.00.00`** est sélectionné. Choisissez Register (S'inscrire), puis OK.
4. Si vous ouvrez e^{2}studio pour la première fois, la fenêtre Code Generator Registration (Inscription Code Generator) s'affiche. Choisissez OK.
5. La fenêtre Code Generator COM component register (Registre du composant COM Code Generator) s'affiche. Sous Please restart e^{2}studio to use Code Generator, sélectionnez OK.

6. La fenêtre Restart e^{2}studio s'affiche. Choisissez OK.
7. e^{2}studio redémarre. Dans la fenêtre Select a directory as a workspace (Sélectionner un annuaire en tant qu'espace de travail), sélectionnez Launch (Lancer).
8. Dans l'écran d'accueil e^{2}studio, sélectionnez l'icône en forme de flèche Go to the e^{2}studio workbench.
9. Right-click dans la fenêtre de l'Explorateur de projets, puis choisissez Importer.
10. Dans l'assistant d'importation, sélectionnez Général, choisissez Existing Projects into Workspace (Projets existants dans l'espace de travail), puis choisissez Next (Suivant).
11. Choisissez Browse (Parcourir), localisez l'annuaire `projects/renesas/rx65n-rsk/e2studio/aws_demos`, puis choisissez Finish (Terminer).
12. Dans le menu Project (Projet), choisissez Project (Projet), puis Build All (Générer tout).

    La console de génération envoie un message avertissant que le programme License Manager n'est pas installé. Vous pouvez ignorer ce message sauf si vous disposez d'une clé de licence pour le CC-RX compilateur. Pour installer le programme License Manager, consultez la page de téléchargement License Manager.

Surveillance des messages MQTT dans le cloud

Avant de lancer le projet de démonstration FreeRTOS, vous pouvez configurer le client MQTT dans la console pour surveiller AWS IoT les messages que votre appareil envoie au Cloud. AWS

Pour vous abonner à la rubrique MQTT avec le AWS IoT Client MQTT

1. Connectez-vous à la console AWS IoT.
2. Dans le volet de navigation, choisissez Test, puis choisissez MQTT test client pour ouvrir le client MQTT.
3. Dans le champ Rubrique d'abonnement, saisissez ***your-thing-name*****/example/topic**, puis choisissez S'abonner à la rubrique.

Lorsque le projet de démonstration s'exécute avec succès sur votre appareil, le message « Hello World ! » s'affiche envoyé plusieurs fois au sujet auquel vous vous êtes abonné.

Exécutez le projet FreeRTOS

Pour exécuter le projet dans e^{2}studio

1. Vérifiez que vous avez connecté le module E2 Lite Debugger à votre RSK+ pour RX65N-2MB
2. Dans le menu supérieur, choisissez Run (Exécuter), puis Debug Configuration (Configuration de débogage).
3. Développez Renesas GDB Hardware Debugging et choisissez aws_demos. HardwareDebug
4. Choisissez l'onglet Debugger, puis choisissez l'onglet Connection Settings (Paramètres de connexion). Vérifiez que vos paramètres de connexion sont corrects.
5. Choisissez Debug (Débogage) pour télécharger le code sur votre carte et commencer le débogage.

   Vous pouvez recevoir un avertissement de pare-feu pour `e2-server-gdb.exe`. Cochez Private networks, such as my home or work network (Réseaux privés, tels que mon réseau domestique ou de travail), puis choisissez Allow access (Autoriser l'accès).
6. e^{2}studio peut demander que Renesas Debug Perspective (Perspective de débogage de Renesas) soit sélectionné. Choisissez Yes (Oui).

   Le voyant LED vert « ACT » sur l'E2 Lite Debugger s'allume.
7. Une fois le code téléchargé sur la carte, choisissez Resume (Reprendre) pour exécuter le code jusqu'à la première ligne de la fonction principale. Choisissez Resume (Reprendre) à nouveau pour exécuter le reste du code.

Pour les derniers projets publiés par Renesas, consultez le `renesas-rx` fork du `amazon-freertos` référentiel sur. GitHub

Résolution des problèmes

Pour obtenir des informations générales sur la résolution des problèmes liés à la prise en main de FreeRTOS, consultez. Résolution des problèmes de mise en route

Mise en route avec la carte STMicroelectronics STM32L4 Discovery Kit IoT Node

> **Important**
>
> Cette intégration de référence est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau

projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Ce didacticiel fournit des instructions sur la mise en route de la carte STMicroelectronics STM32L4 Discovery Kit IoT Node. Si vous ne possédez pas encore le nœud IoT STM32L4 Discovery Kit de STMicroelectronics, consultez AWS le catalogue des appareils partenaires pour en acheter un auprès de notre partenaire.

Assurez-vous d'avoir installé le dernier Wi-Fi microprogramme. Pour télécharger le dernier Wi-Fi firmware, consultez le nœud IoT du kit Discovery STM32L4, sans fil basse consommation, Bluetooth Low Energy, NFC, subGHz,. Wi-Fi Sous Ressources binaires, choisissez la mise à jour du microprogramme du Wi-Fi module Inventek ISM 43362 (lisez le fichier readme pour obtenir des instructions).

Avant de commencer, vous devez configurer AWS IoT votre FreeRTOS Wi-Fi et connecter votre appareil au Cloud. AWS Pour obtenir des instructions, consultez Premiers pas. Dans ce didacticiel, le chemin d'accès au répertoire de téléchargement de FreeRTOS est appelé. *freertos*

Présentation de

Ce didacticiel comprend les instructions de mise en route suivantes :

1. Installation de logiciels sur la machine hôte pour développer et déboguer des applications intégrées pour votre carte de microcontrôleur.
2. Compilation croisée d'une application de démonstration FreeRTOS en une image binaire.
3. Chargement de l'image binaire de l'application dans votre carte et exécution de l'application.

Configurer votre environnement de développement.

Installer System Workbench for STM32

1. naviguez jusqu'à OpenSTM32.org.
2. Inscrivez-vous sur la page web OpenSTM32. Vous devez vous connecter pour télécharger System Workbench.
3. Accédez à System Workbench for STM32 installer pour télécharger et installer System Workbench.

Si vous rencontrez des problèmes lors de l'installation, consultez les FAQ sur le site Web System Workbench.

Créez et exécutez le projet de démonstration FreeRTOS

Importez la démo de FreeRTOS dans le STM32 System Workbench

1. Ouvrez le STM32 System Workbench et saisissez le nom d'un nouvel espace de travail.
2. Dans le menu File (Fichier), choisissez Import (Importer). Développez General (Général), choisissez Existing Projects into Workspace (Projets existants dans l'espace de travail), puis choisissez Next (Suivant).
3. Dans Select Root Directory, entrez `projects/st/stm32l475_discovery/ac6/aws_demos`.
4. Le projet `aws_demos` doit être sélectionné par défaut.
5. Choisissez Finish (Terminer) pour importer le projet dans STM32 System Workbench.
6. Dans le menu Project (Projet), choisissez Build All (Générer tout). Confirmez que le projet se compile sans erreurs.

Surveillance des messages MQTT dans le cloud

Avant de lancer le projet de démonstration FreeRTOS, vous pouvez configurer le client MQTT dans la console pour surveiller AWS IoT les messages que votre appareil envoie au Cloud. AWS

Pour vous abonner à la rubrique MQTT avec AWS IoT Client MQTT

1. Connectez-vous à la console AWS IoT.
2. Dans le volet de navigation, choisissez Test, puis choisissez MQTT test client pour ouvrir le client MQTT.
3. Dans le champ Rubrique d'abonnement, saisissez ***your-thing-name*****/example/topic**, puis choisissez S'abonner à la rubrique.

Lorsque le projet de démonstration s'exécute avec succès sur votre appareil, le message « Hello World ! » s'affiche envoyé plusieurs fois au sujet auquel vous vous êtes abonné.

Exécutez le projet de démonstration FreeRTOS

1. Utilisez un câble USB pour connecter votre STMicroelectronics STM32L4 Discovery Kit IoT Node à votre ordinateur. (Consultez la documentation du fabricant fournie avec votre carte pour connaître le port USB approprié à utiliser.)

2. Dans l'Explorateur de projets`aws_demos`, cliquez avec le bouton droit de la souris, choisissez Debug As, puis choisissez Ac6 C/C STM32 ++ Application.

   Si une erreur de débogage se produit la première fois où une session de débogage est lancée, procédez comme suit :

   1. Dans STM32 System Workbench, dans le menu Run (Exécuter), choisissez Debug Configurations.
   2. Choisissez aws_demos Debug. (Vous devrez peut-être développer Ac6 STM32 Debugging.)
   3. Choisissez l'onglet Debugger (Débogueur).
   4. Dans Configuration Script (Script de configuration), choisissez Show Generator Options (Afficher les options du générateur).
   5. Dans Mode Setup (Configuration du mode), définissez Reset Mode (Mode de réinitialisation) sur Software System Reset (Réinitialisation système du logiciel). Choisissez Apply (Appliquer), puis Debug (Déboguer).
3. Lorsque le débogueur s'arrête au point d'arrêt dans `main()`, dans le menu Run (Exécuter), choisissez Resume (Reprendre).

Utilisation de CMake avec FreeRTOS

Si vous préférez ne pas utiliser d'IDE pour le développement de FreeRTOS, vous pouvez également utiliser CMake pour créer et exécuter les applications de démonstration ou les applications que vous avez développées à l'aide d'éditeurs de code et d'outils de débogage tiers.

Créez d'abord un dossier contenant les fichiers de construction générés (*`build-folder`*).

Utilisez la commande suivante pour générer des fichiers de build :

```
cmake -DVENDOR=st -DBOARD=stm32l475_discovery -DCOMPILER=arm-gcc -S freertos -B build-
folder
```

Si `arm-none-eabi-gcc` n'est pas dans votre chemin d'accès shell, vous devez également définir la variable CMake `AFR_TOOLCHAIN_PATH`. Par exemple :

```
-D AFR_TOOLCHAIN_PATH=/home/user/opt/gcc-arm-none-eabi/bin
```

Pour plus d'informations sur l'utilisation de CMake avec FreeRTOS, consultez. Utilisation de CMake avec FreeRTOS

Résolution des problèmes

Si vous voyez ce qui suit dans la sortie UART de l'application de démonstration, vous devez mettre à jour le microprogramme du Wi-Fi module :

```
[Tmr Svc] WiFi firmware version is: xxxxxxxxxxxx
[Tmr Svc] [WARN] WiFi firmware needs to be updated.
```

Pour télécharger le dernier Wi-Fi firmware, consultez le nœud IoT du kit Discovery STM32L4, sans fil basse consommation, Bluetooth Low Energy, NFC, subGHz,. Wi-Fi Dans Ressources binaires, choisissez le lien de téléchargement pour la mise à jour du microprogramme du Wi-Fi module Inventek ISM 43362.

Pour obtenir des informations générales sur la résolution des problèmes liés à la prise en main de FreeRTOS, consultez. Résolution des problèmes de mise en route

Commencer à utiliser les Texas Instruments CC3220SF-LAUNCHXL

> **Important**
>
> Cette intégration de référence est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Ce didacticiel fournit des instructions pour démarrer avec le Texas Instruments CC3220SF-LAUNCHXL. Si vous ne possédez pas le kit de CC3220SF-LAUNCHXL développement Texas Instruments (TI), consultez le catalogue des appareils AWS partenaires pour en acheter un auprès de notre partenaire.

Avant de commencer, vous devez configurer AWS IoT et télécharger FreeRTOS pour connecter votre appareil au Cloud. AWS Pour obtenir des instructions, consultez Premiers pas. Dans ce didacticiel, le chemin d'accès au répertoire de téléchargement de FreeRTOS est appelé. *freertos*

Présentation de

Ce didacticiel comprend les instructions de mise en route suivantes :

1. Installation de logiciels sur la machine hôte pour développer et déboguer des applications intégrées pour votre carte de microcontrôleur.
2. Compilation croisée d'une application de démonstration FreeRTOS en une image binaire.
3. Chargement de l'image binaire de l'application dans votre carte et exécution de l'application.

Configurer votre environnement de développement.

Suivez les étapes ci-dessous pour configurer votre environnement de développement afin de démarrer avec FreeRTOS.

Notez que FreeRTOS prend en charge deux IDE pour le kit de développement CC3220SF-LAUNCHXL TI : Code Composer Studio et IAR Embedded Workbench version 8.32. Vous pouvez utiliser l'un ou l'autre des IDE pour démarrer.

Installer Code Composer Studio

1. Accédez à IT Code Composer Studio.
2. Téléchargez le programme d'installation en mode hors connexion pour la plateforme de votre machine hôte (Windows, macOS ou Linux 64 bits).
3. Décompressez et exécutez le programme d'installation hors ligne. Suivez les invites.
4. Pour les familles de produits à installer, choisissez les microcontrôleurs sans fil SimpleLink Wi-Fi CC32xx.
5. Sur la page suivante, acceptez les paramètres par défaut pour le débogage des sondes, puis choisissez Terminer.

Si vous rencontrez des problèmes lorsque vous installez Code Composer Studio, consultez TI Development Tools Support, Code Composer Studio FAQs et Troubleshooting CCS.

Installer IAR Embedded Workbench

1. Téléchargez et exécutez le programme d'installation Windows pour la version 8.32 d'IAR Embedded Workbench pour ARM. Dans Debug sonde drivers (Déboguer les pilotes de sonde), assurez-vous que TI XDS est sélectionné :
2. Complétez l'installation et lancez le programme. Sur la page License Wizard, choisissez Register with RAP Systems to get an evaluation license (S'enregistrer auprès d'IAR Systems pour obtenir une licence d'évaluation ou utilisez votre propre licence IAR.

Installation du SDK SimpleLink CC3220

Installez le SDK SimpleLink CC3220. Le SDK SimpleLink Wi-Fi CC3220 contient des pilotes pour le microcontrôleur programmable CC3220SF, plus de 40 exemples d'applications et la documentation requise pour utiliser les échantillons.

Installer Uniflash

Installez Uniflash. CSC Uniflash est un outil autonome utilisée pour programmer la mémoire flash intégrée sur les cartes de contrôleur TI. Uniflash dispose d'une interface utilisateur graphique, d'une interface de ligne de commande et d'une interface de scripts.

Installer le dernier Service Pack

1. Sur votre TI CC3220SF-LAUNCHXL, placez le sauteur SOP sur le jeu de goupilles du milieu (position = 1) et réinitialisez la carte.
2. Démarrez Uniflash. Si votre LaunchPad carte CC3220SF apparaît sous Appareils détectés, choisissez Démarrer. Si votre carte n'est pas détectée, CC3220SF-LAUNCHXLchoisissez-la dans la liste des cartes sous Nouvelle configuration, puis choisissez Start Image Creator.
3. Choisissez New Project (Nouveau projet).
4. Sur la page Démarrer un nouveau projet, saisissez un nom pour votre projet. Pour Type d'appareil, choisissez CC3220SF. Pour Device Mode (Mode d'appareil), choisissez Develop (Développer), puis choisissez Create Project (Créer un projet).
5. Sur le côté droite de la fenêtre d'application Uniflash, choisissez Connect (Connexion).
6. Dans la colonne de gauche, choisissez Advanced (Avancé), Files (Fichiers), puis Service Pack.
7. Choisissez Parcourir, puis naviguez jusqu'à l'endroit où vous avez installé le SDK CC3220SF SimpleLink . Le service pack est situé dans `ti/simplelink_cc32xx_sdk_`*`VERSION`*`/tools/cc32xx_tools/servicepack-cc3x20/sp_`*`VERSION`*`.bin`.
8. Cliquez sur le bouton Burn (Incruster) (), puis choisissez Program Image (Create & Program) (Image du programme (Créer et programmer)) pour installer le service pack. N'oubliez pas de rétablir le jumper SOP en position 0 et de réinitialiser la carte.

Configuration du Wi-Fi provisionnement

Pour configurer les Wi-Fi paramètres de votre tableau, effectuez l'une des opérations suivantes :

- Configurez l'application de démonstration FreeRTOS décrite dans. Configuration des démos de FreeRTOS
- SmartConfigUtilisée par Texas Instruments.

Créez et exécutez le projet de démonstration FreeRTOS

Créez et exécutez le projet de démonstration FreeRTOS dans TI Code Composer

Pour importer la démo de FreeRTOS dans TI Code Composer

1. Ouvrez TI Code Composer, puis choisissez OK pour accepter le nom par défaut de l'espace de travail.
2. Sur la page Getting Started (Mise en route), choisissez Import Project (Importer un projet).
3. Dans Select search-directory, saisissez `projects/ti/cc3220_launchpad/ccs/aws_demos`. Le projet `aws_demos` doit être sélectionné par défaut. Pour importer le projet dans TI Code Composer, choisissez Finish (Terminer).
4. Dans Project Explorer (Explorateur de projet), double-cliquez sur aws_demos pour rendre le projet actif.
5. Dans Project (Projet), choisissez Build Project (Générer le projet) pour veiller à ce que le projet se compile sans erreurs ou avertissements.

Pour exécuter la démo de FreeRTOS dans TI Code Composer

1. Assurez-vous que le sauteur Sense On Power (SOP) de votre Texas Instruments CC3220SF-LAUNCHXL est en position 0. Pour plus d'informations, consultez le Guide de l'utilisateur du processeur réseau SimpleLink Wi-Fi CC3x20, CC3x3x.
2. Utilisez un câble USB pour connecter votre Texas Instruments CC3220SF-LAUNCHXL à votre ordinateur.
3. Dans l'explorateur de projets, assurez-vous que `CC3220SF.ccxml` est sélectionné comme configuration cible active. Pour le rendre actif, cliquez avec le bouton droit sur le fichier et choisissez Définir comme configuration cible active.
4. Dans TI Code Composer, à partir de Exécuter, choisissez Déboguer.

5. Lorsque le débogueur s'arrête au point d'arrêt dans `main()`, consultez le menu Run (Exécuter), puis choisissez Resume (Reprendre).

Surveillance des messages MQTT dans le cloud

Avant de lancer le projet de démonstration FreeRTOS, vous pouvez configurer le client MQTT dans la console pour surveiller AWS IoT les messages que votre appareil envoie au Cloud. AWS

Pour vous abonner à la rubrique MQTT avec AWS IoT Client MQTT

1. Connectez-vous à la console AWS IoT.
2. Dans le volet de navigation, choisissez Test, puis choisissez MQTT test client pour ouvrir le client MQTT.
3. Dans le champ Rubrique d'abonnement, saisissez ***your-thing-name*/example/topic**, puis choisissez S'abonner à la rubrique.

Lorsque le projet de démonstration s'exécute avec succès sur votre appareil, le message « Hello World ! » s'affiche envoyé plusieurs fois au sujet auquel vous vous êtes abonné.

Créez et exécutez un projet de démonstration FreeRTOS dans IAR Embedded Workbench

Pour importer la démo de FreeRTOS dans IAR Embedded Workbench

1. Ouvrez IAR Embedded Workbench, choisissez File (Fichier), puis choisissez Open Workspace (Ouvrir l'espace de travail).
2. Accédez à `projects/ti/cc3220_launchpad/iar/aws_demos`, choisissez aws_demos.eww, puis choisissez OK.
3. Right-click le nom du projet (`aws_demos`), puis choisissez Make.

Pour exécuter la démo de FreeRTOS dans IAR Embedded Workbench

1. Assurez-vous que le sauteur Sense On Power (SOP) de votre Texas Instruments CC3220SF-LAUNCHXL est en position 0. Pour plus d'informations, consultez le Guide de l'utilisateur du processeur réseau SimpleLink Wi-Fi CC3x20, CC3x3x.
2. Utilisez un câble USB pour connecter votre Texas Instruments CC3220SF-LAUNCHXL à votre ordinateur.
3. Générez une nouvelle build pour votre projet.

Pour regénérer le projet, dans le menu Project (Projet), choisissez Make.

4. Dans le menu Project (Projet), choisissez Download et Debug (Télécharger et déboguer). Vous pouvez ignorer le message « Avertissement : échec de l'initialisation EnergyTrace » s'il est affiché. Pour plus d'informations EnergyTrace, consultez la section EnergyTrace Technologie MSP.
5. Lorsque le débogueur s'arrête au point d'arrêt dans `main()`, accédez au menu Debug (Déboguer), puis choisissez Go (OK).

Utilisation de CMake avec FreeRTOS

Si vous préférez ne pas utiliser d'IDE pour le développement de FreeRTOS, vous pouvez également utiliser CMake pour créer et exécuter les applications de démonstration ou les applications que vous avez développées à l'aide d'éditeurs de code et d'outils de débogage tiers.

Pour créer la démo de FreeRTOS avec CMake

1. Créez un dossier contenant les fichiers de compilation générés (*`build-folder`*).
2. Assurez-vous que votre chemin de recherche (variable d'environnement$PATH) contient le dossier où se trouve le binaire du compilateur TI CGT (par exemple `C:\ti\ccs910\ccs\tools\compiler\ti-cgt-arm_18.12.2.LTS\bin`).

   Si vous utilisez le compilateur BRAS IT avec votre carte TI, utilisez la commande suivante pour générer des fichiers de build à partir du code source :

   ```
   cmake -DVENDOR=ti -DBOARD=cc3220_launchpad -DCOMPILER=arm-ti -S freertos -B build-folder
   ```

Pour de plus amples informations, veuillez consulter Utilisation de CMake avec FreeRTOS.

Résolution des problèmes

Si aucun message ne s'affiche dans le client MQTT de la AWS IoT console, vous devrez peut-être configurer les paramètres de débogage de la carte.

Pour configurer les paramètres de débogage des cartes TI

1. Dans Code Composer, dans Project Explorer (Explorateur de projet), choisissez aws_demos.

2. Dans le menu Run (Exécuter), choisissez Debug Configurations.
3. Dans le volet de navigation, sélectionnez aws_demos.
4. Sous l'onglet Target (Cible), sous Connection Options (Options de connexion), choisissez Reset the target on a connect (Réinitialiser la cible sur une connexion).
5. Choisissez Apply, puis Close.

Si ces étapes ne fonctionnent pas, regardez la sortie du programme dans le terminal série. Vous devez voir un texte qui indique la source du problème.

Pour obtenir des informations générales sur la résolution des problèmes liés à la prise en main de FreeRTOS, consultez. Résolution des problèmes de mise en route

Mise en route avec le simulateur d'appareils Windows

Ce didacticiel fournit des instructions pour démarrer avec le simulateur de périphériques Windows FreeRTOS.

Avant de commencer, vous devez configurer AWS IoT et télécharger FreeRTOS pour connecter votre appareil au Cloud. AWS Pour obtenir des instructions, consultez Premiers pas. Dans ce didacticiel, le chemin d'accès au répertoire de téléchargement de FreeRTOS est appelé. *`freertos`*

FreeRTOS est publié sous forme de fichier zip contenant les bibliothèques FreeRTOS et des exemples d'applications pour la plate-forme que vous spécifiez. Pour exécuter les exemples sur un ordinateur Windows, téléchargez les bibliothèques et les exemples portés à exécuter sur Windows. Cet ensemble de fichiers est désigné comme le simulateur FreeRTOS pour Windows.

> **Note**
>
> Ce didacticiel ne peut pas être exécuté correctement sur les instances Windows Amazon EC2.

Configurer votre environnement de développement.

1. Installez la dernière version de Npcap. Sélectionnez le « WinPcap API-compatible mode » lors de l'installation.
2. Installez Microsoft Visual Studio.

Les versions Visual Studio 2017 et 2019 fonctionnent. Toutes les éditions de ces versions de Visual Studio sont prises en charge (Community, Professionnel ou Entreprise).

En plus de l'IDE, installez le composant Développement de bureau avec C++.

Installez le dernier kit SDK Windows 10. Vous pouvez le choisir sous la section Facultatif du composant Développement de bureau avec C++.

3. Assurez-vous que vous avez une connexion Ethernet câblée active.
4. (Facultatif) Si vous souhaitez utiliser le système de CMake-based compilation pour créer vos projets FreeRTOS, installez la dernière version de CMake. FreeRTOS nécessite la version 3.13 ou ultérieure de CMake.

Surveillance des messages MQTT dans le cloud

Avant de lancer le projet de démonstration FreeRTOS, vous pouvez configurer le client MQTT dans la console pour surveiller AWS IoT les messages que votre appareil envoie au Cloud. AWS

Pour vous abonner à la rubrique MQTT avec le AWS IoT Client MQTT

1. Connectez-vous à la console AWS IoT.
2. Dans le volet de navigation, choisissez Test, puis choisissez MQTT test client pour ouvrir le client MQTT.
3. Dans le champ Rubrique d'abonnement, saisissez ***your-thing-name*****/example/topic**, puis choisissez S'abonner à la rubrique.

Lorsque le projet de démonstration s'exécute avec succès sur votre appareil, vous voyez « Hello World ! » envoyé plusieurs fois au sujet auquel vous vous êtes abonné.

Créez et exécutez le projet de démonstration FreeRTOS

Vous pouvez utiliser Visual Studio ou CMake pour créer des projets FreeRTOS.

Création et exécution du projet de démonstration FreeRTOS avec l'IDE Visual Studio

1. Chargez le projet dans Visual Studio.

Dans Visual Studio, dans le menu Fichier, choisissez Ouvrir. Choisissez File/Solution, naviguez jusqu'au `projects/pc/windows/visual_studio/aws_demos/aws_demos.sln` fichier, puis choisissez Ouvrir.

2. Reciblez le projet de démonstration.

   Le projet de démonstration fourni dépend du kit SDK Windows, mais il n'a pas de version du kit SDK Windows spécifiée. Par défaut, l'IDE peut tenter de générer la démonstration avec une version du kit SDK qui n'est pas présente sur votre ordinateur. Pour définir la version du kit SDK Windows, cliquez avec le bouton droit de la souris sur `aws_demos` puis choisissez Recibler les projets. Cette opération permet d'ouvrir la fenêtre Examiner les actions de la solution. Choisissez une version du SDK Windows présente sur votre ordinateur (la valeur initiale dans la liste déroulante est correcte), puis cliquez sur OK.

3. Créez et exécutez le projet.

   Dans le menu Générer, choisissez Générer la solution, et assurez-vous que la solution est générée sans erreurs ou avertissements. Choisissez Dégoguer, Démarrer le débogage pour exécuter le projet. Lors de la première exécution, vous devez sélectionner une interface réseau.

Création et exécution du projet de démonstration FreeRTOS avec CMake

Nous vous recommandons d'utiliser l'interface graphique CMake au lieu de l'outil de ligne de commande CMake pour créer le projet de démonstration du simulateur Windows.

Une fois que vous avez installé CMake, ouvrez son interface utilisateur graphique. Sous Windows, vous pouvez la trouver dans le menu Démarrer sous CMake, CMake (cmake-gui).

1. Définissez le répertoire du code source de FreeRTOS.

   Dans l'interface graphique, définissez le répertoire du code source FreeRTOS *`freertos`* () pour Where is the source code.

   Définissez *`freertos`*`/build` pour Where to build the binaries (Où générer les fichiers binaires).

2. Configurez le projet CMake.

   Dans l'interface utilisateur graphique de CMake, choisissez Ajouter une entrée, puis dans la fenêtre Ajouter une entrée de cache, définissez les valeurs suivantes :

Nom

AFR_BOARD

Type

CHAÎNE

Value

pc.windows

Description

(Facultatif)

3. Choisissez Configurer. Si CMake vous invite à créer le dossier de génération, choisissez Oui, puis sélectionnez un générateur sous Spécifier le générateur pour ce projet. Nous vous recommandons d'utiliser Visual Studio comme générateur, mais Ninja est également pris en charge. (Notez que lorsque vous utilisez Visual Studio 2019, la plateforme doit être définie sur Win32 au lieu de son paramètre par défaut.) Conservez les autres options du générateur inchangées et choisissez Terminer.
4. Générer et ouvrez le projet CMake.

   Une fois que vous avez configuré le projet, l'interface utilisateur graphique de CMake affiche toutes les options disponibles pour le projet généré. Dans le cadre de ce didacticiel, vous pouvez conserver les valeurs par défaut des options.

   Choisissez Générer pour créer une solution Visual Studio, puis choisissez Ouvrir le projet pour ouvrir le projet dans Visual Studio.

   Dans Visual Studio, cliquez avec le bouton droit sur le `aws_demos` projet et choisissez Définir en tant que StartUp projet. Cela vous permet de générer et d'exécuter le projet. Lors de la première exécution, vous devez sélectionner une interface réseau.

Pour plus d'informations sur l'utilisation de CMake avec FreeRTOS, consultez. Utilisation de CMake avec FreeRTOS

Configurer votre interface réseau

Lors de la première exécution du projet de démonstration, vous devez sélectionner l'interface réseau à utiliser. Le programme compte vos interfaces réseau. Recherchez le numéro de votre interface Ethernet câblée. La sortie doit se présenter comme suit :

```
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.
```

Une fois que vous avez identifié le numéro de votre interface Ethernet câblée, fermez la fenêtre de l'application. Dans l'exemple précédent, le numéro à utiliser est1.

Ouvrez `FreeRTOSConfig.h` et définissez `configNETWORK_INTERFACE_TO_USE` sur le numéro correspondant à votre interface réseau câblée.

> **Important**
>
> Seules les interfaces Ethernet sont prises en charge. Wi-Fi n'est pas pris en charge.

Résolution des problèmes

Dépannage des problèmes courants sur Windows

Vous rencontrerez peut-être l'erreur suivante en tenant de générer le projet de démonstration avec Visual Studio :

```
Error "The Windows SDK version X.Y was not found" when building the provided Visual
 Studio solution.
```

Le projet doit être ciblé sur une version du kit SDK Windows présente sur votre ordinateur.

Pour obtenir des informations générales sur la résolution des problèmes liés à la prise en main de FreeRTOS, consultez. Résolution des problèmes de mise en route

Commencer à utiliser le kit IoT industriel Xilinx Avnet MicroZed

> **Important**
>
> Cette intégration de référence est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Ce didacticiel fournit des instructions pour démarrer avec le kit MicroZed IoT industriel Xilinx Avnet. Si vous ne possédez pas le kit Xilinx Avnet Industrial MicroZed IoT, consultez le catalogue des appareils AWS partenaires pour en acheter un auprès de notre partenaire.

Avant de commencer, vous devez configurer AWS IoT et télécharger FreeRTOS pour connecter votre appareil au Cloud. AWS Pour obtenir des instructions, consultez Premiers pas. Dans ce didacticiel, le chemin d'accès au répertoire de téléchargement de FreeRTOS est appelé. *freertos*

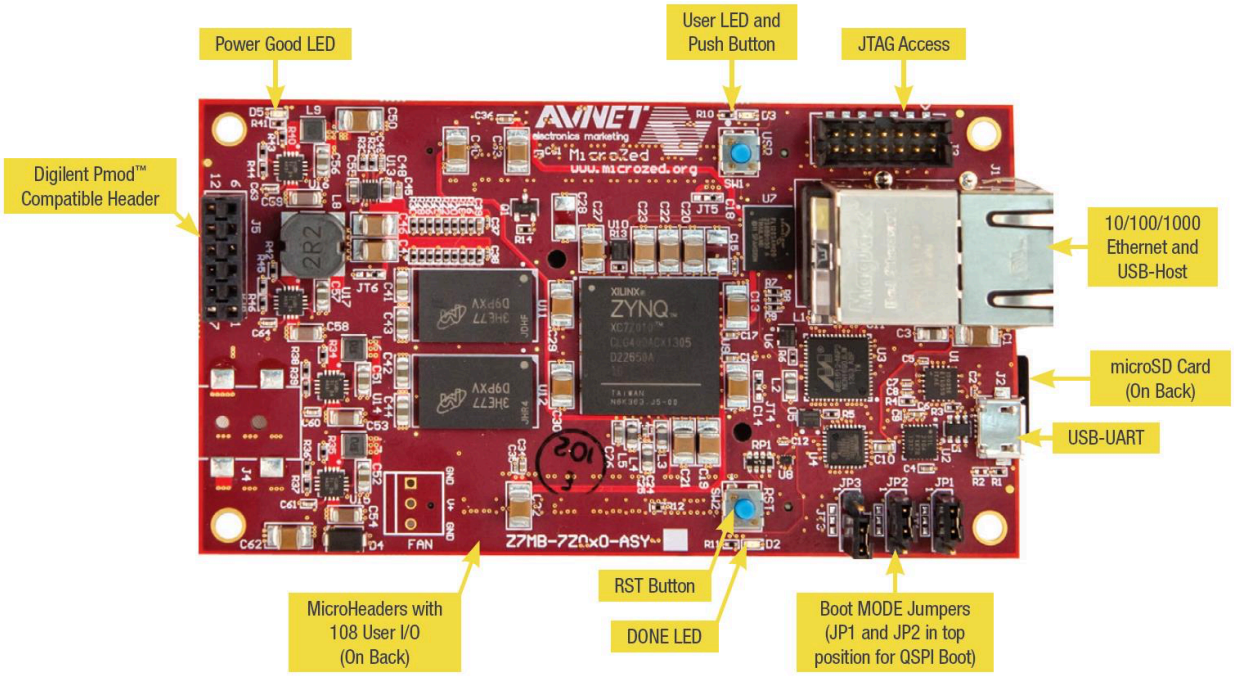
Présentation de

Ce didacticiel comprend les instructions de mise en route suivantes :

1. Connexion de votre carte à un appareil hôte.
2. Installation de logiciels sur la machine hôte pour développer et déboguer des applications intégrées pour votre carte de microcontrôleur.
3. Compilation croisée d'une application de démonstration FreeRTOS en une image binaire.
4. Chargement de l'image binaire de l'application dans votre carte et exécution de l'application.

Configuration du MicroZed matériel

Le schéma suivant peut être utile lors de la configuration du MicroZed matériel :



Pour configurer le MicroZed tableau

1. Connectez votre ordinateur au USB-UART port de votre MicroZed carte mère.
2. Connectez votre ordinateur au port JTAG Access de votre MicroZed carte mère.
3. Connectez un routeur ou un port Ethernet connecté à Internet à l'Ethernet et au USB-Host port de votre MicroZed carte mère.

Configurer votre environnement de développement.

Pour configurer les configurations FreeRTOS pour MicroZed le kit, vous devez utiliser le kit de développement logiciel (XSDK) Xilinx. XSDK est pris en charge sur Windows et Linux.

Télécharger et installer XSDK

Pour installer le logiciel Xilinx, vous avez besoin d'un compte Xilinx gratuit.

Pour télécharger XSDK

1. Accédez à la page de téléchargement du WebInstall client autonome du kit de développement logiciel.

2. Choisissez l'option correspondant à votre système d'exploitation.
3. Vous êtes redirigé vers une page de connexion Xilinx.

   Si vous avez un compte chez Xilinx, entrez vos informations de connexion, puis choisissez Se connecter.

   Si vous n'avez pas de compte, choisissez Create your account. Une fois que vous êtes inscrit, vous devez recevoir un e-mail contenant un lien d'activation de votre compte Xilinx.
4. Sur la page Name and Address Verification, entrez les informations requises, puis choisissez Next. Le téléchargement devrait être prêt à démarrer.
5. Enregistrez le fichier `Xilinx_SDK_`*`version`*`_`*`os`*.

Pour installer XSDK

1. Ouvrez le fichier `Xilinx_SDK_`*`version`*`_`*`os`*.
2. Dans Select Edition to Install, choisissez Xilinx Software Development Kit (XSDK), puis Next.
3. Sur la page suivante de l'assistant d'installation, sous Installation Options, sélectionnez Installation Cable Drivers, puis choisissez Next.

Si votre ordinateur ne détecte pas la MicroZed USB-UART connexion, installez les pilotes VCP du USB-to-UART pont CP210x manuellement. Pour obtenir des instructions, consultez le guide d' USB-to-UART installation du Silicon Labs CP210x.

Pour plus d'informations sur XSDK, consultez le guide de mise en route du kit SDK Xilinx sur le site Web de Xilinx.

Surveillance des messages MQTT dans le cloud

Avant de lancer le projet de démonstration FreeRTOS, vous pouvez configurer le client MQTT dans la console pour surveiller AWS IoT les messages que votre appareil envoie au Cloud. AWS

Pour vous abonner à la rubrique MQTT avec AWS IoT Client MQTT

1. Connectez-vous à la console AWS IoT.
2. Dans le volet de navigation, choisissez Test, puis choisissez MQTT test client pour ouvrir le client MQTT.

3. Dans le champ Rubrique d'abonnement, saisissez ***your-thing-name*/example/topic**, puis choisissez S'abonner à la rubrique.

Créez et exécutez le projet de démonstration FreeRTOS

Ouvrez la démo FreeRTOS dans l'IDE XSDK

1. Lancez l'IDE XSDK en définissant le répertoire de l'espace de travail sur *freertos*/projects/xilinx/microzed/xsdk.
2. Fermez la page de bienvenue. Dans le menu, choisissez Project, puis décochez Build Automatically.
3. Dans le menu, choisissez File, puis Import.
4. Sur la page Select, développez General, puis choisissez Existing Projects into Workspace et Next.
5. Sur la page Importer des projets, choisissez Sélectionner le répertoire racine, puis entrez le répertoire racine de votre projet de démonstration : *freertos*/projects/xilinx/microzed/xsdk/aws_demos. Pour parcourir le répertoire, choisissez Browse.

   Une fois que vous avez spécifié un répertoire racine, les projets de ce répertoire s'affichent sur la page Import Projects. Tous les projets disponibles sont sélectionnés par défaut.

   > **Note**
   >
   > Si vous voyez un avertissement en haut de la page Import Projects (« Some projects cannot be imported because they already exist in the workspace »), vous pouvez l'ignorer.

6. Lorsque tous les projets sont sélectionnés, choisissez Finish.
7. Si vous ne voyez pas les projets aws_bsp, fsbl et MicroZed_hw_platform_0 dans le volet des projets, répétez les étapes précédentes à partir de #3, mais avec le répertoire racine défini sur *freertos*/vendors/xilinx, et importez aws_bsp, fsbl et MicroZed_hw_platform_0.
8. Dans le menu, choisissez Window, puis Preferences.
9. Dans le volet de navigation, développez Run/Debug, choisissez Substitution de chaînes, puis choisissez Nouveau.

10. Dans New String Substitution Variable, pour Name, saisissez **`AFR_ROOT`**. Pour Valeur, entrez le chemin d'accès racine du fichier *`freertos`*`/projects/xilinx/microzed/xsdk/aws_demos`. Choisissez OK, puis choisissez OK à nouveau pour enregistrer la variable et fermez Preferences.

Créez le projet de démonstration FreeRTOS

1. À partir de l'IDE XSDK, dans le menu, sélectionnez Project, puis Clean.
2. Dans Clean, conservez les valeurs par défaut, puis choisissez OK. XSDK nettoie et crée tous les projets, puis génère les fichiers `.elf`.

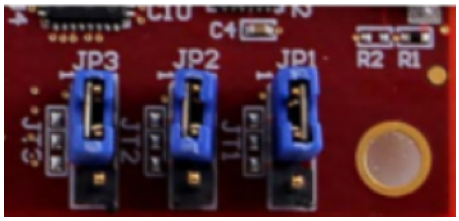
> **Note**
>
> Pour créer tous les projets sans les nettoyer, choisissez Project, puis Build All.
> Pour générer des projets individuels, sélectionnez le projet que vous voulez créer et choisissez Project, puis Build Project.

Génération de l'image de démarrage pour le projet de démonstration FreeRTOS

1. Dans l'IDE XSDK, cliquez avec le bouton droit de la souris sur aws_demos, puis choisissez Create Boot Image.
2. Dans Create Boot Image, choisissez Create new BIF file.
3. En regard de l'option Output BIF file path (Chemin d'accès au fichier BIF de sortie), choisissez Browse (Parcourir), puis `aws_demos.bif` situé dans *`<freertos>`*`/vendors/xilinx/microzed/aws_demos/aws_demos.bif`.
4. Choisissez Ajouter.
5. Dans Add new boot image partition, à côté de l'option File path, choisissez Browse, puis `fsbl.elf`, situé dans `vendors/xilinx/fsbl/Debug/fsbl.elf`.
6. Pour Partition type, choisissez bootloader, puis OK.
7. Dans Create Boot Image, choisissez Create Image. Dans Override Files, choisissez OK pour remplacer le fichier `aws_demos.bif` existant et générer le fichier `BOOT.bin` sous `projects/xilinx/microzed/xsdk/aws_demos/BOOT.bin`.

Débogage JTAG

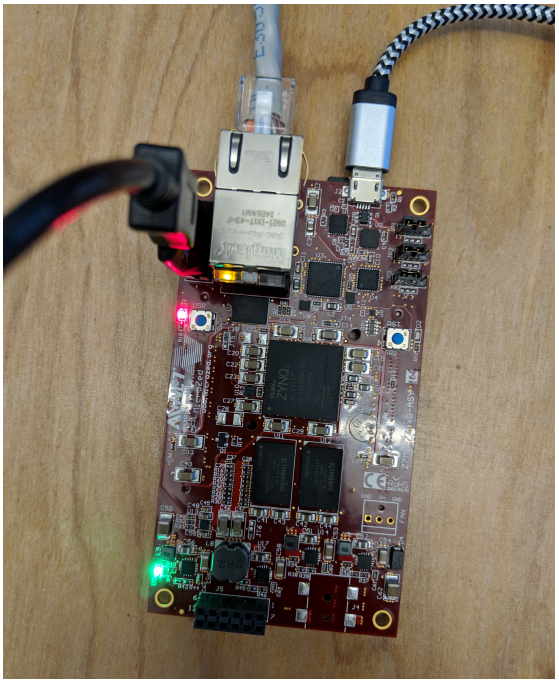
1. Réglez les sauteurs du mode de démarrage de votre MicroZed carte sur le mode de démarrage JTAG.



2. Insérez votre carte microSD dans l'emplacement pour carte microSD situé directement sous le port. USB-UART

> **Note**
>
> Avant de procéder au débogage, assurez-vous de sauvegarder tout le contenu de la carte MicroSD.

La carte doit être similaire à cela :



3. Dans l'IDE XSDK, cliquez avec le bouton droit de la souris sur aws_demos, choisissez Debug As, puis choisissez 1 Launch on System Hardware (System Debugger).

4. Lorsque le débogueur s'arrête au point d'arrêt dans `main()`, dans le menu, choisissez Run, puis Resume.

> **Note**
>
> La première fois que vous exécutez l'application, une nouvelle paire clé-certificat est importée en mémoire non volatile. Pour les exécutions suivantes, vous pouvez mettre en commentaire `vDevModeKeyProvisioning()` dans le fichier `main.c` avant de reconstruire les images et le fichier `BOOT.bin`. Cela empêche la copie des certificats et des clés dans le stockage à chaque exécution.

Vous pouvez choisir de démarrer votre carte à partir MicroZed d'une carte microSD ou d'un flash QSPI pour exécuter le projet de démonstration FreeRTOS. Pour obtenir les instructions, consultez Génération de l'image de démarrage pour le projet de démonstration FreeRTOS et Exécutez le projet de démonstration FreeRTOS.

Exécutez le projet de démonstration FreeRTOS

Pour exécuter le projet de démonstration FreeRTOS, vous pouvez démarrer MicroZed votre carte à partir d'une carte microSD ou d'un flash QSPI.

Lorsque vous configurez votre MicroZed tableau pour exécuter le projet de démonstration FreeRTOS, reportez-vous au schéma ci-dessous. Configuration du MicroZed matériel Vérifiez que vous avez connecté votre MicroZed carte à votre ordinateur.

Démarrez le projet FreeRTOS à partir d'une carte microSD

Formatez la carte microSD fournie avec le kit Xilinx MicroZed Industrial IoT.

1. Copiez le fichier `BOOT.bin` sur la carte MicroSD.
2. Insérez la carte dans l'emplacement pour carte microSD situé directement sous le USB-UART port.
3. Réglez les sauteurs du mode de MicroZed démarrage sur le mode de démarrage SD.



4. Appuyez sur le bouton RST pour réinitialiser l'appareil et lancer le démarrage de l'application. Vous pouvez également débrancher le USB-UART câble du USB-UART port, puis le réinsérer.

Démarrez le projet de démonstration FreeRTOS depuis QSPI Flash

1. Réglez les sauteurs du mode de démarrage de votre MicroZed carte sur le mode de démarrage JTAG.



2. Vérifiez que votre ordinateur est connecté aux ports USB-UART et JTAG Access. Le voyant vert Power Good doit être illuminé.
3. À partir de l'IDE XSDK, dans le menu, choisissez Xilinx, puis Program Flash.
4. Dans Program Flash Memory, la plateforme matérielle doit être renseignée automatiquement. Dans Connexion, choisissez votre serveur MicroZed matériel pour connecter votre carte à votre ordinateur hôte.

> **Note**
>
> Si vous utilisez le câble Xilinx Smart Lync JTAG, vous devez créer un serveur matériel dans l'IDE XSDK. Choisissez New, puis définissez votre serveur.

5. Dans Image File, entrez le nom du chemin d'accès au fichier image `BOOT.bin`. Vous pouvez également rechercher le fichier via l'option Browse.
6. Dans Offset, saisissez **`0x0`**.
7. Dans FSBL File, entrez le nom du chemin d'accès au fichier `fsbl.elf`. Vous pouvez également rechercher le fichier via l'option Browse.
8. Choisissez Program pour programmer la carte.
9. Une fois la programmation QSPI terminée, retirez le USB-UART câble pour mettre la carte hors tension.
10. Réglez les sauteurs du mode de démarrage de votre MicroZed carte sur le mode de démarrage QSPI.

11. Insérez votre carte dans l'emplacement pour carte microSD situé directement sous le USB-UART port.

> **Note**
>
> Assurez-vous de sauvegarder tout le contenu de la carte MicroSD.

12. Appuyez sur le bouton RST pour réinitialiser l'appareil et lancer le démarrage de l'application. Vous pouvez également débrancher le USB-UART câble du USB-UART port, puis le réinsérer.

Résolution des problèmes

Si vous rencontrez des erreurs de génération liées à des chemins d'accès incorrects, essayez de nettoyer et de recréer le projet, comme décrit dans Créez le projet de démonstration FreeRTOS.

Sur Windows, assurez-vous que vous utilisez des barres obliques standards lorsque vous définissez les variables de substitution de chaîne dans l'IDE Windows XSDK.

Pour obtenir des informations générales sur la résolution des problèmes liés à la prise en main de FreeRTOS, consultez. Résolution des problèmes de mise en route

Prochaines étapes avec FreeRTOS

> **Important**
>
> Cette page fait référence au Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Après avoir créé, flashé et exécuté le projet de démonstration FreeRTOS pour votre forum, vous pouvez visiter FreeRTOS.org le site Web pour en savoir plus sur la création d'un nouveau projet FreeRTOS. Il existe également des démos pour de nombreuses bibliothèques FreeRTOS qui montrent comment effectuer des tâches importantes, interagir AWS IoT avec les services et programmer des fonctionnalités spécifiques aux cartes mères (telles que les modems cellulaires). Pour plus d'informations, consultez la page Catégories de la bibliothèque FreeRTOS.

Le FreeRTOS.org site Web contient également des informations détaillées sur le noyau FreeRTOS ainsi que sur les concepts fondamentaux du système d'exploitation en temps réel. Pour plus d'informations, consultez les pages FreeRTOS Kernel Developer Docs et FreeRTOS Kernel Secondary Docs.

Mises à jour de FreeRTOS Over-the-Air

> **Note**
>
> Voir AWS IoT Over-the-Air (OTA) sur le site Web de FreeRTOS pour obtenir des informations récentes sur les mises à jour (OTA Over-the-air).

Over-the-air Les mises à jour (OTA) vous permettent de déployer des mises à jour du microprogramme sur un ou plusieurs appareils de votre parc. Même si les mises à jour OTA ont été conçues pour mettre à jour le microprogramme de l'appareil, vous pouvez les utiliser pour envoyer les fichiers à un ou plusieurs appareils enregistrés auprès d' AWS IoT. Lorsque vous envoyez des mises à jour OTA, nous vous recommandons de les signer numériquement afin que les périphériques qui reçoivent les fichiers puissent vérifier qu'elles n'ont pas été altérées en cours de route.

Vous pouvez utiliser Code Signing for AWS IoT afin de signer vos fichiers, ou vous pouvez signer vos fichiers avec vos propres outils de signature de code.

Lorsque vous créez une mise à jour OTA, Service Gestionnaire de mise à jour OTA crée une tâche AWS IoT pour informer vos appareils qu'une mise à jour est disponible. L'application de démonstration OTA s'exécute sur votre appareil et crée une tâche FreeRTOS qui s'abonne aux sujets de notification AWS IoT pour les tâches et écoute les messages de mise à jour. Lorsqu'une mise à jour est disponible, l'agent OTA publie des demandes AWS IoT et reçoit des mises à jour à l'aide du protocole HTTP ou MQTT, selon les paramètres que vous avez choisis. L'agent OTA vérifie la signature numérique des fichiers téléchargés et, si les fichiers sont valides, installe la mise à jour du microprogramme. Si vous n'utilisez pas l'application de démonstration FreeRTOS OTA Update, vous devez l'intégrer à votre propre application pour bénéficier de AWS IoT Bibliothèque en direct (OTA) la fonctionnalité de mise à jour du microprogramme.

Les mises à jour en direct de FreeRTOS vous permettent de :

- Signer numériquement le microprogramme avant le déploiement.
- Déployer les nouvelles images du microprogramme sur un seul appareil, un groupe d'appareils ou l'ensemble de votre flotte.

- Déployer les microprogrammes sur les appareils au fur et à mesure qu'ils sont ajoutés à des groupes, réinitialisés ou remis en service.
- Vérifier l'authenticité et l'intégrité du nouveau microprogramme après qu'il a été déployé sur les appareils.
- Surveiller la progression d'un déploiement.
- Déboguer un déploiement ayant échoué.

Balisage des ressources OTA

Pour vous aider à gérer vos ressources OTA, vous pouvez, si vous le souhaitez, affecter vos propres métadonnées à des mises à jour et des flux sous la forme de balises. Les balises vous permettent de classer vos AWS IoT ressources de différentes manières (par exemple, par objectif, propriétaire ou environnement). Cela est utile lorsque vous avez de nombreuses ressources du même type. Vous pouvez identifier rapidement une ressource en fonction des balises qui lui ont été attribuées.

Pour plus d'informations, veuillez consulter la rubrique Balisage de vos ressources AWS IoT.

Conditions préalables aux mises à jour OTA

Pour utiliser les mises à jour OTA, procédez comme suit :

- Vérifiez les Conditions préalables aux mises à jour OTA via HTTP ou les Conditions préalables aux mises à jour OTA via MQTT.
- Créez un compartiment Amazon S3 pour stocker votre mise à jour.
- Créer un rôle de service de mise à jour OTA.
- Créer une stratégie utilisateur OTA.
- Créer un certificat de signature de code.
- Si vous utilisez la signature de code pour AWS IoT,Accorder l'accès à la signature de code pour AWS IoT.
- Téléchargez FreeRTOS avec la bibliothèque OTA.

Créez un compartiment Amazon S3 pour stocker votre mise à jour

Les fichiers de mise à jour OTA sont stockés dans des compartiments Amazon S3.

Si vous utilisez Code Signing for AWS IoT, la commande que vous utilisez pour créer une tâche de signature de code utilise un compartiment source (où se trouve l'image du microprogramme non

signée) et un compartiment de destination (où l'image du microprogramme signée est écrite). Vous pouvez spécifier le même compartiment pour la source et la destination. Les noms de fichier sont modifiés en GUID afin que les fichiers d'origine ne soient pas écrasés.

Pour créer un compartiment Amazon S3

1. Connectez-vous à la console Amazon S3 à l'adresse https://console.aws.amazon.com/s3/.
2. Choisissez Créer un compartiment.
3. Entrez un nom de compartiment.
4. Dans les paramètres du compartiment pour Bloquer l'accès public, maintenez l'option Bloquer tout accès public sélectionnée pour accepter les autorisations par défaut.
5. Sous Gestion des versions des compartiments, sélectionnez Activer pour conserver toutes les versions dans le même compartiment.
6. Choisissez Créer un compartiment.

Pour plus d'informations sur Amazon S3, consultez le guide de l'utilisateur d'Amazon Simple Storage Service.

Créer un rôle de service de mise à jour OTA

Le service de mise à jour OTA assume ce rôle pour créer et gérer les tâches de mise à jour OTA en votre nom.

Pour créer un rôle de service OTA

1. Connectez-vous à la https://console.aws.amazon.com/iam/.
2. Dans le volet de navigation, choisissez Rôles.
3. Choisissez Créer un rôle.
4. Sous Sélectionner un type d'entité de confiance, choisissez AWS Service.
5. Choisissez l'IoT dans la liste des AWS services.
6. Sous Select your use case (Sélectionner votre cas d'utilisation), choisissez IoT.
7. Choisissez Suivant : Autorisations.
8. Choisissez Suivant : Balises.
9. Choisissez Suivant : Vérification.
10. Entrez un nom de rôle et une description, puis choisissez Créer un rôle.

Pour plus d'informations sur les rôles IAM, consultez la section Rôles IAM.

> **Important**
>
> Pour résoudre le problème confus de sécurité des adjoints, vous devez suivre les instructions du AWS IoT Coreguide.

Pour ajouter les autorisations de mise à jour OTA à votre rôle de service OTA

1. Dans le champ de recherche de la page de la console IAM, entrez le nom de votre rôle, puis choisissez-le dans la liste.
2. Choisissez Attacher des politiques.
3. Dans la zone de recherche, entrez AmazonFreeRTOSOTAUpdate « », AmazonFreeRTOSOTAUpdatesélectionnez dans la liste des politiques filtrées, puis choisissez Attacher une politique pour associer la stratégie à votre rôle de service.

Pour ajouter les autorisations IAM requises à votre rôle de service OTA

1. Dans le champ de recherche de la page de la console IAM, entrez le nom de votre rôle, puis choisissez-le dans la liste.
2. Sélectionnez Ajouter une politique en ligne.
3. Sélectionnez l'onglet JSON.
4. Copiez et collez le document de stratégie suivant dans la zone de texte :

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "iam:PassRole"
            ],
            "Resource": "arn:aws:iam::111122223333:role/your_role_name"
        }
```

```
    ]
}
```

Assurez-vous de le *your_account_id* remplacer par votre identifiant de AWS compte et *your_role_name* par le nom du rôle de service OTA.

5. Choisissez Examiner une politique.
6. Entrez un nom pour la stratégie, puis choisissez Create policy (Créer une stratégie).

> **Note**
>
> La procédure suivante n'est pas obligatoire si le nom de votre compartiment Amazon S3 commence par « afr-ota ». Si c'est le cas, la politique AWS gérée inclut `AmazonFreeRTOSOTAUpdate` déjà les autorisations requises.

Pour ajouter les autorisations Amazon S3 requises à votre rôle de service OTA

1. Dans le champ de recherche de la page de la console IAM, entrez le nom de votre rôle, puis choisissez-le dans la liste.
2. Sélectionnez Ajouter une politique en ligne.
3. Sélectionnez l'onglet JSON.
4. Copiez et collez le document de stratégie suivant dans la zone.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObjectVersion",
                "s3:GetObject",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::example-bucket/*"
```

```
            ]
        }
    ]
}
```

Cette politique accorde à votre rôle de service OTA l'autorisation de lire les objets Amazon S3. Assurez-vous de le remplacer *example-bucket* par le nom de votre bucket.

5. Choisissez Examiner une politique.
6. Entrez un nom pour la stratégie, puis choisissez Create policy (Créer une stratégie).

Créer une stratégie utilisateur OTA

Vous devez accorder à votre utilisateur l'autorisation d'effectuer des mises à jour OTA. Votre utilisateur doit disposer des autorisations suivantes :

- Accéder au compartiment S3 où les mises à jour de votre microprogramme sont stockées.
- Certificats d'accès stockés dans AWS Certificate Manager.
- Accédez à la fonctionnalité de livraison de AWS IoT MQTT-based fichiers.
- Accédez aux mises à jour de FreeRTOS OTA.
- Accédez aux AWS IoT offres d'emploi.
- Accédez à IAM.
- Signature du code d'accès pour AWS IoT. Consultez Accorder l'accès à la signature de code pour AWS IoT.
- Répertoriez les plateformes matérielles FreeRTOS.
- Marquez et décochez les AWS IoT ressources.

Pour accorder à votre utilisateur les autorisations requises, consultez la section Politiques IAM. Consultez également la section Autorisation des utilisateurs et des services cloud à utiliser les AWS IoT tâches.

Pour activer l'accès, ajoutez des autorisations à vos utilisateurs, groupes ou rôles :

- Utilisateurs et groupes dans AWS IAM Identity Center :

  Créez un jeu d'autorisations. Suivez les instructions de la rubrique Création d'un jeu d'autorisations du Guide de l'utilisateur AWS IAM Identity Center .

- Utilisateurs gérés dans IAM par un fournisseur d'identité :

  Créez un rôle pour la fédération d'identité. Suivez les instructions de la rubrique Création d'un rôle pour un fournisseur d'identité tiers (fédération) dans le Guide de l'utilisateur IAM.
- Utilisateurs IAM :
  - Créez un rôle que votre utilisateur peut assumer. Suivez les instructions de la rubrique Création d'un rôle pour un utilisateur IAM dans le Guide de l'utilisateur IAM.
  - (Non recommandé) Attachez une politique directement à un utilisateur ou ajoutez un utilisateur à un groupe d'utilisateurs. Suivez les instructions de la rubrique Ajout d'autorisations à un utilisateur (console) du Guide de l'utilisateur IAM.

Créer un certificat de signature de code

Pour signer numériquement les images du microprogramme, vous avez besoin d'un certificat de signature de code et d'une clé privée. À des fins de test, vous pouvez créer un certificat auto-signé et une clé privée. Pour les environnements de production, achetez un certificat auprès d'une autorité de certification (CA) bien connue.

Les différentes plateformes nécessitent différents types de certificats de signature de code. Les sections suivantes décrivent comment créer des certificats de signature de code pour différentes FreeRTOS-qualified plateformes.

Rubriques

- Création d'un certificat de signature de code pour le Texas Instruments CC3220SF-LAUNCHXL
- Création d'un certificat de signature de code pour les cartes Espressif ESP32
- Création d'un certificat de signature pour le code du Nordic nrf52840-dk
- Création d'un certificat de signature de code pour le simulateur FreeRTOS pour Windows
- Création d'un certificat de signature de code pour le matériel personnalisé

Création d'un certificat de signature de code pour le Texas Instruments CC3220SF-LAUNCHXL

> **Important**
>
> Cette intégration de référence est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-

FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Le kit de développement Launchpad du microcontrôleur sans fil SimpleLink Wi-Fi CC3220SF prend en charge deux chaînes de certificats pour la signature du code du microprogramme :

- Production (certificat-catalogue)

  Pour utiliser la chaîne de certificats de production, vous devez acheter un certificat de signature de code et utilisez l'outil TI Uniflash pour définir la carte en mode production.

- Test et développement (certificat-playground)

  La chaîne de certificats Playground vous permet d'essayer les mises à jour OTA avec un certificat de signature de code autosigné.

Utilisez le AWS Command Line Interface pour importer votre certificat de signature de code, votre clé privée et votre chaîne de certificats dans. AWS Certificate Manager Pour plus d'informations, voir Installation du AWS CLI dans le guide de AWS Command Line Interface l'utilisateur.

Téléchargez et installez la dernière version du SDK SimpleLink CC3220. Par défaut, les fichiers dont vous avez besoin se trouvent ici :

`C:\ti\simplelink_cc32xx_sdk_version\tools\cc32xx_tools\certificate-playground` (Windows)

`/Applications/Ti/simplelink_cc32xx_version/tools/cc32xx_tools/certificate-playground` (macOS)

Les certificats du SDK SimpleLink CC3220 sont au format DER. Pour créer un certificat de signature de code autosigné, vous devez le convertir au format PEM.

Suivez ces étapes pour créer un certificat de signature de code lié à la hiérarchie des certificats du terrain de jeu de Texas Instruments AWS Certificate Manager et répondant aux critères de signature de AWS IoT code.

> **Note**
>
> Pour créer un certificat de signature de code, vous devez installer OpenSSL sur votre machine. Une fois que vous avez installé OpenSSL, assurez-vous qu'`openssl` est attribué à l'exécutable OpenSSL dans votre invite de commande ou dans l'environnement de terminal.

Pour créer un certificat de signature de code autosigné

1. Ouvrez une invite de commande ou un terminal avec des autorisations d'administrateur.
2. Dans votre répertoire de travail, utilisez le texte suivant pour créer un fichier nommé `cert_config.txt`. Remplacez *`test_signer@amazon.com`* par votre adresse e-mail.

   ```
   [ req ]
   prompt             = no
   distinguished_name = my dn

   [ my dn ]
   commonName = test_signer@amazon.com

   [ my_exts ]
   keyUsage         = digitalSignature
   extendedKeyUsage = codeSigning
   ```

3. Créez une clé privée et une demande de signature de certificat (CSR) :

   ```
   openssl req -config cert_config.txt -extensions my_exts -nodes -days 365 -newkey
    rsa:2048 -keyout tisigner.key -out tisigner.csr
   ```

4. Convertissez la clé privée de la CA racine du playground Texas Instruments du format DER au format PEM.

   La clé privée de la CA racine du playground TI réside ici :

   `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. Convertissez le certificat de la CA racine du playground Texas Instruments du format DER au format PEM.

   Le certificat racine du playground TI réside ici :

   `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. Signez la CSR avec la CA racine 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. Convertissez votre certificat de signature de code (`tisigner.crt.pem`) au format DER :

   ```
   openssl x509 -in tisigner.crt.pem -out tisigner.crt.der -outform DER
   ```

   > **Note**
   >
   > Vous écrirez le certificat `tisigner.crt.der` sur la carte de développement TI ultérieurement.

8. Importez le certificat de signature de code, la clé privée et la chaîne de certificats dans : 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
   ```

   Cette commande affiche l'ARN de votre certificat. Vous avez besoin de cet ARN lorsque vous créez une tâche de mise à jour OTA.

> **Note**
>
> Cette étape part du principe que vous allez utiliser la signature de code AWS IoT pour signer les images de votre microprogramme. Bien que l'utilisation de la signature de code AWS IoT soit recommandée, vous pouvez signer les images de votre microprogramme manuellement.

Création d'un certificat de signature de code pour les cartes Espressif ESP32

> **Important**
>
> Cette intégration de référence est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Les cartes Espressif ESP32 prennent en charge un certificat autosigné SHA-256 avec signature de code ECDSA.

> **Note**
>
> Pour créer un certificat de signature de code, vous devez installer OpenSSL sur votre machine. Une fois que vous avez installé OpenSSL, assurez-vous qu'`openssl` est attribué à l'exécutable OpenSSL dans votre invite de commande ou dans l'environnement de terminal. Utilisez le AWS Command Line Interface pour importer votre certificat de signature de code, votre clé privée et votre chaîne de certificats dans. AWS Certificate Manager Pour plus d'informations sur l'installation du AWS CLI, reportez-vous à la section Installation du AWS CLI.

1. Dans votre répertoire de travail, utilisez le texte suivant pour créer un fichier nommé `cert_config.txt`. Remplacez *`test_signer@amazon.com`* par votre adresse e-mail :

```
[ req ]
```

```
prompt             = no
distinguished_name = my_dn

[ my_dn ]
commonName = test_signer@amazon.com

[ my_exts ]
keyUsage         = digitalSignature
extendedKeyUsage = codeSigning
```

2. Créer une clé privée de signature de code ECDSA :

```
openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -pkeyopt
 ec_param_enc:named_curve -outform PEM -out ecdsasigner.key
```

3. Créer un certificat de signature de code ECDSA :

```
openssl req -new -x509 -config cert_config.txt -extensions my_exts -nodes -days 365
 -key ecdsasigner.key -out ecdsasigner.crt
```

4. Importez le certificat de signature de code, la clé privée et la chaîne de certificats dans : AWS Certificate Manager

```
aws acm import-certificate --certificate fileb://ecdsasigner.crt --private-key
 fileb://ecdsasigner.key
```

Cette commande affiche l'ARN de votre certificat. Vous avez besoin de cet ARN lorsque vous créez une tâche de mise à jour OTA.

> **Note**
>
> Cette étape part du principe que vous allez utiliser la signature de code AWS IoT pour signer les images de votre microprogramme. Bien que l'utilisation de la signature de code AWS IoT soit recommandée, vous pouvez signer les images de votre microprogramme manuellement.

Création d'un certificat de signature pour le code du Nordic nrf52840-dk

> **Important**
>
> Cette intégration de référence est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Le Nordic nrf52840-dk prend en charge un SHA256 auto-signé avec le certificat de signature de code ECDSA.

> **Note**
>
> Pour créer un certificat de signature de code, vous devez installer OpenSSL sur votre machine. Une fois que vous avez installé OpenSSL, assurez-vous qu'`openssl` est attribué à l'exécutable OpenSSL dans votre invite de commande ou dans l'environnement de terminal. Utilisez le AWS Command Line Interface pour importer votre certificat de signature de code, votre clé privée et votre chaîne de certificats dans. AWS Certificate Manager Pour plus d'informations sur l'installation du AWS CLI, reportez-vous à la section Installation du AWS CLI.

1. Dans votre répertoire de travail, utilisez le texte suivant pour créer un fichier nommé `cert_config.txt`. Remplacez *`test_signer@amazon.com`* par votre adresse e-mail :

```
[ req ]
prompt             = no
distinguished_name = my_dn

[ my_dn ]
commonName = test_signer@amazon.com

[ my_exts ]
keyUsage         = digitalSignature
extendedKeyUsage = codeSigning
```

2. Créer une clé privée de signature de code ECDSA :

```
openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -pkeyopt
 ec_param_enc:named_curve -outform PEM -out ecdsasigner.key
```

3. Créer un certificat de signature de code ECDSA :

```
openssl req -new -x509 -config cert_config.txt -extensions my_exts -nodes -days 365
 -key ecdsasigner.key -out ecdsasigner.crt
```

4. Importez le certificat de signature de code, la clé privée et la chaîne de certificats dans : AWS Certificate Manager

```
aws acm import-certificate --certificate fileb://ecdsasigner.crt --private-key
 fileb://ecdsasigner.key
```

Cette commande affiche l'ARN de votre certificat. Vous avez besoin de cet ARN lorsque vous créez une tâche de mise à jour OTA.

> **Note**
>
> Cette étape part du principe que vous allez utiliser la signature de code AWS IoT pour signer les images de votre microprogramme. Bien que l'utilisation de la signature de code AWS IoT soit recommandée, vous pouvez signer les images de votre microprogramme manuellement.

Création d'un certificat de signature de code pour le simulateur FreeRTOS pour Windows

Le simulateur FreeRTOS pour Windows nécessite un certificat de signature de code avec une clé P-256 ECDSA et un hachage pour effectuer les mises à jour OTA. SHA-256 Si vous n'avez pas de certificat de signature de code, suivez la procédure ci-dessous pour en créer un :

> **Note**
>
> Pour créer un certificat de signature de code, vous devez installer OpenSSL sur votre machine. Une fois que vous avez installé OpenSSL, assurez-vous qu'`openssl` est attribué à l'exécutable OpenSSL dans votre invite de commande ou dans l'environnement de terminal. Utilisez le AWS Command Line Interface pour importer votre certificat de signature de code, votre clé privée et votre chaîne de certificats dans. AWS Certificate Manager Pour plus

d'informations sur l'installation du AWS CLI, reportez-vous à la section Installation du AWS CLI.

1. Dans votre répertoire de travail, utilisez le texte suivant pour créer un fichier nommé `cert_config.txt`. Remplacez *`test_signer@amazon.com`* par votre adresse e-mail :

```
[ req ]
prompt             = no
distinguished_name = my_dn

[ my_dn ]
commonName = test_signer@amazon.com

[ my_exts ]
keyUsage         = digitalSignature
extendedKeyUsage = codeSigning
```

2. Créer une clé privée de signature de code ECDSA :

```
openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -pkeyopt
 ec_param_enc:named_curve -outform PEM -out ecdsasigner.key
```

3. Créer un certificat de signature de code ECDSA :

```
openssl req -new -x509 -config cert_config.txt -extensions my_exts -nodes -days 365
 -key ecdsasigner.key -out ecdsasigner.crt
```

4. Importez le certificat de signature de code, la clé privée et la chaîne de certificats dans : AWS Certificate Manager

```
aws acm import-certificate --certificate fileb://ecdsasigner.crt --private-key
 fileb://ecdsasigner.key
```

Cette commande affiche l'ARN de votre certificat. Vous avez besoin de cet ARN lorsque vous créez une tâche de mise à jour OTA.

> **Note**
>
> Cette étape part du principe que vous allez utiliser la signature de code AWS IoT pour signer les images de votre microprogramme. Bien que l'utilisation de la signature de code AWS IoT soit recommandée, vous pouvez signer les images de votre microprogramme manuellement.

Création d'un certificat de signature de code pour le matériel personnalisé

À l'aide d'un ensemble d'outils appropriés, créez un certificat auto-signé et une clé privée pour votre matériel.

Utilisez le AWS Command Line Interface pour importer votre certificat de signature de code, votre clé privée et votre chaîne de certificats dans. AWS Certificate Manager Pour plus d'informations sur l'installation du AWS CLI, reportez-vous à la section Installation du AWS CLI.

Après avoir créé votre certificat de signature de code, vous pouvez utiliser le AWS CLI pour l'importer dans ACM :

```
aws acm import-certificate --certificate fileb://code-sign.crt --private-key fileb://
code-sign.key
```

La sortie de cette commande affiche l'ARN de votre certificat. Vous avez besoin de cet ARN lorsque vous créez une tâche de mise à jour OTA.

ACM a besoin de certificats pour utiliser des algorithmes et des tailles de clés spécifiques. Pour plus d'informations, consultez Conditions requises pour l'importation de certificats. Pour plus d'informations sur ACM, consultez la section Importation de certificats dans. AWS Certificate Manager

Vous devez copier, coller et formater le contenu de votre certificat de signature de code dans le `vendors/vendor/boards/board/aws_demos/config_files/ota_demo_config.h` fichier qui fait partie du code FreeRTOS que vous téléchargerez ultérieurement.

Accorder l'accès à la signature de code pour AWS IoT

Pour activer l'accès, ajoutez des autorisations à vos utilisateurs, groupes ou rôles :

- Utilisateurs et groupes dans AWS IAM Identity Center :

Créez un jeu d'autorisations. Suivez les instructions de la rubrique Création d'un jeu d'autorisations du Guide de l'utilisateur AWS IAM Identity Center .

- Utilisateurs gérés dans IAM par un fournisseur d'identité :

  Créez un rôle pour la fédération d'identité. Suivez les instructions de la rubrique Création d'un rôle pour un fournisseur d'identité tiers (fédération) dans le Guide de l'utilisateur IAM.

- Utilisateurs IAM :
  - Créez un rôle que votre utilisateur peut assumer. Suivez les instructions de la rubrique Création d'un rôle pour un utilisateur IAM dans le Guide de l'utilisateur IAM.
  - (Non recommandé) Attachez une politique directement à un utilisateur ou ajoutez un utilisateur à un groupe d'utilisateurs. Suivez les instructions de la rubrique Ajout d'autorisations à un utilisateur (console) du Guide de l'utilisateur IAM.

Téléchargez FreeRTOS avec la bibliothèque OTA

Vous pouvez cloner ou télécharger FreeRTOS depuis. GitHub Consultez le README.mdfichier pour obtenir des instructions.

Pour plus d'informations sur la configuration et l'exécution de l'application de démonstration OTA, consultez Over-the-air met à jour l'application de démonstration.

> **Important**
>
> - Dans cette rubrique, le chemin d'accès au répertoire de téléchargement de FreeRTOS est appelé. *`freertos`*
> - Les espaces dans le chemin d'accès *`freertos`* peuvent provoquer des échecs de construction. Lorsque vous clonez ou copiez le référentiel, assurez-vous que le chemin d'accès que vous créez ne contient pas d'espaces.
> - La longueur maximale d'un chemin sous Microsoft Windows est de 260 caractères. Les longs chemins de répertoire de téléchargement de FreeRTOS peuvent provoquer des échecs de compilation.
> - Le code source pouvant contenir des liens symboliques, si vous utilisez Windows pour extraire l'archive, vous devrez peut-être :
>   - Activez le mode développeur ou
>   - Utilisez une console élevée en tant qu'administrateur.

Ainsi, Windows peut créer correctement des liens symboliques lors de l'extraction de l'archive. Dans le cas contraire, les liens symboliques seront écrits sous forme de fichiers normaux contenant les chemins des liens symboliques sous forme de texte ou seront vides. Pour plus d'informations, consultez le billet de blog Symlinks in Windows 10 ! .

Si vous utilisez Git sous Windows, vous devez activer le mode développeur ou vous devez :

- Définissez `core.symlinks` ce paramètre sur true à l'aide de la commande suivante :

```
git config --global core.symlinks true
```

- Utilisez une console élevée en tant qu'administrateur chaque fois que vous utilisez une commande git qui écrit sur le système (par exemplegit pull,git clone, etgit submodule update --init --recursive).

Conditions préalables aux mises à jour OTA via MQTT

Cette section décrit les exigences générales relatives à l'utilisation du protocole MQTT pour effectuer des mises à jour OTA.

Configuration requise

- Le microprogramme de l'appareil doit inclure les bibliothèques FreeRTOS nécessaires (agent CoreMQTT, mise à jour OTA et leurs dépendances).
- La version 1.4.0 ou ultérieure de FreeRTOS est requise. Cependant, nous vous recommandons d'utiliser la dernière version dans la mesure du possible.

Configurations

À partir de la version 201912.00, FreeRTOS OTA peut utiliser le protocole HTTP ou MQTT pour transférer les images de mise à jour du microprogramme depuis les appareils. AWS IoT Si vous spécifiez les deux protocoles lorsque vous créez une mise à jour OTA dans FreeRTOS, chaque appareil déterminera le protocole utilisé pour transférer l'image. Pour plus d'informations, consultez Conditions préalables aux mises à jour OTA via HTTP.

Par défaut, la configuration des protocoles OTA `ota_config.h`consiste à utiliser le protocole MQTT.

Configurations spécifiques aux périphériques

Aucune.

Utilisation de la mémoire

Lorsque le protocole MQTT est utilisé pour le transfert de données, aucune mémoire supplémentaire n'est requise pour la connexion MQTT, car elle est partagée entre les opérations de contrôle et de données.

Politique en matière d'appareils

Chaque appareil qui reçoit une mise à jour OTA à l'aide de MQTT doit être enregistré en tant qu'objet AWS IoT et doit être associé à une politique similaire à celle répertoriée ici. Vous trouverez plus d'informations sur les éléments dans les objets `"Action"` et `"Resource"` dans Actions de stratégie AWS IoT et Ressources d'action AWS IoT Core.

Remarques

- Les `iot:Connect` autorisations permettent à votre appareil de se connecter AWS IoT via MQTT.
- Les `iot:Publish` autorisations `iot:Subscribe` et sur les sujets des AWS IoT tâches (`.../jobs/*`) permettent à l'appareil connecté de recevoir des notifications de tâches et des documents relatifs aux tâches, et de publier l'état d'achèvement de l'exécution d'une tâche.
- Les `iot:Publish` autorisations `iot:Subscribe` et sur le thème des flux AWS IoT OTA (`.../streams/*`) permettent à l'appareil connecté de récupérer les données de mise à jour OTA à partir de AWS IoT. Ces autorisations sont requises pour effectuer des mises à jour du firmware sur MQTT.
- Les `iot:Receive` autorisations permettent AWS IoT Core de publier des messages sur ces sujets sur l'appareil connecté. Cette autorisation est vérifiée à chaque remise d'un message MQTT. Vous pouvez utiliser cette autorisation pour révoquer l'accès aux clients actuellement abonnés à une rubrique.

Conditions préalables aux mises à jour OTA via HTTP

Cette section décrit les exigences générales relatives à l'utilisation du protocole HTTP pour effectuer des mises à jour OTA. À partir de la version 201912.00, FreeRTOS OTA peut utiliser le protocole HTTP ou MQTT pour transférer les images de mise à jour du microprogramme depuis les appareils. AWS IoT

> **Note**
>
> - Bien que le protocole HTTP puisse être utilisé pour transférer l'image du microprogramme, la bibliothèque de l'agent CoreMQTT est toujours requise car d'autres interactions AWS IoT Core utilisent la bibliothèque de l'agent CoreMQTT, notamment l'envoi ou la réception de notifications d'exécution de tâches, de documents de tâche et de mises à jour du statut d'exécution.
> - Lorsque vous spécifiez les deux protocoles MQTT et HTTP pour la tâche de mise à jour OTA, la configuration du logiciel de l'agent OTA sur chaque périphérique détermine le protocole à utiliser pour transférer l'image du microprogramme. Pour remplacer la méthode de protocole MQTT par défaut de l'agent OTA par le protocole HTTP, vous pouvez modifier les fichiers d'en-tête utilisés pour compiler le code source FreeRTOS pour le périphérique.

Configuration requise

- Le microprogramme de l'appareil doit inclure les bibliothèques FreeRTOS nécessaires (agent CoreMQTT, HTTP, agent OTA et leurs dépendances).
- La version 201912.00 ou ultérieure de FreeRTOS est requise pour modifier la configuration des protocoles OTA afin de permettre le transfert de données OTA via HTTP.

Configurations

Veuillez consulter la configuration suivante des protocoles OTA dans le fichier `\vendors\boards\board\aws_demos\config_files\ota_config.h`.

Pour activer le transfert de données OTA via HTTP

1. Remplacez `configENABLED_DATA_PROTOCOLS` par `OTA_DATA_OVER_HTTP`
2. Lorsque de la mise à jour OTA, vous pouvez spécifier les deux protocoles afin que le protocole MQTT ou HTTP puisse être utilisé. Pour définir le protocole principal utilisé par le périphérique sur HTTP, remplacez `configOTA_PRIMARY_DATA_PROTOCOL` par `OTA_DATA_OVER_HTTP`.

> **Note**
>
> HTTP n'est pris en charge que pour les opérations de données OTA. Pour les opérations de contrôle, vous devez utiliser MQTT.

Configurations spécifiques aux périphériques

ESP32

En raison d'une quantité limitée de RAM, vous devez désactiver BLE lorsque vous activez HTTP comme protocole de données OTA. Dans le fichier [vendors/espressif/boards/esp32/aws_demos/config_files/aws_iot_network_config.h](), remplacez `configENABLED_NETWORKS` par `AWSIOT_NETWORK_TYPE_WIFI` uniquement.

```
/**
     * @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 )
```

Utilisation de la mémoire

Lorsque MQTT est utilisé pour le transfert de données, aucune mémoire de segment supplémentaire n'est requise pour la connexion MQTT, car elle est partagée entre les opérations de contrôle et de données. Toutefois, l'activation de données via HTTP nécessite une mémoire de segment supplémentaire. Voici les données d'utilisation de la mémoire par segment de mémoire pour toutes les plateformes prises en charge, calculées à l'aide de l'API FreeRTOS`xPortGetFreeHeapSize`. Assurez-vous de disposer de suffisamment de RAM pour utiliser la bibliothèque OTA.

Instruments texans CC3220SF-LAUNCHXL

Opérations de contrôle (MQTT) : 12 Ko

Opérations de données (HTTP) : 10 Ko

> **Note**
>
> TI utilise beaucoup moins de RAM, car il utilise le protocole SSL sur le matériel. Il n'a donc pas recours à la bibliothèque mbedtls.

Microchip Curiosity PIC32MZEF

Opérations de contrôle (MQTT) : 65 Ko

Opérations de données (HTTP) : 43 Ko

Espressif ESP32

Opérations de contrôle (MQTT) : 65 Ko

Opérations de données (HTTP) : 45 Ko

> **Note**
>
> BLE sur ESP32 consomme environ 87 Ko de RAM. La quantité de RAM ne suffit pas pour toutes les opérations, ce qui est mentionné dans les configurations spécifiques aux périphériques ci-dessus.

Simulateur Windows

Opérations de contrôle (MQTT) : 82 Ko

Opérations de données (HTTP) : 63 Ko

Nordic nrf52840-dk

HTTP n'est pas pris en charge.

Politique en matière d'appareils

Cette stratégie vous permet d'utiliser MQTT ou HTTP pour les mises à jour OTA.

Chaque appareil qui reçoit une mise à jour OTA via le protocole HTTP doit être enregistré en tant qu'objet dans AWS IoT et l'objet doit avoir une stratégie attachée comme celle répertoriée ici. Vous trouverez plus d'informations sur les éléments dans les objets `"Action"` et `"Resource"` dans Actions de stratégie AWS IoT et Ressources d'action AWS IoT Core.

Remarques

- Les autorisations `iot:Connect` permettent à votre appareil de se connecter à AWS IoT via MQTT.
- Les `iot:Publish` autorisations `iot:Subscribe` et sur les sujets des AWS IoT tâches (`.../jobs/*`) permettent à l'appareil connecté de recevoir des notifications de tâches et des documents relatifs aux tâches, et de publier l'état d'achèvement de l'exécution d'une tâche.
- Les `iot:Receive` autorisations permettent AWS IoT Core de publier des messages sur ces sujets sur l'appareil actuellement connecté. Cette autorisation est vérifiée à chaque remise d'un message MQTT. Vous pouvez utiliser cette autorisation pour révoquer l'accès aux clients actuellement abonnés à une rubrique.

Didacticiel OTA

Cette section contient un didacticiel pour la mise à jour du microprogramme sur les appareils exécutant FreeRTOS à l'aide des mises à jour OTA. En plus des images du microprogramme, vous pouvez utiliser une mise à jour OTA pour envoyer n'importe quel type de fichier à un périphérique connecté à AWS IoT.

Vous pouvez utiliser la AWS IoT console ou le AWS CLI pour créer une mise à jour OTA. La console est le moyen le plus simple de vous familiariser avec OTA, car elle effectue une grande partie du travail à votre place. AWS CLI C'est utile lorsque vous automatisez des tâches de mise à jour OTA, que vous travaillez avec un grand nombre d'appareils ou que vous utilisez des appareils qui ne sont pas qualifiés pour FreeRTOS. Pour plus d'informations sur les appareils éligibles à FreeRTOS, consultez le site Web des partenaires FreeRTOS.

Pour créer une mise à jour OTA

1. Déployez une version initiale de votre microprogramme sur un ou plusieurs appareils.
2. Vérifiez que le microprogramme s'exécute correctement.
3. Lorsqu'une mise à jour du microprogramme est nécessaire, apportez les modifications au code et créez la nouvelle image.

4. Si vous signez manuellement votre microprogramme, signez puis téléchargez l'image du microprogramme signée dans votre compartiment Amazon S3. Si vous utilisez la signature de code pour AWS IoT, téléchargez votre image de microprogramme non signée dans un compartiment Amazon S3.
5. Créez une mise à jour OTA.

Lorsque vous créez une mise à jour OTA, vous spécifiez le protocole de livraison d'image (MQTT ou HTTP). Vous pouvez spécifier les deux pour permettre au périphérique de choisir. L'agent FreeRTOS OTA de l'appareil reçoit l'image du microprogramme mise à jour et vérifie la signature numérique, la somme de contrôle et le numéro de version de la nouvelle image. Si la mise à jour du microprogramme est vérifiée, l'appareil est réinitialisé et, en fonction de la logique définie par l'application, valide la mise à jour. Si vos appareils n'exécutent pas FreeRTOS, vous devez implémenter un agent OTA qui s'exécute sur vos appareils.

Installation du micrologiciel initial

Pour mettre à jour le micrologiciel, vous devez installer une version initiale du micrologiciel qui utilise la bibliothèque de l'agent OTA pour écouter les tâches de mise à jour OTA. Si vous n'utilisez pas FreeRTOS, ignorez cette étape. Vous devez à la place copier votre implémentation de l'agent OTA sur vos appareils.

Rubriques

- Installez la version initiale du microprogramme sur le Texas Instruments CC3220SF-LAUNCHXL
- Installer la version initiale du micrologiciel sur la carte Espressif ESP32
- Installer la version initiale du micrologiciel sur la Nordic nRF52840 DK
- Micrologiciel initial sur le simulateur Windows
- Installer la version initiale du microprogramme sur une carte personnalisée

Installez la version initiale du microprogramme sur le Texas Instruments CC3220SF-LAUNCHXL

> **Important**
>
> Cette intégration de référence est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-

FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Ces instructions supposent que vous avez déjà généré le projet `aws_demos`, comme décrit dansTéléchargez, compilez, flashez et exécutez la démo de FreeRTOS OTA sur Texas Instruments CC3220SF-LAUNCHXL.

1. Sur votre Texas Instruments CC3220SF-LAUNCHXL, placez le sauteur SOP sur le jeu de goupilles du milieu (position = 1) et réinitialisez le plateau.
2. Téléchargez et installez l'outil TI Uniflash.
3. Démarrez Uniflash. Dans la liste des configurations, choisissez CC3220SF-LAUNCHXL, puis choisissez Start Image Creator.
4. Choisissez New Project (Nouveau projet).
5. Sur la page Démarrer un nouveau projet, saisissez un nom pour votre projet. Pour Type d'appareil, choisissez CC3220SF. Pour Device Mode (Mode d'appareil), choisissez Develop (Développer). Choisissez Create Project (Créer un projet).
6. Déconnectez votre émulateur de terminal.
7. Sur le côté droite de la fenêtre d'application Uniflash, choisissez Connect (Connexion).
8. Sous Avancé, Fichiers, choisissez Fichiers utilisateur.
9. Dans le volet de sélection File (Fichier), choisissez l'icône Add File (Ajouter un fichier)

.
10. Accédez au répertoire `/Applications/Ti/simplelink_cc32xx_sdk_`*`version`*`/tools/cc32xx_tools/certificate-playground`, sélectionnez `dummy-root-ca-cert`, choisissez Open (Ouvrir), puis choisissez Write (Écrire).
11. Dans le volet de sélection File (Fichier), choisissez l'icône Add File (Ajouter un fichier)

.
12. Accédez au répertoire de travail où vous avez créé le certificat de signature de code et la clé privée, choisissez `tisigner.crt.der`, choisissez Open (Ouvrir), puis choisissez Write (Écrire).
13. Dans la liste déroulante Action, choisissez Select MCU Image (Sélectionner l'image de la carte de microcontrôleur), puis choisissez Browse (Parcourir) pour choisir l'image du microprogramme à utiliser pour écrire sur votre appareil (aws_demos.bin). Ce fichier est situé dans le répertoire

freertos/vendors/ti/boards/cc3220_launchpad/aws_demos/ccs/Debug. Choisissez Ouvrir.

a. Dans la boîte de dialogue du fichier, confirmez que le nom du fichier est défini sur `mcuflashimg.bin`.

b. Activez la case à cocher Vendor (Fournisseur).

c. Sous File Token (Jeton de fichier), tapez **1952007250**.

d. Sous Private Key File Name (Nom de fichier de clé privée), choisissez Browse (Parcourir), puis choisissez `tisigner.key` dans le répertoire de travail où vous avez créé le certificat de signature de code et la clé privée.

e. Sous Certification File Name (Nom de fichier de certification), choisissez `tisigner.crt.der`.

f. Choisissez Write (Écrire).

14. Dans le volet gauche, sous Files (Fichiers), choisissez Service Pack.

15. Sous Service Pack File Name (Nom du fichier du Service Pack), choisissez Browse (Parcourir), accédez à `simplelink_cc32x_sdk_`*version*`/tools/cc32xx_tools/servicepack-cc3x20`, choisissez `sp_3.7.0.1_2.0.0.0_2.2.0.6.bin`, puis choisissez Open (Ouvrir).

16. Dans le volet gauche, sous Fichiers, sélectionnez Trusted Root-Certificate Catalog.

17. Désactivez la case à cocher Utiliser le Root-Certificate catalogue sécurisé par défaut.

18. Sous Fichier source, choisissez Parcourir, choisissez ***version***simplelink_cc32xx_sdk_// certificate- .lst, puis choisissez Ouvrir. tools/cc32xx_tools playground/ certcatalogPlayGround20160911

19. Sous Fichier source de signature, choisissez Parcourir, choisissez simplelink_cc32xx_sdk_// certificate- .lst.signed_3220.bin ***version***, puis choisissez Ouvrir. tools/cc32xx_tools playground/ certcatalogPlayGround20160911

20. Choisissez le bouton



pour enregistrer votre projet.

21. Choisissez le bouton



.

22. Choisissez Program Image (Image du programme).

23. Une fois le processus de programmation terminé, placez le jumper SOP sur la première série de broches (position = 0), réinitialisez la carte et reconnectez votre émulateur de terminal pour vous assurer que la sortie est la même que lorsque vous déboguez la démonstration avec Code Composer Studio. Notez le numéro de version de l'application dans la sortie du terminal. Vous utiliserez ce numéro de version ultérieurement pour vérifier que votre micrologiciel a été mis à jour par une mise à jour OTA.

    Le terminal doit afficher une sortie semblable à la suivante.

```
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
```

Installer la version initiale du micrologiciel sur la carte Espressif ESP32

> **Important**
>
> Cette intégration de référence est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Ce guide est écrit en partant du principe que vous avez déjà suivi les étapes décrites dans Getting Started with the Espressif ESP32-DevKitC ESP-WROVER-KIT et les prérequis de Over-the-Air mise à jour. Avant de tenter une mise à jour OTA, vous souhaiterez peut-être exécuter le projet de démonstration MQTT décrit dans Getting Started with FreeRTOS pour vous assurer que votre carte et votre chaîne d'outils sont correctement configurées.

Pour une image d'usine initiale sur la carte

1. Ouvrez*freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h, commentez #define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED et définissez CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED ouCONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED.
2. Copiez le certificat de SHA-256/ECDSA PEM-formatted signature de code que vous avez généré dans le Conditions préalables aux mises à jour OTA to. vendors/*vendor*/boards/*board*/aws_demos/config_files/ota_demo_config.h Il doit être formaté comme suit.

   ```
   #define otapalconfigCODE_SIGNING_CERTIFICATE  \
   "-----BEGIN CERTIFICATE-----\n" \
   "...base64 data...\n" \
   "-----END CERTIFICATE-----\n";
   ```

3. Avec la démonstration de mise à jour OTA sélectionnée, suivez les mêmes étapes que celles décrites dans Mise en route avec ESP32 pour générer et flasher l'image. Si vous avez précédemment généré et flashé le projet, il se peut que vous ayez besoin d'exécuter d'abord make clean. Après avoir exécuté make flash monitor, vous devriez voir quelque chose

de semblable à ce qui suit. La commande de certains messages peut varier, car l'application de démonstration exécute plusieurs tâches à la fois.

```
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. La carte ESP32 écoute désormais les mises à jour OTA. Le ESP-IDF moniteur est lancé par la `make flash monitor` commande. Vous pouvez appuyer sur Ctrl+] pour quitter. Vous pouvez également utiliser votre programme terminal TTY favori (par exemple, PuTTY, Tera Term ou GNU Screen) pour écouter la sortie série de la carte. Sachez que la connexion au port série de la carte peut entraîner son redémarrage.

Installer la version initiale du micrologiciel sur la Nordic nRF52840 DK

> **⚠ Important**
>
> Cette intégration de référence est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Ce guide est écrit en partant du principe que vous avez déjà effectué les étapes de la section Over-the-Air Mettre à jour Débuter avec le Nordic n RF52840-DK les prérequis. Avant de tenter une mise à jour OTA, vous souhaiterez peut-être exécuter le projet de démonstration MQTT décrit dans Getting Started with FreeRTOS pour vous assurer que votre carte et votre chaîne d'outils sont correctement configurées.

Pour une image d'usine initiale sur la carte

1. Ouvrir *freertos*`/vendors/nordic/boards/nrf52840-dk/aws_demos/config_files/aws_demo_config.h`.
2. Remplacez `#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED` par `CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` ou `CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`.
3. Avec la démonstration de la mise à jour OTA sélectionnée, suivez les mêmes étapes que celles décrites dans Débuter avec le Nordic n RF52840-DK pour générer et flasher l'image.

   Vous devez visualiser des résultats similaires à ce qui suit.

```
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
```

La carte écoute désormais les mises à jour OTA.

Micrologiciel initial sur le simulateur Windows

Lorsque vous utilisez le simulateur Windows, il n'est pas nécessaire de flasher une version initiale du microprogramme. Le simulateur Windows fait partie de l'application `aws_demos`, qui comprend également le microprogramme.

Installer la version initiale du microprogramme sur une carte personnalisée

À l'aide de votre IDE, générez le projet `aws_demos`, en veillant à inclure la bibliothèque OTA. Pour plus d'informations sur la structure du code source de FreeRTOS, consultez. Démos de FreeRTOS

Assurez-vous d'inclure votre certificat de signature de code, votre clé privée et votre chaîne de confiance du certificat dans le projet FreeRTOS ou sur votre appareil.

À l'aide de l'outil approprié, gravez l'application sur votre carte et assurez-vous qu'elle s'exécute correctement.

Mettre à jour la version de votre microprogramme

L'agent OTA inclus dans FreeRTOS vérifie la version de toute mise à jour et ne l'installe que si elle est plus récente que la version du microprogramme existante. Les étapes suivantes vous montrent comment incrémenter la version du microprogramme de l'application de démonstration OTA.

1. Ouvrez le projet `aws_demos` dans votre IDE.
2. Localisez le fichier `/vendors/vendor/boards/board/aws_demos/config_files/ota_demo_config.h` et incrémentez la valeur de APP_VERSION_BUILD.
3. Pour planifier une mise à jour d'une plate-forme Renesas rx65n avec un type de fichier autre que 0 (fichiers autres que le microprogramme), vous devez signer le fichier avec l'outil Renesas Secure Flash Programmer, sinon la vérification de signature sur l'appareil échouera. L'outil crée un package de fichiers signé avec l'extension `.rsu` qui est un type de fichier propriétaire pour Renesas. L'outil se trouve sur Github. Vous pouvez utiliser l'exemple de commande suivant pour générer l'image :

   ```
   "Renesas Secure Flash Programmer.exe" CUI Update "RX65N(ROM 2MB)/Secure
    Bootloader=256KB" "sig-sha256-ecdsa" 1 "file_name" "output_file_name.rsu"
   ```

4. Regénérez le projet.

Vous devez copier la mise à jour du microprogramme dans le compartiment Amazon S3 que vous avez créé, comme décrit dansCréez un compartiment Amazon S3 pour stocker votre mise à jour. Le nom du fichier que vous devez copier sur Amazon S3 dépend de la plate-forme matérielle que vous utilisez :

- Instruments texans CC3220SF-LAUNCHXL : `vendors/ti/boards/cc3220_launchpad/aws_demos/ccs/debug/aws_demos.bin`
- Espressif ESP32 : `vendors/espressif/boards/esp32/aws_demos/make/build/aws_demos.bin`

Création d'une mise à jour OTA (AWS IoT console)

1. Dans le volet de navigation de la AWS IoT console, sous Gérer, sélectionnez Actions à distance, puis choisissez Jobs.
2. Choisissez Créer une tâche.
3. Sous Type de tâche, sélectionnez Créer une tâche de mise à jour FreeRTOS OTA, puis choisissez Next.
4. Dans Propriétés de la tâche, entrez un nom de tâche et (éventuellement) une description de la tâche, puis choisissez Next.
5. Vous pouvez déployer une mise à jour OTA sur un seul appareil ou sur un groupe d'appareils. Sous Appareils à mettre à jour, sélectionnez un ou plusieurs objets ou groupes d'objets dans la liste déroulante.
6. Sous Sélectionnez le protocole pour le transfert de fichiers, sélectionnez HTTP ou MQTT, ou sélectionnez les deux pour permettre à chaque appareil de déterminer le protocole à utiliser.
7. Sous Signer et choisissez votre fichier, sélectionnez Signer un nouveau fichier pour moi.
8. Sous Profil de signature de code, choisissez Créer un nouveau profil.
9. Dans Créer un profil de signature de code, saisissez un nom pour votre profil de signature de code.

    a. Sous Plate-forme matérielle de l'appareil, choisissez votre plateforme matérielle.

> **Note**
>
> Seules les plateformes matérielles qualifiées pour FreeRTOS sont affichées dans cette liste. Si vous testez une plate-forme non qualifiée et que vous utilisez la P-256 SHA-256 suite de chiffrement ECDSA pour la signature, vous pouvez choisir le profil de signature de code Windows Simulator pour produire une signature compatible. Si vous utilisez une plate-forme non qualifiée et que vous utilisez une suite de chiffrement autre que l'ECDSA P-256 SHA-256 pour la signature, vous pouvez utiliser Code Signing pour AWS IoT, ou vous pouvez signer vous-même la mise à jour de votre microprogramme. Pour de plus amples informations, veuillez consulter Signature numérique de votre mise à jour de microprogramme.

b. Sous Certificat de signature de code, choisissez Sélectionner un certificat existant, puis sélectionnez un certificat précédemment importé, ou choisissez Importer un nouveau certificat de signature de code, choisissez vos fichiers et sélectionnez Importer pour importer un nouveau certificat.

c. Sous Nom de chemin du certificat de signature de code sur l'appareil, entrez le nom complet du chemin d'accès au certificat de signature de code sur votre appareil. Pour la plupart des appareils, vous pouvez laisser ce champ vide. Pour le simulateur Windows et pour les appareils qui placent le certificat dans un emplacement de fichier spécifique, entrez le chemin ici.

> **Important**
>
> Sur le Texas Instruments CC3220SF-LAUNCHXL, n'insérez pas de barre oblique (**/**) devant le nom du fichier si votre certificat de signature de code existe à la racine du système de fichiers. Sinon, la mise à jour OTA échoue lors de l'authentification avec une erreur `file not found`.

d. Sélectionnez Créer.

10. Sous Fichier, sélectionnez Sélectionnez un fichier existant, puis choisissez Parcourir S3. La liste de vos compartiments Amazon S3 s'affiche. Choisissez le compartiment qui contient la mise à jour du microprogramme, puis choisissez la mise à jour dans le compartiment.

> **Note**
>
> Les projets de démonstration Microchip Curiosity PIC32MZEF produisent deux images binaires avec les noms par défaut `mplab.production.bin` et `mplab.production.ota.bin`. Utilisez le second fichier lorsque vous chargez une image pour la mise à jour OTA.

11. Sous Nom du fichier sur l'appareil, entrez le nom du chemin complet vers l'emplacement sur votre appareil où la tâche OTA copiera l'image du microprogramme. Cet emplacement varie en fonction de la plateforme.

    > **Important**
    >
    > Sur le Texas Instruments CC3220SF-LAUNCHXL, pour des raisons de sécurité, le nom du chemin de l'image du microprogramme doit être `/sys/mcuflashimg.bin`.

12. Ouvrez le type de fichier et entrez une valeur entière comprise entre 0 et 255. Le type de fichier que vous entrez sera ajouté au document Job envoyé au microcontrôleur. Le firmware/software développeur du microcontrôleur est pleinement responsable de l'utilisation de cette valeur. Les scénarios possibles incluent un microcontrôleur doté d'un processeur secondaire dont le microprogramme peut être mis à jour indépendamment du processeur principal. Lorsque l'appareil reçoit une tâche de mise à jour OTA, il peut utiliser le type de fichier pour identifier le processeur auquel la mise à jour est destinée.
13. Sous Rôle IAM, choisissez un rôle conformément aux instructions figurant dansCréer un rôle de service de mise à jour OTA.
14. Choisissez Suivant.
15. Entrez un ID et une description pour votre tâche de mise à jour OTA.
16. Sous Type de tâche, choisissez Votre tâche sera terminée après le déploiement sur le poste sélectionné devices/groups (instantané).
17. Choisissez toutes les configurations facultatives appropriées pour votre tâche (Configuration du lancement des exécutions de tâche, Annulation des tâches, Délai d'expiration des exécutions de tâche et Balises).
18. Choisissez Créer.

Pour utiliser une image de microprogramme précédemment signée

1. Sous Sélectionner et signer votre image de microprogramme, choisissez Sélectionner une image de microprogramme précédemment signée.
2. Sous Nom de chemin de l'image du microprogramme sur l'appareil, entrez le nom complet du chemin d'accès à l'emplacement, sur votre appareil, où la tâche OTA copiera l'image du microprogramme. Cet emplacement varie en fonction de la plateforme.
3. Sous Tâche de signature de code précédente, choisissez Sélectionner, puis choisissez la tâche de signature de code précédente utilisée pour signer l'image du microprogramme que vous utilisez pour la mise à jour OTA.

Utilisation d'une image de microprogramme signée personnalisée

1. Sous Sélectionner et signer votre image de microprogramme, choisissez Utiliser mon image de microprogramme signée personnalisée.
2. Sous Nom de chemin du certificat de signature de code sur l'appareil, entrez le nom complet du chemin d'accès au certificat de signature de code sur votre appareil. Pour la plupart des appareils, vous pouvez laisser ce champ vide. Pour le simulateur Windows et pour les appareils qui placent le certificat dans un emplacement de fichier spécifique, entrez le chemin ici.
3. Sous Nom de chemin de l'image du microprogramme sur l'appareil, entrez le nom complet du chemin d'accès à l'emplacement, sur votre appareil, où la tâche OTA copiera l'image du microprogramme. Cet emplacement varie en fonction de la plateforme.
4. Sous Signature, collez votre signature au format PEM.
5. Sous Algorithme de hachage d'origine, sélectionnez l'algorithme de hachage utilisé lors de la création de votre signature de fichier.
6. Sous Algorithme de chiffrement d'origine, sélectionnez l'algorithme utilisé lors de la création de votre signature de fichier.
7. Sous Sélectionnez votre image de microprogramme dans Amazon S3, choisissez le compartiment Amazon S3 et l'image de microprogramme signée dans le compartiment Amazon S3.

Une fois que vous avez spécifié les informations de signature de code, spécifiez le type de tâche de la mise à jour OTA, le rôle de service et un ID pour votre mise à jour.

> **Note**
>
> N'utilisez pas d'informations personnelles identifiables dans l'ID de tâche de votre mise à jour OTA. Voici quelques exemples d'informations personnelles identifiables :
>
> - Noms
> - Adresses IP
> - Adresses e-mail
> - Adresses
> - Informations bancaires.
> - Informations médicales.

1. Sous Type de tâche, choisissez Votre tâche sera terminée après le déploiement sur le poste sélectionné devices/groups (instantané).
2. Sous Rôle IAM pour la tâche de mise à jour OTA, choisissez votre rôle de service OTA.
3. Entrez un ID alphanumérique pour votre tâche, puis choisissez Créer.

La tâche apparaît dans la AWS IoT console avec le statut EN COURS.

> **Note**
>
> - La AWS IoT console ne met pas automatiquement à jour l'état des tâches. Actualisez votre navigateur pour voir les mises à jour.

Connectez votre terminal UART série à votre appareil. Vous devez voir une sortie qui indique que l'appareil télécharge le microprogramme mis à jour.

Une fois que l'appareil a téléchargé le microprogramme mis à jour, il redémarre, puis installe le microprogramme. Vous pouvez voir ce qui se passe dans le terminal UART.

Pour obtenir un didacticiel qui montre comment utiliser la console pour créer une mise à jour OTA, veuillez consulter Over-the-air met à jour l'application de démonstration.

Création d'une mise à jour OTA avec AWS CLI

Lorsque vous utilisez le AWS CLI pour créer une mise à jour OTA, vous :

1. Signez numériquement votre image de microprogramme.
2. Créez un flux de votre image de microprogramme signée numériquement.
3. Démarrez une tâche de mise à jour OTA.

Signature numérique de votre mise à jour de microprogramme

Lorsque vous utilisez le AWS CLI pour effectuer des mises à jour OTA, vous pouvez utiliser la signature de code pour AWS IoT, ou vous pouvez signer vous-même la mise à jour de votre microprogramme. Pour obtenir la liste des algorithmes de signature et de hachage cryptographiques pris en charge par Code Signing for AWS IoT, consultez. SigningConfigurationOverrides Si vous souhaitez utiliser un algorithme cryptographique qui n'est pas pris en charge par Code Signing for AWS IoT, vous devez signer le binaire de votre microprogramme avant de le télécharger sur Amazon S3.

Signature de l'image de votre microprogramme avec Code Signing pour AWS IoT

Pour signer l'image de votre microprogramme à l'aide de Code Signing for AWS IoT, vous pouvez utiliser l'un des AWS SDK ou des outils de ligne de commande. Pour plus d'informations sur la signature de code pour AWS IoT, consultez la section Signature de code pour AWS IoT.

Après avoir installé et configuré les outils de signature de code, copiez l'image de votre microprogramme non signée dans votre compartiment Amazon S3 et lancez une tâche de signature de code à l'aide des commandes suivantes. AWS CLI La commande put-signing-profile crée un profil de signature de code réutilisable. La commande start-signing-job démarre la tâche de signature.

```
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*et il *your-destination-bucket-name* peut s'agir du même compartiment Amazon S3.

Voici les paramètres pour les commandes put-signing-profile et start-signing-job :

`source`

Spécifie l'emplacement du microprogramme non signé dans un compartiment S3.

- `bucketName` : nom de votre compartiment S3.
- `key` : clé (nom de fichier) de votre microprogramme dans votre compartiment S3.
- `version` : version S3 de votre microprogramme dans votre compartiment S3. Cela est différent de votre version de microprogramme. Vous pouvez le trouver en accédant à la console Amazon S3, en choisissant votre compartiment, et en haut de la page, à côté de Versions, en choisissant Afficher.

`destination`

Destination sur le périphérique vers laquelle le microprogramme signé dans le compartiment S3 sera copié. Le format de ce paramètre est le même que le paramètre `source`.

`signing-material`

L'ARN de votre certificat de signature de code. Cet ARN est généré lorsque vous importez votre certificat dans ACM.

`signing-parameters`

Tableau de mappage des paires clé-valeur pour la signature. Ces informations peuvent inclure toutes celles que vous souhaitez utiliser lors de la signature.

> **Note**
>
> Ce paramètre est obligatoire lorsque vous créez un profil de signature de code pour la signature des mises à jour OTA avec la signature de code pour AWS IoT.

`platform`

Valeur `platformId` de la plateforme matérielle sur laquelle vous distribuez la mise à jour OTA.

Pour renvoyer la liste des plateformes disponibles et leurs valeurs `platformId`, utilisez la commande `aws signer list-signing-platforms`.

La tâche de signature démarre et écrit l'image du microprogramme signée dans le compartiment Amazon S3 de destination. Le nom de fichier de l'image de microprogramme signée est un GUID. Vous avez besoin de ce nom de fichier lorsque vous créez un flux. Vous pouvez trouver le nom du fichier en accédant à la console Amazon S3 et en choisissant votre compartiment. Si vous ne voyez pas de fichier avec un nom de fichier GUID, actualisez votre navigateur.

La commande affiche un ARN de tâche et un ID de tâche. Vous aurez besoin de ces valeurs plus tard. Pour plus d'informations sur la signature de code pour AWS IoT, consultez la section Signature de code pour AWS IoT.

Signature manuelle de votre image de microprogramme

Signez numériquement l'image de votre microprogramme et téléchargez l'image de microprogramme signée dans votre compartiment Amazon S3.

Création d'un flux de votre mise à jour du microprogramme

Un flux est une interface abstraite avec les données, qui peut être utilisée par un périphérique. Le flux masque la complexité de l'accès aux données stockées dans différents emplacements ou différents services basés sur le cloud. Le service OTA Update Manager vous permet d'utiliser plusieurs données, stockées à différents endroits dans Amazon S3, pour effectuer une mise à jour OTA.

Lorsque vous créez une mise à jour AWS IoT OTA, vous pouvez également créer un flux contenant votre mise à jour de microprogramme signée. Créez un fichier JSON (`stream.json`) qui identifie votre image de microprogramme signée. Le fichier JSON doit contenir ce qui suit.

```
[
  {
    "fileId":"your_file_id",
    "s3Location":{
      "bucket":"your_bucket_name",
      "key":"your_s3_object_key"
    }
```

```
  }
]
```

Voici les attributs dans le fichier JSON :

`fileId`

Nombre entier arbitraire compris entre 0 et 255 qui identifie l'image de votre microprogramme.

`s3Location`

Le compartiment et la clé du microprogramme pour la création du flux.

`bucket`

Le compartiment Amazon S3 dans lequel est stockée votre image de microprogramme non signée.

`key`

Le nom de fichier de l'image de microprogramme signée dans le compartiment Amazon S3. Vous pouvez trouver cette valeur dans la console Amazon S3 en consultant le contenu de votre compartiment.

Si vous utilisez Code Signing pour AWS IoT, le nom du fichier est un GUID généré par Code Signing for AWS IoT.

Pour créer un flux, utilisez la commande create-stream AWS CLI .

```
aws iot create-stream \
    --stream-id your_stream_id \
    --description your_description \
    --files file://stream.json \
    --role-arn your_role_arn
```

Voici les arguments de la create-stream AWS CLI commande :

`stream-id`

Chaîne arbitraire d'identification du flux.

`description`

Description facultative du flux.

`files`

Une ou plusieurs références à des fichiers JSON qui contiennent des données sur les images de microprogramme pour créer le flux. Le fichier JSON doit contenir les attributs suivants :

`fileId`

Un ID fichier arbitraire.

`s3Location`

Le nom du compartiment dans lequel l'image de micrologiciel signée est stockée et la clé (nom de fichier) de l'image de microprogramme signée.

`bucket`

Le compartiment Amazon S3 dans lequel l'image du microprogramme signée est stockée.

`key`

La clé (nom de fichier) de l'image de microprogramme signée.

Lorsque vous utilisez la signature de code pour AWS IoT, cette clé est un GUID.

Voici un exemple de fichier `stream.json`.

```
[
    {
        "fileId":123,
        "s3Location": {
            "bucket":"codesign-ota-bucket",
            "key":"48c67f3c-63bb-4f92-a98a-4ee0fbc2bef6"
        }
    }
]
```

`role-arn`

Rôle de service OTA qui accorde également l'accès au compartiment Amazon S3 dans lequel l'image du microprogramme est stockée.

Pour trouver la clé d'objet Amazon S3 de votre image de microprogramme signée, utilisez la aws signer describe-signing-job --job-id ***`my-job-id`*** commande où se `my-job-id` trouve l'ID de tâche

affiché par la create-signing-job AWS CLI commande. La sortie de la commande describe-signing-job contient la clé de l'image de microprogramme signée.

```
... text deleted for brevity ...
  "signedObject": {
    "s3": {
      "bucketName": "ota-bucket",
      "key": "7309da2c-9111-48ac-8ee4-5a4262af4429"
    }
  }
... text deleted for brevity ...
```

Création d'une mise à jour OTA

Utilisez la create-ota-update AWS CLI commande pour créer une tâche de mise à jour OTA.

> **Note**
>
> N'utilisez pas d'informations personnelles identifiables dans l'ID de tâche de votre mise à jour OTA. Voici quelques exemples d'informations personnelles identifiables :
>
> - Noms
> - Adresses IP
> - Adresses e-mail
> - Adresses
> - Informations bancaires.
> - Informations médicales.

```
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]
```

Format de `cli-input-json`

```
{
  "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"
    }
  ]
}
```

champs cli-input-json

Nom	Type	Description
otaUpdateId	chaîne (max : 128, min : 1)	ID de la mise à jour OTA à créer.
description	chaîne (max : 2028)	Description de la mise à jour OTA.
targets	liste	Appareils ciblés pour recevoir les mises à jour OTA.
protocols	liste	Protocole utilisé pour transférer l'image de mise à jour OTA. Les valeurs valides sont [HTTP], [MQTT], [HTTP, MQTT]. Lorsque HTTP et MQTT sont tous deux

Nom	Type	Description
		spécifiés, le dispositif cible peut choisir le protocole.
`targetSelection`	chaîne	Indique si la mise à jour continuera à s'exécuter (CONTINUOUS) ou sera terminée une fois que tous les objets spécifiés comme cibles l'auront effectuée (SNAPSHOT). Si elle est continue, la mise à jour peut également être exécutée sur un objet lorsqu'une modificat ion est détectée dans une cible. Par exemple, une mise à jour s'exécute sur un objet lorsque ce dernier est ajouté à un groupe cible, même après que la mise à jour a été effectuée par tous les objets qui se trouvaient à l'origine dans le groupe. Valeurs valides : CONTINUOUS | SNAPSHOT. énum : CONTINUOUS | SNAPSHOT
`awsJobExecutionsRo lloutConfig`		Configuration pour le déploiement des mises à jour OTA.
`maximumPerMinute`	entier (max : 1 000, min : 1)	Nombre maximal d'exécutions de tâches de mise à jour OTA commencées par minute.

Nom	Type	Description
`exponentialRate`		Taux d'augmentation pour le déploiement d'une tâche. Ce paramètre vous permet de définir un accroissement du taux exponentiel pour le déploiement d'une tâche.
`baseRatePerMinute`	entier (max : 1 000, min : 1)	Nombre minimal d'objets qui seront informés de la présence d'une tâche en attente, par minute, au début du déploiement d'une tâche. Il s'agit du taux initial de déploiement.
`rateIncreaseCriteria`		Critères pour initier l'augment ation du taux de déploiement pour une tâche. AWS IoT prend en charge jusqu'à un chiffre après la virgule (par exemple, 1,5, mais pas 1,55).
`numberOfNotifiedTh ings`	entier (min : 1)	Lorsque ce nombre d'objets a été notifié, cela entraîne une augmentation du taux de déploiement.
`numberOfSucceededT hings`	entier (min : 1)	Lorsque ce nombre d'objets a réussi l'exécution de leur tâche, cela entraîne une augmentation du taux de déploiement.

Nom	Type	Description
`awsJobPresignedUrlConfig`		Informations de configuration pour les URL présignées.
`expiresInSec`	long	Durée de validité (en secondes) des URL présignées. Les valeurs valides sont comprises entre 60 et 3 600, la valeur par défaut est de 1 800 secondes. Pre-signed Les URL sont générées lorsqu'une demande pour le document de travail est reçue.
`awsJobAbortConfig`		Les critères qui déterminent quand et comment un arrêt de travail a lieu.
`abortCriteriaList`	liste	Liste des critères qui déterminent quand et comment arrêter le travail.
`failureType`	chaîne	Type d'échec de l'exécution d'une tâche susceptible de provoquer un arrêt de tâche. enum : FAILED | REJECTED | TIMED_OUT | ALL
`action`	chaîne	Type de mesure à prendre pour déclencher l'arrêt de travail. enum : CANCEL

Nom	Type	Description
`minNumberOfExecutedThings`	entier (min : 1)	Le nombre minimum d'objets qui doivent recevoir des notifications d'exécution de la tâche avant que la tâche puisse être arrêtée.
`awsJobTimeoutConfig`		Spécifie la durée pendant laquelle chaque appareil doit terminer son exécution de la tâche. Un minuteur est démarré quand l'état de l'exécution de la tâche a la valeur `IN_PROGRESS` . Si l'état de l'exécution de la tâche n'est pas défini sur un autre état terminal avant l'expiration du minuteur, il sera automatiquement défini avec la valeur `TIMED_OUT` .

Nom	Type	Description
`inProgressTimeoutInMinutes`	long	Spécifie la durée (en minutes) pendant laquelle cet appareil doit terminer l'exécution de la tâche. Le délai peut être n'importe quelle valeur comprise entre 1 minute et 7 jours (1 à 10 080 minutes). Le minuteur en cours ne peut pas être mis à jour et s'applique à toutes les exécutions de tâche pour la tâche. Chaque fois qu'une tâche d'exécution demeure dans l'état IN_PROGRESS plus longtemps que l'intervalle défini, la tâche d'exécution échoue et passe à l'état terminal `TIMED_OUT` .
`files`	liste	Fichiers qui seront diffusés par la mise à jour OTA.
`fileName`	chaîne	Nom du fichier.
`fileType`	entier plage - max. : 255 min. : 0	Une valeur entière que vous pouvez inclure dans le document de travail pour permettre à vos appareils d'identifier le type de fichier reçu du cloud.
`fileVersion`	chaîne	Version du fichier.
`fileLocation`		Emplacement du microprogramme mis à jour.

Nom	Type	Description
`stream`		Flux contenant la mise à jour OTA.
`streamId`	chaîne (max : 128, min : 1)	ID du flux.
`fileId`	entier (max : 255, min : 0)	ID d'un fichier associé à un flux.
`s3Location`		Emplacement du microprog ramme mis à jour dans S3.
`bucket`	chaîne (min : 1)	Compartiment S3.
`key`	chaîne (min : 1)	Clé d'S3
`version`	chaîne	Version du compartiment S3.
`codeSigning`		Méthode de signature de code du fichier.
`awsSignerJobId`	chaîne	L'ID du AWSSignerJob qui a été créé pour signer le fichier.
`startSigningJobPar ameter`		Décrire la tâche de signature du code.
`signingProfilePara meter`		Décrit le profil de signature du code.
`certificateArn`	chaîne	ARN de certificat.

Nom	Type	Description
`platform`	chaîne	Plateforme matérielle de votre appareil.
`certificatePathOnDevice`	chaîne	Emplacement du certificat de signature de code sur votre appareil.
`signingProfileName`	chaîne	Nom du profil de signature du code.
`destination`		Emplacement pour écrire fichier dont le code est signé.
`s3Destination`		Décrit l'emplacement dans S3 du microprogramme mis à jour.
`bucket`	chaîne (min : 1)	Compartiment S3 qui contient le microprogramme mis à jour.
`prefix`	chaîne	Préfixe S3.
`customCodeSigning`		Méthode personnalisée pour la signature de code d'un fichier.
`signature`		Signature du fichier.
`inlineDocument`	blob	Représentation binaire codée en base64 de la signature de code.
`certificateChain`		Chaîne de certificats.
`certificateName`	chaîne	Nom du certificat.

Nom	Type	Description
`inlineDocument`	chaîne	Représentation binaire codée en base64 de la chaîne de certificats de signature de code.
`hashAlgorithm`	chaîne	Algorithme de hachage utilisé pour la signature de code du fichier.
`signatureAlgorithm`	chaîne	Algorithme de signature utilisé pour la signature de code du fichier.
`attributes`	map	Une liste de name/attribute paires.
`roleArn`	chaîne (max : 2048, min : 20)	Rôle IAM qui donne AWS IoT accès à Amazon S3, aux AWS IoT tâches et aux ressource s de signature de AWS code pour créer une tâche de mise à jour OTA.
`additionalParameters`	map	Liste de paramètres de mise à jour OTA supplémentaires qui sont des paires nom-valeur.
`tags`	liste	Métadonnées qui peuvent être utilisées pour gérer les mises à jour.
`Key`	chaîne (max : 128, min : 1)	Clé de la balise.

Nom	Type	Description
`Value`	chaîne (max : 256, min : 1)	Valeur de la balise.

Output

```
{
  "otaUpdateId": "string",
  "awsIotJobId": "string",
  "otaUpdateArn": "string",
  "awsIotJobArn": "string",
  "otaUpdateStatus": "string"
}
```

AWS CLI champs de sortie

Nom	Type	Description
`otaUpdateId`	chaîne (max : 128, min : 1)	ID de mise à jour OTA.
`awsIotJobId`	chaîne	L'ID de AWS IoT tâche associé à la mise à jour OTA.
`otaUpdateArn`	chaîne	ARN de mise à jour OTA.
`awsIotJobArn`	chaîne	L'ARN de la AWS IoT tâche associé à la mise à jour OTA.
`otaUpdateStatus`	chaîne	Statut de la mise à jour OTA. énum : CREATE_PE NDING CREATE_IN _PROGRESS CREATE_FA ILED CREATE_COMPLETE | | |

Voici un exemple de fichier JSON transmis à la create-ota-update commande qui utilise la signature de code pour AWS IoT.

```
[
  {
    "fileName": "firmware.bin",
    "fileType": 1,
    "fileLocation": {
      "stream": {
        "streamId": "004",
        "fileId":123
      }
    },
    "codeSigning": {
      "awsSignerJobId": "48c67f3c-63bb-4f92-a98a-4ee0fbc2bef6"
    }
  }
]
```

Voici un exemple de fichier JSON transmis à la create-ota-update AWS CLI commande qui utilise un fichier en ligne pour fournir du matériel de signature de code personnalisé.

```
[
  {
    "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"
```

```
      }
    }
  }
]
```

Voici un exemple de fichier JSON transmis à la create-ota-update AWS CLI commande qui permet à FreeRTOS OTA de démarrer une tâche de signature de code et de créer un profil et un flux de signature de code.

```
[
  {
    "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"
          }
        }
      }
    }
  }
]
```

Voici un exemple de fichier JSON transmis à la create-ota-update AWS CLI commande qui crée une mise à jour OTA qui lance une tâche de signature de code avec un profil existant et utilise le flux spécifié.

```
[
  {
    "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"
          }
        }
      }
    }
  }
]
```

Voici un exemple de fichier JSON transmis à la create-ota-update AWS CLI commande qui permet à FreeRTOS OTA de créer un flux avec un identifiant de tâche de signature de code existant.

```
[
  {
    "fileName": "your_firmware_path_on_device",
    "fileType": 1,
    "fileVersion": "1",
    "codeSigning":{
      "awsSignerJobId": "your_signer_job_id"
    }
  }
]
```

Voici un exemple de fichier JSON transmis à la create-ota-update AWS CLI commande qui crée une mise à jour OTA. La mise à jour crée un flux à partir de l'objet S3 spécifié et utilise une signature de code personnalisée.

```
[
  {
    "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"
      }
    }
  }
]
```

Affichage des mises à jour OTA

Vous pouvez utiliser la list-ota-updates AWS CLI commande pour obtenir une liste de toutes les mises à jour OTA.

```
aws iot list-ota-updates
```

La sortie de la commande list-ota-updates se présente comme suit.

```
{
  "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
    }
  ]
}
```

Obtention d'informations sur une mise à jour OTA

Vous pouvez utiliser la get-ota-update AWS CLI commande pour obtenir l'état de création ou de suppression d'une mise à jour OTA.

```
aws iot get-ota-update --ota-update-id your-ota-update-id
```

La sortie de la commande get-ota-update ressemble à ce qui suit.

```
{
    "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"
    }
}
```

Les valeurs renvoyées pour `otaUpdateStatus` incluent les éléments suivants :

`CREATE_PENDING`

La création d'une mise à jour OTA est en attente.

`CREATE_IN_PROGRESS`

Une mise à jour OTA est en cours de création.

`CREATE_COMPLETE`

Une mise à jour OTA a été créée.

CREATE_FAILED

La création d'une mise à jour OTA a échoué.

DELETE_IN_PROGRESS

Une mise à jour OTA est en cours de suppression.

DELETE_FAILED

La suppression d'une mise à jour OTA a échoué.

> **Note**
>
> Pour obtenir l'état d'exécution d'une mise à jour OTA après sa création, vous devez utiliser la commande describe-job-execution. Pour plus d'informations, consultez Describe Job Execution.

Supprimer des OTA-related données

Actuellement, vous ne pouvez pas utiliser la AWS IoT console pour supprimer des flux ou des mises à jour OTA. Vous pouvez utiliser le AWS CLI pour supprimer les flux, les mises à jour OTA et les AWS IoT tâches créées lors d'une mise à jour OTA.

Suppression d'un flux OTA

Lorsque vous créez une mise à jour OTA qui utilise MQTT, vous pouvez utiliser la ligne de commande ou la AWS IoT console pour créer un flux afin de diviser le microprogramme en plusieurs parties afin qu'il puisse être envoyé via MQTT. Vous pouvez supprimer ce flux à l'aide de la delete-stream AWS CLI commande, comme indiqué dans l'exemple suivant.

```
aws iot delete-stream --stream-id your_stream_id
```

Suppression d'une mise à jour OTA

Lorsque vous créez une mise à jour OTA, les éléments suivants sont créés :

- Une entrée dans la base de données des tâches de mise à jour OTA.
- Une AWS IoT tâche pour effectuer la mise à jour.

- Exécution AWS IoT d'une tâche pour chaque appareil mis à jour.

La commande delete-ota-update supprime l'entrée dans la base de données des tâches de mise à jour OTA uniquement. Vous devez utiliser la commande delete-job pour supprimer la tâche AWS IoT .

Utilisez la commande delete-ota-update pour supprimer une mise à jour OTA.

```
aws iot delete-ota-update --ota-update-id your_ota_update_id
```

`ota-update-id`

ID de la mise à jour OTA à supprimer.

`delete-stream`

Supprime le flux associé à la mise à jour OTA.

`force-delete-aws-job`

Supprime la AWS IoT tâche associée à la mise à jour OTA. Si cet indicateur n'est pas défini et que la tâche est dans l'état `In_Progress`, la tâche n'est pas supprimée.

Suppression d'une tâche IoT créée pour une mise à jour OTA

FreeRTOS crée AWS IoT une tâche lorsque vous créez une mise à jour OTA. Une exécution de tâche est également créée pour chaque appareil qui traite la tâche. Vous pouvez utiliser la delete-job AWS CLI commande pour supprimer une tâche et les exécutions de tâches associées.

```
aws iot delete-job --job-id your-job-id --no-force
```

Le paramètre `no-force` spécifie que seules les tâches qui sont dans un état final (COMPLETED ou CANCELLED) peuvent être supprimées. Vous pouvez supprimer une tâche qui est dans un état non final en transmettant le paramètre `force`. Pour plus d'informations, consultez DeleteJob API.

> **Note**
>
> La suppression d'une tâche avec le statut IN_PROGRESS interrompt toutes les exécutions de tâche IN_PROGRESS sur vos appareils et peut se traduire par le fait qu'un appareil se retrouve dans un état non déterministe. Assurez-vous que chaque appareil exécutant une tâche qui a été supprimée peut être rétabli dans un état connu.

Selon le nombre d'exécutions de tâche créées pour la tâche et plusieurs autres facteurs, la suppression d'une tâche peut prendre du temps. Pendant la suppression de la tâche, son statut est DELETION_IN_PROGRESS. Toute tentative de suppression ou d'annulation d'une tâche dont le statut est DELETION_IN_PROGRESS entraîne une erreur.

Vous pouvez utiliser la commande delete-job-execution pour supprimer une exécution de tâche. Il se peut que vous souhaitiez supprimer une exécution de tâche lorsqu'un petit nombre d'appareils ne sont pas en mesure de traiter une tâche. Cela supprime l'exécution de tâche pour un seul périphérique, comme illustré dans l'exemple suivant.

```
aws iot delete-job-execution --job-id your-job-id --thing-name
                    your-thing-name --execution-number your-job-execution-number --no-
force
```

Comme pour la delete-job AWS CLI commande, vous pouvez passer le `--force` paramètre au pour delete-job-execution forcer la suppression de l'exécution d'une tâche. Pour plus d'informations, consultez la section DeleteJobExecutionAPI.

> **Note**
>
> La suppression d'une exécution de tâche avec le statut IN_PROGRESS interrompt toutes les exécutions de tâche IN_PROGRESS sur vos appareils et peut se traduire par le fait qu'un appareil se retrouve dans un état non déterministe. Assurez-vous que chaque appareil exécutant une tâche qui a été supprimée peut être rétabli dans un état connu.

Pour plus d'informations sur l'application de démonstration de mise à jour OTA, consultez Over-the-air met à jour l'application de démonstration.

Service Gestionnaire de mise à jour OTA

Voici les opérations possibles avec le service Gestionnaire de mise à jour OTA (Over-the-Air) :

- Créez une mise à jour OTA et les ressources qu'elle utilise, notamment une AWS IoT tâche, un AWS IoT flux et une signature de code.
- Obtenir des informations sur une mise à jour OTA.
- Répertoriez toutes les mises à jour OTA associées à votre AWS compte.
- Supprime une mise à jour OTA.

Une mise à jour OTA est une structure de données gérée par le service Gestionnaire de mise à jour OTA. Il contient :

- Un ID de mise à jour OTA.
- Une description de mise à jour OTA facultative.
- La liste des appareils à mettre à jour (cibles).
- Le type de mise à jour OTA : CONTINUOUS ou SNAPSHOT. Consultez la section Offres d'emploi du Guide du AWS IoT développeur pour une discussion sur le type de mise à jour dont vous avez besoin.
- Protocole utilisé pour effectuer la mise à jour OTA : [MQTT], [HTTP] ou [MQTT, HTTP]. Lorsque vous spécifiez MQTT et HTTP, la configuration du périphérique détermine le protocole utilisé.
- La liste des fichiers à envoyer aux appareils cible.
- Rôle IAM qui donne AWS IoT accès à Amazon S3, aux AWS IoT tâches et aux ressources de signature de AWS code pour créer une tâche de mise à jour OTA.
- Une liste facultative de paires nom défini par l'utilisateur / valeur.

Les mises à jour OTA ont été conçues pour mettre à jour le micrologiciel de l'appareil, mais vous pouvez les utiliser pour envoyer les fichiers de votre choix à un ou plusieurs appareils enregistrés auprès d'un ou de plusieurs appareils AWS IoT. Lorsque vous effectuez des mises à jour OTA de microprogramme, nous vous recommandons de les signer numériquement pour que les appareils qui les reçoivent puissent vérifier qu'elles n'ont pas été altérées en cours de route.

Vous pouvez envoyer des images de microprogramme mises à jour à l'aide du protocole HTTP ou MQTT, en fonction des paramètres que vous choisissez. Vous pouvez signer les mises à jour de votre firmware avec Code Signing for FreeRTOS ou utiliser vos propres outils de signature de code.

Pour mieux contrôler le processus, vous pouvez utiliser l' CreateStreamAPI pour créer un flux lorsque vous envoyez des mises à jour via MQTT. Dans certains cas, vous pouvez modifier le code de l'agent FreeRTOS pour ajuster la taille des blocs que vous envoyez et recevez.

Lorsque vous créez une mise à jour OTA, le service OTA Manager crée une tâche AWS IoT pour informer vos appareils qu'une mise à jour est disponible. L'agent FreeRTOS OTA s'exécute sur vos appareils et écoute les messages de mise à jour. Lorsqu'une mise à jour est disponible, elle demande l'image de mise à jour du microprogramme via HTTP ou MQTT et stocke les fichiers localement. Elle vérifie la signature numérique des fichiers téléchargés et, si elle est valide, installe la mise à jour du

microprogramme. Si vous n'utilisez pas FreeRTOS, vous devez implémenter votre propre agent OTA pour écouter et télécharger les mises à jour et effectuer toutes les opérations d'installation.

Intégration de l'agent OTA dans votre application

L'agent OTA (Over-the-Air) est conçu pour simplifier la quantité de code que vous devez écrire pour ajouter la fonctionnalité de mise à jour OTA à votre produit. Cette charge d'intégration consiste principalement à initialiser l'agent OTA et à créer une fonction de rappel personnalisée pour répondre aux messages d'événement de l'agent OTA. Lors de l'initialisation du système d'exploitation, les interfaces MQTT, HTTP (si le protocole HTTP est utilisé pour le téléchargement de fichiers) et PAL (Platform Specific Implementation) sont transmises à l'agent OTA. Les tampons peuvent également être initialisés et transmis à l'agent OTA.

> **Note**
>
> Bien que l'intégration de la fonctionnalité de mise à jour OTA dans votre application soit assez simple, le système de mise à jour OTA nécessite une compréhension qui ne se limite pas à l'intégration du code. Pour vous familiariser avec la configuration de votre AWS compte avec les AWS IoT éléments, les informations d'identification, les certificats de signature de code, les appareils d'approvisionnement et les tâches de mise à jour OTA, consultez les conditions préalables à FreeRTOS.

Gestion des connexions

L'agent OTA utilise le protocole MQTT pour toutes les opérations de communication de contrôle impliquant des AWS IoT services, mais il ne gère pas la connexion MQTT. Pour vérifier que l'agent OTA n'interfère pas avec la stratégie de gestion de la connexion de votre application, la connexion MQTT (y compris les fonctionnalités de déconnexion et de reconnexion) doit être gérée par l'application utilisateur principal. Le fichier peut être téléchargé via le protocole MQTT ou HTTP. Vous pouvez choisir le protocole lors de la création de la tâche OTA. Si vous choisissez MQTT, l'agent OTA utilise la même connexion pour les opérations de contrôle et pour le téléchargement de fichiers.

Démo OTA simple

Voici un extrait d'une démonstration OTA simple qui illustre comment l'agent se connecte à l'agent MQTT et initialise l'agent OTA. Dans cet exemple, nous configurons la démo pour utiliser le rappel de

l'application OTA par défaut et pour renvoyer certaines statistiques une fois par seconde. Pour des raisons de concision, nous ignorons certains détails de la démonstration.

La démo OTA montre également la gestion des connexions MQTT en surveillant le rappel de déconnexion et en rétablissant la connexion. Lorsqu'une déconnexion se produit, la démo suspend d'abord les opérations de l'agent OTA, puis tente de rétablir la connexion MQTT. Les tentatives de reconnexion MQTT sont retardées d'un temps qui augmente de façon exponentielle jusqu'à une valeur maximale et une instabilité est également ajoutée. Si la connexion est rétablie, l'agent OTA poursuit ses opérations.

Pour un exemple pratique utilisant le broker AWS IoT MQTT, consultez le code de démonstration OTA dans le `demos/ota` répertoire.

Étant donné que l'agent OTA est sa propre tâche, le délai intentionnel d'une seconde de cet exemple affecte cette application uniquement. Il n'a aucun impact sur les performances de l'agent.

```
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;
}
```

Voici le flux de haut niveau de cette application de démonstration :

- Créez un contexte d'agent MQTT.
- Connectez-vous à votre AWS IoT terminal.
- Initialisez l'agent OTA.
- Boucle qui permet une tâche de mise à jour OTA et produit des statistiques une fois par seconde.
- Si MQTT se déconnecte, suspendez les opérations de l'agent OTA.
- Essayez de vous reconnecter avec un délai et une instabilité exponentiels.
- En cas de reconnexion, reprenez les opérations de l'agent OTA.
- Si l'agent s'arrête, attendez une seconde, puis essayez de vous reconnecter.

Utilisation du rappel d'application pour les événements de l'agent OTA

L'exemple précédent a été utilisé `prvOtaAppCallback` comme gestionnaire de rappel pour les événements de l'agent OTA. (Voir le quatrième paramètre de l'appel `OTA_Init` d'API). Si vous souhaitez implémenter une gestion personnalisée des événements d'achèvement, vous devez modifier la gestion par défaut dans l'OTA demo/application. Au cours du processus OTA, l'agent OTA peut envoyer l'une des énumérations d'événements suivantes au gestionnaire de rappel. Il appartient à l'application de décider quand et comment gérer ces événements.

```
/**
 * @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;
```

L'agent OTA peut recevoir une mise à jour en arrière-plan pendant le traitement actif de l'application principale. L'objectif de la fourniture de ces événements consiste à autoriser l'application à décider si l'action peut être prise immédiatement ou si elle doit être reportée après la fin du traitement spécifique à une autre application. Cela empêche une interruption imprévue de votre appareil pendant le traitement actif (par exemple, une action « vacuum ») qui serait provoquée par une

réinitialisation après une mise à jour du microprogramme. Voici les événements de tâche reçus par le gestionnaire de rappel :

`OtaJobEventActivate`

Lorsque le gestionnaire de rappel reçoit cet événement, vous pouvez soit réinitialiser l'appareil immédiatement, soit planifier un appel pour le réinitialiser ultérieurement. Cela vous permet de reporter la phase de réinitialisation de l'appareil et de test automatique, si nécessaire.

`OtaJobEventFail`

Lorsque le gestionnaire de rappel reçoit cet événement, la mise à jour a échoué. Aucune action n'est nécessaire dans ce cas. Vous pouvez, si vous le souhaitez, afficher un message de journal ou effectuer une action spécifique à l'application.

`OtaJobEventStartTest`

La phase d'autotest vise à permettre au microprogramme récemment mis à jour de s'exécuter et de se tester lui-même avant de déterminer s'il fonctionne correctement et de s'engager à être la dernière image permanente de l'application. Quand une nouvelle mise à jour est reçue et authentifiée, et que l'appareil a été réinitialisé, l'agent OTA envoie l'événement `OtaJobEventStartTest` à la fonction de rappel lorsqu'il est prêt à être testé. Le développeur peut ajouter n’importe quel test requis pour déterminer si le microprogramme de l’appareil fonctionne correctement après la mise à jour. Lorsque le microprogramme de l'appareil est considéré comme fiable par les tests automatiques, le code doit valider le microprogramme en tant que nouvelle image permanente en appelant la fonction `OTA_SetImageState( OtaImageStateAccepted )`.

`OtaJobEventProcessed`

L'événement OTA mis en file d'attente `OTA_SignalEvent` est traité, ce qui permet d'effectuer des opérations de nettoyage telles que la libération des tampons OTA.

`OtaJobEventSelfTestFailed`

L'autotest OTA a échoué pour la tâche en cours. La gestion par défaut de cet événement consiste à arrêter l'agent OTA et à le redémarrer afin que le périphérique revienne à l'image précédente.

`OtaJobEventUpdateComplete`

L'événement de notification indiquant la fin de la mise à jour des tâches OTA.

Sécurité OTA

Voici trois aspects de la sécurité OTA (Over-the-Air) :

Sécurité de connexion

Le service OTA Update Manager s'appuie sur les mécanismes de sécurité existants, tels que l'authentification mutuelle TLS (Transport Layer Security), utilisés par AWS IoT. Le trafic de mise à jour OTA passe par la passerelle de l' AWS IoT appareil et utilise des mécanismes AWS IoT de sécurité. Chaque message MQTT entrant et sortant via la passerelle d'appareils est soumis à une authentification et une autorisation strictes.

Authenticité et intégrité des mises à jour OTA

Le microprogramme peut être signé numériquement avant une mise à jour OTA pour s'assurer qu'il s'agit d'une source fiable et qu'il n'a pas été falsifié.

Le service FreeRTOS OTA Update Manager utilise la signature de code AWS IoT pour signer automatiquement votre microprogramme. Pour plus d'informations, consultez Code Signing for AWS IoT.

L'agent OTA, qui s'exécute sur vos appareils, effectue les vérifications d'intégrité sur le microprogramme lorsqu'il arrive sur l'appareil.

Sécurité de l'opérateur

Chaque appel d'API effectué via l'API du plan de contrôle est soumis à une authentification et à une autorisation standard IAM Signature version 4. Pour créer un déploiement, vous devez disposer d'autorisations pour appeler les API `CreateStream` et `CreateDeployment`, `CreateJob`. En outre, dans votre politique de compartiment ou ACL Amazon S3, vous devez accorder des autorisations de lecture au principal du AWS IoT service afin que la mise à jour du microprogramme stockée dans Amazon S3 soit accessible pendant le streaming.

Signature de code pour AWS IoT

La AWS IoT console utilise la signature de code AWS IoT pour signer automatiquement l'image de votre microprogramme pour tout appareil pris en charge par AWS IoT.

Code Signing for AWS IoT utilise un certificat et une clé privée que vous importez dans ACM. Vous pouvez utiliser un certificat autosigné à des fins de test, mais nous vous recommandons d'obtenir un certificat auprès d'une autorité de certification commerciale (CA) bien connue.

Les certificats de signature de code utilisent la X.509 version 3 `Key Usage` et `Extended Key Usage` les extensions. L'extension `Key Usage` est définie sur `Digital Signature` et l'extension `Extended Key Usage` sur `Code Signing`. Pour plus d'informations sur la signature de votre image de code, consultez le guide de signature de code pour les AWS IoT développeurs et le manuel de référence sur la signature de code pour les AWS IoT API.

> **Note**
>
> Vous pouvez télécharger le kit de signature de code pour le AWS IoT SDK depuis Tools for Amazon Web Services.

Résolution des problèmes OTA

Les sections suivantes contiennent des informations pour vous aider à résoudre les problèmes avec les mises à jour OTA.

Rubriques

- Configurer les CloudWatch journaux pour les mises à jour OTA
- Journal AWS IoT Appels d'API OTA avec AWS CloudTrail
- Obtenez les informations relatives à l'échec de CreateOtaUpdate à l'aide du AWS CLI
- Obtenez les codes de défaillance OTA avec le AWS CLI
- Dépannage des mises à jour OTA de plusieurs appareils
- Résolution des problèmes liés aux mises à jour OTA avec la plateforme Texas Instruments CC3220SF Launchpad

Configurer les CloudWatch journaux pour les mises à jour OTA

Le service OTA Update prend en charge la journalisation avec Amazon CloudWatch. Vous pouvez utiliser la AWS IoT console pour activer et configurer la CloudWatch journalisation Amazon pour les mises à jour OTA. Pour plus d'informations sur consultez Cloudwatch Logs.

Pour activer la journalisation, vous devez créer un rôle IAM et configurer la journalisation des mises à jour OTA.

> **Note**
>
> Avant d'activer la journalisation des mises à jour OTA, assurez-vous de bien comprendre les autorisations d'accès aux CloudWatch journaux. Les utilisateurs ayant accès aux CloudWatch journaux peuvent voir vos informations de débogage. Pour plus d'informations, consultez Authentification et contrôle d'accès pour Amazon CloudWatch Logs.

Créer un rôle de journalisation et activer la journalisation

Utilisez la console AWS IoT pour créer un rôle de journalisation et activer la journalisation.

1. Choisissez Paramètres dans le volet de navigation.
2. Sous Journaux, choisissez Modifier.
3. Sous Niveau de détail, choisissez Déboguer.
4. Sous Définir le rôle, choisissez Create new pour créer un rôle IAM pour la journalisation.
5. Sous Nom, saisissez un nom unique pour votre rôle. Votre rôle sera créé avec toutes les autorisations requises.
6. Choisissez Mettre à jour.

Journaux de mise à jour OTA

Le service de mise à jour OTA publie les journaux sur votre compte lorsque l'une des conditions suivantes se produit :

- Une mise à jour OTA est créée.
- Une mise à jour OTA est achevée.
- Une tâche de signature de code est créée.
- Une tâche de signature de code est terminée.
- Un AWS IoT emploi est créé.
- Une AWS IoT tâche est terminée.
- Un flux est créé.

Vous pouvez afficher vos journaux dans la console CloudWatch .

Pour afficher une mise à jour OTA dans CloudWatch Logs

1. Dans le volet de navigation, choisissez Logs (Journaux).
2. Dans Log Groups, sélectionnez AWSIoTLogsV2.

Les journaux de mise à jour OTA peuvent contenir les propriétés suivantes :

accountId

L'identifiant du AWS compte sous lequel le journal a été généré.

actionType

L'action qui a généré le journal. Cette propriété peut avoir l'une des valeurs suivantes :

- `CreateOTAUpdate` : une mise à jour OTA a été créée.
- `DeleteOTAUpdate` : une mise à jour OTA a été supprimée.
- `StartCodeSigning` : une tâche de signature de code a été démarrée.
- `CreateAWSJob`: Un AWS IoT emploi a été créé.
- `CreateStream` : un flux a été créé.
- `GetStream`: une demande de flux a été envoyée à la fonctionnalité de livraison de AWS IoT MQTT-based fichiers.
- `DescribeStream`: une demande d'informations sur un flux a été envoyée à la fonction de livraison de AWS IoT MQTT-based fichiers.

awsJobId

L'ID de la AWS IoT tâche qui a généré le journal.

clientId

L'ID du client MQTT qui a émis la demande ayant généré le journal.

clientToken

Le jeton client associé à la demande ayant généré le journal.

détails

Informations supplémentaires sur l'opération ayant généré le journal.

logLevel

Niveau de journalisation du journal. Pour les journaux de mise à jour OTA, la valeur est toujours définie sur DEBUG.

ota UpdateId

ID de la mise à jour OTA ayant généré le journal.

protocole ;

Protocole utilisé pour effectuer la demande ayant généré le journal.

status

Statut de l'opération ayant généré le journal. Les valeurs valides sont :

- Réussite
- Échec

streamId

L'ID du AWS IoT flux qui a généré le journal.

timestamp

Heure à laquelle le journal a été généré.

topicName

Rubrique MQTT utilisée pour effectuer la demande ayant généré le journal.

Exemples de journaux

Voici un exemple de journal généré lorsqu'une tâche de signature de code est démarrée :

```
{
    "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."
```

```
}
```

Voici un exemple de journal généré lors de la création AWS IoT d'une tâche :

```
{
    "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."
}
```

Voici un exemple de journal généré lorsqu'une mise à jour OTA est créée :

```
{
    "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."
}
```

Voici un exemple de journal généré lorsqu'un flux est créé :

```
{
    "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."
}
```

Voici un exemple de journal généré lorsqu'une mise à jour OTA est supprimée :

```
{
    "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."
}
```

Voici un exemple de journal généré lorsqu'un appareil demande un flux à partir de la fonctionnalité de livraison de MQTT-based fichiers :

```
{
    "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."
}
```

Voici un exemple de journal généré lorsqu'un appareil appelle l'API `DescribeStream` :

```
{
    "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."
```

```
}
```

Journal AWS IoT Appels d'API OTA avec AWS CloudTrail

FreeRTOS est intégré CloudTrail à un service qui AWS IoT capture les appels d'API OTA et envoie les fichiers journaux à un compartiment Amazon S3 que vous spécifiez. CloudTrail capture les appels d'API de votre code vers les API AWS IoT OTA. À l'aide des informations collectées par CloudTrail, vous pouvez déterminer la demande qui a été faite à AWS IoT OTA, l'adresse IP source à partir de laquelle la demande a été faite, qui a fait la demande, quand elle a été faite, etc.

Pour plus d'informations CloudTrail, notamment sur la manière de le configurer et de l'activer, consultez le guide de AWS CloudTrail l'utilisateur.

Informations FreeRTOS dans CloudTrail

Lorsque la CloudTrail journalisation est activée dans votre AWS compte, les appels d'API effectués aux actions AWS IoT OTA sont suivis dans des fichiers CloudTrail journaux où ils sont écrits avec d'autres enregistrements de AWS service. CloudTrail détermine à quel moment créer et écrire dans un nouveau fichier en fonction d'une période et de la taille du fichier.

Les actions du plan de contrôle AWS IoT OTA suivantes sont enregistrées par CloudTrail :

- CreateStream
- DescribeStream
- ListStreams
- UpdateStream
- DeleteStream
- CreateOTAUpdate
- GetOTAUpdate
- ListOTAUpdates
- DeleteOTAUpdate

> **Note**
>
> AWS IoT Les actions du plan de données OTA (côté appareil) ne sont pas enregistrées CloudTrail. CloudWatch Utilisez-le pour les surveiller.

Chaque entrée du journal contient des informations sur la personne qui a généré la demande. Les informations d'identité de l'utilisateur dans l'entrée de journal permettent de déterminer les éléments suivants :

- Si la demande a été effectuée avec des informations d'identification d'utilisateur root ou IAM.
- Si la demande a été effectuée avec des informations d'identification de sécurité temporaires pour un rôle ou un utilisateur fédéré.
- Si la demande a été faite par un autre AWS service.

Pour plus d'informations, consultez l'élément CloudTrail UserIdentity. AWS IoT Les actions OTA sont documentées dans la référence des API AWS IoT OTA.

Vous pouvez stocker vos fichiers journaux dans votre compartiment Amazon S3 aussi longtemps que vous le souhaitez, mais vous pouvez également définir des règles de cycle de vie Amazon S3 pour archiver ou supprimer automatiquement les fichiers journaux. Par défaut, vos fichiers journaux sont chiffrés avec le chiffrement côté serveur (SSE) d'Amazon S3.

Si vous souhaitez être averti lorsque les fichiers journaux sont livrés, vous pouvez configurer la publication CloudTrail des notifications Amazon SNS. Pour plus d'informations, consultez Configuration des notifications Amazon SNS pour. CloudTrail

Vous pouvez également agréger les fichiers journaux AWS IoT OTA provenant de plusieurs AWS régions et de plusieurs AWS comptes dans un seul compartiment Amazon S3.

Pour plus d'informations, consultez les sections Réception de fichiers CloudTrail journaux de plusieurs régions et Réception de fichiers CloudTrail journaux de plusieurs comptes.

Comprendre les entrées du fichier journal FreeRTOS

CloudTrail les fichiers journaux peuvent contenir une ou plusieurs entrées de journal. Chaque entrée répertorie plusieurs JSON-formatted événements. Une entrée de journal représente une demande individuelle à partir de n'importe quelle source et comprend des informations sur l'action demandée, sur la date et l'heure de l'action, sur les paramètres de la demande, etc. Les entrées de journal ne sont pas des arborescences des appels de procédure des appels d'API publique. Elles ne s'affichent donc dans aucun ordre précis.

L'exemple suivant montre une entrée de CloudTrail journal qui illustre le journal à partir d'un appel à `CreateOTAUpdate` l'action.

```
{
```

```
    "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"
}
```

Obtenez les informations relatives à l'échec de CreateOtaUpdate à l'aide du AWS CLI

Si le processus de création d'une tâche de mise à jour OTA échoue, vous pouvez peut-être prendre des mesures pour remédier au problème. Lorsque vous créez une tâche de mise à jour OTA, le service OTA Manager crée une tâche IoT et la planifie pour les appareils cibles. Ce processus crée ou utilise également d'autres types de AWS ressources dans votre compte (une tâche de signature de code, un AWS IoT flux, un objet Amazon S3). Toute erreur rencontrée peut entraîner l'échec du processus sans créer de AWS IoT tâche. Dans cette section de résolution des problèmes, nous donnons des instructions sur la façon de récupérer les détails de la panne.

1. Installez et configurez l'AWS CLI.
2. Exécutez `aws configure` et entrez les informations suivantes.

   ```
   $ aws configure
   AWS Access Key ID [None]: AccessID
   AWS Secret Access Key [None]: AccessKey
   Default region name [None]: Region
   Default output format [None]: json
   ```

   Pour plus d'informations, voir Configuration rapide avec `aws configure`.
3. Exécuter :

   ```
   aws iot get-ota-update --ota-update-id ota_update_job_001
   ```

   Où se *ota_update_job_001* trouve l'identifiant que vous avez donné à la mise à jour OTA lorsque vous l'avez créée.
4. La sortie doit se présenter comme suit :

```
{
    "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)"
        }
    }
}
```

Si la création échoue, le `otaUpdateStatus` champ de la sortie de commande contiendra `CREATE_FAILED` et contiendra les détails de l'échec. `errorInfo`

Obtenez les codes de défaillance OTA avec le AWS CLI

1. Installez et configurez l'AWS CLI.
2. Exécutez `aws configure` et entrez les informations suivantes.

```
$ aws configure
AWS Access Key ID [None]: AccessID
AWS Secret Access Key [None]: AccessKey
Default region name [None]: Region
Default output format [None]: json
```

Pour plus d'informations, voir Configuration rapide avec `aws configure`.

3. Exécuter :

```
aws iot describe-job-execution --job-id JobID --thing-name ThingName
```

Où *`JobID`* se trouvent la chaîne d'identification complète de la tâche dont nous voulons obtenir le statut (elle était associée à la tâche de mise à jour OTA lors de sa création) et *`ThingName`* le nom de l' AWS IoT objet sous lequel l'appareil est enregistré dans AWS IoT

4. La sortie doit se présenter comme suit :

```
{
    "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
    }
}
```

Dans cet exemple de sortie, « `reason` » dans « `detailsmap` » comporte deux champs : le champ « 0xEEEEEEEE » contient le code d'erreur générique de l'agent OTA et le champ « 0xffffffff » contient le sous-code. Les codes d'erreur génériques sont répertoriés dans https://docs.aws.amazon.com/freertos/latest/lib-ref/html1/aws__ota__agent_8h.html. Consultez les codes d'erreur avec le préfixe « `kOTA_Err_` ». Le sous-code peut être un code spécifique à la plateforme ou fournir plus de détails sur l'erreur générique.

Dépannage des mises à jour OTA de plusieurs appareils

Pour effectuer des OTA sur plusieurs appareils (objets) en utilisant la même image de microprogramme, implémentez une fonction (par exemple `getThingName()`) qui récupère `clientcredentialIOT_THING_NAME` de la mémoire non volatile. Assurez-vous que cette fonction lit le nom de l'objet à partir d'une partie de la mémoire non volatile qui n'est pas écrasée par l'OTA, et que le nom de l'objet est provisionné avant d'exécuter la première tâche. Si vous utilisez le flux JITP, vous pouvez lire le nom de l'objet dans le nom commun de votre certificat d'appareil.

Résolution des problèmes liés aux mises à jour OTA avec la plateforme Texas Instruments CC3220SF Launchpad

La plateforme CC3220SF Launchpad offre un mécanisme de détection de logiciel inviolable. Celui-ci utilise un compteur d'alerte de sécurité incrémenté chaque fois qu'il y a violation d'intégrité. L'appareil est verrouillé lorsque le compteur d'alerte de sécurité atteint un seuil prédéterminé (la valeur par défaut est de 15) et que l'hôte reçoit l'événement asynchrone `SL_ERROR_DEVICE_LOCKED_SECURITY_ALERT`. L'appareil verrouillé possède alors une accessibilité limitée. Pour récupérer l'appareil, vous pouvez le reprogrammer ou utiliser la « restauration usine » pour rétablir l'image d'usine par défaut. Vous devez programmer le comportement souhaité en mettant à jour le gestionnaire d'événements asynchrones dans `network_if.c`.

Bibliothèques FreeRTOS

Les bibliothèques FreeRTOS fournissent des fonctionnalités supplémentaires au noyau FreeRTOS et à ses bibliothèques internes. Vous pouvez utiliser les bibliothèques FreeRTOS pour la mise en réseau et la sécurité des applications embarquées. Les bibliothèques FreeRTOS permettent également à vos applications d'interagir avec les services. AWS IoT FreeRTOS inclut des bibliothèques qui permettent de :

- Connectez les appareils au AWS IoT cloud en toute sécurité à l'aide de MQTT et de Device Shadows.
- Découvrez les AWS IoT Greengrass cœurs et connectez-vous à ceux-ci.
- Gérez Wi-Fi les connexions.
- Ecoutez et traitez Mises à jour de FreeRTOS Over-the-Air.

Le `libraries` répertoire contient le code source des bibliothèques FreeRTOS. Il s'agit de fonctions d'assistance destinées à faciliter la mise en œuvre de la bibliothèque. Nous vous déconseillons de modifier ces fonctions d'assistance.

bibliothèques de portage FreeRTOS

Les bibliothèques de portage suivantes sont incluses dans les configurations de FreeRTOS disponibles en téléchargement sur la console FreeRTOS. Ces bibliothèques dépendent de la plateforme. Leur contenu change en fonction de votre plateforme matérielle. Pour plus d'informations sur le portage de ces bibliothèques sur un appareil, consultez le Guide de portage de FreeRTOS.

bibliothèques de portage FreeRTOS

Bibliothèque	Référence d'API	Description
Bluetooth Low Energy	Référence d'API Bluetooth Low Energy	À l'aide de la bibliothèque FreeRTOS Bluetooth Low Energy, votre microcontrôleur peut communiquer avec AWS IoT le courtier MQTT via une passerell e. Pour de plus amples informati ons, veuillez consulter Bibliothèque Bluetooth Low Energy.

Bibliothèque	Référence d'API	Description
Over-the-Air mises à jour	AWS IoT Over-the-air référence d'API de mise à jour	La bibliothèque de mises à jour AWS IoT Over-the-air FreeRTOS (OTA) vous permet de gérer les notifications de mise à jour, de télécharger les mises à jour et d'effectuer une vérification cryptographique des mises à jour du microprogramme sur votre appareil FreeRTOS. Pour de plus amples informations, veuillez consulter AWS IoT Bibliothèque en direct (OTA).
FreeRTOS+POSIX	Référence d'API FreeRTOS+POSIX	Vous pouvez utiliser la bibliothèque FreeRTOS+POSIX pour porter des applications POSIX-compliant vers l'écosystème FreeRTOS. Pour plus d'informations, consultez FreeRTOS+POSIX.
Secure Sockets	Référence de l'API Secure Sockets	Pour de plus amples informations, veuillez consulter Bibliothèque Secure Sockets.
FreeRTOS+TCP	Référence de l'API FreeRTOS+TCP	FreeRTOS+TCP est une pile évolutive, open source et sécurisée pour FreeRTOS. TCP/IP Pour plus d'informations, consultez FreeRTOS+TCP.

Bibliothèque	Référence d'API	Description
Wi-Fi	Wi-Fi API Reference	La bibliothèque FreeRTOS Wi-Fi vous permet d'interagir avec la pile sans fil de niveau inférieur de votre microcontrôleur. Pour de plus amples informations, veuillez consulter Wi-Fi bibliothèque.
Code PKCS11		La bibliothèque CorePKCS11 est une implémentation de référence de la norme de cryptographie à clé publique #11, pour prendre en charge le provisionnement et l'authentification des clients TLS. Pour de plus amples informations, veuillez consulter bibliothèque CorePKCS11.
TLS		Pour de plus amples informations, veuillez consulter : acte de révision dans un pipeline se poursuivant d'une étape à l'autre dans un flux de travail..
Commun I/O	Référence I/O d'API commune	Pour de plus amples informations, veuillez consulter Commun I/O.
Interface cellulaire	Référence d'API d'interface cellulaire	La bibliothèque Cellular Interface expose les capacités de quelques modems cellulaires populaires via une API uniforme. Pour de plus amples informations, veuillez consulter Bibliothèque d'interfaces cellulaires.

bibliothèques d'applications FreeRTOS

Vous pouvez éventuellement inclure les bibliothèques d'applications autonomes suivantes dans votre configuration FreeRTOS pour interagir AWS IoT avec les services sur le cloud.

> **Note**
>
> Certaines bibliothèques d'applications possèdent les mêmes API que les bibliothèques du AWS IoT Device SDK for Embedded C. Pour ces bibliothèques, consultez le manuel AWS IoT Device SDK C API Reference. Pour plus d'informations sur le SDK du AWS IoT périphérique pour Embedded C, consultezAWS IoT SDK de périphérique pour Embedded C.

bibliothèques d'applications FreeRTOS

Bibliothèque	Référence d'API	Description
AWS IoT Device Defender	Référence de l'API du SDK Device Defender C	La bibliothèque FreeRTOS connecte votre appareil AWS IoT Device Defender FreeRTOS à. AWS IoT Device Defender Pour de plus amples informations, veuillez consulter AWS IoT Device Defender bibliothèque.
AWS IoT Greengrass	Référence de l'API Greengrass	La bibliothèque FreeRTOS connecte votre appareil AWS IoT Greengrass FreeRTOS à. AWS IoT Greengrass Pour de plus amples informati ons, veuillez consulter AWS IoT Greengrass Bibliothèque Discovery.
MQTT	Référence de l'API de bibliothèque MQTT (v1.x.x)	La bibliothèque CoreMQTT fournit un client permettant à votre appareil FreeRTOS de publier des sujets MQTT et de s'y abonner. MQTT

Bibliothèque	Référence d'API	Description
	Référence d'API de l'agent MQTT (v1) Référence de l'API du SDK MQTT (v2.x.x) C	est le protocole avec lequel les appareils interagissent AWS IoT. Pour plus d'informations sur la version 3.0.0 de la bibliothèque CoreMQTT, consultez. bibliothèque CoreMQTT
Agent CoreMQTT	Référence de l'API de la bibliothèque d'agents CoreMQTT	La bibliothèque de l'agent CoreMQTT est une API de haut niveau qui ajoute la sécurité des threads à la bibliothèque CoreMQTT. Il vous permet de créer une tâche d'agent MQTT dédiée qui gère une connexion MQTT en arrière-plan et ne nécessite aucune intervention de la part d'autres tâches. La bibliothèque fournit des équivalents thread-safe aux API de CoreMQTT, de sorte qu'elle peut être utilisée dans des environnements multithread. Pour plus d'informations sur la bibliothèque de l'agent CoreMQTT, consultez. Bibliothèque de l'agent CoreMQTT
AWS IoT Device Shadow	Référence d'API de kits SDK C pour Device Shadow	La bibliothèque AWS IoT Device Shadow permet à votre appareil FreeRTOS d'interagir avec AWS IoT les ombres de l'appareil. Pour de plus amples informations, veuillez consulter AWS IoT Bibliothèque Device Shadow.

Configuration des bibliothèques FreeRTOS

Les paramètres de configuration pour FreeRTOS et AWS IoT le Device SDK for Embedded C sont définis comme des constantes du préprocesseur C. Vous pouvez définir les paramètres de configuration à l'aide d'un fichier de configuration global ou à l'aide d'une option de compilation, comme `-D` dans `gcc`. Comme les paramètres de configuration sont définis comme constantes au moment de la compilation, une bibliothèque doit être recréée si un paramètre de configuration est modifié.

Si vous souhaitez utiliser un fichier de configuration pour définir des options de configuration, créez et enregistrez le fichier avec le nom `iot_config.h`, et placez-le dans votre chemin d'inclusion. Dans le fichier, utilisez `#define` des directives pour configurer les bibliothèques, les démos et les tests FreeRTOS.

Pour plus d'informations sur les options de configuration globale prises en charge, consultez la Référence du fichier de configuration globale.

Bibliothèque d'algorithmes Backoff

> **Note**
>
> Le contenu de cette page n'est peut-être pas à jour. Reportez-vous à la page de la FreeRTOS.org bibliothèque pour obtenir la dernière mise à jour.

Introduction

La bibliothèque BackoffAlgorithm est une bibliothèque utilitaire utilisée pour espacer les retransmissions répétées du même bloc de données, afin d'éviter la congestion du réseau. Cette bibliothèque calcule la période d'attente pour une nouvelle tentative d'opérations réseau (par exemple, une connexion réseau défaillante avec le serveur) à l'aide d'un ralentissement exponentiel avec algorithme de gigue.

L'interruption exponentielle avec instabilité est généralement utilisée lors d'une nouvelle tentative de connexion échouée ou de demande réseau envoyée à un serveur en raison d'un encombrement du réseau ou de charges élevées sur le serveur. Il est utilisé pour étaler le temps des demandes de nouvelle tentative créées par plusieurs appareils tentant de se connecter au réseau en même temps. Dans un environnement où la connectivité est faible, un client peut être déconnecté à tout moment. Une stratégie de sauvegarde permet donc au client d'économiser la batterie en évitant de tenter des reconnexions à plusieurs reprises alors qu'elles ont peu de chances de réussir.

La bibliothèque est écrite en C et conçue pour être conforme aux normes ISO C90 et C:2012MISRA. La bibliothèque ne dépend d'aucune bibliothèque supplémentaire autre que la bibliothèque C standard et ne dispose d'aucune allocation de mémoire, ce qui la rend adaptée aux microcontrôleurs IoT, mais également entièrement portable sur d'autres plateformes.

Cette bibliothèque peut être utilisée librement et est distribuée sous la licence open source du MIT.

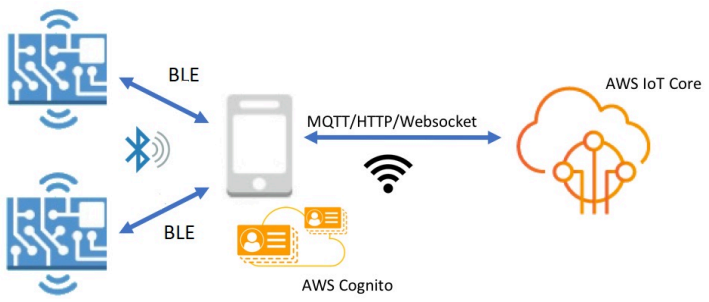
Taille du code de BackoffAlgorithm (exemple généré avec GCC pour ARM) Cortex-M		
Fichier	Avec optimisation -O1	Avec l'optimisation -Os
backoff_algorithm.c	0,1 K	0,1 K
Estimations totales	0,1 K	0,1 K

Bibliothèque Bluetooth Low Energy

> **Important**
>
> Cette bibliothèque est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Présentation de

FreeRTOS prend en charge la publication et l'abonnement à des sujets relatifs au transport télémétrique des messages en file d'attente (MQTT) via Bluetooth Low Energy via un périphérique proxy, tel qu'un téléphone portable. Grâce à la bibliothèque FreeRTOS Bluetooth Low Energy (BLE), votre microcontrôleur peut communiquer en toute sécurité avec le courtier MQTT. AWS IoT

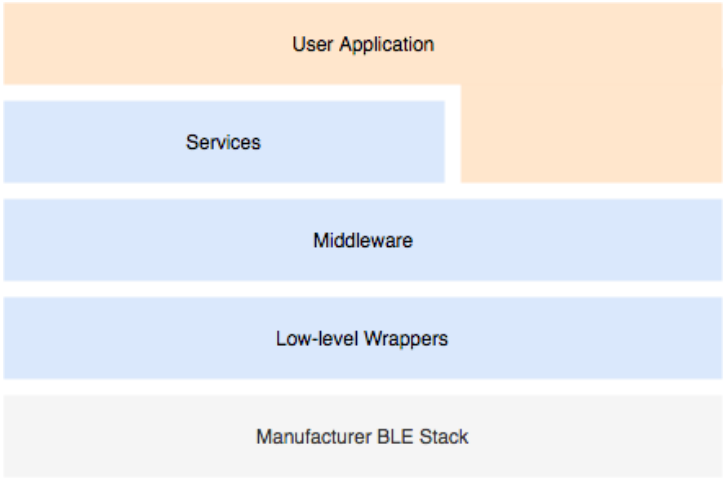


À l'aide des SDK mobiles pour les appareils Bluetooth FreeRTOS, vous pouvez écrire des applications mobiles natives qui communiquent avec les applications intégrées sur votre microcontrôleur via BLE. Pour plus d'informations sur les kits SDK mobiles, consultez SDK mobiles pour appareils Bluetooth FreeRTOS.

La bibliothèque FreeRTOS BLE inclut des services permettant de Wi-Fi configurer des réseaux, de transférer de grandes quantités de données et de fournir des abstractions réseau via BLE. La bibliothèque FreeRTOS BLE inclut également des intergiciels et des API de niveau inférieur pour un contrôle plus direct de votre stack BLE.

Architecture

Trois couches constituent la bibliothèque FreeRTOS BLE : les services, les intergiciels et les wrappers de bas niveau.



Services

La couche de services FreeRTOS BLE comprend quatre services d'attributs génériques (GATT) qui exploitent les API du middleware :

- Informations sur les périphériques
- Wi-Fi approvisionnement
- Abstraction réseau
- Transfert d'objets de grande taille

Informations sur les périphériques

Le service d'informations sur les appareils recueille des informations sur votre microcontrôleur, notamment :

- Version de FreeRTOS utilisée par votre appareil.
- AWS IoT Point de terminaison du compte pour lequel l'appareil est enregistré.
- Unité de transmission maximale Bluetooth Low Energy (MTU).

Wi-Fi approvisionnement

Le service de Wi-Fi provisionnement permet aux microcontrôleurs dotés Wi-Fi des capacités suivantes :

- Répertorier les réseaux dans une plage.
- Enregistrer les réseaux et les informations d'identification correspondantes sur la mémoire flash.
- Définir la priorité du réseau.
- Supprimer les réseaux et les informations d'identification correspondantes de la mémoire flash.

Abstraction réseau

Le service d'abstraction réseau extrait le type de connexion réseau pour les applications. Une API commune interagit avec le matériel Ethernet et Bluetooth Low Energy de votre appareil, permettant à une application d'être compatible avec plusieurs types de connexion. Wi-Fi

Transfert d'objets volumineux

Le service Large Object Transfer envoie des données à un client et en reçoit des données. D'autres services, tels que le Wi-Fi provisionnement et l'abstraction du réseau, utilisent le service Large Object Transfer pour envoyer et recevoir des données. Vous pouvez également utiliser l'API Large Object Transfer pour interagir directement avec le service.

MQTT via BLE

MQTT over BLE contient le profil GATT pour créer un service proxy MQTT sur BLE. Le service proxy MQTT permet à un client MQTT de communiquer avec le courtier AWS MQTT via une passerelle. Par exemple, vous pouvez utiliser le service proxy pour connecter un appareil exécutant FreeRTOS

à MQTT via une AWS application pour smartphone. Le dispositif BLE est le serveur GATT et expose les services et les caractéristiques du dispositif passerelle. Le serveur GATT utilise ces services et caractéristiques exposés pour effectuer des opérations MQTT avec le cloud pour cet appareil. Pour plus d'informations, consultez Annexe A : Profil MQTT sur BLE GATT.

Intergiciel

Le middleware Bluetooth Low Energy FreeRTOS est une abstraction des API de niveau inférieur. Les API d'intergiciel constituent une interface plus conviviale pour la pile Bluetooth Low Energy.

À l'aide des API d'intergiciel, vous pouvez enregistrer plusieurs rappels, dans plusieurs couches, pour un seul événement. L'initialisation de l'intergiciel Bluetooth Low Energy initialise également des services et démarre la publicité.

Abonnement de rappel flexible

Supposons que votre matériel Bluetooth Low Energy se déconnecte et que le service MQTT sur Bluetooth Low Energy ait besoin de détecter la déconnexion. Une application que vous avez créée peut également avoir à détecter le même événement de déconnexion. L'intergiciel BLE peut acheminer l'événement à différentes parties du code où vous avez enregistré des rappels, sans que les couches supérieures n'aient à se disputer les ressources de niveau inférieur.

Low-level emballages

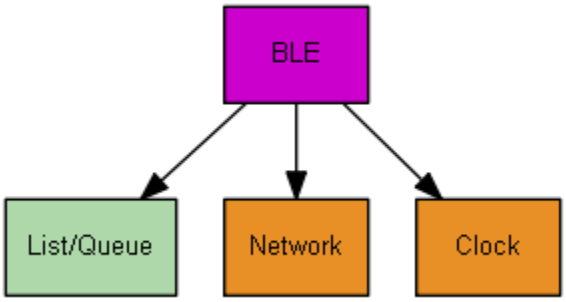
Les wrappers Bluetooth Low Energy FreeRTOS de bas niveau sont une abstraction de la gamme Bluetooth Low Energy du fabricant. Low-level les wrappers proposent un ensemble commun d'API pour un contrôle direct du matériel. Les API de bas niveau optimisent l'utilisation de la RAM, mais leurs fonctionnalités sont limitées.

Utilisez les API du service Bluetooth Low Energy pour interagir avec les services Bluetooth Low Energy. Les API de service exigent plus de ressources que les API de bas niveau.

Dépendances et exigences

La bibliothèque Bluetooth Low Energy comporte les dépendances directes suivantes :

- Bibliothèque Linear Containers
- Une couche de plateforme en interface avec le système d'exploitation pour la gestion des threads, les minuteurs, les fonctions d'horloge et l'accès réseau.



Seul le service Wi-Fi Provisioning possède des dépendances de bibliothèque FreeRTOS :

Service GATT	Dépendance
Wi-Fi Approvisionnement	Wi-Fi bibliothèque

Pour communiquer avec le courtier AWS IoT MQTT, vous devez avoir un AWS compte et enregistrer vos appareils en tant qu' AWS IoT objets. Pour en savoir plus sur la configuration, consultez le Manuel du développeur AWS IoT.

FreeRTOS Bluetooth Low Energy utilise Amazon Cognito pour l'authentification des utilisateurs sur votre appareil mobile. Pour utiliser les services proxy MQTT, vous devez créer une identité Amazon Cognito et des groupes d'utilisateurs. Chaque identité Amazon Cognito doit être associée à la politique appropriée. Pour plus d'informations, consultez le Guide du développeur Amazon Cognito.

Fichier de configuration des bibliothèques

Les applications qui utilisent le service FreeRTOS MQTT over Bluetooth Low Energy doivent fournir `iot_ble_config.h` un fichier d'en-tête dans lequel les paramètres de configuration sont définis. Les paramètres de configuration non définis utilisent les valeurs par défaut spécifiées dans `iot_ble_config_defaults.h`.

Voici quelques paramètres de configuration importants :

`IOT_BLE_ADD_CUSTOM_SERVICES`

Autorise les utilisateurs à créer leurs propres services.

`IOT_BLE_SET_CUSTOM_ADVERTISEMENT_MSG`

Permet aux utilisateurs de personnaliser la publicité et d'analyser les messages de réponse.

Pour de plus amples informations, veuillez consulter Référence d'API Bluetooth Low Energy.

Optimisation

Lors de l'optimisation des performances de votre carte, prenez en considération les éléments suivants :

- Low-level Les API utilisent moins de RAM, mais offrent des fonctionnalités limitées.
- Vous pouvez définir le paramètre `bleconfigMAX_NETWORK` dans le fichier d'en-tête `iot_ble_config.h` sur une valeur inférieure afin de réduire la quantité de pile consommée.
- Vous pouvez augmenter la taille MTU à sa valeur maximale pour limiter la mise en mémoire tampon des messages, accélérer l'exécution du code et consommer moins de RAM.

Restrictions liées à l'utilisation

Par défaut, la bibliothèque FreeRTOS Bluetooth Low Energy définit `eBTpropertySecureConnectionOnly` la propriété sur TRUE, ce qui place l'appareil en mode Connexions sécurisées uniquement. Comme défini par la spécification Bluetooth Core v5.0, vol 3, partie C, 10.2.4, lorsqu'un appareil utilise un mode autorisant uniquement les connexions sécurisées, le plus haut niveau de sécurité basse énergie du mode 1, à savoir le niveau 4, est requis pour l'accès à n'importe quel attribut disposant d'autorisations supérieures au niveau de sécurité basse énergie inférieur du mode 1, à savoir le niveau 1. Dans le mode de sécurité basse énergie 1 de niveau 4, un appareil doit avoir des fonctionnalités d'entrée et de sortie pour la comparaison numérique.

Voici les modes pris en charge et les propriétés associées :

Mode 1, Niveau 1 (aucune sécurité)

```
/* 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 )
```

Mode 1, Niveau 2 (appairage non authentifié avec chiffrement)

```
#define IOT_BLE_ENABLE_NUMERIC_COMPARISON       ( 0 )
#define IOT_BLE_ENABLE_SECURE_CONNECTION        ( 0 )
#define IOT_BLE_INPUT_OUTPUT                    ( eBTIONone )
```

Mode 1, Niveau 3 (appairage authentifié avec chiffrement)

Ce mode n'est pas pris en charge.

Mode 1, Niveau 4 (appairage des connexions sécurisées LE authentifiées avec chiffrement)

Ce mode est pris en charge par défaut.

Pour de plus amples informations sur les modes de sécurité basse énergie, veuillez consulter la spécification Bluetooth Core v5.0, vol 3, partie C, 10.2.1.

Initialisation

Si votre application interagit avec la pile Bluetooth Low Energy via un intergiciel, vous devez uniquement initialiser ce dernier. L'intergiciel s'occupe de l'initialisation des couches inférieures de la pile.

Intergiciel

Pour initialiser le middleware

1. Initialisez n'importe quel pilote matériel Bluetooth Low Energy avant d'appeler l'API d'intergiciel Bluetooth Low Energy.
2. Activez Bluetooth Low Energy.
3. Initialisez l'intergiciel avec `IotBLE_Init()`.

> **Note**
>
> Cette étape d'initialisation n'est pas requise si vous exécutez les AWS démos. L'initialisation des démonstrations est gérée par le gestionnaire de réseau local, situé à *`freertos`*`/demos/network_manager`.

Low-level APIs

Si vous ne souhaitez pas utiliser les services FreeRTOS Bluetooth Low Energy GATT, vous pouvez contourner le middleware et interagir directement avec les API de bas niveau pour économiser des ressources.

Pour initialiser les API de bas niveau

1. Initialisez n'importe quel pilote de matériel Bluetooth Low Energy avant d'appeler les API. L'initialisation des pilotes ne fait pas partie des API de base niveau Bluetooth Low Energy.

2. L'API Bluetooth Low Energy Low-Level fournit un enable/disable appel à la pile Bluetooth Low Energy pour optimiser la puissance et les ressources. Avant d'appeler les API, vous devez activer Bluetooth Low Energy.

```
const BTInterface_t * pxIface = BTGetBluetoothInterface();
xStatus = pxIface->pxEnable( 0 );
```

3. Le gestionnaire Bluetooth contient des API qui sont communes à Bluetooth Low Energy et au Bluetooth classique. Les rappels du gestionnaire doivent ensuite être initialisés.

```
xStatus = xBTInterface.pxBTInterface->pxBtManagerInit( &xBTManagerCb );
```

4. L'adaptateur Bluetooth Low Energy peut être ajouté en plus de l'API courante. Vous devez initialiser ses rappels courants comme vous avez initialisé l'API courante.

```
xBTInterface.pxBTLeAdapterInterface = ( BTBleAdapter_t * )
 xBTInterface.pxBTInterface->pxGetLeAdapter();
xStatus = xBTInterface.pxBTLeAdapterInterface-
>pxBleAdapterInit( &xBTBleAdapterCb );
```

5. Enregistrez votre nouvelle application utilisateur.

```
xBTInterface.pxBTLeAdapterInterface->pxRegisterBleApp( pxAppUuid );
```

6. Initialisez les rappels aux serveurs GATT.

```
xBTInterface.pxGattServerInterface = ( BTGattServerInterface_t * )
 xBTInterface.pxBTLeAdapterInterface->ppvGetGattServerInterface();
xBTInterface.pxGattServerInterface->pxGattServerInit( &xBTGattServerCb );
```

Une fois que vous avez initialisé l'adaptateur Bluetooth Low Energy, vous pouvez ajouter un serveur GATT. Vous pouvez enregistrer un seul serveur GATT à la fois.

```
xStatus = xBTInterface.pxGattServerInterface->pxRegisterServer( pxAppUuid );
```

7. Définissez les propriétés de l'application comme la connexion sécurisée uniquement et la taille MTU.

```
xStatus = xBTInterface.pxBTInterface-
>pxSetDeviceProperty( &pxProperty[ usIndex ] );
```

Référence des API

Pour obtenir une référence d'API complète, veuillez consulter Référence d'API Bluetooth Low Energy.

Exemple d'utilisation

Les exemples ci-dessous montrent comment utiliser la bibliothèque Bluetooth Low Energy pour la publicité et la création de nouveaux services. Pour les applications de démonstration FreeRTOS Bluetooth Low Energy complètes, voir Applications de démonstration Bluetooth Low Energy.

Publicité

1. Dans votre application, définissez les UUID de publicité :

```
static const BTUuid_t _advUUID =
{
    .uu.uu128 = IOT_BLE_ADVERTISING_UUID,
    .ucType   = eBTuuidType128
};
```

2. Ensuite, définissez la fonction de rappel `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;
}
```

Ce rappel envoie l'UUID dans le message de publicité et le nom complet dans la réponse d'analyse.

3. Ouvrez `vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/iot_ble_config.h` et définissez `IOT_BLE_SET_CUSTOM_ADVERTISEMENT_MSG` sur 1. Cela déclenche le rappel `IotBle_SetCustomAdvCb`.

Ajout d'un nouveau service

Pour consulter l'ensemble des exemples de services, accédez à *`freertos`*`/.../ble/services`.

1. Créez des UUID pour la caractéristique et les descripteurs du service :

```
#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. Créez un tampon pour enregistrer les handles de la caractéristique et des descripteurs :

```
static uint16_t usHandlesBuffer[egattDemoNbAttributes];
```

3. Créez la table de l'attribut. Pour enregistrer une RAM, définissez la table en tant que `const`.

> **Important**
>
> Créez toujours les attributs dans l'ordre, avec le service en tant que premier attribut.

```
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. Créez un tableau des rappels. Ce tableau de rappels doit suivre le même ordre que le tableau de table défini ci-dessus.

Par exemple, si `vReadCounter` est déclenché lorsque `xCharCounterUUID_TYPE` est consulté, et si `vWriteCommand` se déclenche lorsque `xCharControlUUID_TYPE` est consulté, définissez le tableau comme suit :

```
static const IotBleAttributeEventCallback_t pxCallBackArray[egattDemoNbAttributes]
 =
    {
  NULL,
  vReadCounter,
  vEnableNotification,
  vWriteCommand
};
```

5. Créez le service :

```
static const BTService_t xGattDemoService =
{
  .xNumberOfAttributes = egattDemoNbAttributes,
  .ucInstId = 0,
  .xType = eBTServiceTypePrimary,
  .pusHandlesBuffer = usHandlesBuffer,
  .pxBLEAttributes = (BTAttribute_t *)pxAttributeTable
};
```

6. Appelez l'API `IotBle_CreateService` avec la structure que vous avez créée à l'étape précédente. L'intergiciel synchronise la création de tous les services, ce qui signifie que tous les nouveaux services doivent être déjà définis lorsque le rappel `IotBle_AddCustomServicesCb` est déclenché.

   a. Définissez `IOT_BLE_ADD_CUSTOM_SERVICES` sur 1 dans `vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/iot_ble_config.h`.

   b. Créez IotBle _ AddCustomServicesCb dans votre application :

```
void IotBle_AddCustomServicesCb(void)
{
    BTStatus_t xStatus;
    /* Select the handle buffer. */
    xStatus = IotBle_CreateService( (BTService_t *)&xGattDemoService,
 (IotBleAttributeEventCallback_t *)pxCallBackArray );
}
```

Portage

Périphériques utilisateur d'entrée et de sortie

Une connexion sécurisée nécessite des entrées et des sorties pour la comparaison numérique. L'événement `eBLENumericComparisonCallback` peut être enregistré à l'aide du gestionnaire d'événements :

```
xEventCb.pxNumericComparisonCb = &prvNumericComparisonCb;
xStatus = BLE_RegisterEventCb( eBLENumericComparisonCallback, xEventCb );
```

Le périphérique doit afficher la clé numérique et utiliser le résultat de la comparaison comme entrée.

Portage des implémentations d'API

Pour porter FreeRTOS vers une nouvelle cible, vous devez implémenter certaines API pour Wi-Fi le service Provisioning et la fonctionnalité Bluetooth Low Energy.

API Bluetooth Low Energy

Pour utiliser le middleware Bluetooth Low Energy FreeRTOS, vous devez implémenter certaines API.

API communes entre GAP pour Bluetooth Classic et GAP pour Bluetooth Low Energy

- `pxBtManagerInit`
- `pxEnable`
- `pxDisable`
- `pxGetDeviceProperty`
- `pxSetDeviceProperty` (toutes les options sont obligatoires, sauf `eBTpropertyRemoteRssi` et `eBTpropertyRemoteVersionInfo`)
- `pxPair`
- `pxRemoveBond`
- `pxGetConnectionState`
- `pxPinReply`
- `pxSspReply`
- `pxGetTxpower`
- `pxGetLeAdapter`

- pxDeviceStateChangedCb
- pxAdapterPropertiesCb
- pxSspRequestCb
- pxPairingStateChangedCb
- pxTxPowerCb

API propres à pour GAP pour 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`

Pour plus d'informations sur le portage de la bibliothèque Bluetooth Low Energy FreeRTOS sur votre plateforme, voir Portage de la bibliothèque Bluetooth Low Energy dans le Guide de portage FreeRTOS.

SDK mobiles pour appareils Bluetooth FreeRTOS

> **Important**
>
> Cette bibliothèque est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous

possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Vous pouvez utiliser les SDK mobiles pour les appareils Bluetooth FreeRTOS pour créer des applications mobiles qui interagissent avec votre microcontrôleur via Bluetooth Low Energy. Les SDK mobiles peuvent également communiquer avec les AWS services, en utilisant Amazon Cognito pour l'authentification des utilisateurs.

SDK Android pour appareils Bluetooth FreeRTOS

Utilisez le SDK Android pour les appareils Bluetooth FreeRTOS pour créer des applications mobiles Android qui interagissent avec votre microcontrôleur via Bluetooth Low Energy. Le SDK est disponible sur. GitHub

Pour installer le SDK Android pour les appareils Bluetooth FreeRTOS, suivez les instructions de « Configuration du SDK » dans le fichier du projet. README.md

Pour plus d'informations sur la configuration et l'exécution de l'application mobile de démonstration qui est incluse dans le kit SDK, consultez Conditions préalables et Application de démonstration du SDK FreeRTOS Bluetooth Low Energy Mobile.

SDK iOS pour appareils Bluetooth FreeRTOS

Utilisez le SDK iOS pour les appareils Bluetooth FreeRTOS pour créer des applications mobiles iOS qui interagissent avec votre microcontrôleur via Bluetooth Low Energy. Le SDK est disponible sur. GitHub

Pour installer le kit SDK iOS

1. Installez CocoaPods :

```
$ gem install cocoapods
$ pod setup
```

> **Note**
>
> Vous devrez peut-être utiliser `sudo` pour installer CocoaPods.

2. Installez le SDK avec CocoaPods (ajoutez ceci à votre podfile) :

```
$ pod 'FreeRTOS', :git => 'https://github.com/aws/amazon-freertos-ble-ios-sdk.git'
```

Pour plus d'informations sur la configuration et l'exécution de l'application mobile de démonstration qui est incluse dans le kit SDK, consultez Conditions préalables et Application de démonstration du SDK FreeRTOS Bluetooth Low Energy Mobile.

Annexe A : Profil MQTT sur BLE GATT

Détails des services du GATT

MQTT over BLE utilise une instance du service de transfert de données GATT pour envoyer des messages MQTT Concise Binary Object Representation (CBOR) entre le périphérique FreeRTOS et le périphérique proxy. Le service de transfert de données expose certaines caractéristiques qui permettent d'envoyer et de recevoir des données brutes via le protocole BLE GATT. Il gère également la fragmentation et l'assemblage de charges utiles supérieures à la taille maximale de l'unité de transfert (MTU) BLE.

UUID du service

`A9D7-166A-D72E-40A9-A002-4804-4CC3-FF00`

Instances de service

Une instance du service GATT est créée pour chaque session MQTT avec le courtier. Chaque service possède un UUID unique (deux octets) qui identifie son type. Chaque instance individuelle est différenciée par son ID d'instance.

Chaque service est instancié en tant que service principal sur chaque périphérique serveur BLE. Vous pouvez créer plusieurs instances du service sur un appareil donné. Le type de service proxy MQTT possède un UUID unique.

Caractéristiques

Format de contenu caractéristique : CBOR

Taille maximale de la valeur caractéristique : 512 octets

Caractéristiques	Exigence	Propriétés obligatoires	Propriétés facultatives	Autorisations de sécurité	Brève description	UUID
Contrôle	M	Écrire	Aucune	L'écriture a besoin d'un chiffrement	Utilisé pour démarrer et arrêter le proxy MQTT.	A9D7-166A-D72E-40A9-A002-4804-4CC3-FF01
Message TX	M	Lire, Notifier	Aucune	La lecture a besoin d'un chiffrement	Utilisé pour envoyer une notification contenant un message à un courtier via un proxy.	A9D7-166A-D72E-40A9-A002-4804-4CC3-FF02
RX Message	M	Lire, écrire sans répondre	Aucune	La lecture et l'écriture ont besoin d'être chiffrées	Utilisé pour recevoir un message d'un courtier via un proxy.	A9D7-166A-D72E-40A9-A002-4804-4CC3-FF03
TXLargeMessage	M	Lire, Notifier	Aucune	La lecture a besoin d'un	Utilisé pour envoyer un	A9D7-166A-D72E-40A9-

Caractéristiques	Exigence	Propriétés obligatoires	Propriétés facultatives	Autorisations de sécurité	Brève description	UUID
				chiffrement	message volumineux (Message > Taille MTU BLE) à un courtier via un proxy.	A002-4804-4CC3-FF04
RXLargeMessage	M	Lire, écrire sans répondre	Aucune	La lecture et l'écriture ont besoin d'être chiffrées	Utilisé pour recevoir un message volumineux (Message > Taille MTU BLE) d'un courtier via un proxy.	A9D7-166A-D72E-40A9-A002-4804-4CC3-FF05

Exigences relatives aux procédures du GATT

Lire les valeurs caractéristiques	Obligatoire
Lire les valeurs caractéristiques longues	Obligatoire
Écrire des valeurs caractéristiques	Obligatoire
Écrire des valeurs caractéristiques longues	Obligatoire

Lire les descripteurs de caractéristiques	Obligatoire
Écrire des descripteurs de caractéristiques	Obligatoire
Notifications	Obligatoire
Indications	Obligatoire

Types de messages

Les types de messages suivants sont échangés.

Type de message	Message	Cartographiez avec ces paires clé/valeur
0x01	CONNECT	• Clé = « w », valeur = type 0 entier, type de message (1) • Clé = « d », valeur = type 3, chaîne de texte, identifiant du client pour la session • Clé = « a », valeur = type 3, chaîne de texte, point de terminaison du courtier pour la session • Clé = « c », valeur = type de valeur simple True/False
0x02	CONNACK	• Clé = « w », valeur = type 0 entier, type de message (2) • Clé = « s », valeur = type 0 entier, code d'état
0x03	PUBLISH	• Clé = « w », valeur = type 0 entier, type de message (3)

Type de message	Message	Cartographiez avec ces paires clé/valeur
		• Clé = « u », valeur = Type 3, chaîne de texte, sujet à publier • Clé = « n », valeur = Type 0, entier, QoS pour la publication • Clé = « i », valeur = type 0, entier, identifiant de message, uniquement pour les publications de QoS 1 • Clé = « k », valeur = type 2, chaîne d'octets, charge utile pour la publication
0x04	PUBACK	• Envoyé uniquement pour les messages QoS 1. • Clé = « w », valeur = type 0 entier, type de message (4) • Clé = « i », valeur = Type 0, entier, identifiant du message

Type de message	Message	Cartographiez avec ces paires clé/valeur
0x08	SUBSCRIBE	• Clé = « w », valeur = type 0 entier, type de message (8) • Key = « v », valeur = Type 4, tableau de chaînes de texte, sujets d'abonnement • Clé = « o », valeur = Type 4, tableau d'entiers, QoS pour l'abonnement • Clé = « i », valeur = Type 0, entier, identifiant du message
0x09	SUBACK	• Clé = « w », valeur = type 0 entier, type de message (9) • Clé = « i », valeur = Type 0, entier, identifiant du message • Clé = « s », valeur = Type 0, entier, code d'état pour l'abonnement
0X0A	SE DÉSABONNER	• Clé = « w », valeur = type 0 entier, type de message (10) • Key = « v », valeur = Type 4, tableau de chaînes de texte, sujets de désinscription • Clé = « i », valeur = Type 0, entier, identifiant du message

Type de message	Message	Cartographiez avec ces paires clé/valeur
0x0B	DÉBAUCHER	• Clé = « w », valeur = type 0 entier, type de message (11) • Clé = « i », valeur = Type 0, entier, identifiant du message • Clé = « s », valeur = Type 0, entier, code d'état pour UnSubscription
0X0C	PINGREQ	• Clé = « w », valeur = type 0 entier, type de message (12)
0x0D	PINGRESP	• Clé = « w », valeur = type 0 entier, type de message (13)
0x0E	DÉCONNECTER	• Clé = « w », valeur = type 0 entier, type de message (14)

Caractéristiques de transfert de charges utiles importantes

TXLargeMessage

TXLargeMessage est utilisé par le périphérique pour envoyer une charge utile importante supérieure à la taille MTU négociée pour la connexion BLE.

- Le dispositif envoie les premiers octets MTU de la charge utile sous forme de notification par le biais de la caractéristique.
- Le proxy envoie une demande de lecture sur cette caractéristique pour les octets restants.
- Le périphérique envoie jusqu'à la taille de la MTU ou jusqu'au nombre d'octets restants de la charge utile, la valeur la plus faible étant retenue. À chaque fois, il augmente le décalage lu en fonction de la taille de la charge utile envoyée.

- Le proxy continuera à lire la caractéristique jusqu'à ce qu'il obtienne une charge utile de longueur nulle ou une charge utile inférieure à la taille de la MTU.
- Si le périphérique ne reçoit pas de demande de lecture dans un délai spécifié, le transfert échoue et le proxy et la passerelle libèrent la mémoire tampon.
- Si le proxy n'obtient pas de réponse de lecture dans le délai spécifié, le transfert échoue et le proxy libère la mémoire tampon.

RXLargeMessage

RXLargeMessage est utilisé par le périphérique pour recevoir une charge utile importante supérieure à la taille MTU négociée pour la connexion BLE.

- Le proxy écrit les messages, jusqu'à la taille MTU, un par un, en utilisant l'option d'écriture avec réponse sur cette caractéristique.
- Le périphérique met le message en mémoire tampon jusqu'à ce qu'il reçoive une demande d'écriture d'une longueur nulle ou inférieure à la taille de la MTU.
- Si le périphérique ne reçoit pas de demande d'écriture dans le délai imparti, le transfert échoue et le périphérique libère la mémoire tampon.
- Si le proxy ne reçoit pas de réponse en écriture dans le délai spécifié, le transfert échoue et le proxy libère la mémoire tampon.

Bibliothèque d'interfaces cellulaires

> **Note**
>
> Le contenu de cette page n'est peut-être pas à jour. Reportez-vous à la page de la FreeRTOS.org bibliothèque pour la dernière mise à jour.

Introduction

La bibliothèque Cellular Interface implémente une API unifiée simple qui masque la complexité des commandes AT spécifiques aux modems cellulaires et expose une interface semblable à un socket aux programmeurs C.

La plupart des modems cellulaires implémentent plus ou moins les commandes AT définies par la norme 3GPP TS v27.007. Ce projet fournit une implémentation de telles commandes AT standard dans un composant commun réutilisable. Les trois bibliothèques d'interface cellulaire de ce projet

tirent toutes parti de ce code commun. La bibliothèque de chaque modem implémente uniquement les commandes AT spécifiques au fournisseur, puis expose l'API complète de la bibliothèque d'interface cellulaire.

Le composant commun qui implémente la norme 3GPP TS v27.007 a été écrit conformément aux critères de qualité du code suivants :

- Les scores de complexité GNU ne sont pas supérieurs à 8
- Norme de C:2012 codage MISRA. Tout écart par rapport à la norme est documenté dans les commentaires du code source marqués de la mention « couverture ».

Dépendances et exigences

Il n'existe aucune dépendance directe pour la bibliothèque de l'interface cellulaire. Cependant, Ethernet Wi-Fi et cellulaire ne peuvent pas coexister dans la pile réseau FreeRTOS. Les développeurs doivent choisir l'une des interfaces réseau à intégrer à la bibliothèque Secure Sockets.

Portage

Pour plus d'informations sur le portage de la bibliothèque d'interface cellulaire sur votre plateforme, consultez la section Portage de la bibliothèque d'interface cellulaire dans le guide de portage de FreeRTOS.

Utilisation de la mémoire

Taille du code de la bibliothèque d'interface cellulaire (exemple généré avec GCC pour ARM Cortex-M)		
Fichier	Avec optimisation -O1	Avec l'optimisation -Os
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 KM	1,2 K
cellular_common_api.	0,5 K	0,5 K
cellular_common.c	1,6 K	1,4 KM

Taille du code de la bibliothèque d'interface cellulaire (exemple généré avec GCC pour ARM Cortex-M)		
Fichier	Avec optimisation -O1	Avec l'optimisation -Os
cellular_pkthandler.c	1,4 KM	1,2 K
cellular_pktio.c	1,8 K	1,6 K
Estimations totales	13,9 K	12,4 K

Prise en main

Téléchargez le code source

Le code source peut être téléchargé dans le cadre des bibliothèques FreeRTOS ou seul.

Pour cloner la bibliothèque depuis Github à l'aide du protocole HTTPS :

```
git clone https://github.com/FreeRTOS/FreeRTOS-Cellular-Interface.git
```

En utilisant SSH :

```
git clone git@github.com:FreeRTOS/FreeRTOS-Cellular-Interface.git
```

Structure du dossier

À la racine de ce dépôt, vous verrez les dossiers suivants :

- `source`: code commun réutilisable qui implémente les commandes AT standard définies par 3GPP TS v27.007
- `doc`: documentation
- `test`: test unitaire et cbmc
- `tools`: outils pour l'analyse statique de Coverity et cMock

Configuration et création de la bibliothèque

La bibliothèque d'interface cellulaire doit être créée dans le cadre d'une application. Pour ce faire, vous devez fournir certaines configurations. Le Cellular_Interface_Windows_Simulator projet

FreeRTOS_ fournit un exemple de configuration du build. Vous trouverez de plus amples informations dans les références de l'API cellulaire.

Reportez-vous à la page de l'interface cellulaire pour plus d'informations.

Intégrez la bibliothèque d'interfaces cellulaires aux plateformes de microcontrôleurs

La bibliothèque d'interface cellulaire fonctionne sur les microcontrôleurs à l'aide d'une interface abstraite, l'interface de communication, pour communiquer avec les modems cellulaires. Une interface de communication doit également être implémentée sur la plate-forme MCU. Les implémentations les plus courantes de l'interface de communication communiquent via le matériel UART, mais peuvent également être implémentées via d'autres interfaces physiques, telles que SPI. La documentation de l'interface de communication se trouve dans les références de l'API de la bibliothèque cellulaire. Les exemples d'implémentation suivants de l'interface de communication sont disponibles :

- Interface de communication du simulateur FreeRTOS pour Windows
- FreeRTOS : interface de communication E/UART commune
- Interface de communication de la carte de découverte STM32 L475
- Interface de communication pour carte Sierra Sensor Hub

Commun I/O

> **Important**
>
> Cette bibliothèque est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Présentation

En général, les pilotes de périphériques sont indépendants du système d'exploitation sous-jacent et sont spécifiques à une configuration matérielle donnée. Une couche d'abstraction matérielle (HAL) fournit une interface commune entre les pilotes et le code d'application de niveau supérieur. La HAL ignore les détails du fonctionnement d'un pilote spécifique et fournit une API uniforme pour

contrôler de tels périphériques. Vous pouvez utiliser les mêmes API pour accéder à divers pilotes de périphériques sur plusieurs cartes de référence basées sur des microcontrôleurs (MCU).

FreeRTOS I/O Common agit comme cette couche d'abstraction matérielle. Elles fournissent un ensemble d'API standard permettant d'accéder aux périphériques série courants sur les cartes de référence prises en charge. Ces API communes communiquent et interagissent avec ces périphériques et permettent à votre code de fonctionner sur toutes les plates-formes. Si ce n'est pas courant I/O, l'écriture de code pour fonctionner avec des appareils de bas niveau est spécifique au fournisseur de silicium.

Périphériques pris en charge

- UART
- SPI
- I2C

Fonctionnalités prises en charge

- Synchrone read/write : la fonction ne revient pas tant que la quantité de données demandée n'est pas transférée.
- Asynchrone read/write — La fonction revient immédiatement et le transfert de données s'effectue de manière asynchrone. Lorsque l'exécution de la fonction est terminée, un rappel utilisateur enregistré est invoqué.

Spécificités relatives au périphérique

- I2C — Combinez plusieurs opérations en une seule transaction. Permet d'écrire puis de lire des actions dans une seule et même transaction.
- SPI — Transférez des données entre le primaire et le secondaire, ce qui signifie que l'écriture et la lecture se font simultanément.

Portage

Pour plus d'informations, consultez le guide de portage de FreeRTOS.

AWS IoT Device Defender bibliothèque

> **Note**
>
> Le contenu de cette page n'est peut-être pas à jour. Reportez-vous à la page de la FreeRTOS.org bibliothèque pour la dernière mise à jour.

Introduction

Vous pouvez utiliser la AWS IoT Device Defender bibliothèque pour envoyer les métriques de sécurité de vos appareils IoT à AWS IoT Device Defender. Vous pouvez les utiliser AWS IoT Device Defender pour surveiller en permanence ces indicateurs de sécurité sur les appareils afin de détecter les écarts par rapport au comportement que vous avez défini comme étant approprié pour chaque appareil. Si quelque chose ne semble pas correct, AWS IoT Device Defender envoie une alerte afin que vous puissiez prendre des mesures pour résoudre le problème. Les interactions AWS IoT Device Defender utilisent MQTT, un protocole léger de publication et d'abonnement. Cette bibliothèque fournit une API pour composer et reconnaître les chaînes de rubriques MQTT utilisées par AWS IoT Device Defender.

Pour plus d'informations, consultez AWS IoT Device Defender dans le Guide du développeur AWS IoT .

La bibliothèque est écrite en C et conçue pour être conforme aux normes ISO C90 et C:2012MISRA. La bibliothèque ne dépend d'aucune bibliothèque supplémentaire autre que la bibliothèque C standard. Il n'a pas non plus de dépendance à la plate-forme, telle que le threading ou la synchronisation. Il peut être utilisé avec n'importe quelle bibliothèque MQTT et n'importe quelle bibliothèque JSON ou CBOR. La bibliothèque possède des preuves démontrant une utilisation sûre de la mémoire et l'absence d'allocation de tas, ce qui la rend adaptée aux microcontrôleurs IoT, mais également entièrement portable sur d'autres plateformes.

La AWS IoT Device Defender bibliothèque peut être utilisée librement et est distribuée sous la licence open source du MIT.

Taille du code de AWS IoT Device Defender (exemple généré avec GCC pour ARM Cortex-M)		
Fichier	Avec optimisation -O1	Avec l'optimisation -Os
defender.c	1,1 K	0,6 K

Taille du code de AWS IoT Device Defender (exemple généré avec GCC pour ARM Cortex-M)		
Fichier	Avec optimisation -O1	Avec l'optimisation -Os
Estimations totales	1,1 K	0,6 K

AWS IoT Greengrass Bibliothèque Discovery

> **Note**
>
> Le contenu de cette page n'est peut-être pas à jour. Reportez-vous à la page de la FreeRTOS.org bibliothèque pour la dernière mise à jour.

Présentation de

La bibliothèque AWS IoT Greengrass Discovery est utilisée par vos microcontrôleurs pour découvrir un cœur Greengrass sur votre réseau. Avec les API AWS IoT Greengrass Discovery, votre appareil peut envoyer des messages à un noyau Greengrass après avoir trouvé le point de terminaison du noyau.

Dépendances et exigences

Pour utiliser la bibliothèque Greengrass Discovery, vous devez y créer un objet AWS IoT, notamment un certificat et une politique. Pour plus d'informations, consultez Mise en route avec AWS IoT.

Vous devez définir des valeurs pour les constantes suivantes dans le fichier *freertos*`/demos/include/aws_clientcredential.h` :

`clientcredentialMQTT_BROKER_ENDPOINT`

Votre AWS IoT point de terminaison.

`clientcredentialIOT_THING_NAME`

Nom de votre objet IoT.

`clientcredentialWIFI_SSID`

Le SSID de votre Wi-Fi réseau.

`clientcredentialWIFI_PASSWORD`

Ton Wi-Fi mot de passe.

`clientcredentialWIFI_SECURITY`

Type de sécurité utilisé par votre Wi-Fi réseau.

Vous devez aussi définir des valeurs pour les constantes suivantes dans le fichier *`freertos`*`/demos/include/aws_clientcredential_keys.h` :

`keyCLIENT_CERTIFICATE_PEM`

PEM de certificat associé à votre objet.

`keyCLIENT_PRIVATE_KEY_PEM`

PEM de clé privée associé à votre objet.

Vous devez disposer d'un groupe Greengrass et d'un appareil principal configuré dans la console. Pour plus d'informations, consultez Getting Started with AWS IoT Greengrass.

Bien que la bibliothèque CoreMQTT ne soit pas requise pour la connectivité Greengrass, nous vous recommandons vivement de l'installer. La bibliothèque peut être utilisée pour communiquer avec le noyau Greengrass une fois qu'il a été détecté.

Référence des API

Pour obtenir une référence d'API complète, consultez la Référence d'API Greengrass.

Exemple d'utilisation

Flux de travail Greengrass

Le périphérique MCU lance le processus de découverte en demandant à AWS IoT un fichier JSON contenant les principaux paramètres de connectivité de Greengrass. Il existe deux méthodes pour récupérer les paramètres de connectivité Greengrass Core à partir du fichier JSON :

- La sélection automatique parcourt tous les noyaux Greengrass répertoriés dans le fichier JSON et se connecte au premier disponible.
- La sélection manuelle utilise les informations contenues dans `aws_ggd_config.h` pour se connecter au noyau Greengrass spécifié.

Comment utiliser l'API Greengrass

Toutes les options de configuration par défaut de l'API Greengrass sont définies dans `aws_ggd_config_defaults.h`.

Si un seul cœur Greengrass est présent, appelez `GGD_GetGGCIPandCertificate` pour demander le fichier JSON avec les informations de connectivité du noyau Greengrass. Lors du retour de `GGD_GetGGCIPandCertificate`, le paramètre `pcBuffer` contient le texte du fichier JSON. Le paramètre `pxHostAddressData` contient l'adresse IP et le port du noyau Greengrass auquel vous pouvez vous connecter.

Pour plus d'options de personnalisation, telles que l'allocation dynamique des certificats, vous devez appeler les API suivantes :

`GGD_JSONRequestStart`

Envoie une requête HTTP GET pour AWS IoT lancer la demande de découverte visant à découvrir un noyau Greengrass. `GD_SecureConnect_Send`est utilisé pour envoyer la demande à AWS IoT.

`GGD_JSONRequestGetSize`

Permet d'obtenir la taille du fichier JSON à partir de la réponse HTTP.

`GGD_JSONRequestGetFile`

Permet d'obtenir la chaîne d'objet JSON. `GGD_JSONRequestGetSize` et `GGD_JSONRequestGetFile` utilisent `GGD_SecureConnect_Read` pour obtenir les données JSON à partir du socket. `GGD_JSONRequestStart`, `GGD_SecureConnect_Send` et `GGD_JSONRequestGetSize` doivent être appelés pour recevoir les données JSON depuis AWS IoT.

`GGD_GetIPandCertificateFromJSON`

Extrait l'adresse IP et le certificat du noyau Greengrass depuis les données JSON. Vous pouvez activer la sélection automatique en définissant `xAutoSelectFlag` avec la valeur `True`. La sélection automatique détecte le premier appareil principal auquel votre appareil FreeRTOS peut se connecter. Pour vous connecter à un noyau Greengrass, appelez la fonction `GGD_SecureConnect_Connect`, en transmettant l'adresse IP, le port et le certificat de l'appareil principal. Pour utiliser la sélection manuelle, définissez les champs suivants du paramètre `HostParameters_t` :

`pcGroupName`

ID du groupe Greengrass auquel le cœur appartient. Vous pouvez utiliser la commande `aws greengrass list-groups` de l'interface de ligne de commande pour trouver l'ID de vos groupes Greengrass.

`pcCoreAddress`

ARN du noyau Greengrass auquel vous vous connectez.

bibliothèque CoreHTTP

> **Note**
>
> Le contenu de cette page n'est peut-être pas à jour. Reportez-vous à la page de la FreeRTOS.org bibliothèque pour la dernière mise à jour.

bibliothèque cliente HTTP C pour petits appareils IoT (MCU ou petit MPU)

Introduction

La bibliothèque CoreHTTP est une implémentation cliente d'un sous-ensemble de la HTTP/1.1norme. La norme HTTP fournit un protocole sans état qui s'exécute au-dessus des systèmes d'information hypertexte collaboratifs TCP/IP et distribués et qui est souvent utilisé dans ces systèmes.

La bibliothèque CoreHTTP implémente un sous-ensemble de la norme de HTTP/1.1protocole. Cette bibliothèque a été optimisée pour un faible encombrement mémoire. La bibliothèque fournit une API entièrement synchrone afin que les applications puissent gérer complètement leur simultanéité. Il utilise uniquement des tampons fixes, de sorte que les applications ont le contrôle total de leur stratégie d'allocation de mémoire.

La bibliothèque est écrite en C et conçue pour être conforme aux normes ISO C90 et C:2012MISRA. Les seules dépendances de la bibliothèque sont la bibliothèque C standard et la version LTS (v12.19.1) de l'analyseur http de. Node.js La bibliothèque possède des preuves démontrant une utilisation sûre de la mémoire et l'absence d'allocation de tas, ce qui la rend adaptée aux microcontrôleurs IoT, mais également entièrement portable sur d'autres plateformes.

Lorsque vous utilisez des connexions HTTP dans des applications IoT, nous vous recommandons d'utiliser une interface de transport sécurisée, telle qu'une interface utilisant le protocole TLS, comme illustré dans leDémo d'authentification mutuelle CoreHTTP.

Cette bibliothèque peut être utilisée librement et est distribuée sous la licence open source du MIT.

Taille du code de CoreHTTP (exemple généré avec GCC pour ARM) Cortex-M		
Fichier	Avec optimisation -O1	Avec l'optimisation -Os
core_http_client.c	3,2 K	2,6 K
api.c (llhttp)	2,6 K	2,0 KM
http.c (llhttp)	0,3 K	0,3 K
llhttp.c (llhttp)	17,9	15,9
Estimations totales	23,9 K	20,7 K

Bibliothèque CoreJSON

> **Note**
>
> Le contenu de cette page n'est peut-être pas à jour. Reportez-vous à la page de la FreeRTOS.org bibliothèque pour obtenir la dernière mise à jour.

Introduction

JSON (JavaScript Object Notation) est un format de sérialisation de données lisible par l'homme. Il est largement utilisé pour échanger des données, notamment avec le service AWS IoT Device Shadow, et fait partie de nombreuses API, telles que l' GitHub API REST. Le JSON est maintenu en tant que norme par Ecma International.

La bibliothèque CoreJSON fournit un analyseur qui prend en charge les recherches de touches tout en appliquant strictement la syntaxe d'échange de données JSON ECMA-404 standard. La bibliothèque est écrite en C et conçue pour être conforme aux normes ISO C90 et C:2012 MISRA. Il possède des preuves démontrant une utilisation sûre de la mémoire et l'absence d'allocation de tas, ce qui le rend adapté aux microcontrôleurs IoT, mais également entièrement portable sur d'autres plateformes.

Utilisation de la mémoire

La bibliothèque CoreJSON utilise une pile interne pour suivre les structures imbriquées dans un document JSON. La pile existe pendant la durée d'un seul appel de fonction ; elle n'est pas préservée. La taille de la pile peut être spécifiée en définissant la macro`JSON_MAX_DEPTH`, dont la valeur par défaut est de 32 niveaux. Chaque niveau consomme un octet.

Taille du code de CoreJSON (exemple généré avec GCC pour ARM) Cortex-M		
Fichier	Avec optimisation -O1	Avec l'optimisation -Os
core_json.c	2,9 K	2,4 K
Estimations totales	2,9 K	2,4 K

bibliothèque CoreMQTT

> **Note**
>
> Le contenu de cette page n'est peut-être pas à jour. Reportez-vous à la page de la FreeRTOS.org bibliothèque pour obtenir la dernière mise à jour.

Introduction

La bibliothèque CoreMQTT est une implémentation cliente de la norme MQTT (Message Queue Telemetry Transport). La norme MQTT fournit un protocole de messagerie léger publish/subscribe (ou PubSub) qui s'exécute par-dessus TCP/IP et est souvent utilisé dans les cas d'utilisation de machine à machine (M2M) et d'Internet des objets (IoT).

La bibliothèque CoreMQTT est conforme à la norme de protocole MQTT 3.1.1. Cette bibliothèque a été optimisée pour un faible encombrement mémoire. La conception de cette bibliothèque couvre différents cas d'utilisation, allant des plateformes aux ressources limitées utilisant uniquement des messages QoS 0 MQTT PUBLISH aux plateformes riches en ressources utilisant des connexions QoS 2 MQTT PUBLISH sur TLS (Transport Layer Security). La bibliothèque fournit un menu de fonctions composables, qui peuvent être choisies et combinées pour répondre précisément aux besoins d'un cas d'utilisation particulier.

La bibliothèque est écrite en C et conçue pour être conforme aux normes ISO C90 et C:2012MISRA. Cette bibliothèque MQTT ne dépend d'aucune bibliothèque supplémentaire, à l'exception des bibliothèques suivantes :

- La bibliothèque C standard
- Interface de transport réseau mise en œuvre par le client
- (Facultatif) Une fonction de temps de plateforme implémentée par l'utilisateur

La bibliothèque est découplée des pilotes réseau sous-jacents grâce à une spécification d'interface de transport simple d'envoi et de réception. Le rédacteur de l'application peut sélectionner une interface de transport existante ou implémenter la sienne en fonction de son application.

La bibliothèque fournit une API de haut niveau pour se connecter à un broker MQTT, subscribe/unsubscribe à un sujet, publier un message sur un sujet et recevoir des messages entrants. Cette API prend l'interface de transport décrite ci-dessus comme paramètre et l'utilise pour envoyer et recevoir des messages depuis et vers le broker MQTT.

La bibliothèque expose également des serializer/deserializer API de bas niveau. Cette API peut être utilisée pour créer une application IoT simple composée uniquement du sous-ensemble requis de fonctionnalités MQTT, sans aucune autre surcharge. L' serializer/deserializer API peut être utilisée conjointement avec n'importe quelle API de couche de transport disponible, comme les sockets, pour envoyer et recevoir des messages à destination et en provenance du courtier.

Lorsque vous utilisez des connexions MQTT dans des applications IoT, nous vous recommandons d'utiliser une interface de transport sécurisée, telle qu'une interface utilisant le protocole TLS.

Cette bibliothèque MQTT ne comporte aucune dépendance à la plate-forme, telle que le threading ou la synchronisation. Cette bibliothèque possède des preuves démontrant une utilisation sûre de la mémoire et l'absence d'allocation de tas, ce qui la rend adaptée aux microcontrôleurs IoT, mais également entièrement portable sur d'autres plateformes. Il peut être utilisé librement et est distribué sous la licence open source du MIT.

Taille du code de CoreMQTT (exemple généré avec GCC pour ARM) Cortex-M		
Fichier	Avec optimisation -O1	Avec l'optimisation -Os
core_mqtt.c	4,0 K	3,4 K

Taille du code de CoreMQTT (exemple généré avec GCC pour ARM) Cortex-M		
Fichier	Avec optimisation -O1	Avec l'optimisation -Os
core_mqtt_state.c	1,7 K	1,3 K
core_mqtt_serializer.c	2,8 K	2,2 K
Estimations totales	8,5 K	6,9 K

Bibliothèque de l'agent CoreMQTT

> **Note**
>
> Le contenu de cette page n'est peut-être pas à jour. Reportez-vous à la page de la FreeRTOS.org bibliothèque pour obtenir la dernière mise à jour.

Introduction

La bibliothèque de l'agent CoreMQTT est une API de haut niveau qui renforce la sécurité des threads. bibliothèque CoreMQTT Il vous permet de créer une tâche d'agent MQTT dédiée qui gère une connexion MQTT en arrière-plan et ne nécessite aucune intervention de la part d'autres tâches. La bibliothèque fournit des équivalents thread-safe aux API de CoreMQTT, de sorte qu'elle peut être utilisée dans des environnements multithread.

L'agent MQTT est une tâche indépendante (ou thread d'exécution). Elle garantit la sécurité des threads en étant la seule tâche autorisée à accéder à l'API de la bibliothèque MQTT. Il sérialise l'accès en isolant tous les appels d'API MQTT vers une seule tâche et élimine le besoin de sémaphores ou d'autres primitives de synchronisation.

La bibliothèque utilise une file d'attente de messagerie sécurisée par thread (ou un autre mécanisme de communication inter-processus) pour sérialiser toutes les demandes d'appel des API MQTT. L'implémentation de la messagerie est découplée de la bibliothèque via une interface de messagerie, ce qui permet de porter la bibliothèque vers d'autres systèmes d'exploitation. L'interface de messagerie est composée de fonctions pour envoyer et recevoir des pointeurs vers les structures de commande de l'agent, et de fonctions pour allouer ces objets de commande, ce qui permet au

rédacteur de l'application de décider de la stratégie d'allocation de mémoire appropriée pour son application.

La bibliothèque est écrite en C et conçue pour être conforme aux normes ISO C90 et C:2012MISRA. La bibliothèque ne dépend d'aucune bibliothèque supplémentaire autre que bibliothèque CoreMQTT la bibliothèque C standard. La bibliothèque possède des preuves démontrant une utilisation sûre de la mémoire et l'absence d'allocation de tas. Elle peut donc être utilisée pour les microcontrôleurs IoT, mais elle est également entièrement portable sur d'autres plateformes.

Cette bibliothèque peut être utilisée librement et est distribuée sous la licence open source du MIT.

Taille du code de l'agent CoreMQTT (exemple généré avec GCC pour ARM) Cortex-M		
Fichier	Avec optimisation -O1	Avec l'optimisation -Os
core_mqtt_agent.c	1,7 K	1,5 KM
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)	2,8 K	2,2 K
Estimations totales	10,5 K	8,6 K

AWS IoT Bibliothèque en direct (OTA)

> **Note**
>
> Le contenu de cette page n'est peut-être pas à jour. Reportez-vous à la page de la FreeRTOS.org bibliothèque pour la dernière mise à jour.

Introduction

La bibliothèque de mises à jour AWS IoT Over-the-air (OTA) vous permet de gérer la notification, le téléchargement et la vérification des mises à jour du microprogramme pour les appareils FreeRTOS en utilisant le protocole HTTP ou MQTT. En utilisant la bibliothèque de l'agent OTA, vous pouvez séparer logiquement les mises à jour des microprogrammes de l'application s'exécutant sur vos appareils. L'agent OTA peut partager une connexion réseau avec l'application. En partageant une connexion réseau, vous pouvez éventuellement enregistrer une quantité importante de mémoire RAM. De plus, la bibliothèque de l'agent OTA vous permet de définir une logique spécifique à l'application pour tester, valider ou annuler une mise à jour du microprogramme.

L'Internet des objets (IoT) étend la connectivité Internet aux appareils embarqués qui n'étaient traditionnellement pas connectés. Ces appareils peuvent être programmés pour communiquer des données utilisables sur Internet et peuvent être surveillés et contrôlés à distance. Grâce aux avancées technologiques, ces appareils embarqués traditionnels obtiennent rapidement des fonctionnalités Internet dans les espaces grand public, industriel et professionnel.

Les appareils IoT sont généralement déployés en grande quantité et souvent dans des endroits difficiles ou peu pratiques d'accès pour un opérateur humain. Imaginez un scénario dans lequel une faille de sécurité susceptible d'exposer des données est découverte. Dans de tels scénarios, il est important de mettre à jour les appareils concernés avec des correctifs de sécurité rapidement et de manière fiable. Sans la possibilité d'effectuer des mises à jour OTA, il peut également être difficile de mettre à jour des appareils géographiquement dispersés. La mise à jour de ces appareils par un technicien sera coûteuse, chronophage et souvent peu pratique. Le temps nécessaire à la mise à jour de ces appareils les expose à des failles de sécurité pendant une période plus longue. Le rappel de ces appareils à des fins de mise à jour sera également coûteux et peut entraîner des perturbations importantes pour les consommateurs en raison des temps d'arrêt.

Les mises à jour sans fil (OTA) permettent de mettre à jour le microprogramme de l'appareil sans rappel coûteux ni visite d'un technicien. Cette méthode apporte les avantages suivants :

- Sécurité : capacité à réagir rapidement aux failles de sécurité et aux bogues logiciels découverts après le déploiement des appareils sur le terrain.
- Innovation - Les produits peuvent être fréquemment mis à jour à mesure que de nouvelles fonctionnalités sont développées, ce qui stimule le cycle d'innovation. Les mises à jour peuvent prendre effet rapidement avec un temps d'arrêt minimal par rapport aux méthodes de mise à jour traditionnelles.

- Coût - Les mises à jour OTA peuvent réduire les coûts de maintenance de manière significative par rapport aux méthodes traditionnellement utilisées pour mettre à jour ces appareils.

La fourniture de la fonctionnalité OTA nécessite les considérations de conception suivantes :

- Communication sécurisée : les mises à jour doivent utiliser des canaux de communication cryptés pour éviter que les téléchargements ne soient falsifiés pendant le transfert.
- Restauration : les mises à jour peuvent échouer pour des raisons telles que la connectivité réseau intermittente ou la réception d'une mise à jour non valide. Dans ces scénarios, l'appareil doit pouvoir revenir à un état stable et éviter de se bloquer.
- Vérification de l'auteur : les mises à jour doivent être vérifiées pour provenir d'une source fiable, tout comme d'autres validations, telles que la vérification de la version et de la compatibilité.

Pour plus d'informations sur la configuration des mises à jour OTA avec FreeRTOS, consultez. Mises à jour de FreeRTOS Over-the-Air

AWS IoT Bibliothèque en direct (OTA)

La bibliothèque AWS IoT OTA vous permet de gérer les notifications relatives aux nouvelles mises à jour disponibles, de les télécharger et d'effectuer une vérification cryptographique des mises à jour du microprogramme. À l'aide de la bibliothèque cliente OTA (over-the-air), vous pouvez séparer logiquement les mécanismes de mise à jour du microprogramme de l'application exécutée sur votre appareil. La bibliothèque cliente OTA (over-the-air) peut partager une connexion réseau avec l'application, ce qui permet d'économiser de la mémoire sur les appareils aux ressources limitées. En outre, la bibliothèque cliente OTA (over-the-air) vous permet de définir une logique spécifique à l'application pour tester, valider ou annuler une mise à jour du microprogramme. La bibliothèque prend en charge différents protocoles d'application tels que Message Queuing Telemetry Transport (MQTT) et Hypertext Transfer Protocol (HTTP) et propose diverses options de configuration que vous pouvez ajuster en fonction du type et des conditions de votre réseau.

Les API de cette bibliothèque fournissent les principales fonctions suivantes :

- Inscrivez-vous pour recevoir des notifications ou interrogez les nouvelles demandes de mise à jour disponibles.
- Recevez, analysez et validez la demande de mise à jour.
- Téléchargez et vérifiez le fichier conformément aux informations contenues dans la demande de mise à jour.

- Exécutez un autotest avant d'activer la mise à jour reçue afin de garantir la validité fonctionnelle de la mise à jour.
- Mettez à jour l'état de l'appareil.

Cette bibliothèque utilise des AWS services pour gérer diverses fonctions liées au cloud, telles que l'envoi de mises à jour du microprogramme, la surveillance d'un grand nombre d'appareils dans plusieurs régions, la réduction du rayon d'action des déploiements défectueux et la vérification de la sécurité des mises à jour. Cette bibliothèque peut être utilisée avec n'importe quelle bibliothèque MQTT ou HTTP.

Les démos de cette bibliothèque présentent des mises à jour complètes en direct à l'aide de la bibliothèque et des services AWS CoreMQTT sur un appareil FreeRTOS.

Caractéristiques

Voici l'interface complète de l'agent OTA :

`OTA_Init`

Initialise le moteur OTA en démarrant l'agent OTA (« tâche OTA ») dans le système. Un seul agent OTA peut exister.

`OTA_Shutdown`

Signale à l'agent OTA de s'arrêter. L'agent OTA peut éventuellement se désabonner de tous les sujets de notification des tâches MQTT, arrêter les tâches OTA en cours, le cas échéant, et effacer toutes les ressources.

`OTA_GetState`

Permet d'obtenir l'état actuel de l'agent OTA.

`OTA_ActivateNewImage`

Active l'image la plus récente du microprogramme du microcontrôleur reçue via OTA. (Le statut détaillé de la tâche doit être désormais test automatique.)

`OTA_SetImageState`

Définit l'état de validation de l'image du microprogramme du microcontrôleur actuellement en cours d'exécution (test, acceptée ou rejetée).

`OTA_GetImageState`

Obtient l'état de validation de l'image du microprogramme du microcontrôleur actuellement en cours d'exécution (test, acceptée ou rejetée).

`OTA_CheckForUpdate`

Demande la prochaine mise à jour OTA disponible à partir du service de mise à jour OTA.

`OTA_Suspend`

Suspendez toutes les opérations de l'agent OTA.

`OTA_Resume`

Reprenez les opérations de l'agent OTA.

`OTA_SignalEvent`

Signalez un événement à la tâche de l'agent OTA.

`OTA_EventProcessingTask`

Boucle de traitement des événements de l'agent OTA.

`OTA_GetStatistics`

Obtenez les statistiques des paquets de messages OTA, notamment le nombre de paquets reçus, mis en file d'attente, traités et abandonnés.

`OTA_Err_strerror`

Conversion du code d'erreur en chaîne pour les erreurs OTA.

`OTA_JobParse_strerror`

Convertissez un code d'erreur OTA Job Parsing en chaîne.

`OTA_PalStatus_strerror`

Conversion du code d'état en chaîne pour le statut OTA PAL.

`OTA_OsStatus_strerror`

Conversion du code d'état en chaîne pour l'état du système d'exploitation OTA.

Référence des API

Pour plus d'informations, consultez la section AWS IoT Over-the-air Mise à jour : fonctions.

Exemple d'utilisation

Une application de OTA-capable périphérique typique utilisant le protocole MQTT pilote l'agent OTA en utilisant la séquence d'appels d'API suivante.

1. Connectez-vous à l'agent AWS IoT CoreMQTT. Pour de plus amples informations, veuillez consulter Bibliothèque de l'agent CoreMQTT.
2. Initialisez l'agent OTA en appelant`OTA_Init`, y compris les tampons, les interfaces OTA requises, le nom de l'objet et le rappel de l'application. Le rappel implémente une logique spécifique à l'application qui s'exécute après avoir terminé la tâche de mise à jour OTA.
3. Lorsque la mise à jour OTA est terminée, FreeRTOS appelle le rappel de fin de tâche avec l'un des événements `accepted` suivants :, ou. `rejected` `self test`
4. Si la nouvelle image du microprogramme a été rejetée (par exemple, en raison d'une erreur de validation), l'application peut généralement ignorer la notification et attendre la prochaine mise à jour.
5. Si la mise à jour est valide et a été marquée comme acceptée, appelez `OTA_ActivateNewImage` pour réinitialiser l'appareil et démarrer la nouvelle image du microprogramme.

Portage

Pour plus d'informations sur le portage des fonctionnalités OTA sur votre plateforme, consultez la section Portage de la bibliothèque OTA dans le Guide de portage de FreeRTOS.

Utilisation de la mémoire

Taille du code de l' AWS IoT OTA (exemple généré avec GCC pour ARM Cortex-M)		
Fichier	Avec optimisation -O1	Avec l'optimisation -Os
otac.	8,3 K	7,5 K
ota_interface.c	0,1 K	0,1 K
ota_base64c	0,6 K	0,6 K
ota_mqtt.c	2,4 K	2,2 K

Taille du code de l' AWS IoT OTA (exemple généré avec GCC pour ARM Cortex-M)		
Fichier	Avec optimisation -O1	Avec l'optimisation -Os
ota_cbor.c	0,8 K	0,6 K
ota_http.c	0,3 K	0,3 K
Estimations totales	12,5 K	11,3 K

bibliothèque CorePKCS11

> **Note**
>
> Le contenu de cette page n'est peut-être pas à jour. Reportez-vous à la page de la FreeRTOS.org bibliothèque pour obtenir la dernière mise à jour.

Présentation de

La norme de cryptographie à clé publique #11 définit une API indépendante de la plate-forme pour gérer et utiliser des jetons cryptographiques. PKCS #11 fait référence à l'API définie par la norme et à la norme elle-même. L'API cryptographique PKCS #11 fait abstraction du stockage des clés, des get/set propriétés des objets cryptographiques et de la sémantique des sessions. Il est largement utilisé pour manipuler des objets cryptographiques courants, et il est important car les fonctions qu'il spécifie permettent aux logiciels d'application d'utiliser, de créer, de modifier et de supprimer des objets cryptographiques, sans jamais exposer ces objets à la mémoire de l'application. Par exemple, les intégrations de référence AWS FreeRTOS utilisent un petit sous-ensemble de l'API PKCS #11 pour accéder à la clé secrète (privée) nécessaire pour créer une connexion réseau authentifiée et sécurisée par le protocole TLS (Transport Layer Security) sans que l'application ne « voie » la clé.

La bibliothèque CorePKCS11 contient une implémentation fictive logicielle de l'interface (API) PKCS #11 qui utilise les fonctionnalités cryptographiques fournies par Mbed TLS. L'utilisation d'une maquette logicielle permet un développement et une flexibilité rapides, mais il est prévu que vous remplaciez la maquette par une implémentation spécifique au stockage sécurisé des clés utilisé dans vos appareils de production. En général, les fournisseurs de cryptoprocesseurs sécurisés, tels que le Trusted Platform Module (TPM), le Hardware Security Module (HSM), le Secure Element ou tout autre type d'enclave matérielle sécurisée, distribuent une implémentation PKCS #11 avec le matériel.

L'objectif de la bibliothèque fictive du logiciel CorePKCS11 est donc de fournir une implémentation PKCS #11 non spécifique au matériel qui permet un prototypage et un développement rapides avant de passer à une implémentation PKCS #11 spécifique au cryptoprocesseur dans les appareils de production.

Seul un sous-ensemble de la norme PKCS #11 est mis en œuvre, en mettant l'accent sur les opérations impliquant des clés asymétriques, la génération de nombres aléatoires et le hachage. Les cas d'utilisation ciblés incluent la gestion des certificats et des clés pour l'authentification TLS et la vérification des signatures de code sur les petits appareils intégrés. Consultez le fichier `pkcs11.h` (obtenu auprès d'OASIS, le corps standard) dans le référentiel de code source FreeRTOS. Dans l'implémentation de référence FreeRTOS, les appels d'API PKCS #11 sont effectués par l'interface d'assistance TLS afin d'effectuer l'authentification du client TLS pendant. `SOCKETS_Connect` Les appels d'API PKCS #11 sont également effectués par notre flux de travail de provisionnement unique pour les développeurs afin d'importer un certificat client TLS et une clé privée à des fins d'authentification auprès du AWS IoT courtier MQTT. Ces deux cas d'utilisation, le provisionnement et l'authentification du client TLS, ne nécessitent la mise en œuvre que d'un petit sous-ensemble de la norme d'interface PKCS #11.

Caractéristiques

Le sous-ensemble suivant de PKCS #11 est utilisé. Cette liste est à peu près dans l'ordre où les routines sont appelées lors de la prise en charge de la mise en service, de l'authentification du client TLS et du nettoyage. Pour une description détaillée des fonctions, consultez la documentation PKCS #11 fournie par le comité de normalisation.

Configuration générale et API de destruction

- `C_Initialize`
- `C_Finalize`
- `C_GetFunctionList`
- `C_GetSlotList`
- `C_GetTokenInfo`
- `C_OpenSession`
- `C_CloseSession`
- `C_Login`

API d’allocation

- `C_CreateObject CKO_PRIVATE_KEY` (pour la clé privée de l'appareil)
- `C_CreateObject CKO_CERTIFICATE` (pour le certificat d'appareil et le certificat de vérification de code)
- `C_GenerateKeyPair`
- `C_DestroyObject`

Authentification client

- `C_GetAttributeValue`
- `C_FindObjectsInit`
- `C_FindObjects`
- `C_FindObjectsFinal`
- `C_GenerateRandom`
- `C_SignInit`
- `C_Sign`
- `C_VerifyInit`
- `C_Verify`
- `C_DigestInit`
- `C_DigestUpdate`
- `C_DigestFinal`

Support cryptosystème asymétrique

L'implémentation de référence FreeRTOS utilise PKCS #11 RSA 2048 bits (signature uniquement) et ECDSA avec la courbe NIST. P-256 Les instructions suivantes décrivent comment créer un AWS IoT objet basé sur un certificat P-256 client.

Assurez-vous d'utiliser les versions suivantes (ou plus récentes) de AWS CLI et d'OpenSSL :

```
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
```

La procédure suivante suppose que vous avez utilisé la `aws configure` commande pour configurer le AWS CLI. Pour plus d'informations, voir Configuration rapide avec `aws configure` dans le Guide de AWS Command Line Interface l'utilisateur.

Pour créer un AWS IoT chose basée sur un certificat P-256 client

1. Créez n'importe AWS IoT quoi.

   ```
   aws iot create-thing --thing-name thing-name
   ```

2. Utilisez OpenSSL pour créer une clé. P-256

   ```
   openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -pkeyopt
    ec_param_enc:named_curve -outform PEM -out thing-name.key
   ```

3. Créez une demande d'inscription de certificat signée par la clé créée à l'étape 2.

   ```
   openssl req -new -nodes -days 365 -key thing-name.key -out thing-name.req
   ```

4. Soumettez la demande d'enregistrement du certificat à 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. Attachez le certificat (référencé par la sortie de l'ARN de la commande précédente) à l'objet.

   ```
   aws iot attach-thing-principal --thing-name thing-name \
     --principal "arn:aws:iot:us-
   east-1:123456789012:cert/
   86e41339a6d1bbc67abf31faf455092cdebf8f21ffbc67c4d238d1326c7de729"
   ```

6. Créez une stratégie. (Cette politique est trop permissive. Il ne doit être utilisé qu'à des fins de développement.)

   ```
   aws iot create-policy --policy-name FullControl --policy-document file://
   policy.json
   ```

Voici le fichier policy.json spécifié dans la commande `create-policy`. Vous pouvez omettre cette `greengrass:*` action si vous ne souhaitez pas lancer la démo de FreeRTOS pour la connectivité et la découverte de Greengrass.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:*",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "greengrass:*",
      "Resource": "*"
    }
  ]
}
```

7. Attachez le mandataire (certificat) et la stratégie à l'objet.

```
aws iot attach-principal-policy --policy-name FullControl \
  --principal "arn:aws:iot:us-
east-1:123456789012:cert/
86e41339a6d1bbc67abf31faf455092cdebf8f21ffbc67c4d238d1326c7de729"
```

Maintenant, suivez les étapes de la section Mise en route d'AWS IoT de ce guide.
N'oubliez pas de copier le certificat et la clé privée que vous avez créés dans votre fichier `aws_clientcredential_keys.h`. Copiez votre nom d'objet dans `aws_clientcredential.h`.

> **Note**
>
> Le certificat et la clé privée sont codés en dur à des fins de démonstration uniquement. Production-level les applications doivent stocker ces fichiers dans un emplacement sécurisé.

Portage

Pour plus d'informations sur le portage de la bibliothèque CorePKCS11 sur votre plateforme, voir Portage de la bibliothèque CorePKCS11 dans le Guide de portage de FreeRTOS.

Utilisation de la mémoire

Taille du code de CorePKCS11 (exemple généré avec GCC pour ARM) Cortex-M		
Fichier	Avec optimisation -O1	Avec l'optimisation -Os
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
Estimations totales	10,2 K	8,6 K

Bibliothèque Secure Sockets

> **Important**
>
> Cette bibliothèque est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Présentation de

Vous pouvez utiliser la bibliothèque FreeRTOS Secure Sockets pour créer des applications intégrées qui communiquent en toute sécurité. Cette bibliothèque a été conçue pour faciliter les premiers pas des développeurs logiciels aux expériences de programmation réseau diverses.

La bibliothèque FreeRTOS Secure Sockets est basée sur l'interface des sockets de Berkeley, avec une option de communication sécurisée supplémentaire par le protocole TLS. Pour plus d'informations sur les différences entre la bibliothèque FreeRTOS Secure Sockets et l'interface des sockets Berkeley, `SOCKETS_SetSockOpt` consultez la référence de l'API Secure Sockets.

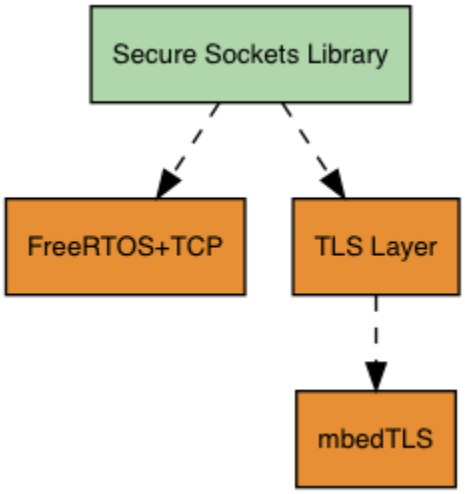
> **Note**
>
> Actuellement, seules les API clientes, ainsi qu'une implémentation IP légère (LWiP) de l'API `Bind` côté serveur, sont prises en charge pour les sockets FreeRTOS Secure.

Dépendances et exigences

La bibliothèque FreeRTOS Secure Sockets dépend d' TCP/IP une pile et d'une implémentation TLS. Les ports pour FreeRTOS répondent à ces dépendances de trois manières :

- Une implémentation personnalisée des deux technologies TCP/IP et du protocole TLS
- Une implémentation personnalisée de et de la TCP/IP couche TLS FreeRTOS avec MbedTLS
- FreeRTOS+TCP et la couche FreeRTOS TLS avec mbedTLS

Le diagramme de dépendance ci-dessous montre l'implémentation de référence incluse dans la bibliothèque FreeRTOS Secure Sockets. Cette implémentation de référence prend en charge le TLS et via TCP/IP Ethernet et Wi-Fi avec FreeRTOS+TCP et MBEDTLS comme dépendances. Pour plus d'informations sur la couche TLS FreeRTOS, consultez. : acte de révision dans un pipeline se poursuivant d'une étape à l'autre dans un flux de travail.



Caractéristiques

Les fonctionnalités de la bibliothèque FreeRTOS Secure Sockets incluent :

- Une interface standard de Berkeley Sockets-based
- Thread-safe API pour envoyer et recevoir des données

- Easy-to-enable TLS

Résolution des problèmes

Codes d'erreur

Les codes d'erreur renvoyés par la bibliothèque FreeRTOS Secure Sockets sont des valeurs négatives. Pour plus d'informations sur chaque code d'erreur, consultez Codes d'erreur Secure Sockets dans la Référence d'API Secure Sockets.

> **Note**
>
> Si l'API FreeRTOS Secure Sockets renvoie un code d'erreur, bibliothèque CoreMQTT le, qui dépend de la bibliothèque FreeRTOS Secure Sockets, renvoie le code d'erreur. `AWS_IOT_MQTT_SEND_ERROR`

Developer Support

La bibliothèque FreeRTOS Secure Sockets inclut deux macros d'assistance pour gérer les adresses IP :

`SOCKETS_inet_addr_quick`

Cette macro convertit une adresse IP qui est exprimée sous la forme de quatre octets numériques distincts en adresse IP exprimée sous la forme d'un nombre 32 bits dans l'ordre des octets de réseau.

`SOCKETS_inet_ntoa`

Cette macro convertit une adresse IP exprimée sous la forme d'un nombre 32 bits dans l'ordre des octets de réseau en chaîne de décimales dans la notation décimale séparée par des points.

Restrictions liées à l'utilisation

Seuls les sockets TCP sont pris en charge par la bibliothèque FreeRTOS Secure Sockets. Les sockets UDP ne sont pas pris en charge.

Les API de serveur ne sont pas prises en charge par la bibliothèque FreeRTOS Secure Sockets, à l'exception d'une implémentation IP légère (LWiP) de l'API côté serveur. `Bind` Les API client sont prises en charge.

Initialisation

Pour utiliser la bibliothèque FreeRTOS Secure Sockets, vous devez initialiser la bibliothèque et ses dépendances. Pour initialiser la bibliothèque Secure Sockets, utilisez le code suivant dans votre application :

```
BaseType_t xResult = pdPASS;
xResult = SOCKETS_Init();
```

Les bibliothèques dépendantes doivent être initialisées séparément. Par exemple, si FreeRTOS+TCP est une dépendance, vous devez également invoquer `FreeRTOS_IPInit` dans votre application.

Référence des API

Pour une référence complète de l'API, consultez la section Référence de l'API Secure Sockets.

Exemple d'utilisation

Le code suivant connecte un client à un serveur.

```
#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 );
}
```

Pour obtenir un exemple complet, consultez Démonstration du client Secure Sockets Echo.

Portage

Les sockets sécurisés FreeRTOS dépendent d' TCP/IP une pile et d'une implémentation TLS. En fonction de votre pile, pour effectuer le portage de la bibliothèque Secure Sockets, vous devez parfois inclure les éléments suivants :

- La pile FreeRTOS+TCP TCP/IP
- La bibliothèque CorePKCS11
- La : acte de révision dans un pipeline se poursuivant d'une étape à l'autre dans un flux de travail.

Pour plus d'informations sur le portage, consultez la section Portage de la bibliothèque Secure Sockets dans le guide de portage de FreeRTOS.

AWS IoT Bibliothèque Device Shadow

> **Note**
>
> Le contenu de cette page n'est peut-être pas à jour. Reportez-vous à la page de la FreeRTOS.org bibliothèque pour la dernière mise à jour.

Introduction

Vous pouvez utiliser la bibliothèque AWS IoT Device Shadow pour stocker et récupérer l'état actuel (l'ombre) de chaque appareil enregistré. L'ombre de l'appareil est une représentation virtuelle persistante de votre appareil avec laquelle vous pouvez interagir dans vos applications Web même si l'appareil est hors ligne. L'état de l'appareil est capturé sous forme d'ombre dans un document JSON. Vous pouvez envoyer des commandes au service AWS IoT Device Shadow via MQTT ou HTTP pour demander le dernier état connu du périphérique ou pour le modifier. L'ombre de chaque appareil est identifiée de manière unique par le nom de l'objet correspondant, une représentation d'un appareil spécifique ou d'une entité logique sur le AWS Cloud. Pour plus d'informations, consultez la section Gestion des appareils avec AWS IoT. Vous trouverez plus de détails sur les ombres dans AWS IoT la documentation.

La bibliothèque AWS IoT Device Shadow ne dépend pas de bibliothèques supplémentaires autres que la bibliothèque C standard. Il n'a pas non plus de dépendance à la plate-forme, telle que le threading ou la synchronisation. Il peut être utilisé avec n'importe quelle bibliothèque MQTT et n'importe quelle bibliothèque JSON.

Cette bibliothèque peut être utilisée librement et est distribuée sous la licence open source du MIT.

Taille du code de AWS IoT Device Shadow (exemple généré avec GCC pour ARM Cortex-M)		
Fichier	Avec optimisation -O1	Avec l'optimisation -Os
shadow.c	1,2 K	0,9 K
Estimations totales	1,2 K	0,9 K

AWS IoT Bibliothèque d'emplois

> **Note**
>
> Le contenu de cette page n'est peut-être pas à jour. Reportez-vous à la page de la FreeRTOS.org bibliothèque pour la dernière mise à jour.

Introduction

AWS IoT Jobs est un service qui avertit un ou plusieurs appareils connectés d'une tâche en attente. Vous pouvez utiliser une tâche pour gérer votre parc d'appareils, mettre à jour le microprogramme et les certificats de sécurité de vos appareils, ou effectuer des tâches administratives telles que le redémarrage des appareils et l'exécution de diagnostics. Pour plus d'informations, consultez la section Offres d'emploi du guide du AWS IoT développeur. Les interactions avec le service AWS IoT Jobs utilisent MQTT, un protocole léger de publication et d'abonnement. Cette bibliothèque fournit une API pour composer et reconnaître les chaînes de rubriques MQTT utilisées par le service AWS IoT Jobs.

La bibliothèque AWS IoT Jobs est écrite en C et conçue pour être conforme aux normes ISO C90 et C:2012MISRA. La bibliothèque ne dépend d'aucune bibliothèque supplémentaire autre que la bibliothèque C standard. Il peut être utilisé avec n'importe quelle bibliothèque MQTT et n'importe quelle bibliothèque JSON. La bibliothèque possède des preuves démontrant une utilisation sûre de la mémoire et l'absence d'allocation de tas, ce qui la rend adaptée aux microcontrôleurs IoT, mais également entièrement portable sur d'autres plateformes.

Cette bibliothèque peut être utilisée librement et est distribuée sous la licence open source du MIT.

Taille du code des AWS IoT tâches (exemple généré avec GCC pour ARM Cortex-M)		
Fichier	Avec optimisation -O1	Avec l'optimisation -Os
jobs.c	1,9 K	1,6 K
Estimations totales	1,9 K	1,6 K

: acte de révision dans un pipeline se poursuivant d'une étape à l'autre dans un flux de travail.

> **Important**
>
> Cette bibliothèque est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

L'interface FreeRTOS Transport Layer Security (TLS) est un wrapper léger et optionnel utilisé pour soustraire les détails de l'implémentation cryptographique à l'interface SSL (Secure Sockets Layer) située au-dessus d'elle dans la pile de protocoles. L'objectif de l'interface TLS est de faciliter le remplacement de la bibliothèque cryptographique logicielle courante, mbed TLS, par une autre implémentation de primitives TLS de chiffrement et de négociation du protocole TLS. L'interface TLS peut être échangée sans avoir à modifier l'interface SSL. Voir `iot_tls.h` dans le dépôt du code source de FreeRTOS.

L'interface TLS est facultative, car vous pouvez choisir de communiquer directement avec une bibliothèque de chiffrement à partir du protocole SSL. L'interface n'est pas utilisée pour les solutions de microcontrôleur qui incluent une implémentation de déchargement complet de la pile pour le protocole TLS et le transport réseau.

Pour plus d'informations sur le portage de l'interface TLS, voir Portage de la bibliothèque TLS dans le Guide de portage de FreeRTOS.

Wi-Fi bibliothèque

> **Important**
>
> Cette bibliothèque est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Présentation de

La bibliothèque Wi-FiFreeRTOS extrait les implémentations Wi-Fi spécifiques aux ports dans une API commune qui simplifie le développement et le portage des applications pour toutes les cartes dotées de fonctionnalités. FreeRTOS-qualified Wi-Fi À l'aide de cette API courante, les applications peuvent communiquer avec leur pile sans fil de niveau inférieur via une interface commune.

Dépendances et exigences

La bibliothèque FreeRTOS nécessite le Wi-Fi noyau FreeRTOS+TCP.

Caractéristiques

La Wi-Fi bibliothèque inclut les fonctionnalités suivantes :

- Support pour l'authentification WEP, WPA, WPA2 et WPA3
- Balayage de point d'accès
- Gestion de l'alimentation
- Profils réseau

Pour plus d'informations sur les fonctionnalités de la Wi-Fi bibliothèque, voir ci-dessous.

Wi-Fi modes

Wi-Fi les appareils peuvent être dans l'un des trois modes suivants : station, point d'accès ou P2P. Vous pouvez obtenir le mode actuel d'un Wi-Fi appareil en appelant`WIFI_GetMode`. Vous pouvez définir le mode Wi-Fi d'un appareil Wi-Fi en appelant `WIFI_SetMode`. Le changement de mode en appelant `WIFI_SetMode` déconnecte l'appareil, s'il est déjà connecté à un réseau.

Mode Station

Configurez votre appareil en mode Station pour connecter la carte à un point d'accès existant.

Mode Access Point (AP)

Configurez votre appareil en mode AP de façon à rendre en faire un point d'accès auquel d'autres appareils peuvent se connecter. Lorsque votre appareil est en mode AP, vous pouvez connecter un autre appareil à votre appareil FreeRTOS et configurer les nouvelles informations d'identification. Wi-Fi Pour configurer le mode AP, appelez `WIFI_ConfigureAP`. Pour placer votre appareil en mode AP, appelez `WIFI_StartAP`. Pour désactiver le mode AP, appelez `WIFI_StopAP`.

> **Note**
>
> Les bibliothèques FreeRTOS ne Wi-Fi fournissent pas de provisionnement en mode AP. Vous devez fournir les fonctionnalités supplémentaires, y compris les capacités de serveur DHCP et HTTP, pour obtenir la prise en charge complète du mode point d'accès.

Mode P2P

Configurez votre appareil en mode P2P pour permettre à plusieurs appareils de se connecter directement l'un à l'autre, sans point d'accès.

Sécurité

L' Wi-Fi API prend en charge les types de sécurité WEP, WPA, WPA2 et WPA3. Lorsqu'un appareil se trouve en mode Station, vous devez spécifier le type de sécurité réseau lors de l'appel de la fonction `WIFI_ConnectAP`. Lorsqu'un appareil est en mode AP, l'appareil peut être configuré pour utiliser n'importe lequel des types de sécurité pris en charge :

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

Analyse et connexion

Pour rechercher des points d'accès à proximité, configurez votre appareil en mode Station et appelez la fonction `WIFI_Scan`. Si vous trouvez un certain réseau dans l'analyse, vous pouvez vous y connecter en appelant `WIFI_ConnectAP` et en fournissant les informations d'identification du réseau. Vous pouvez déconnecter un Wi-Fi appareil du réseau en appelant`WIFI_Disconnect`. Pour plus d'informations sur l'analyse et la connexion, consultez Exemple d'utilisation et Référence des API.

Gestion de l'alimentation

Les différents Wi-Fi appareils ont des besoins en énergie différents, en fonction de l'application et des sources d'alimentation disponibles. Un appareil peut toujours être allumé pour réduire la latence ou

il peut être connecté par intermittence et passer en mode basse consommation lorsque cela n' Wi-Fi est pas nécessaire. L'API d'interface prend en charge divers modes de gestion de l'alimentation, comme toujours activée, alimentation basse et mode normal. Vous définissez le mode d'alimentation d'un appareil à l'aide de la fonction `WIFI_SetPMMode`. Vous pouvez obtenir le mode d'alimentation en cours d'un appareil Wi-Fi en appelant la fonction `WIFI_GetPMMode`.

Profils réseau

La Wi-Fi bibliothèque vous permet d'enregistrer des profils réseau dans la mémoire non volatile de vos appareils. Cela vous permet d'enregistrer les paramètres réseau afin qu'ils puissent être récupérés lorsqu'un appareil se reconnecte à un Wi-Fi réseau, éliminant ainsi le besoin de réapprovisionner les appareils une fois qu'ils ont été connectés à un réseau. `WIFI_NetworkAdd`ajoute un profil réseau. `WIFI_NetworkGet`récupère un profil réseau. `WIFI_NetworkDel`supprime un profil réseau. Le nombre de profils que vous pouvez enregistrer dépend de la plateforme.

Configuration

Pour utiliser la Wi-Fi bibliothèque, vous devez définir plusieurs identifiants dans un fichier de configuration. Pour plus d'informations sur ces identifiants, consultez Référence des API.

> **Note**
>
> La bibliothèque n'inclut pas le fichier de configuration nécessaire. Vous devez en créer un. Lorsque vous créez votre fichier de configuration, veillez à inclure les identifiants de configuration spécifiques à votre carte.

Initialisation

Avant d'utiliser la Wi-Fi bibliothèque, vous devez initialiser certains composants spécifiques à la carte, en plus des composants FreeRTOS. En utilisant le fichier `vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/application_code/main.c` comme modèle pour l'initialisation, procédez de la façon suivante :

1. Supprimez l'exemple de logique de Wi-Fi connexion `main.c` si votre application gère Wi-Fi les connexions. Remplacez l'appel de fonction `DEMO_RUNNER_RunDemos()` suivant :

   ```
   if( SYSTEM_Init() == pdPASS )
   ```

```
    {
    ...
        DEMO_RUNNER_RunDemos();
    ...
        }
```

Par un appel à votre propre application :

```
if( SYSTEM_Init() == pdPASS )
    {
    ...
        // This function should create any tasks
        // that your application requires to run.
        YOUR_APP_FUNCTION();
    ...
        }
```

2. Appelez `WIFI_On()` pour initialiser et allumer votre Wi-Fi puce.

> **Note**
>
> Certaines cartes peuvent nécessiter une initialisation du matériel supplémentaire.

3. Transmettez une `WIFINetworkParams_t` structure configurée `WIFI_ConnectAP()` pour connecter votre carte à un Wi-Fi réseau disponible. Pour plus d'informations sur la structure `WIFINetworkParams_t`, consultez Exemple d'utilisation et Référence des API.

Référence des API

Pour une référence complète sur les API, consultez la section Référence des Wi-Fi API.

Exemple d'utilisation

Connexion à un point d'accès connu

```
#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
}
```

Recherche des points d'accès à proximité

```
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);
}
```

Portage

L'implémentation `iot_wifi.c` nécessite l'implémentation des fonctions définies dans `iot_wifi.h`. Au minimum, l'implémentation doit retourner `eWiFiNotSupported` pour toute fonction non essentielle ou non prise en charge.

Pour plus d'informations sur le portage de la Wi-Fi bibliothèque, voir Portage de la Wi-Fi bibliothèque dans le Guide de portage de FreeRTOS.

Démos de FreeRTOS

FreeRTOS inclut quelques applications de démonstration dans `demos` le dossier, sous le répertoire principal de FreeRTOS. Tous les exemples pouvant être exécutés par FreeRTOS apparaissent dans `common` le dossier, sous. `demos` Il existe également un dossier pour chaque FreeRTOS-qualified plateforme sous le `demos` dossier.

Avant de tester la démonstration des applications, nous vous recommandons de terminer le didacticiel Commencez avec FreeRTOS. Il explique comment configurer et exécuter la démo de l'agent CoreMQTT.

Exécution des démos de FreeRTOS

Les rubriques suivantes expliquent comment configurer et exécuter les démos de FreeRTOS :

- Applications de démonstration Bluetooth Low Energy
- Démonstration du chargeur de démarrage pour la carte Microchip Curiosity PIC32MZEF
- AWS IoT Device Defender démo
- AWS IoT Greengrass Application de démonstration V1 Discovery
- AWS IoT Greengrass V2
- Démos CoreHTTP
- AWS IoT Démo de la bibliothèque d'emplois
- Démos de CoreMQTT
- Over-the-air met à jour l'application de démonstration
- Démonstration du client Secure Sockets Echo
- AWS IoT Application de démonstration Device Shadow

La `DEMO_RUNNER_RunDemos` fonction, située dans le *`freertos`*`/demos/demo_runner/iot_demo_runner.c` fichier, initialise un thread détaché sur lequel s'exécute une seule application de démonstration. Par défaut, appelle et démarre `DEMO_RUNNER_RunDemos` uniquement la démo de l'agent CoreMQTT. En fonction de la configuration que vous avez sélectionnée lors du téléchargement de FreeRTOS et de l'endroit où vous avez téléchargé FreeRTOS, les autres exemples de fonctions d'exécution peuvent démarrer par défaut. Pour activer une application de démonstration, ouvrez le *`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/aws_demo_config.h` fichier et définissez la démo que vous souhaitez exécuter.

> **Note**
>
> Les combinaisons d'exemples ne fonctionnent pas toutes ensemble. Selon la combinaison, il est possible que le logiciel ne s'exécute pas sur la cible sélectionnée en raison de contraintes de mémoire. Nous vous recommandons d'exécuter une démonstration à la fois.

Configuration des démonstrations

Les démos ont été configurées pour vous aider à faire vos premiers pas rapidement. Il se peut que vous souhaitiez modifier certaines configurations pour que votre projet puisse créer une version qui s'exécute sur votre plateforme. Vous pouvez trouver les fichiers de configuration dans `vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files`.

Applications de démonstration Bluetooth Low Energy

> **Important**
>
> Cette démo est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Présentation de

FreeRTOS Bluetooth Low Energy inclut trois applications de démonstration :

- Démonstration MQTT sur Bluetooth Low Energy

Cette application montre comment utiliser le MQTT sur le service Bluetooth Low Energy.

- Démonstration Wi-Fi approvisionnement

  Cette application explique comment utiliser le service Bluetooth Low Energy Wi-Fi Provisioning.

- Démonstration Serveur d'attributs générique (GATT)

  Cette application montre comment utiliser les API intergicielles Bluetooth Low Energy FreeRTOS pour créer un serveur GATT simple.

> **Note**
>
> Pour configurer et exécuter les démos de FreeRTOS, suivez les étapes décrites dans. Commencez avec FreeRTOS

Conditions préalables

Pour suivre ces démonstrations, vous avez besoin d'un microcontrôleur équipé de fonctionnalités Bluetooth Low Energy. Vous avez également besoin du SDK iOS pour appareils Bluetooth FreeRTOS ou du SDK Android pour appareils Bluetooth FreeRTOS.

Configuration AWS IoT et Amazon Cognito pour FreeRTOS Bluetooth Low Energy

Pour connecter vos AWS IoT appareils à MQTT, vous devez configurer Amazon AWS IoT Cognito.

Pour configurer AWS IoT

1. Créez un AWS compte sur https://aws.amazon.com/.
2. Ouvrez la console AWS IoT, puis, dans le volet de navigation, choisissez Gérer, puis Objets.
3. Choisissez Créer, puis Créer un seul objet.
4. Entrez un nom pour votre appareil, puis choisissez Suivant.
5. Si vous connectez votre microcontrôleur au cloud via un appareil mobile, choisissez Créer un objet sans certificat. Dans la mesure où les SDK mobiles utilisent Amazon Cognito pour l'authentification des appareils, il n'est pas nécessaire de créer un certificat d'appareil pour les démos utilisant le Bluetooth Low Energy.

Si vous connectez directement votre microcontrôleur au cloud Wi-Fi, choisissez Créer un certificat, sélectionnez Activer, puis téléchargez le certificat, la clé publique et la clé privée de l'objet.

6. Choisissez l'objet que vous venez de créer dans la liste des objets enregistrés, puis choisissez Interagir à partir de la page de l'objet. Notez le point de terminaison de l' AWS IoT API REST.

Pour plus d'informations sur la configuration, consultez la section Getting Started with AWS IoT.

Pour créer un groupe d'utilisateurs Amazon Cognito

1. Ouvrez la console Amazon Cognito et choisissez Gérer les groupes d'utilisateurs.
2. Sélectionnez Create a user pool.
3. Attribuez un nom au groupe d'utilisateurs, puis choisissez Consulter les valeurs par défaut.
4. Dans le volet de navigation, choisissez Clients d'application, puis Ajouter un client d'application.
5. Tapez un nom pour le client d'application, puis cliquez sur Créer un client d'application.
6. Dans le volet de navigation, sélectionnez Vérification, puis Créer un groupe.

   Prenez note de l'ID du groupe qui s'affiche sur la page Paramètres généraux de votre groupe d'utilisateurs.
7. Dans le volet de navigation, choisissez Clients d'application, puis Afficher les détails. Notez l'ID et le code secret du client d'application.

Pour créer un pool d'identités Amazon Cognito

1. Ouvrez la console Amazon Cognito et choisissez Manage Identity Pools.
2. Saisissez un nom pour le groupe d'identités.
3. Développez Fournisseurs d'authentification, choisissez l'onglet Cognito, puis entrez l'ID de votre groupe d'utilisateurs et l'ID du client d'application.
4. Sélectionnez Créer une réserve.
5. Développez Afficher les détails et notez les deux noms de rôle IAM. Choisissez Autoriser pour créer les rôles IAM pour les identités authentifiées et non authentifiées afin d'accéder à Amazon Cognito.

6. Choisissez Edit identity groupe (Modifier le groupe d'identités).
   Prenez note de l'ID du groupe d'identités. Il devrait avoir la forme `us-west-2:12345678-1234-1234-1234-123456789012`.

Pour plus d'informations sur la configuration d'Amazon Cognito, consultez la section Getting Started with Amazon Cognito.

Pour créer et associer une politique IAM à l'identité authentifiée

1. Ouvrez la console IAM, puis dans le volet de navigation, sélectionnez Rôles.
2. Recherchez et choisissez le rôle de votre identité authentifiée, choisissez Attacher des stratégies, puis Ajouter une stratégie en ligne.
3. Choisissez l'onglet JSON, puis collez le JSON suivant :

   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. Choisissez Examiner une stratégie, entrez un nom pour la stratégie, puis choisissez Créer une stratégie.

Gardez vos informations AWS IoT et celles d'Amazon Cognito à portée de main. Vous avez besoin du point de terminaison et des identifiants pour authentifier votre application mobile auprès du AWS Cloud.

Configurez votre environnement FreeRTOS pour Bluetooth Low Energy

Pour configurer votre environnement, vous devez télécharger FreeRTOS sur votre microcontrôleur, puis télécharger et configurer Bibliothèque Bluetooth Low Energy le SDK mobile pour les appareils Bluetooth FreeRTOS sur votre appareil mobile.

Pour configurer l'environnement de votre microcontrôleur avec FreeRTOS Bluetooth Low Energy

1. Téléchargez ou clonez FreeRTOS depuis. GitHub Consultez le README.mdfichier pour obtenir des instructions.
2. Configurez FreeRTOS sur votre microcontrôleur.

   Pour plus d'informations sur la prise en main de FreeRTOS sur FreeRTOS-qualified un microcontrôleur, consultez le guide de votre carte dans Getting Started with FreeRTOS.

   > **Note**
   >
   > Vous pouvez exécuter les démos sur n'importe quel Energy-enabled microcontrôleur Bluetooth Low doté de FreeRTOS et de bibliothèques FreeRTOS Bluetooth Low Energy portées. Actuellement, le projet de démonstration MQTT sur Bluetooth Low Energy FreeRTOS est entièrement porté sur les appareils Bluetooth Low suivants : Energy-enabled
   >
   > - Espressif ESP32-DevKitC et le ESP-WROVER-KIT
   > - Nordic n RF52840-DK

Composants communs

Les applications de démonstration FreeRTOS ont deux composants communs :

- Network Manager

- Application de démonstration du kit SDK pour appareils mobiles Bluetooth Low Energy

Network Manager

Le gestionnaire de réseau gère la connexion réseau de vos microcontrôleurs. Il se trouve dans votre répertoire FreeRTOS à l'adresse. `demos/network_manager/aws_iot_network_manager.c` Si le gestionnaire de réseau est activé à la fois pour Bluetooth Low Energy Wi-Fi et pour Bluetooth Low Energy, les démos démarrent avec Bluetooth Low Energy par défaut. Si la connexion Bluetooth Low Energy est interrompue et que votre carte mère l'est Wi-Fi-enabled, le gestionnaire de réseau passe à une Wi-Fi connexion disponible pour vous empêcher de vous déconnecter du réseau.

Pour activer un type de connexion réseau avec le Network Manager, ajoutez le type de connexion réseau au `configENABLED_NETWORKS` paramètre dans `vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/aws_iot_network_config.h` (où *`vendor`* il s'agit du nom du fournisseur et du nom de la carte que vous utilisez pour exécuter les démos). *`board`*

Par exemple, si le Bluetooth Low Energy est Wi-Fi activé et que le Bluetooth est activé, la ligne commençant par `#define configENABLED_NETWORKS` in `aws_iot_network_config.h` se lit comme suit :

```
#define  configENABLED_NETWORKS  ( AWSIOT_NETWORK_TYPE_BLE | AWSIOT_NETWORK_TYPE_WIFI )
```

Pour obtenir une liste des types de connexion réseau actuellement pris en charge, consultez les lignes qui commencent par `#define AWSIOT_NETWORK_TYPE` dans `aws_iot_network.h`.

Application de démonstration du SDK FreeRTOS Bluetooth Low Energy Mobile

L'application de démonstration du SDK FreeRTOS Bluetooth Low Energy Mobile se trouve GitHub sur le SDK Android pour les appareils Bluetooth FreeRTOS sous et `amazon-freertos-ble-android-sdk/app` sur le SDK iOS pour les appareils Bluetooth FreeRTOS sous. `amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo` Dans cet exemple, nous utilisons des captures d'écran de la version iOS de l'application de démonstration pour mobile.

> **Note**
>
> Si vous utilisez un appareil iOS, vous avez besoin de Xcode pour générer l'application de démonstration pour mobile. Si vous utilisez un appareil Android, vous pouvez utiliser Android Studio pour générer l'application de démonstration pour mobile.

Pour configurer l'application de démonstration du kit SDK iOS

Lorsque vous définissez des variables de configuration, utilisez le format des valeurs d'espace réservé prévu dans les fichiers de configuration.

1. Confirmez que le SDK iOS pour appareils Bluetooth FreeRTOS est installé.
2. Exécutez la commande suivante à partir de `amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/` :

   ```
   $ pod install
   ```

3. Ouvrez le projet `amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/AmazonFreeRTOSDemo.xcworkspace` avec Xcode et remplacez le compte développeur de signature par votre compte.
4. Créez une AWS IoT politique dans votre région (si ce n'est pas déjà fait).

   > **Note**
   >
   > Cette politique est différente de la politique IAM créée pour l'identité authentifiée Amazon Cognito.

   a. Ouvrez la AWS IoT console.

   b. Dans le volet de navigation, choisissez successivement Sécurisé, Stratégies et Créer. Entrez un nom pour identifier votre stratégie. Dans la section Ajouter des instructions, choisissez Mode avancé. Copiez et collez le code JSON suivant dans la fenêtre de l'éditeur de stratégie. Remplacez *`aws-region`* et *`aws-account`* par votre AWS région et votre numéro de compte.

   c. Choisissez Créer.
5. Ouvrez `amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/AmazonFreeRTOSDemo/Amazon/AmazonConstants.swift` et redéfinissez les variables suivantes :

   - `region`: Votre AWS région.
   - `iotPolicyName`: le nom AWS IoT de votre police.
   - `mqttCustomTopic` : la rubrique MQTT dans laquelle vous voulez publier.

6. Ouvrir `amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/AmazonFreeRTOSDemo/Support/awsconfiguration.json`.

   Sous `CognitoIdentity`, redéfinissez les variables suivantes :

   - `PoolId`: l'identifiant de votre pool d'identités Amazon Cognito.
   - `Region`: Votre AWS région.

   Sous `CognitoUserPool`, redéfinissez les variables suivantes :

   - `PoolId`: l'identifiant de votre groupe d'utilisateurs Amazon Cognito.
   - `AppClientId` : ID de votre client d'application.
   - `AppClientSecret` : clé secrète de votre client d'application.
   - `Region`: Votre AWS région.

Pour configurer l'application de démonstration du kit SDK Android

Lorsque vous définissez des variables de configuration, utilisez le format des valeurs d'espace réservé prévu dans les fichiers de configuration.

1. Confirmez que le SDK Android pour appareils Bluetooth FreeRTOS est installé.
2. Créez une AWS IoT politique dans votre région (si ce n'est pas déjà fait).

   > **Note**
   >
   > Cette politique est différente de la politique IAM créée pour l'identité authentifiée Amazon Cognito.

   a. Ouvrez la AWS IoT console.

   b. Dans le volet de navigation, choisissez successivement Sécurisé, Stratégies et Créer. Entrez un nom pour identifier votre stratégie. Dans la section Ajouter des instructions, choisissez Mode avancé. Copiez et collez le code JSON suivant dans la fenêtre de l'éditeur de stratégie. Remplacez *`aws-region`* et *`aws-account`* par votre AWS région et votre numéro de compte.

   c. Choisissez Créer.

3. Ouvrez https://github.com/aws/amazon-freertos-ble-android-sdk/blob/master/app/src/main/java/software/amazon/freertos/demo/DemoConstants.javaet redéfinissez les variables suivantes :

   - `AWS_IOT_POLICY_NAME`: le nom AWS IoT de votre police.
   - `AWS_IOT_REGION`: Votre AWS région.

4. Ouvert https://github.com/aws/amazon-freertos-ble-android-sdk/blob/master/app/src/main/res/raw/awsconfiguration.json.

   Sous `CognitoIdentity`, redéfinissez les variables suivantes :

   - `PoolId`: l'identifiant de votre pool d'identités Amazon Cognito.
   - `Region`: Votre AWS région.

   Sous `CognitoUserPool`, redéfinissez les variables suivantes :

   - `PoolId`: l'identifiant de votre groupe d'utilisateurs Amazon Cognito.
   - `AppClientId` : ID de votre client d'application.
   - `AppClientSecret` : clé secrète de votre client d'application.
   - `Region`: Votre AWS région.

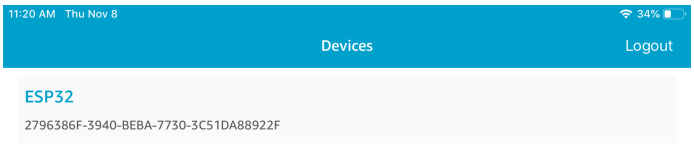
Pour découvrir et établir des connexions sécurisées avec votre microcontrôleur sur Bluetooth Low Energy

1. Pour coupler votre microcontrôleur et votre appareil mobile en toute sécurité (étape 6), vous avez besoin d'un émulateur de terminal série doté à la fois de capacités d'entrée et de sortie (par exemple). TeraTerm Configurez le terminal pour vous connecter à votre carte via une connexion série, comme indiqué dans Installation d'un émulateur de terminal.
2. Exécutez le projet de démonstration Bluetooth Low Energy sur votre microcontrôleur.
3. Exécutez l'application de démonstration du kit SDK mobile Bluetooth Low Energy sur votre appareil mobile.

   Pour démarrer l'application de démonstration dans le kit SDK Android à partir de la ligne de commande, exécutez la commande suivante :

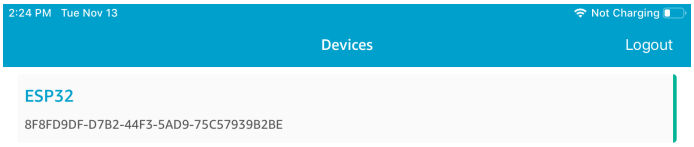
   ```
   $ ./gradlew installDebug
   ```

4. Confirmez que votre microcontrôleur s'affiche sous Appareils dans l'application de démonstration du kit SDK mobile Bluetooth Low Energy.

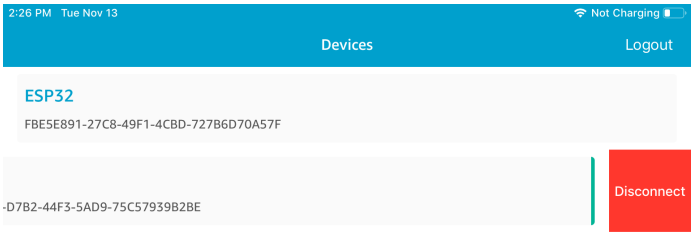


> **Note**
>
> Tous les appareils équipés de FreeRTOS et du service d'information sur les appareils *freertos*/.../device_information () situés à portée apparaissent dans la liste.

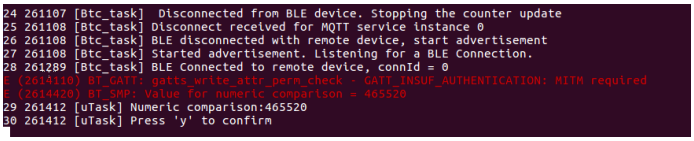
5. Choisissez votre microcontrôleur dans la liste d'appareils. L'application établit une connexion avec la carte, et une ligne verte s'affiche en regard de l'appareil connecté.

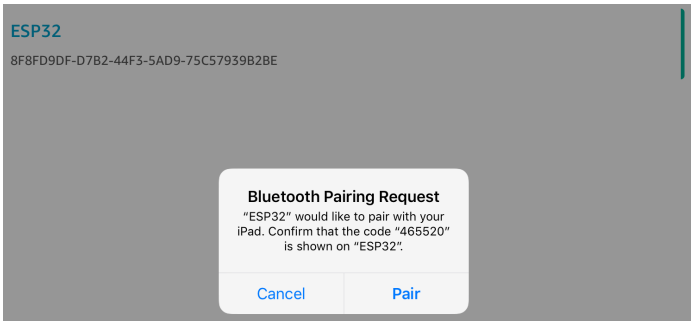


Vous pouvez vous déconnecter de votre microcontrôleur en faisant glisser la ligne vers la gauche.



6. Si vous y êtes invité, couplez votre microcontrôleur et votre appareil mobile.





Si le code de comparaison numérique est le même pour les deux appareils, associez-les.

> **Note**
>
> L'application de démonstration du SDK Bluetooth Low Energy Mobile utilise Amazon Cognito pour l'authentification des utilisateurs. Assurez-vous que vous avez configuré un utilisateur et des groupes d'identités Amazon Cognito, et que vous avez associé des politiques IAM aux identités authentifiées.

MQTT sur Bluetooth Low Energy

Dans la démonstration MQTT over Bluetooth Low Energy, votre microcontrôleur publie des messages dans le AWS Cloud via un proxy MQTT.

Pour vous abonner à une rubrique MQTT de démonstration

1. Connectez-vous à la AWS IoT console.
2. Dans le volet de navigation, choisissez Test, puis choisissez MQTT test client pour ouvrir le client MQTT.
3. Dans le champ Rubrique d'abonnement, saisissez ***thing-name*****/example/topic1**, puis choisissez S'abonner à la rubrique.

Si vous utilisez Bluetooth Low Energy pour associer le microcontrôleur à votre appareil mobile, les messages MQTT sont acheminés via l'application de démonstration du kit SDK Bluetooth Low Energy mobile sur votre appareil mobile.

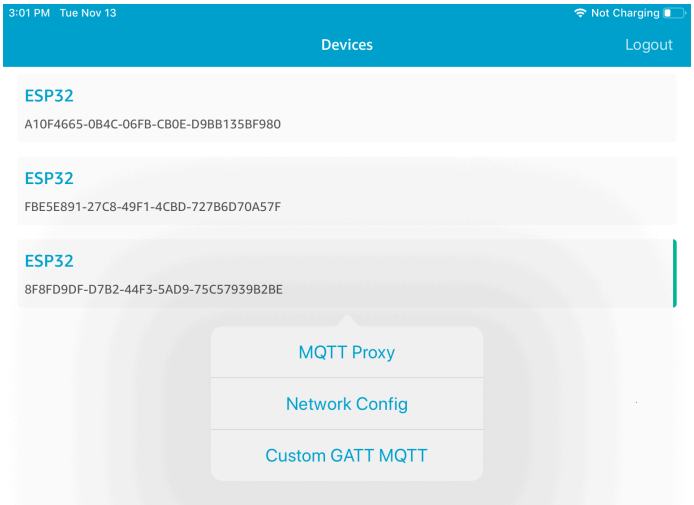
Pour activer la démonstration via Bluetooth Low Energy

1. Ouvrez `vendors/vendor/boards/board/aws_demos/config_files/aws_demo_config.h` et définissez `CONFIG_MQTT_BLE_TRANSPORT_DEMO_ENABLED`.
2. Ouvrez `demos/include/aws_clientcredential.h` et configurez `clientcredentialMQTT_BROKER_ENDPOINT` avec le point de terminaison du AWS IoT courtier. Configurez `clientcredentialIOT_THING_NAME` avec le nom de l'objet du microcontrôleur BLE. Le point de terminaison du AWS IoT broker peut être obtenu à partir de la AWS IoT console en choisissant Paramètres dans le volet de navigation de gauche, ou via la CLI en exécutant la commande :`aws iot describe-endpoint --endpoint-type=iot:Data-ATS`.

> **Note**
>
> Le point de terminaison du AWS IoT courtier et le nom de l'objet doivent tous deux se trouver dans la même région où l'identité cognito et le groupe d'utilisateurs sont configurés.

Pour exécuter la démonstration

1. Créez et exécutez le projet de démonstration sur votre microcontrôleur.
2. Assurez-vous que vous avez associé votre carte et votre appareil mobile à l'aide de l'Application de démonstration du SDK FreeRTOS Bluetooth Low Energy Mobile.
3. Dans la liste Appareils de d'application mobile de démonstration, choisissez votre microcontrôleur, puis choisissez MQTT Proxy pour ouvrir les paramètres de proxy MQTT.



4. Une fois que vous avez activé le proxy MQTT, les messages MQTT apparaissent dans la rubrique *thing-name*/example/topic1, et les données sont imprimées sur le terminal UART.

Wi-Fi approvisionnement

Wi-Fi Provisioning est un service FreeRTOS Bluetooth Low Energy qui vous permet d' Wi-Fi envoyer en toute sécurité des informations d'identification réseau depuis un appareil mobile vers un microcontrôleur via Bluetooth Low Energy. Le code source du service Wi-Fi Provisioning se trouve à l'adresse*freertos*/.../wifi_provisioning.

> **Note**
>
> La démo de Wi-Fi Provisioning est actuellement prise en charge sur l' ESP32-DevKitCEspressif.

Pour activer la démonstration

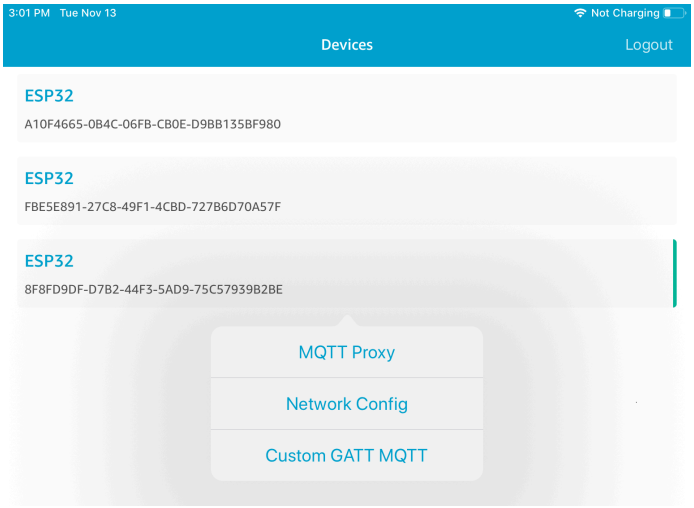
1. Activez le service Wi-Fi Provisioning. Ouvrez vendors/*vendor*/boards/*board*/aws_demos/config_files/iot_ble_config.h et réglez #define IOT_BLE_ENABLE_WIFI_PROVISIONING sur 1 (où *vendor* il s'agit du nom du fournisseur et du *board* nom de la carte que vous utilisez pour exécuter les démos).

   > **Note**
   >
   > Le service Wi-Fi Provisioning est désactivé par défaut.

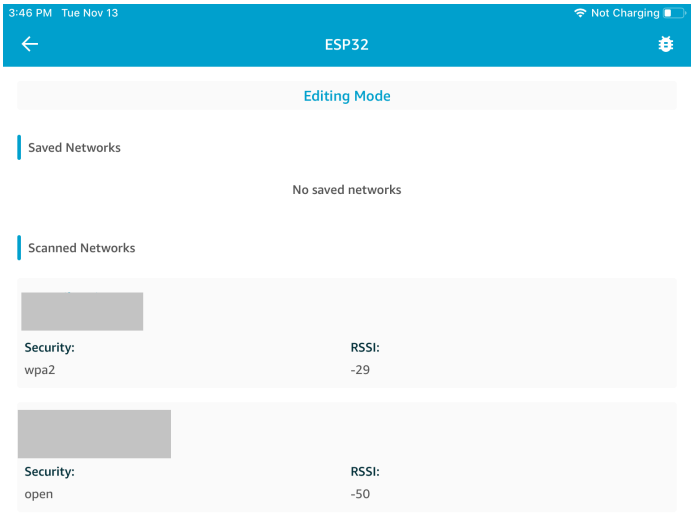
2. Configurez le Network Manager pour activer à la fois Bluetooth Low Energy et Wi-Fi.

Pour exécuter la démonstration

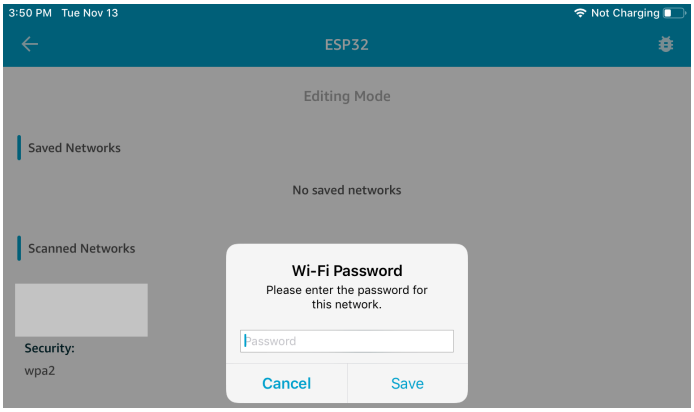
1. Créez et exécutez le projet de démonstration sur votre microcontrôleur.
2. Assurez-vous d'avoir jumelé votre microcontrôleur et votre appareil mobile à l'aide duApplication de démonstration du SDK FreeRTOS Bluetooth Low Energy Mobile.
3. Dans la liste Devices (Appareils) de l'application mobile de démonstration, choisissez votre microcontrôleur, puis choisissez Network Config (Configuration réseau) pour ouvrir les paramètres de configuration du réseau.



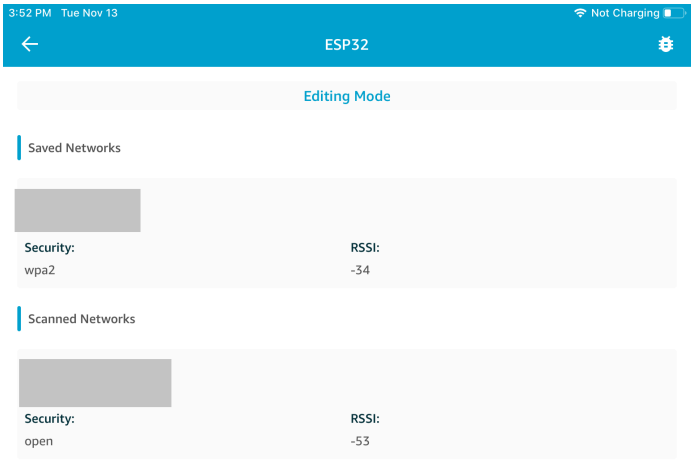
4. Une fois que vous avez choisi Network Config pour votre carte, le microcontrôleur envoie une liste des réseaux à proximité à l'appareil mobile. Wi-Fi Les réseaux disponibles apparaissent dans une liste sous Réseaux scannés.



Dans la liste Scanned Networks, choisissez votre réseau, puis saisissez le SSID et le mot de passe, le cas échéant.

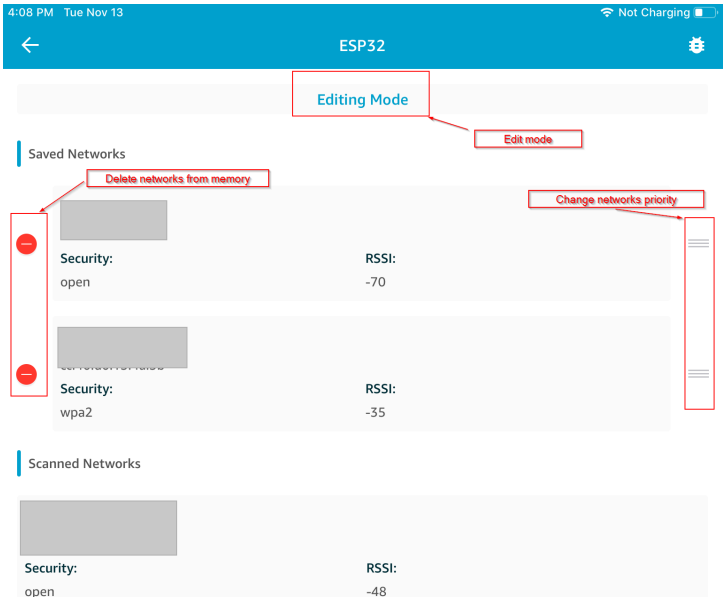


Le microcontrôleur se connecte au réseau et l'enregistre. Le réseau s'affiche sous Réseaux enregistrés.



Vous pouvez enregistrer plusieurs réseaux dans l'application mobile de démonstration. Lorsque vous redémarrez l'application et la démonstration, le microcontrôleur se connecte au premier réseau enregistré disponible, en commençant par le haut de la liste Saved Networks.

Pour modifier l'ordre de priorité des réseaux ou pour supprimer des réseaux, sur la page Network Configuration, choisissez Editing Mode. Pour modifier l'ordre de priorité des réseaux, sélectionnez le côté droit du réseau dont vous souhaitez modifier la priorité, puis faites-le glisser vers le haut ou vers le bas. Pour supprimer un réseau, sélectionnez le bouton rouge sur le côté gauche du réseau que vous souhaitez supprimer.



Serveur d'attributs générique (GATT)

Dans cet exemple, une application de serveur d'attributs génériques (GATT) de démonstration sur votre microcontrôleur envoie une simple valeur de compteur à l'Application de démonstration du SDK FreeRTOS Bluetooth Low Energy Mobile.

Avec le kit SDK Bluetooth Low Energy Mobile, vous pouvez créer votre propre client GATT pour un appareil mobile qui se connecte au serveur GATT sur votre microcontrôleur et s'exécute en parallèle avec l'application mobile de démonstration.

Pour activer la démonstration

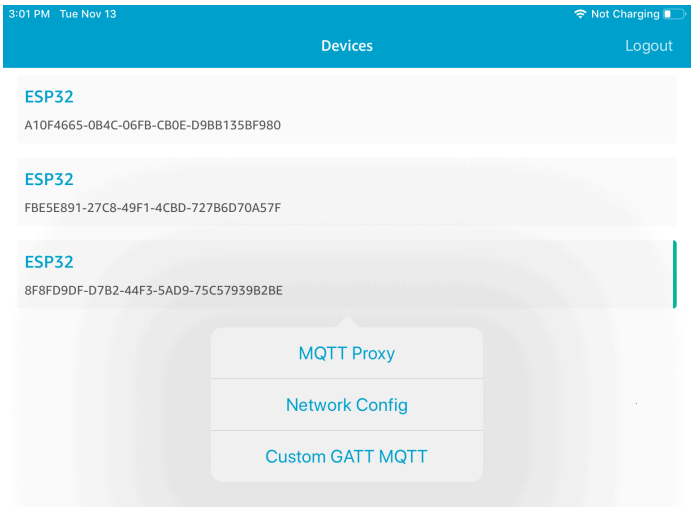
1. Activez la démonstration GATT Bluetooth Low Energy. Dans `vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/iot_ble_config.h` (où *`vendor`* il s'agit du nom du fournisseur et du *`board`* tableau que vous utilisez pour exécuter les démos), ajoutez `#define IOT_BLE_ADD_CUSTOM_SERVICES ( 1 )` à la liste des instructions de définition.

   > **Note**
   >
   > La démonstration GATT Bluetooth Low Energy est désactivée par défaut.

2. Ouvrez *`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/aws_demo_config.h`, commentez `#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED` et définissez `CONFIG_BLE_GATT_SERVER_DEMO_ENABLED`.

Pour exécuter la démonstration

1. Créez et exécutez le projet de démonstration sur votre microcontrôleur.
2. Assurez-vous que vous avez associé votre carte et votre appareil mobile à l'aide de l'Application de démonstration du SDK FreeRTOS Bluetooth Low Energy Mobile.
3. Dans la liste Appareils de d'application, choisissez votre carte, puis choisissez MQTT Proxy pour ouvrir les options de proxy MQTT.



4. Revenez à la liste Appareils, choisissez votre carte, puis choisissez Custom GATT MQTT pour ouvrir les options personnalisées du service GATT.
5. Choisissez Start Counter pour commencer à publier des données à la rubrique MQTT ***your-thing-name*/example/topic**.

   Une fois que vous avez activé le proxy MQTT, Hello World et des messages de compteur incrémentiels s'affichent sur la rubrique *your-thing-name*`/example/topic`.

Démonstration du chargeur de démarrage pour la carte Microchip Curiosity PIC32MZEF

> **Important**
>
> Cette démo est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

> **Note**
>
> En accord avec Microchip, nous retirons le Curiosity PIC32MZEF (DM320104) de la branche principale du référentiel FreeRTOS Reference Integration et ne le proposerons plus dans les nouvelles versions. Microchip a publié un avis officiel indiquant que le PIC32MZEF (DM320104) n'est plus recommandé pour les nouveaux modèles. Les projets et le code source PIC32MZEF sont toujours accessibles via les balises de version précédentes. Microchip recommande aux clients d'utiliser la carte de PIC32MZ-EF-2.0 développement Curiosity (DM320209) pour les nouvelles conceptions. La plate-forme Pic32MZv1 se trouve toujours dans la version v202012.00 du référentiel FreeRTOS Reference Integration. Cependant, la plateforme n'est plus prise en charge par la version v202107.00 du FreeRTOS Reference.

Cette démonstration du chargeur de démarrage implémente le contrôle de la version du microprogramme, la vérification de la signature de chiffrement et le test automatique de l'application. Ces fonctionnalités prennent en charge les mises à jour du microprogramme par liaison radio (OTA) pour FreeRTOS.

La vérification du microprogramme inclut la vérification de l'intégrité et l'authenticité des nouveaux micrologiciels reçus à distance. Le chargeur de démarrage vérifie la signature cryptographique de l'application avant le démarrage. La démonstration utilise l'algorithme de signature numérique à courbe elliptique (ECDSA). SHA-256 Les utilitaires fournis peuvent être utilisés pour générer une application signée qui peut être flashée sur le périphérique.

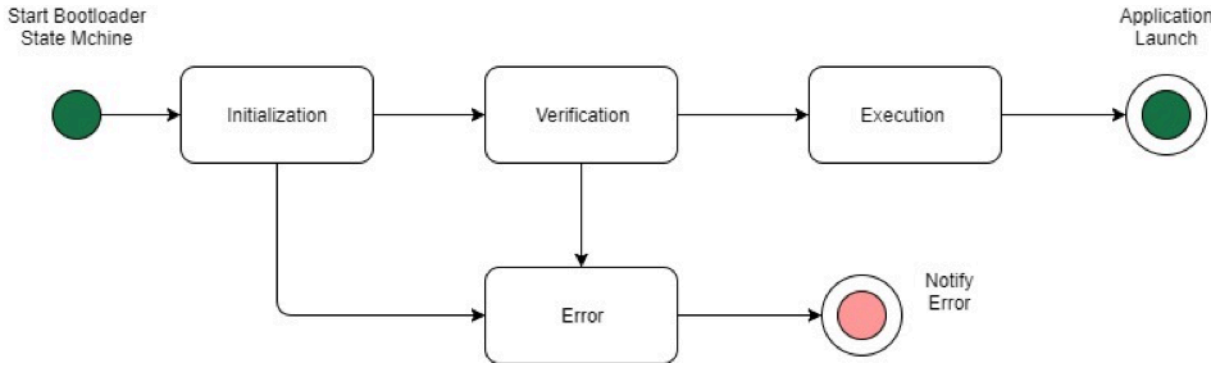
Le chargeur de démarrage prend en charge les fonctionnalités suivantes requises pour OTA :

- Gère les images d'application sur l'appareil et les basculements de l'une à l'autre.
- Autorise l'exécution d'un test automatique de l'image OTA reçue et annule en cas d'échec.
- Vérifie la signature et la version de l'image de la mise à jour OTA.

> **Note**
>
> Pour configurer et exécuter les démos de FreeRTOS, suivez les étapes décrites dans. Commencez avec FreeRTOS

États du chargeur de démarrage

Le processus du chargeur de démarrage est présenté dans la machine d'état suivante.



Le tableau suivant décrit les états du chargeur de démarrage.

État du chargeur de démarrage	Description
Initialisation	Le chargeur de démarrage est dans l'état d'initialisation.
Vérification	Le chargeur de démarrage vérifie les images présentes sur l'appareil.
Exécuter l'image	Le chargeur de démarrage lance l'image sélectionnée.
Exécuter par défaut	Le chargeur de démarrage lance l'image par défaut.
Erreur	Le chargeur de démarrage est dans l'état d'erreur.

Dans le schéma précédent, `Execute Image` et `Execute Default` ont tous deux l'état `Execution`.

Etat d'exécution du chargeur de démarrage

Le chargeur de démarrage est dans l'état `Execution` et est prêt à lancer l'image vérifiée sélectionnée. Si l'image à lancer se trouve dans la banque supérieure, les banques sont

échangées avant l'exécution de l'image, car l'application est toujours créée pour la banque inférieure.

Etat d'exécution du chargeur de démarrage par défaut

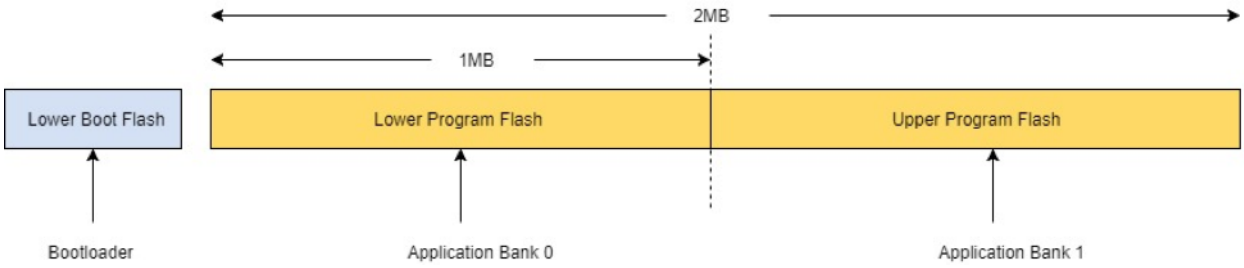
Si l'option de configuration pour lancer l'image par défaut est activée, le chargeur de démarrage lance l'application à partir d'une adresse d'exécution par défaut. Cette option doit être désactivée sauf pendant le débogage.

État d'erreur du chargeur de démarrage

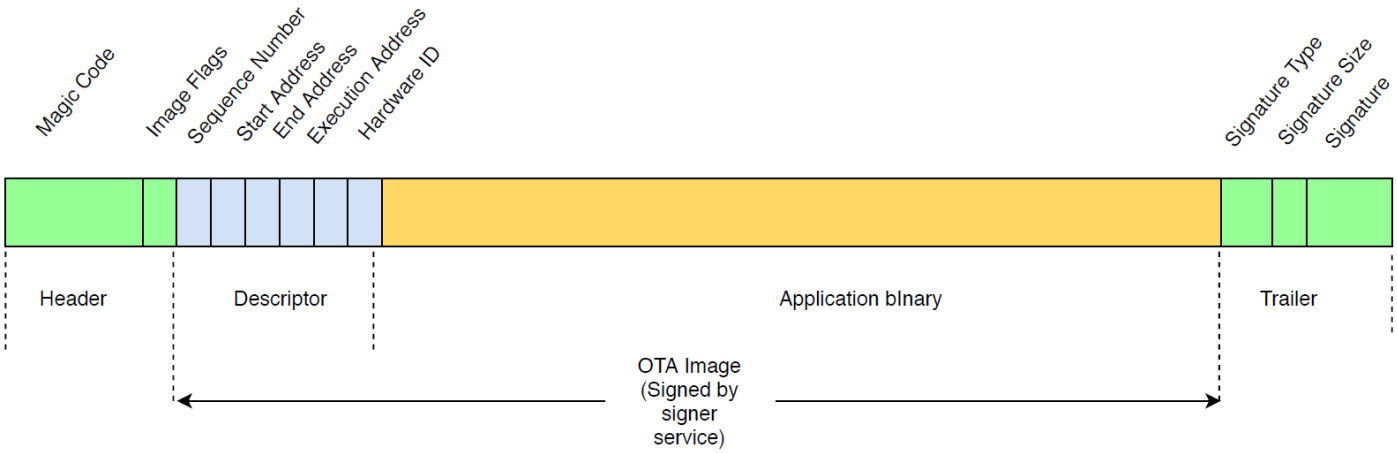
Le chargeur de démarrage est dans un état d'erreur et aucune image valide n'est présente sur l'appareil. Le chargeur de démarrage doit informer l'utilisateur. L'implémentation par défaut envoie un message de journal à la console et le voyant lumineux de la carte clignote rapidement et indéfiniment.

Périphérique flash

La carte Microchip Curiosity PIC32MZEF contient un programme flash interne de deux mégaoctets divisé en deux banques. Il prend en charge la permutation des cartographies mémoire entre ces deux banques, ainsi que les mises à jour en direct. Le chargeur de démarrage de démonstration est programmé dans une autre région flash du démarrage inférieur.



Structure d'image d'application



Le schéma illustre les principaux composants de l'image d'application stockée sur chaque banque de l'appareil.

Composant	Taille (en octets)
En-tête d'image	8 octets
Descripteur d'image	24 bytes
Application binaire	< 1 Mo - (324)
Trailer	292 bytes

En-tête d'image

Les images des applications présentes sur l'appareil doivent commencer par un en-tête qui se compose d'un code magique et d'indicateurs d'image.

Champ d'en-tête	Taille (en octets)
Code magique	7 octets
Indicateurs d'image	1 octet

Code magique

L'image sur l'appareil flash doit commencer par un code magique. Le code magique par défaut est `@AFRTOS`. Le chargeur de démarrage vérifie si un code magique valide est présent avant de démarrer l'image. Il s'agit de la première étape de la vérification.

Indicateurs d'image

Les indicateurs d'image sont utilisés pour stocker le statut des images d'application. Les indicateurs sont utilisés dans le processus OTA. Les indicateurs d'image des deux banques déterminent l'état de l'appareil. Si l'image en cours d'exécution est marquée comme validation en attente, cela signifie que l'appareil est dans la phase de test automatique OTA. Même si les images sur les appareils sont marquées comme valides, elles passent par les mêmes étapes de vérification à chaque démarrage. Si une image est marquée comme nouvelle, le chargeur de démarrage la marque comme en attente de validation et la lance pour le test automatique après la vérification. Le chargeur de démarrage initialise et démarre aussi le temporisateur de surveillance de telle sorte que si la nouvelle image OTA échoue au test automatique, l'appareil redémarre et le chargeur de démarrage rejette l'image en l'effaçant et en exécutant l'image précédente valide.

L'appareil ne peut avoir qu'une seule image valide. L'autre image peut être une nouvelle image OTA ou une validation en attente (test automatique). Après une mise à jour OTA réussie, l'ancienne image est effacée de l'appareil.

Statut	Value	Description
Nouvelle image	0xFF	L'image d'application est nouvelle et jamais exécutée.
Validation en attente	0xFE	L'image d'application est marquée en vue de l'exécution du test.
Valide	0xFC	L'image d'application est marquée comme valide et validée.
Non valide	0xF8	L'image d'application est marquée comme non valide.

Descripteur d'image

L'image d'application sur l'appareil flash doit contenir le descripteur d'image après l'en-tête d'image. Le descripteur d'image est généré par un utilitaire de post-build qui utilise les fichiers de configuration (`ota-descriptor.config`) pour générer le descripteur approprié et l'ajouter à l'application binaire. Le résultat de cette étape post-build est l'image binaire qui peut être utilisée pour l'OTA.

Champ descripteur	Taille (en octets)
Numéro de séquence	4 octets
Adresse de début	4 octets
Adresse de fin	4 octets
Adresse d'exécution	4 octets
ID matériel	4 octets
Instances réservées	4 octets

Numéro de séquence

Le numéro de séquence doit être incrémenté avant de créer une nouvelle image OTA. Consultez le fichier `ota-descriptor.config`. Le chargeur de démarrage utilise ce nombre pour déterminer l'image à démarrer. Les valeurs valides sont comprises entre 1 et 4294967295.

Adresse de début

Adresse de début de l'image d'application sur l'appareil. Comme le descripteur d'image est préajouté à l'application binaire, cette adresse est le début du descripteur d'image.

Adresse de fin

Adresse de fin de l'image d'application sur l'appareil, à l'exclusion de la fin de l'image.

Adresse d'exécution

Adresse d'exécution de l'image.

ID matériel

ID matériel unique utilisé par le chargeur de démarrage pour vérifier que l'image OTA est conçue pour la bonne plateforme.

Instances réservées

Ce champ est réservé pour une utilisation future.

Fin d'image

La fin d'image est ajoutée à l'application binaire. Elle contient la chaîne de type signature, la taille de la signature et la signature de l'image.

Champ de fin	Taille (en octets)
Type de signature	32 bytes
Taille de la signature	4 octets
Signature	256 bytes

Type de signature

Le type de signature est une chaîne qui représente l'algorithme de chiffrement utilisé et qui sert de marqueur pour la fin. Le chargeur de démarrage prend en charge l'algorithme ECDSA. La valeur par défaut est sig-sha256-ecdsa.

Taille de la signature

Taille de la signature de chiffrement, en octets.

Signature

Signature de chiffrement de l'application binaire préfixée par le descripteur de l'image.

Configuration du chargeur de démarrage

Les options de base de configuration du chargeur de démarrage sont fournies dans *freertos*/vendors/microchip/boards/curiosity_pic32mzef/bootloader/config_files/aws_boot_config.h. Certaines options sont fournies à des fins de débogage uniquement.

Activer le démarrage par défaut

Active l'exécution de l'application à partir de l'adresse par défaut et doit être activé pour le débogage uniquement. L'image est exécutée à partir de l'adresse par défaut sans aucune vérification.

Activer la vérification de la signature de chiffrement

Active la vérification de la signature de chiffrement au démarrage. Les images défaillantes sont effacées de l'appareil. Cette option est fournie à des fins de débogage uniquement et doit rester activée en production.

Effacer l'image non valide

Active un effacement complet de la banque si la vérification de l'image sur cette banque échoue. Cette option est fournie à des fins de débogage uniquement et doit rester activée en production.

Activer la vérification de l'ID du matériel

Active la vérification de l'ID du matériel dans le descripteur de l'image OTA et l'ID matériel programmé dans le chargeur de démarrage. Facultatif. Peut être désactivé si la vérification de l'ID du matériel n'est pas obligatoire.

Activer la vérification de l'adresse

Active la vérification des adresses de début, de fin et d'exécution, dans le descripteur de l'image OTA. Nous vous recommandons de garder cette option activée.

Création du chargeur de démarrage

Le bootloader de démonstration est inclus en tant que projet chargeable dans le `aws_demos` projet situé *`freertos`*`/vendors/microchip/boards/curiosity_pic32mzef/aws_demos/mplab/` dans le référentiel de code source FreeRTOS. Lorsque le projet `aws_demos` est créé, il crée d'abord le chargeur de démarrage, puis l'application. La sortie finale est une image hexadécimale unifiée incluant le chargeur de démarrage et l'application. L'utilitaire `factory_image_generator.py` est fourni pour générer une image hexadécimale unifiée avec signature de chiffrement. Les scripts du chargeur de démarrage sont situés dans *`freertos`*`/demos/ota/bootloader/utility/`.

Étape de pré-build du chargeur de démarrage

Cette étape de pré-compilation exécute un script utilitaire appelé `codesigner_cert_utility.py` qui extrait la clé publique du certificat de signature de code et génère un fichier d'en-tête C contenant

la clé publique au format codé Abstract Syntax Notation One ()ASN.1. Cet en-tête est compilé dans le projet du chargeur de démarrage. L'en-tête généré contient deux constantes : un tableau de la clé publique et la longueur de la clé. Le projet du chargeur de démarrage peut également être généré sans `aws_demos` et débogué comme une application normale.

AWS IoT Device Defender démo

> **Important**
>
> Cette démo est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Introduction

Cette démo vous montre comment utiliser la bibliothèque AWS IoT Device Defender pour vous connecter à AWS IoT Device Defender. La démo utilise la bibliothèque CoreMQTT pour établir une connexion MQTT via TLS (authentification mutuelle) avec le courtier AWS IoT MQTT et la bibliothèque CoreJSON pour valider et analyser les réponses reçues du service. AWS IoT Device Defender La démo vous montre comment créer un rapport au format JSON à l'aide des métriques collectées à partir de l'appareil, et comment soumettre le rapport créé au AWS IoT Device Defender service. La démo montre également comment enregistrer une fonction de rappel dans la bibliothèque CoreMQTT pour gérer les réponses du AWS IoT Device Defender service afin de confirmer si un rapport envoyé a été accepté ou rejeté.

> **Note**
>
> Pour configurer et exécuter les démos de FreeRTOS, suivez les étapes décrites dans. Commencez avec FreeRTOS

Fonctionnalité

Cette démonstration crée une tâche d'application unique qui montre comment collecter des métriques, créer un rapport Device Defender au format JSON et le soumettre au AWS IoT Device Defender service via une connexion MQTT sécurisée au courtier AWS IoT MQTT. La démo

inclut les métriques réseau standard ainsi que les métriques personnalisées. Pour les métriques personnalisées, la démo inclut :

- Une métrique nommée « `task_numbers` » qui est une liste d'identifiants de tâches FreeRTOS. Le type de cette métrique est « liste de nombres ».
- Une métrique nommée « `stack_high_water_mark` » qui est le filigrane correspondant à la taille maximale de la tâche de l'application de démonstration. Le type de cette métrique est « nombre ».

La manière dont nous collectons les métriques réseau dépend de la TCP/IP pile utilisée. Pour FreeRTOS+TCP et les configurations LWip prises en charge, nous proposons des implémentations de collecte de métriques qui collectent des métriques réelles à partir de l'appareil et les soumettent dans le rapport. AWS IoT Device Defender Vous pouvez trouver les implémentations pour FreeRTOS +TCP et LWiP sur. GitHub

Pour les cartes utilisant une autre TCP/IP pile, nous fournissons des définitions provisoires des fonctions de collecte de métriques qui renvoient des zéros pour toutes les métriques réseau. Implémentez les fonctions *`freertos`*`/demos/device_defender_for_aws/metrics_collector/stub/metrics_collector.c` de votre pile réseau afin d'envoyer des métriques réelles. Le fichier est également disponible sur le GitHubsite Web.

Pour l'ESP32, la configuration LWip par défaut n'utilise pas le verrouillage du noyau et, par conséquent, la démo utilisera des métriques bloquées. Si vous souhaitez utiliser l'implémentation de la collecte de métriques LWip de référence, définissez les macros suivantes dans : `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
```

Voici un exemple de sortie lorsque vous exécutez la démo.

```
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-------
```

Si votre carte n'utilise pas FreeRTOS+TCP ou une configuration LWip prise en charge, le résultat ressemblera à ce qui suit.

```
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-------
```

Le code source de la démo se trouve dans le *freertos*`/demos/device_defender_for_aws/` répertoire de téléchargement ou sur le GitHubsite Web.

S'abonner à AWS IoT Device Defender topics

La ToDefenderTopics fonction d'abonnement s'abonne aux sujets MQTT sur lesquels les réponses aux rapports Device Defender publiés seront reçues. Il utilise la macro `DEFENDER_API_JSON_ACCEPTED` pour créer la chaîne thématique sur laquelle les réponses aux signalements Device Defender acceptés sont reçues. Il utilise la macro `DEFENDER_API_JSON_REJECTED` pour créer la chaîne thématique sur laquelle les réponses aux signalements rejetés par Device Defender seront reçues.

Collecte des métriques relatives aux appareils

La DeviceMetrics fonction de collecte collecte des métriques réseau à l'aide des fonctions définies dans`metrics_collector.h`. Les métriques collectées sont le nombre d'octets et de paquets envoyés et reçus, les ports TCP ouverts, les ports UDP ouverts et les connexions TCP établies.

Génération du AWS IoT Device Defender report

La DeviceMetricsReport fonction de génération génère un rapport Device Defender à l'aide de la fonction définie dans`report_builder.h`. Cette fonction prend les métriques réseau et une mémoire tampon, crée un document JSON au format attendu AWS IoT Device Defender et l'écrit dans la mémoire tampon fournie. Le format du document JSON attendu par AWS IoT Device Defender est spécifié dans Device-side les métriques du Guide du AWS IoT développeur.

Publier le AWS IoT Device Defender report

Le AWS IoT Device Defender rapport est publié sur la rubrique MQTT pour la publication de AWS IoT Device Defender rapports JSON. Le rapport est construit à l'aide de la macro`DEFENDER_API_JSON_PUBLISH`, comme indiqué dans cet extrait de code sur le GitHub site Web.

Rappel pour gérer les réponses

La fonction PublishCallback gère les messages de publication MQTT entrants. Il utilise l'`Defender_MatchTopic`API de la AWS IoT Device Defender bibliothèque pour vérifier si le message MQTT entrant provient du AWS IoT Device Defender service. Si le message provient du AWS IoT Device Defender service, il analyse la réponse JSON reçue et extrait l'ID du rapport dans la réponse. L'identifiant du rapport est ensuite vérifié pour être le même que celui envoyé dans le rapport.

AWS IoT Greengrass Application de démonstration V1 Discovery

> **Important**
>
> Cette démo est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Avant de lancer la démo AWS IoT Greengrass Discovery pour FreeRTOS, vous devez AWS configurer AWS IoT Greengrass, et. AWS IoT Pour procéder à la configuration AWS, suivez les instructions indiquées surConfiguration de votre AWS compte et autorisations. Pour le configurer AWS IoT Greengrass, vous devez créer un groupe Greengrass, puis ajouter un noyau Greengrass. Pour plus d'informations sur la configuration AWS IoT Greengrass, consultez Getting Started with AWS IoT Greengrass.

Après avoir configuré AWS et AWS IoT Greengrass, vous devez configurer des autorisations supplémentaires pour AWS IoT Greengrass.

Pour configurer AWS IoT Greengrass des autorisations

1. Accédez à la console IAM.
2. Dans le volet de navigation, choisissez Rôles, puis recherchez et choisissez Greengrass_ServiceRole.
3. Choisissez Joindre des politiques, sélectionnez AmazonS3FullAccesset AWSIoTFullAccess, puis choisissez Joindre une politique.
4. Accédez à la console AWS IoT.
5. Dans le panneau de navigation, choisissez Greengrass, Groupes, puis sélectionnez le groupe Greengrass que vous avez créé précédemment.
6. Choisissez Paramètres, puis Ajouter un rôle.
7. Choisissez Greengrass_ServiceRole, puis cliquez sur Enregistrer.

Connectez votre tableau à votre démo FreeRTOS AWS IoT et configurez-la.

1. Enregistrement de votre carte MCU auprès AWS IoT

Une fois que vous avez enregistré votre carte, vous devrez créer et attacher une nouvelle stratégie Greengrass au certificat de l'appareil.

Pour créer un nouveau AWS IoT Greengrass policy

1. Accédez à la console AWS IoT.
2. Dans le volet de navigation, choisissez successivement Sécurisé, Stratégies et Créer.
3. Entrez un nom pour identifier votre stratégie.
4. Dans la section Ajouter des instructions, choisissez Mode avancé. Copiez et collez le code JSON suivant dans la fenêtre de l'éditeur de stratégie :

   ```
   {
         "Effect": "Allow",
         "Action": [
           "greengrass:*"
         ],
         "Resource": "*"
   }
   ```

   Cette politique accorde des AWS IoT Greengrass autorisations à toutes les ressources.
5. Choisissez Créer.

Pour joindre le AWS IoT Greengrass politique relative au certificat de votre appareil

1. Accédez à la console AWS IoT.
2. Dans le panneau de navigation, choisissez Gérer, choisissez Objets, puis choisissez l'objet que vous avez créé précédemment.
3. Choisissez Sécurité, puis choisissez le certificat attaché à votre appareil.
4. Choisissez Stratégies, Actions, puis Attacher une stratégie.
5. Recherchez et choisissez la stratégie Greengrass que vous avez créée précédemment, puis choisissez Attacher.

2. Téléchargement de FreeRTOS

> **Note**
>
> Si vous téléchargez FreeRTOS depuis la console FreeRTOS, choisissez Connect to - au lieu de Connect to AWS IoT Greengrass-. ***Platform*** AWS IoT***Platform***

3. Configuration des démos de FreeRTOS.

   Ouvrez *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h, commentez `#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED` et définissez `CONFIG_GREENGRASS_DISCOVERY_DEMO_ENABLED`.

Après avoir configuré AWS IoT et téléchargé et configuré FreeRTOS AWS IoT Greengrass, vous pouvez créer, flasher et exécuter la démo de Greengrass sur votre appareil. Pour configurer l'environnement de développement du matériel et des logiciels de votre carte, suivez les instructions fournies dans Board-specific guides de démarrage.

La démo de Greengrass publie une série de messages destinés au noyau de Greengrass et au client MQTT. AWS IoT Pour afficher les messages dans le client AWS IoT MQTT, ouvrez la AWS IoT console, choisissez Test, choisissez le client de test MQTT, puis ajoutez un abonnement à. `freertos/demos/ggd`

Dans le client MQTT, vous devez voir les chaînes suivantes :

```
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
```

Utilisation d'une instance Amazon EC2

Si vous travaillez avec une instance Amazon EC2

1. Trouvez le DNS public (IPv4) associé à votre instance Amazon EC2 : accédez à la console Amazon EC2, puis dans le panneau de navigation de gauche, sélectionnez Instances. Choisissez votre instance Amazon EC2, puis sélectionnez le panneau Description. Recherchez l'entrée pour le DNS public (IPv4) et prenez note de celui-ci.

2. Recherchez l'entrée relative aux groupes de sécurité et choisissez le groupe de sécurité attaché à votre instance Amazon EC2.
3. Choisissez l'onglet Règles entrantes puis choisissez Modifier les règles entrantes et ajoutez les règles suivantes.

   Règles entrantes

Type	Protocole	Plage de ports	Source	Description - facultative
HTTP	TCP	80	0,0.0. 0/0	-
HTTP	TCP	80	::/0	-
SSH	TCP	22	0,0.0. 0/0	-
TCP personnal isé	TCP	8883	0,0.0. 0/0	Communica tions MQTT
TCP personnal isé	TCP	8883	::/0	Communica tions MQTT
HTTPS	TCP	443	0,0.0. 0/0	-
HTTPS	TCP	443	::0/0	-
Tous les ICMP - IPv4	ICMP	Tous	0,0.0. 0/0	-
Tous les ICMP - IPv4	ICMP	Tous	::0/0	-

4. Dans la AWS IoT console, choisissez Greengrass, puis Groups, puis choisissez le groupe Greengrass que vous avez créé précédemment. Cliquez sur Paramètres. Modifiez la Détection de la connexion locale en Gérer manuellement les informations de connexion.
5. Dans le panneau de navigation, choisissez Noyaux puis sélectionnez votre noyau de groupe.
6. Choisissez Connectivité et assurez-vous que vous n'avez qu'un seul point de terminaison principal (supprimez tout le reste) et qu'il ne s'agit pas d'une adresse IP (car elle peut être modifiée). La meilleure option est d'utiliser le DNS public (IPv4) que vous avez noté dans la première étape.

7. Ajoutez l'objet IoT FreeRTOS que vous avez créé au groupe GG.

   a. Cliquez sur la flèche de retour pour revenir à la page AWS IoT Greengrass du groupe. Dans le panneau de navigation, choisissez Périphériques puis Ajouter un périphérique.

   b. Choisissez Sélectionner un objet IoT. Choisissez votre appareil, puis sélectionnez Terminer.

8. Ajoutez les abonnements nécessaires : sur la page Greengrass Group, choisissez Abonnements, puis choisissez Ajouter un abonnement et entrez les informations comme indiqué ici.

   Abonnements

Source	Cible	Rubrique
TIGG1	Cloud IoT	freertos/demos/ggd

   Où « Source » est le nom donné à l' AWS IoT objet créé dans la AWS IoT console lorsque vous avez enregistré votre carte, « TIGG1 » dans l'exemple donné ici.

9. Démarrez le déploiement de votre AWS IoT Greengrass groupe et assurez-vous que le déploiement est réussi. Vous devriez maintenant être en mesure d'exécuter correctement la démonstration de AWS IoT Greengrass découverte.

AWS IoT Greengrass V2

Compatibilité avec AWS IoT Greengrass V2 devices

AWS IoT Greengrass V2 le support des appareils clients est rétrocompatible avec. AWS IoT Greengrass V1 Vous pouvez connecter les appareils clients FreeRTOS aux appareils principaux de la V2 sans modifier le code de l'application. Pour permettre aux appareils clients de se connecter à un périphérique principal V2, procédez comme suit.

- Déployez le logiciel Greengrass sur l'appareil principal de Greengrass. Voir Connecter les appareils clients aux appareils principaux auxquels connecter un appareil AWS IoT Greengrass V2.
- Pour relayer des messages (y compris les fonctions Lambda) entre les appareils clients, le service AWS IoT Core cloud et les composants Greengrass, déployez et configurez le composant de pont MQTT.
- Déployez le composant de détection IP pour détecter automatiquement les informations de connectivité ou gérez manuellement les points de terminaison.

- Voir Interagir avec les AWS IoT appareils locaux pour plus d'informations.

Pour plus de détails, consultez AWS la documentation sur l'exécution d'AWS IoT Greengrass V1 applications sur AWS IoT Greengrass V2.

Démos CoreHTTP

> **Important**
>
> Cette démo est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Ces démos peuvent vous aider à apprendre à utiliser la bibliothèque CoreHTTP.

Rubriques

- Démo d'authentification mutuelle CoreHTTP
- Démo de téléchargement de CoreHTTP Basic sur Amazon S3
- Démo de téléchargement de CoreHTTP basic S3
- Démo multithread de base de CoreHTTP

Démo d'authentification mutuelle CoreHTTP

> **Important**
>
> Cette démo est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Introduction

Le projet de démonstration CoreHTTP (authentification mutuelle) vous montre comment établir une connexion à un serveur HTTP à l'aide du protocole TLS avec authentification mutuelle entre le client

et le serveur. Cette démonstration utilise une implémentation d'interface de TLS-based transport mbed pour établir une connexion TLS authentifiée par le serveur et le client, et illustre un flux de travail de réponse aux demandes en HTTP.

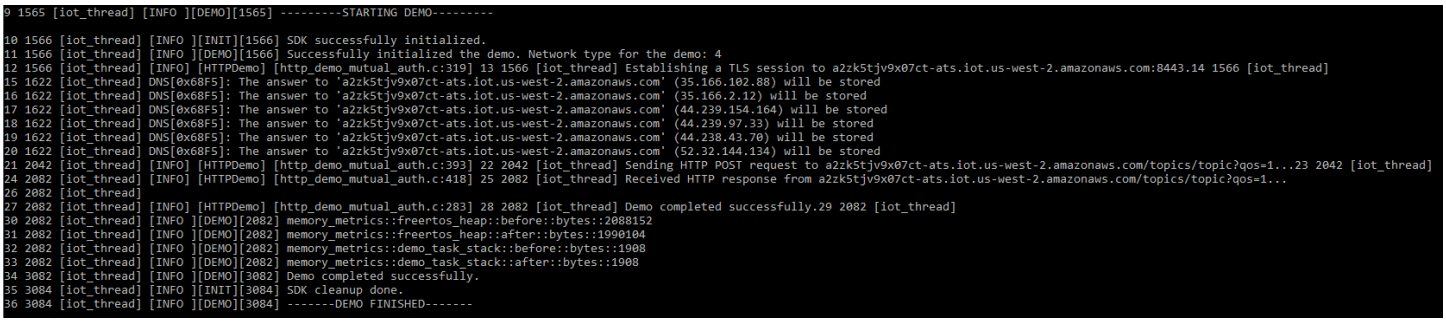
> **Note**
>
> Pour configurer et exécuter les démos de FreeRTOS, suivez les étapes décrites dans. Commencez avec FreeRTOS

Fonctionnalité

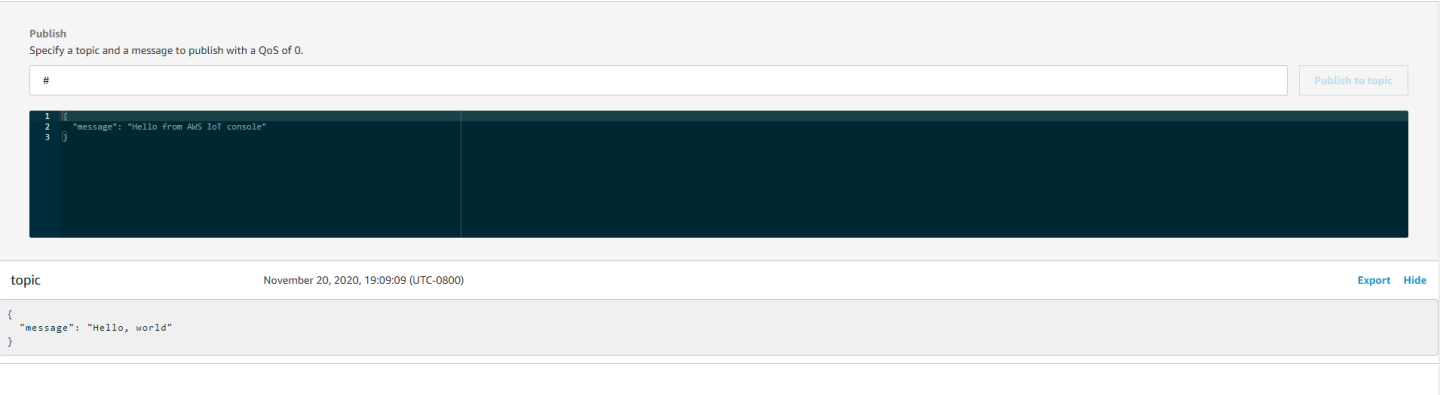
Cette démonstration crée une tâche d'application unique avec des exemples qui montrent comment effectuer les opérations suivantes :

- Connectez-vous au serveur HTTP sur le AWS IoT terminal.
- Envoyez une requête POST.
- Recevez la réponse.
- Déconnectez-vous du serveur.

Une fois ces étapes terminées, la démo génère un résultat similaire à la capture d'écran suivante.



La AWS IoT console génère une sortie similaire à la capture d'écran suivante.



Organisation du code source

Le fichier source de démonstration est nommé `http_demo_mutual_auth.c` et se trouve dans le *`freertos`*`/demos/coreHTTP/` répertoire et sur le GitHubsite Web.

Connexion au AWS IoT Serveur HTTP

La ToServerWithBackoffRetries fonction de connexion tente d'établir une connexion TLS mutuellement authentifiée avec le serveur AWS IoT HTTP. Si la connexion échoue, elle réessaie après un délai d'expiration. La valeur du délai d'attente augmente de façon exponentielle jusqu'à ce que le nombre maximum de tentatives soit atteint ou que la valeur maximale du délai d'attente soit atteinte. La `RetryUtils_BackoffAndSleep` fonction fournit des valeurs de délai d'attente qui augmentent de façon exponentielle et revient `RetryUtilsRetriesExhausted` lorsque le nombre maximum de tentatives a été atteint. La `connectToServerWithBackoffRetries` fonction renvoie un état d'échec si la connexion TLS au broker ne peut pas être établie après le nombre de tentatives configuré.

Envoi d'une requête HTTP et réception de la réponse

La SendHttpRequest fonction prv montre comment envoyer une requête POST au serveur AWS IoT HTTP. Pour plus d'informations sur l'envoi d'une requête à l'API REST dans AWS IoT, voir Protocoles de communication entre appareils - HTTPS. La réponse est reçue avec le même appel d'API CoreHTTP,`HTTPClient_Send`.

Démo de téléchargement de CoreHTTP Basic sur Amazon S3

> **Important**
>
> Cette démo est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez

déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Introduction

Cet exemple montre comment envoyer une requête PUT au serveur HTTP Amazon Simple Storage Service (Amazon S3) et comment télécharger un petit fichier. Il exécute également une requête GET pour vérifier la taille du fichier après le téléchargement. Cet exemple utilise une interface de transport réseau qui utilise mbedTLS pour établir une connexion authentifiée mutuellement entre un client d'appareil IoT exécutant CoreHTTP et le serveur HTTP Amazon S3.

> **Note**
>
> Pour configurer et exécuter les démos de FreeRTOS, suivez les étapes décrites dans. Commencez avec FreeRTOS

À filetage unique ou à filetage multiple

Il existe deux modèles d'utilisation de CoreHTTP, monothread et multithread (multitâche). Bien que la démonstration présentée dans cette section exécute la bibliothèque HTTP dans un thread, elle montre en fait comment utiliser CoreHTTP dans un environnement à thread unique. Dans cette démo, une seule tâche utilise l'API HTTP. Bien que les applications à thread unique doivent appeler à plusieurs reprises la bibliothèque HTTP, les applications multithread peuvent à la place envoyer des requêtes HTTP en arrière-plan dans le cadre d'une tâche d'agent (ou de démon).

Organisation du code source

Le fichier source de démonstration est nommé `http_demo_s3_upload.c` et se trouve dans le *`freertos`*`/demos/coreHTTP/` répertoire et sur le GitHubsite Web.

Configuration de la connexion au serveur HTTP Amazon S3

Cette démonstration utilise une URL pré-signée pour se connecter au serveur HTTP Amazon S3 et autoriser l'accès à l'objet à télécharger. La connexion TLS du serveur HTTP Amazon S3 utilise uniquement l'authentification du serveur. Au niveau de l'application, l'accès à l'objet est authentifié à l'aide de paramètres dans la requête URL pré-signée. Suivez les étapes ci-dessous pour configurer votre connexion à AWS.

1. Créez un AWS compte :

   a. Si ce n'est pas déjà fait, créez un AWS compte.

   b. Les comptes et les autorisations sont définis à l'aide de Gestion des identités et des accès AWS (IAM). Vous utilisez IAM pour gérer les autorisations de chaque utilisateur de votre compte. Par défaut, un utilisateur ne dispose d'aucune autorisation tant que le propriétaire root ne l'a pas accordée.

      i. Pour ajouter un utilisateur à votre AWS compte, consultez le guide de l'utilisateur IAM.

      ii. Autorisez votre AWS compte à accéder à FreeRTOS AWS IoT et en ajoutant cette politique :

      - AmazonS3FullAccess

2. Créez un compartiment dans Amazon S3 en suivant les étapes décrites dans Comment créer un compartiment S3 ? dans le guide de l'utilisateur d'Amazon Simple Storage Service.
3. Chargez un fichier sur Amazon S3 en suivant les étapes décrites dans Comment télécharger des fichiers et des dossiers dans un compartiment S3 ? .
4. Générez une URL pré-signée à l'aide du script situé dans le `FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/presigned_urls_gen.py` fichier.

   Pour les instructions d'utilisation, consultez le `FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/README.md` fichier.

Fonctionnalité

La démo se connecte d'abord au serveur HTTP Amazon S3 avec l'authentification du serveur TLS. Il crée ensuite une requête HTTP pour télécharger les données spécifiées dans`democonfigDEMO_HTTP_UPLOAD_DATA`. Après avoir chargé le fichier, il vérifie que le fichier a bien été téléchargé en demandant la taille du fichier. Le code source de la démo se trouve sur le GitHubsite Web.

Connexion au serveur HTTP Amazon S3

La ToServerWithBackoffRetries fonction de connexion tente d'établir une connexion TCP avec le serveur HTTP. Si la connexion échoue, elle réessaie après un délai d'expiration. La valeur du délai d'attente augmentera de façon exponentielle jusqu'à ce que le nombre maximum

de tentatives soit atteint ou que la valeur maximale du délai d'expiration soit atteinte. La `connectToServerWithBackoffRetries` fonction renvoie un état d'échec si la connexion TCP au serveur ne peut pas être établie après le nombre de tentatives configuré.

La `prvConnectToServer` fonction montre comment établir une connexion au serveur HTTP Amazon S3 en utilisant uniquement l'authentification du serveur. Il utilise l'interface de TLS-based transport mbed implémentée dans le `FreeRTOS-Plus/Source/Application-Protocols/network_transport/freertos_plus_tcp/using_mbedtls/using_mbedtls.c` fichier. La définition de `prvConnectToServer` se trouve sur le GitHubsite Web.

Charger des données

La `prvUploadS3ObjectFile` fonction montre comment créer une requête PUT et spécifier le fichier à télécharger. Le compartiment Amazon S3 dans lequel le fichier est chargé et le nom du fichier à télécharger sont spécifiés dans l'URL pré-signée. Pour économiser de la mémoire, la même mémoire tampon est utilisée à la fois pour les en-têtes de demande et pour recevoir la réponse. La réponse est reçue de manière synchrone à l'aide de la fonction `HTTPClient_Send` API. Un code d'état de `200 OK` réponse est attendu du serveur HTTP Amazon S3. Tout autre code d'état est une erreur.

Le code source de `prvUploadS3ObjectFile()` se trouve sur le GitHubsite Web.

Vérification du téléchargement

La `prvVerifyS3ObjectFileSize` fonction appelle `prvGetS3ObjectFileSize` pour récupérer la taille de l'objet dans le compartiment S3. Le serveur HTTP Amazon S3 ne prend actuellement pas en charge les requêtes HEAD utilisant une URL pré-signée. Le 0ème octet est donc demandé. La taille du fichier est indiquée dans le champ d'`Content-Range`en-tête de la réponse. Une `206 Partial Content` réponse est attendue du serveur. Tout autre code d'état de réponse est une erreur.

Le code source de `prvGetS3ObjectFileSize()` se trouve sur le GitHubsite Web.

Démo de téléchargement de CoreHTTP basic S3

> **Important**
>
> Cette démo est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez

déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Introduction

Cette démonstration montre comment utiliser les requêtes de plage pour télécharger des fichiers depuis le serveur HTTP Amazon S3. Les requêtes Range sont prises en charge de manière native dans l'API CoreHTTP lorsque vous les utilisez `HTTPClient_AddRangeHeader` pour créer la requête HTTP. Pour un environnement de microcontrôleur, les demandes de plage sont vivement encouragées. En téléchargeant un fichier volumineux dans des plages séparées, au lieu d'une seule demande, chaque section du fichier peut être traitée sans bloquer le socket réseau. Les demandes de plage réduisent le risque de perte de paquets, qui nécessitent des retransmissions sur la connexion TCP, et réduisent ainsi la consommation d'énergie de l'appareil.

Cet exemple utilise une interface de transport réseau qui utilise mbedTLS pour établir une connexion authentifiée mutuellement entre un client d'appareil IoT exécutant CoreHTTP et le serveur HTTP Amazon S3.

> **Note**
>
> Pour configurer et exécuter les démos de FreeRTOS, suivez les étapes décrites dans. Commencez avec FreeRTOS

À filetage unique ou à filetage multiple

Il existe deux modèles d'utilisation de CoreHTTP, monothread et multithread (multitâche). Bien que la démonstration présentée dans cette section exécute la bibliothèque HTTP dans un thread, elle montre en fait comment utiliser CoreHTTP dans un environnement à thread unique (une seule tâche utilise l'API HTTP dans la démo). Bien que les applications à thread unique doivent appeler à plusieurs reprises la bibliothèque HTTP, les applications multithread peuvent à la place envoyer des requêtes HTTP en arrière-plan dans le cadre d'une tâche d'agent (ou de démon).

Organisation du code source

Le projet de démonstration porte un nom `http_demo_s3_download.c` et se trouve dans le *`freertos`*`/demos/coreHTTP/` répertoire et sur le GitHubsite Web.

Configuration de la connexion au serveur HTTP Amazon S3

Cette démonstration utilise une URL pré-signée pour se connecter au serveur HTTP Amazon S3 et autoriser l'accès à l'objet à télécharger. La connexion TLS du serveur HTTP Amazon S3 utilise uniquement l'authentification du serveur. Au niveau de l'application, l'accès à l'objet est authentifié à l'aide de paramètres dans la requête URL pré-signée. Suivez les étapes ci-dessous pour configurer votre connexion à AWS.

1. Créez un AWS compte :
    a. Si ce n'est pas déjà fait, créez et activez un AWS compte.
    b. Les comptes et les autorisations sont définis à l'aide de Gestion des identités et des accès AWS (IAM). IAM vous permet de gérer les autorisations pour chaque utilisateur de votre compte. Par défaut, un utilisateur ne dispose d'aucune autorisation tant que le propriétaire root ne l'a pas accordée.
        i. Pour ajouter un utilisateur à votre AWS compte, consultez le guide de l'utilisateur IAM.
        ii. Autorisez votre AWS compte à accéder à FreeRTOS AWS IoT et en ajoutant les politiques suivantes :
            • AmazonS3FullAccess
2. Créez un compartiment dans S3 en suivant les étapes décrites dans Comment créer un compartiment S3 ? dans le guide de l'utilisateur de la console Amazon Simple Storage Service.
3. Chargez un fichier sur S3 en suivant les étapes décrites dans Comment télécharger des fichiers et des dossiers dans un compartiment S3 ? .
4. Générez une URL pré-signée à l'aide du script situé à `FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/presigned_urls_gen.py` l'adresse. Pour les instructions d'utilisation, voir`FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/README.md`.

Fonctionnalité

La démo récupère d'abord la taille du fichier. Ensuite, il demande chaque plage d'octets de manière séquentielle, en boucle, avec des tailles de plage de`democonfigRANGE_REQUEST_LENGTH`.

Le code source de la démo se trouve sur le GitHubsite Web.

Connexion au serveur HTTP Amazon S3

La fonction connect ToServerWithBackoffRetries () tente d'établir une connexion TCP avec le serveur HTTP. Si la connexion échoue, elle réessaie après un délai d'expiration. La valeur du délai d'attente augmentera de façon exponentielle jusqu'à ce que le nombre maximum de tentatives soit atteint ou que la valeur maximale du délai d'expiration soit atteinte. `connectToServerWithBackoffRetries()`renvoie un état d'échec si la connexion TCP au serveur ne peut pas être établie après le nombre de tentatives configuré.

La fonction `prvConnectToServer()` montre comment établir une connexion au serveur HTTP Amazon S3 en utilisant uniquement l'authentification du serveur. Il utilise l'interface de TLS-based transport mbed implémentée dans le fichier FreeRTOS-Plus/Source/Application-Protocols/network_transport/freertos_plus_tcp/using_mbedtls/using_mbedtls.c.

Le code source de `prvConnectToServer()` se trouve sur GitHub.

Création d'une demande de plage

La fonction API `HTTPClient_AddRangeHeader()` prend en charge la sérialisation d'une plage d'octets dans les en-têtes de requête HTTP pour former une demande de plage. Les demandes de plage sont utilisées dans cette démo pour récupérer la taille du fichier et pour demander chaque section du fichier.

La fonction `prvGetS3ObjectFileSize()` récupère la taille du fichier dans le compartiment S3. L'`Connection: keep-alive`en-tête est ajouté dans cette première demande à Amazon S3 afin de maintenir la connexion ouverte après l'envoi de la réponse. Le serveur HTTP S3 ne prend actuellement pas en charge les requêtes HEAD utilisant une URL pré-signée, c'est pourquoi le 0ème octet est demandé. La taille du fichier est indiquée dans le champ d'`Content-Range`en-tête de la réponse. Une `206 Partial Content` réponse est attendue du serveur ; tout autre code d'état de réponse reçu est une erreur.

Le code source de `prvGetS3ObjectFileSize()` se trouve sur GitHub.

Après avoir récupéré la taille du fichier, cette démonstration crée une nouvelle demande de plage pour chaque plage d'octets du fichier à télécharger. Il utilise `HTTPClient_AddRangeHeader()` pour chaque section du fichier.

Envoi de demandes de portée et réception de réponses

La fonction `prvDownloadS3ObjectFile()` envoie les demandes de plage en boucle jusqu'à ce que le fichier entier soit téléchargé. La fonction API `HTTPClient_Send()` envoie une demande et

reçoit la réponse de manière synchrone. Lorsque la fonction revient, la réponse est reçue dans un fichier`xResponse`. Le code d'état est ensuite vérifié `206 Partial Content` et le nombre d'octets téléchargés jusqu'à présent est incrémenté par la valeur de l'`Content-Length`en-tête.

Le code source de `prvDownloadS3ObjectFile()` se trouve sur GitHub.

Démo multithread de base de CoreHTTP

> **Important**
>
> Cette démo est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Introduction

Cette démo utilise les files d'attente sécurisées de FreeRTOS pour contenir les demandes et les réponses en attente de traitement. Dans cette démo, il y a trois tâches à prendre en compte.

- La tâche principale attend que les demandes apparaissent dans la file d'attente des demandes. Il enverra ces demandes sur le réseau, puis placera la réponse dans la file d'attente des réponses.
- Une tâche de demande crée des objets de requête de bibliothèque HTTP à envoyer au serveur et les place dans la file d'attente des demandes. Chaque objet de demande spécifie une plage d'octets du fichier S3 que l'application a configuré pour le téléchargement.
- Une tâche de réponse attend que les réponses apparaissent dans la file d'attente des réponses. Il enregistre toutes les réponses qu'il reçoit.

Cette démo multithread de base est configurée pour utiliser une connexion TLS uniquement avec authentification du serveur, requise par le serveur HTTP Amazon S3. L'authentification de la couche application est effectuée à l'aide des paramètres Signature Version 4 dans la requête d'URL présignée.

Organisation du code source

Le projet de démonstration porte un nom `http_demo_s3_download_multithreaded.c` et se trouve dans le *`freertos`*`/demos/coreHTTP/` répertoire et GitHubsur le site Web.

Création du projet de démonstration

Le projet de démonstration utilise l'édition communautaire gratuite de Visual Studio. Pour créer la démo, procédez comme suit :

1. Ouvrez le fichier de solution `mqtt_multitask_demo.sln` Visual Studio depuis l'IDE Visual Studio.
2. Sélectionnez Build Solution dans le menu Build de l'IDE.

> **Note**
>
> Si vous utilisez Microsoft Visual Studio 2017 ou une version antérieure, vous devez sélectionner un jeu d'outils de plate-forme compatible avec votre version : Project -> RTOSDemos Properties -> Platform Toolset.

Configuration du projet de démonstration

La démo utilise la _TCP/IP pile FreeRTOS+TCP. Suivez donc les instructions fournies au projet de démarrage pour : TCP/IP

1. Installez les composants requis (tels que WinPcap).
2. Définissez éventuellement une adresse IP statique ou dynamique, une adresse de passerelle et un masque réseau.
3. Définissez éventuellement une adresse MAC.
4. Sélectionnez une interface réseau Ethernet sur votre machine hôte.
5. Il est important de tester votre connexion réseau avant d'essayer d'exécuter la démo HTTP.

Configuration de la connexion au serveur HTTP Amazon S3

Suivez les instructions de la démo Configuration de la connexion au serveur HTTP Amazon S3 de téléchargement de base de CoreHTTP.

Fonctionnalité

La démo crée trois tâches au total :

- Celui qui envoie des demandes et reçoit des réponses sur le réseau.

- Celui qui crée des demandes à envoyer.
- Celui qui traite les réponses reçues.

Dans cette démonstration, la tâche principale est la suivante :

1. Crée les files d'attente de demandes et de réponses.
2. Crée la connexion au serveur.
3. Crée les tâches de demande et de réponse.
4. Attend que la file d'attente envoie des demandes sur le réseau.
5. Place les réponses reçues sur le réseau dans la file d'attente des réponses.

La tâche de demande :

1. Crée chacune des demandes de plage.

La tâche de réponse :

1. Traite chacune des réponses reçues.

Typedefs

La démo définit les structures suivantes pour prendre en charge le multithreading.

Demander des articles

Les structures suivantes définissent un élément de demande à placer dans la file d'attente des demandes. L'élément de demande est copié dans la file d'attente une fois que la tâche de demande crée une requête HTTP.

```
/**
 * @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;
```

Élément de réponse

Les structures suivantes définissent un élément de réponse à placer dans la file d'attente des réponses. L'élément de réponse est copié dans la file d'attente une fois que la tâche HTTP principale reçoit une réponse sur le réseau.

```
/**
 * @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;
```

Tâche d'envoi HTTP principale

Tâche principale de l'application :

1. Analyse l'URL présignée de l'adresse de l'hôte afin d'établir une connexion avec le serveur HTTP Amazon S3.
2. Analyse l'URL présignée pour le chemin d'accès aux objets du compartiment S3.
3. Se connecte au serveur HTTP Amazon S3 à l'aide du protocole TLS avec authentification du serveur.
4. Crée les files d'attente de demandes et de réponses.
5. Crée les tâches de demande et de réponse.

La fonction `prvHTTPDemoTask()` effectue cette configuration et donne le statut de démonstration. Le code source de cette fonction se trouve sur Github.

Dans la fonction`prvDownloadLoop()`, la tâche principale bloque et attend les demandes provenant de la file d'attente de demandes. Lorsqu'il reçoit une demande, il l'envoie à l'aide de la fonction API`HTTPClient_Send()`. Si la fonction API est réussie, elle place la réponse dans la file d'attente des réponses.

Le code source de se `prvDownloadLoop()` trouve sur Github.

Tâche de requête HTTP

La tâche de demande est spécifiée dans la fonction`prvRequestTask`. Le code source de cette fonction se trouve sur Github.

La tâche de demande récupère la taille du fichier dans le compartiment Amazon S3. Cela se fait dans la fonction`prvGetS3ObjectFileSize`. L'en-tête « Connection : keep-alive » est ajouté à cette demande adressée à Amazon S3 afin de maintenir la connexion ouverte après l'envoi de la réponse. Le serveur HTTP Amazon S3 ne prend actuellement pas en charge les requêtes HEAD utilisant une URL présignée. Le 0ème octet est donc demandé. La taille du fichier est indiquée dans le champ d'`Content-Range`en-tête de la réponse. Une `206 Partial Content` réponse est attendue du serveur ; tout autre code d'état de réponse reçu est une erreur.

Le code source de se `prvGetS3ObjectFileSize` trouve sur Github.

Après avoir récupéré la taille du fichier, la tâche de demande continue de demander chaque plage du fichier. Chaque demande de plage est placée dans la file d'attente de la tâche principale à envoyer. Les plages de fichiers sont configurées par l'utilisateur de démonstration dans la macro`democonfigRANGE_REQUEST_LENGTH`. Les demandes de plage sont prises en charge de manière native dans l'API de la bibliothèque cliente HTTP à l'aide de cette fonction`HTTPClient_AddRangeHeader`. La fonction `prvRequestS3ObjectRange` montre comment utiliser`HTTPClient_AddRangeHeader()`.

Le code source de la fonction se `prvRequestS3ObjectRange` trouve sur Github.

Tâche de réponse HTTP

Les tâches de réponse attendent dans la file d'attente les réponses reçues sur le réseau. La tâche principale remplit la file de réponses lorsqu'elle reçoit avec succès une réponse HTTP. Cette tâche traite les réponses en enregistrant le code d'état, les en-têtes et le corps. Une application du monde réel peut traiter la réponse en écrivant le corps de la réponse dans une mémoire flash, par exemple. Si le code d'état de la réponse ne l'est pas`206 partial content`, la tâche indique à la tâche principale que la démonstration doit échouer. La tâche de réponse est spécifiée dans la fonction`prvResponseTask`. Le code source de cette fonction se trouve sur Github.

AWS IoT Démo de la bibliothèque d'emplois

> **Important**
>
> Cette démo est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Introduction

La démo de la bibliothèque de AWS IoT tâches vous montre comment vous connecter au service AWS IoT Jobs via une connexion MQTT, récupérer une tâche et la traiter sur un appareil. AWS IoT Le projet de démonstration AWS IoT Jobs utilise le port FreeRTOS pour Windows, de sorte qu'il peut être créé et évalué avec la version Visual Studio Community sous Windows. Aucun microcontrôleur n'est nécessaire. La démo établit une connexion sécurisée avec le broker AWS IoT MQTT à l'aide du protocole TLS de la même manière que le. Démo de l'authentification mutuelle CoreMQTT

> **Note**
>
> Pour configurer et exécuter les démos de FreeRTOS, suivez les étapes décrites dans. Commencez avec FreeRTOS

Organisation du code source

Le code de démonstration se trouve dans le `jobs_demo.c` fichier et se trouve sur le GitHubsite Web ou dans le *`freertos`*`/demos/jobs_for_aws/` répertoire.

Configurez le AWS IoT Connexion au broker MQTT

Dans cette démonstration, vous utilisez une connexion MQTT au broker AWS IoT MQTT. Cette connexion est configurée de la même manière que leDémo de l'authentification mutuelle CoreMQTT.

Fonctionnalité

La démo montre le flux de travail utilisé pour recevoir des tâches AWS IoT et les traiter sur un appareil. La démo est interactive et vous oblige à créer des tâches à l'aide de la AWS IoT console ou

du AWS Command Line Interface (AWS CLI). Pour plus d'informations sur la création d'une tâche, voir create-job dans le manuel AWS CLI Command Reference. La démo nécessite que le document de travail comporte une `action` clé configurée `print` pour imprimer un message sur la console.

Consultez le format suivant pour ce document de travail.

```
{
    "action": "print",
    "message": "ADD_MESSAGE_HERE"
}
```

Vous pouvez utiliser le AWS CLI pour créer une tâche comme dans l'exemple de commande suivant.

```
aws iot create-job \
    --job-id t12 \
    --targets arn:aws:iot:region:123456789012:thing/device1 \
    --document '{"action":"print","message":"hello world!"}'
```

La démo utilise également un document de travail dont la `action` clé est définie `publish` pour republier le message dans un sujet. Consultez le format suivant pour ce document de travail.

```
{
    "action": "publish",
    "message": "ADD_MESSAGE_HERE",
    "topic": "topic/name/here"
}
```

La démo s'exécute en boucle jusqu'à ce qu'elle reçoive un document de travail dont la `action` touche est réglée `exit` pour quitter la démo. Le format du document de travail est le suivant.

```
{
    "action: "exit"
}
```

Point d'entrée de la démo Jobs

Le code source de la fonction de point d'entrée de démonstration de Jobs se trouve sur GitHub. Cette fonction effectue les opérations suivantes :

1. Établissez une connexion MQTT à l'aide des fonctions d'assistance dans. `mqtt_demo_helpers.c`
2. Abonnez-vous à la rubrique MQTT relative à l'`NextJobExecutionChanged`API, en utilisant les fonctions d'assistance dans. `mqtt_demo_helpers.c` La chaîne de rubrique a été assemblée plus tôt, à l'aide de macros définies par la bibliothèque AWS IoT Jobs.
3. Publiez dans la rubrique MQTT de l'`StartNextPendingJobExecution`API, à l'aide des fonctions d'assistance dans. `mqtt_demo_helpers.c` La chaîne de rubrique a été assemblée plus tôt, à l'aide de macros définies par la bibliothèque AWS IoT Jobs.
4. Appelez `MQTT_ProcessLoop` à plusieurs reprises pour recevoir les messages entrants qui sont remis `prvEventCallback` pour traitement.
5. Une fois que la démo a reçu l'action de sortie, désabonnez-vous du sujet MQTT et déconnectez-vous à l'aide des fonctions d'assistance du `mqtt_demo_helpers.c` fichier.

Rappel pour les messages MQTT reçus

La EventCallback fonction prv appelle `Jobs_MatchTopic` depuis la bibliothèque AWS IoT Jobs pour classer le message MQTT entrant. Si le type de message correspond à une nouvelle tâche, `prvNextJobHandler()` il est appelé.

La NextJobHandler fonction prv et les fonctions qu'elle appelle analysent le document de tâche à partir du JSON-formatted message et exécutent l'action spécifiée par la tâche. La `prvSendUpdateForJob` fonction est particulièrement intéressante.

Envoyer une mise à jour pour une tâche en cours

La fonction prv SendUpdateForJob () appelle `Jobs_Update()` depuis la bibliothèque Jobs pour renseigner la chaîne de rubrique utilisée dans l'opération de publication MQTT qui suit immédiatement.

Démos de CoreMQTT

> **Important**
>
> Cette démo est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Ces démos peuvent vous aider à apprendre à utiliser la bibliothèque CoreMQTT.

Rubriques

- Démo de l'authentification mutuelle CoreMQTT
- Démonstration du partage de connexion avec l'agent CoreMQTT

Démo de l'authentification mutuelle CoreMQTT

> **Important**
>
> Cette démo est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Introduction

Le projet de démonstration d'authentification mutuelle CoreMQTT vous montre comment établir une connexion avec un courtier MQTT à l'aide du protocole TLS avec authentification mutuelle entre le client et le serveur. Cette démonstration utilise une implémentation d'interface de TLS-based transport mbed pour établir une connexion TLS authentifiée par le serveur et le client, et illustre le flux de travail d'abonnement/publication de MQTT au niveau QoS 1. Il s'abonne à un filtre de rubrique, puis publie dans les rubriques correspondant au filtre et attend la réception de ces messages du serveur au niveau QoS 1. Ce cycle de publication auprès du courtier et de réception du même message de la part du courtier se répète indéfiniment. Les messages de cette démo sont envoyés à QoS 1, ce qui garantit au moins une livraison conformément à la spécification MQTT.

> **Note**
>
> Pour configurer et exécuter les démos de FreeRTOS, suivez les étapes décrites dans. Commencez avec FreeRTOS

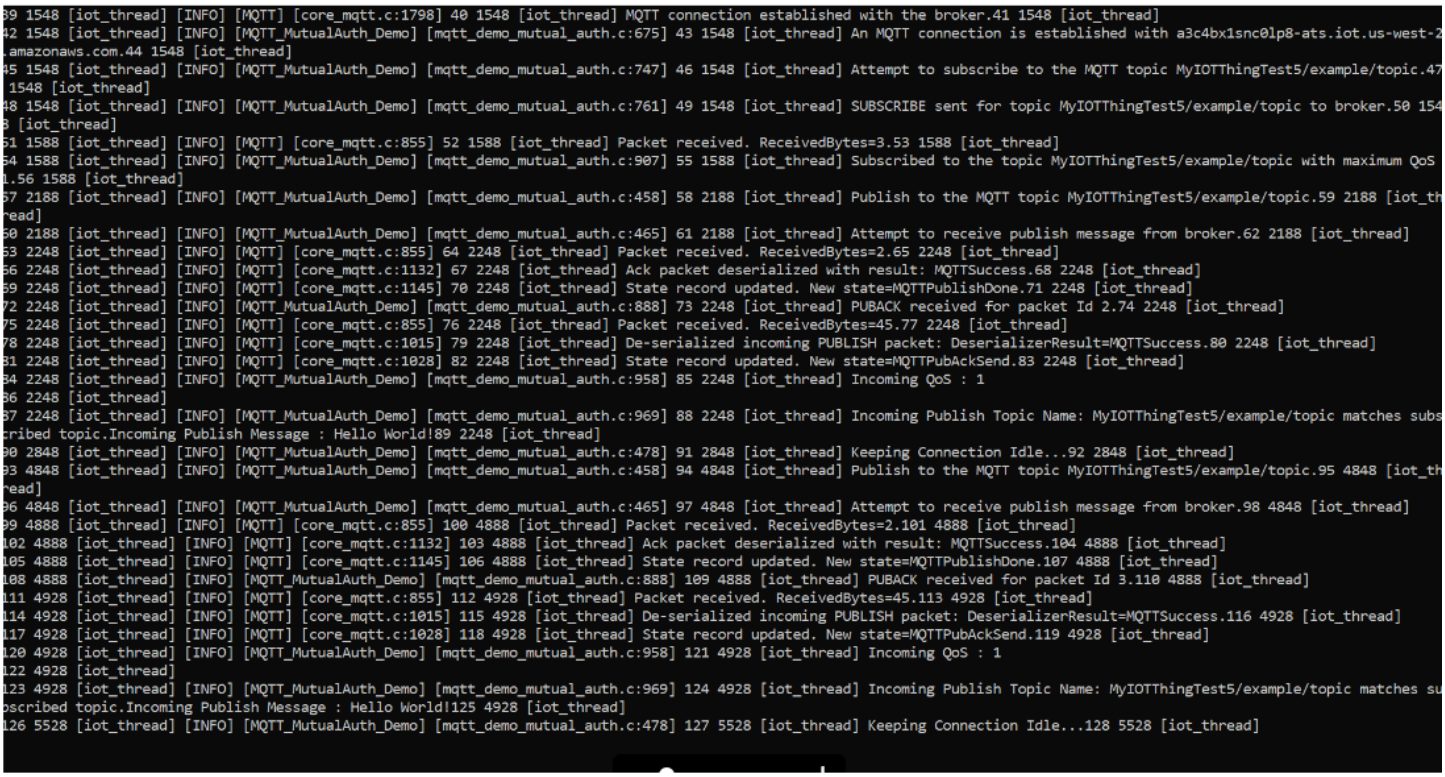
Code source

Le fichier source de démonstration est nommé `mqtt_demo_mutual_auth.c` et se trouve dans le *`freertos`*`/demos/coreMQTT/` répertoire et GitHubsur le site Web.

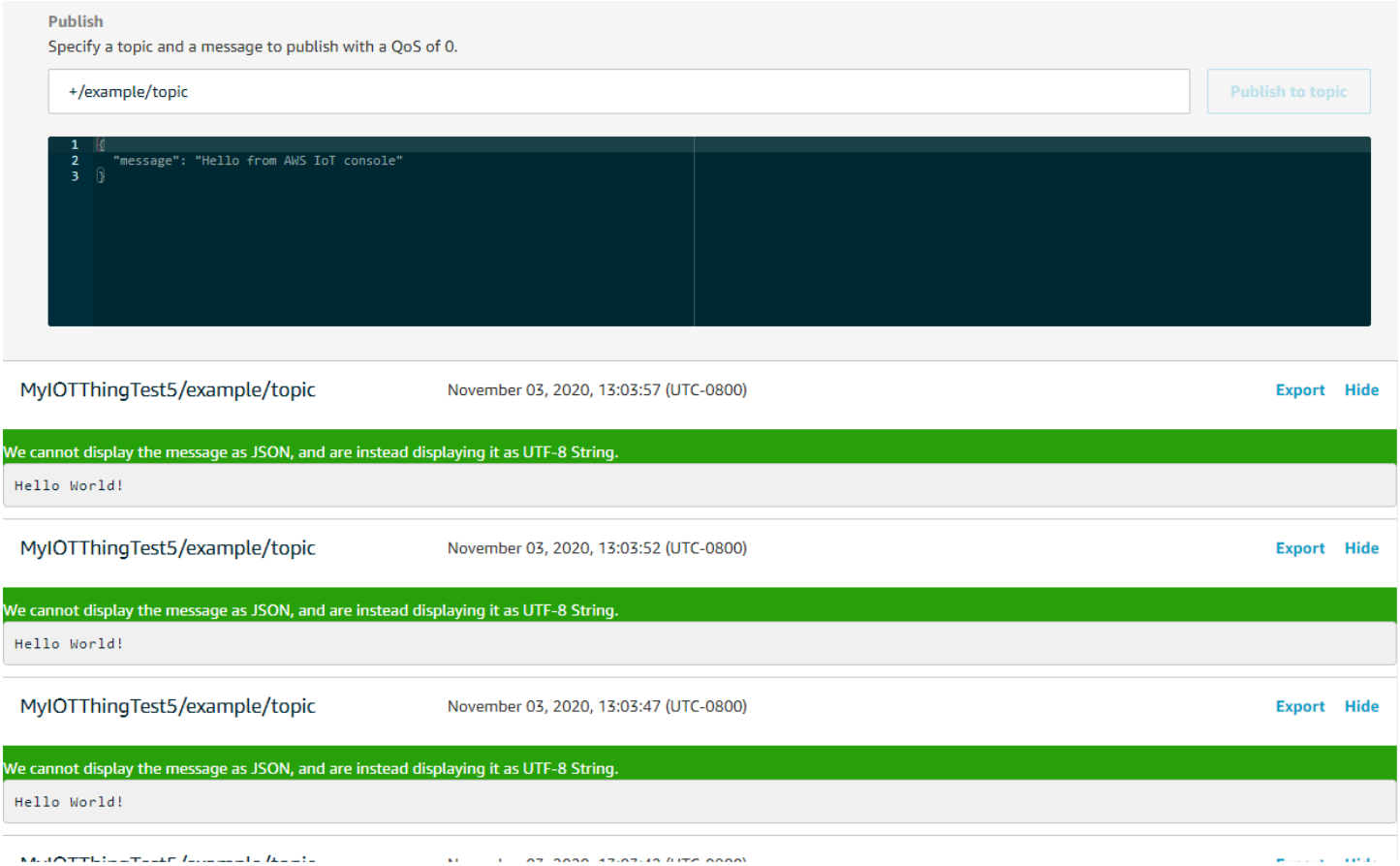
Fonctionnalité

La démo crée une tâche d'application unique qui passe en revue un ensemble d'exemples illustrant comment se connecter au courtier, s'abonner à une rubrique sur le courtier, publier dans une rubrique sur le courtier, puis enfin, se déconnecter du courtier. L'application de démonstration s'abonne et publie sur le même sujet. Chaque fois que la démo publie un message au broker MQTT, celui-ci renvoie le même message à l'application de démonstration.

Une démonstration réussie générera un résultat similaire à l'image suivante.



La AWS IoT console générera une sortie similaire à l'image suivante.



Réessayez la logique avec un recul et une instabilité exponentiels

La BackoffForRetry fonction prv montre comment les opérations réseau ayant échoué avec le serveur, par exemple les connexions TLS ou les demandes d'abonnement MQTT, peuvent être réessayées avec un retard et une instabilité exponentiels. La fonction calcule la période d'attente pour la prochaine tentative et effectue le délai d'annulation si les tentatives n'ont pas été épuisées. Comme le calcul de la période d'attente nécessite la génération d'un nombre aléatoire, la fonction utilise le module PKCS11 pour générer le nombre aléatoire. L'utilisation du module PKCS11 permet d'accéder à un véritable générateur de nombres aléatoires (TRNG) si la plateforme du fournisseur le prend en charge. Nous vous recommandons d'amorcer le générateur de nombres aléatoires avec une source d'entropie spécifique à l'appareil afin de réduire la probabilité de collisions avec des appareils lors de nouvelles tentatives de connexion.

Connexion au courtier MQTT

La ConnectToServerWithBackoffRetries fonction prv tente d'établir une connexion TLS mutuellement authentifiée avec le broker MQTT. Si la connexion échoue, elle réessaie après une période d'attente. La période d'attente augmentera de façon exponentielle jusqu'à ce que le nombre

maximum de tentatives soit atteint ou que la période d'interruption maximale soit atteinte. La `BackoffAlgorithm_GetNextBackoff` fonction fournit une valeur de retard qui augmente de façon exponentielle et revient `RetryUtilsRetriesExhausted` lorsque le nombre maximum de tentatives a été atteint. La `prvConnectToServerWithBackoffRetries` fonction renvoie un état d'échec si la connexion TLS au broker ne peut pas être établie après le nombre de tentatives configuré.

La CreateMQTTConnectionWithBroker fonction prv montre comment établir une connexion MQTT avec un broker MQTT avec une session propre. Il utilise l'interface de transport TLS, qui est implémentée dans le `FreeRTOS-Plus/Source/Application-Protocols/platform/freertos/transport/src/tls_freertos.c` fichier. N'oubliez pas que nous définissons les secondes pendant lesquelles le courtier doit rester actif. `xConnectInfo`

La fonction suivante montre comment l'interface de transport TLS et la fonction de temps sont définies dans un contexte MQTT à l'aide de la `MQTT_Init` fonction. Il montre également comment une fonction de rappel d'événement pointer (`prvEventCallback`) est définie. Ce rappel est utilisé pour signaler les messages entrants.

Abonnement à un sujet MQTT

La MQTTSubscribeWithBackoffRetries fonction prv montre comment s'abonner à un filtre de sujet sur le broker MQTT. L'exemple montre comment s'abonner à un filtre de sujet, mais il est possible de transmettre une liste de filtres de sujets dans le même appel d'API d'abonnement pour s'abonner à plusieurs filtres de sujets. De plus, si le courtier MQTT rejette la demande d'abonnement, l'abonnement réessaiera, avec un retard exponentiel, pour. `RETRY_MAX_ATTEMPTS`

Publication dans une rubrique

La MQTTPublishToTopic fonction prv montre comment publier sur un sujet sur le broker MQTT.

Réception de messages entrants

L'application enregistre une fonction de rappel d'événement avant de se connecter au broker, comme décrit précédemment. La `prvMQTTDemoTask` fonction appelle la `MQTT_ProcessLoop` fonction pour recevoir les messages entrants. Lorsqu'un message MQTT entrant est reçu, il appelle la fonction de rappel d'événements enregistrée par l'application. La EventCallback fonction prv est un exemple d'une telle fonction de rappel d'événements. `prvEventCallback`examine le type de paquet entrant et appelle le gestionnaire approprié. Dans l'exemple ci-dessous, la fonction appelle soit à gérer `prvMQTTProcessIncomingPublish()` les messages de publication entrants, soit `prvMQTTProcessResponse()` à gérer les accusés de réception (ACK).

Traitement des paquets de publication MQTT entrants

La MQTTProcessIncomingPublish fonction prv montre comment traiter un paquet de publication à partir du broker MQTT.

Désinscription d'un sujet

La dernière étape du flux de travail consiste à se désabonner du sujet afin que le courtier n'envoie aucun message publié depuis`mqttexampleTOPIC`. Voici la définition de la fonction prv MQTTUnsubscribeFromTopic.

Modification de l'autorité de certification racine utilisée dans la démo

Par défaut, les démos FreeRTOS utilisent le certificat Amazon Root CA 1 (clé RSA 2048 bits) pour s'authentifier auprès du serveur. AWS IoT Core Il est possible d'utiliser d'autres certificats CA pour l'authentification du serveur, notamment le certificat Amazon Root CA 3 (clé ECC 256 bits). Pour modifier l'autorité de certification racine pour la démonstration d'authentification mutuelle CoreMQTT :

1. Dans un éditeur de texte, ouvrez le fichier *freertos*`/vendors/`*vendor*`/boards/`*board*`/aws_demos/config_files/mqtt_demo_mutual_auth_config.h`.
2. Dans le fichier, repérez la ligne suivante.

   ```
   * #define democonfigROOT_CA_PEM    "...insert here..."
   ```

   Décommentez cette ligne et, si nécessaire, déplacez-la au-delà de la fin `*/` du bloc de commentaires.
3. Copiez le certificat CA que vous souhaitez utiliser, puis collez-le dans le `"...insert here..."` texte. Le résultat doit ressembler à l'exemple suivant :

   ```
   #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. (Facultatif) Vous pouvez modifier l'autorité de certification racine pour les autres démos. Répétez les étapes 1 à 3 pour chaque *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/*demo-name*_config.h fichier.

Démonstration du partage de connexion avec l'agent CoreMQTT

> **Important**
>
> Cette démo est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Introduction

Le projet de démonstration du partage de connexion CoreMQTT vous montre comment utiliser une application multithread pour établir une connexion au broker AWS MQTT à l'aide du protocole TLS avec authentification mutuelle entre le client et le serveur. Cette démonstration utilise une implémentation d'interface de TLS-based transport mbed pour établir une connexion TLS authentifiée par le serveur et le client, et illustre le flux de travail d'abonnement/publication de MQTT au niveau QoS 1. La démo s'abonne à un filtre de rubrique, publie les rubriques correspondant au filtre, puis attend de recevoir ces messages du serveur au niveau QoS 1. Ce cycle de publication auprès du courtier et de réception du même message de la part du courtier est répété plusieurs fois pour chaque tâche créée. Les messages de cette démo sont envoyés à QoS 1, ce qui garantit au moins une livraison conformément à la spécification MQTT.

> **Note**
>
> Pour configurer et exécuter les démos de FreeRTOS, suivez les étapes décrites dans. Commencez avec FreeRTOS

Cette démonstration utilise une file d'attente sécurisée par thread pour contenir les commandes permettant d'interagir avec l'API MQTT. Il y a deux tâches à prendre en compte dans cette démonstration.

- Une tâche (principale) de l'agent MQTT traite les commandes de la file de commandes tandis que les autres tâches les mettent en file d'attente. Cette tâche entre dans une boucle au cours de laquelle elle traite les commandes de la file de commandes. Si une commande de fin est reçue, cette tâche sera interrompue.
- Une tâche de sous-publication de démonstration crée un abonnement à un sujet MQTT, puis crée des opérations de publication et les place dans la file de commandes. Ces opérations de publication sont ensuite exécutées par la tâche de l'agent MQTT. La tâche de sous-publication de démonstration attend la fin de la publication, ce qui est indiqué par l'exécution du rappel de fin de commande, puis entre dans un court délai avant de commencer la publication suivante. Cette tâche montre des exemples de la manière dont les tâches d'application utiliseraient l'API de l'agent CoreMQTT.

Pour les messages de publication entrants, l'agent CoreMQTT invoque une seule fonction de rappel. Cette démo inclut également un gestionnaire d'abonnements qui permet aux tâches de spécifier un rappel à invoquer pour les messages de publication entrants sur les sujets auxquels elles sont abonnées. Dans le cadre de cette démonstration, le rappel de publication entrant de l'agent appelle le gestionnaire d'abonnements pour répartir les publications sur toute tâche ayant enregistré un abonnement.

Cette démonstration utilise une connexion TLS avec authentification mutuelle pour se connecter AWS. Si le réseau se déconnecte de façon inattendue pendant la démonstration, le client tente de se reconnecter en utilisant une logique d'interruption exponentielle. Si le client se reconnecte avec succès, mais que le courtier ne peut pas reprendre la session précédente, le client se réabonnera aux mêmes sujets que lors de la session précédente.

Monothread ou multithread

Il existe deux modèles d'utilisation de CoreMQTT, monothread et multithread (multitâche). Le modèle à thread unique utilise la bibliothèque CoreMQTT uniquement à partir d'un thread et vous oblige à effectuer des appels explicites répétés dans la bibliothèque MQTT. Les cas d'utilisation multithread peuvent à la place exécuter le protocole MQTT en arrière-plan dans une tâche d'agent (ou de démon), comme le montre la démonstration documentée ici. Lorsque vous exécutez le protocole MQTT dans une tâche d'agent, vous n'avez pas besoin de gérer explicitement un état MQTT ni d'appeler la fonction `MQTT_ProcessLoop` API. En outre, lorsque vous utilisez une tâche d'agent, plusieurs tâches d'application peuvent partager une seule connexion MQTT sans avoir besoin de primitives de synchronisation telles que des mutex.

Code source

Les fichiers source de démonstration sont nommés `mqtt_agent_task.c` `simple_sub_pub_demo.c` et se trouvent dans le *`freertos`*`/demos/coreMQTT_Agent/` répertoire et sur le GitHubsite Web.

Fonctionnalité

Cette démonstration crée au moins deux tâches : une tâche principale qui traite les demandes d'appels d'API MQTT, et un nombre configurable de sous-tâches qui créent ces demandes. Dans cette démonstration, la tâche principale crée les sous-tâches, appelle la boucle de traitement et effectue ensuite le nettoyage. La tâche principale crée une connexion MQTT unique avec le broker, qui est partagée entre les sous-tâches. Les sous-tâches créent un abonnement MQTT auprès du courtier, puis y publient des messages. Chaque sous-tâche utilise un sujet unique pour ses publications.

Tâche principale

La tâche principale de l'application établit une session MQTT, crée les sous-tâches et exécute la boucle de traitement MQTT Agent_CommandLoop jusqu'à ce qu'une commande de fin soit reçue. RunCoreMQTTAgentDemo Si le réseau se déconnecte de façon inattendue, la démo se reconnecte au courtier en arrière-plan et rétablit les abonnements auprès du courtier. Une fois la boucle de traitement terminée, elle se déconnecte du broker.

Commandes

Lorsque vous appelez une API d'agent CoreMQTT, elle crée une commande qui est envoyée à la file d'attente de la tâche de l'agent, qui est traitée dans. `MQTTAgent_CommandLoop()` Au moment de la création de la commande, le rappel de fin facultatif et les paramètres de contexte peuvent être transmis. Une fois la commande correspondante terminée, le rappel d'achèvement sera invoqué avec le contexte transmis et toutes les valeurs de retour créées à la suite de la commande. La signature du rappel de fin est la suivante :

```
typedef void (* MQTTAgentCommandCallback_t )( void * pCmdCallbackContext,
                                              MQTTAgentReturnInfo_t * pReturnInfo );
```

Le contexte d'exécution des commandes est défini par l'utilisateur ; pour cette démo, il s'agit de : MQTTAgentCommandContextstruct.

Les commandes sont considérées comme terminées lorsque :

- S'abonne, se désabonne et publie avec QoS > 0 : une fois que le paquet d'accusé de réception correspondant a été reçu.
- Toutes les autres opérations : une fois que l'API CoreMQTT correspondante a été invoquée.

Toutes les structures utilisées par la commande, y compris les informations de publication, les informations d'abonnement et les contextes d'achèvement, doivent rester dans le champ d'application jusqu'à ce que la commande soit terminée. Une tâche appelante ne doit réutiliser aucune des structures d'une commande avant l'invocation du rappel de fin. Notez que puisque le rappel d'achèvement est invoqué par l'agent MQTT, il s'exécutera avec le contexte de thread de la tâche de l'agent, et non avec la tâche qui a créé la commande. Inter-process des mécanismes de communication, tels que des notifications de tâches ou des files d'attente, peuvent être utilisés pour signaler à la tâche appelante la fin de la commande.

Exécution de la boucle de commande

Les commandes sont traitées en continu dans`MQTTAgent_CommandLoop()`. S'il n'y a aucune commande à traiter, la boucle attendra au maximum qu'une commande soit ajoutée à la file d'attente et, si aucune commande n'est ajoutée, elle exécutera une seule itération de`MQTT_ProcessLoop()`. `MQTT_AGENT_MAX_EVENT_QUEUE_WAIT_TIME` Cela garantit à la fois que MQTT Keep-Alive est géré et que toutes les publications entrantes sont reçues même s'il n'y a aucune commande dans la file d'attente.

La fonction de boucle de commande sera renvoyée pour les raisons suivantes :

- Une commande renvoie tout autre code d'état`MQTTSuccess`. L'état de l'erreur est renvoyé par la boucle de commande, vous pouvez donc décider comment le gérer. Dans cette démonstration, la connexion TCP est rétablie et une tentative de reconnexion est effectuée. En cas d'erreur, une reconnexion peut se produire en arrière-plan sans aucune intervention de la part d'autres tâches utilisant MQTT.
- Une commande de déconnexion (de`MQTTAgent_Disconnect`) est traitée. La boucle de commande se termine afin que le protocole TCP puisse être déconnecté.
- Une commande de fin (depuis`MQTTAgent_Terminate`) est traitée. Cette commande marque également comme une erreur toute commande encore dans la file d'attente ou en attente d'un paquet d'accusé de réception, avec un code de retour de. `MQTTRecvFailed`

Gestionnaire d'abonnements

Comme la démo utilise plusieurs sujets, un gestionnaire d'abonnements est un moyen pratique d'associer les sujets souscrits à des rappels ou à des tâches uniques. Le gestionnaire d'abonnement de cette démo est mono-thread, il ne doit donc pas être utilisé par plusieurs tâches simultanément. Dans cette démo, les fonctions du gestionnaire d'abonnements ne sont appelées qu'à partir des fonctions de rappel transmises à l'agent MQTT et ne sont exécutées qu'avec le contexte du thread de la tâche de l'agent.

Tâche simple d'abonnement/de publication

Chaque instance du prv SimpleSubscribePublishTask crée un abonnement à un sujet MQTT et crée des opérations de publication pour ce sujet. Pour illustrer plusieurs types de publication, les tâches à numéro pair utilisent QoS 0 (qui sont terminées une fois le paquet de publication envoyé) et les tâches impaires utilisent QoS 1 (qui sont terminées à la réception d'un paquet PUBACK).

Over-the-air met à jour l'application de démonstration

FreeRTOS inclut une application de démonstration qui démontre les fonctionnalités de la bibliothèque over-the-air (OTA). L'application de démonstration OTA se trouve dans le *freertos*`/demos/ota/ota_demo_core_mqtt/ota_demo_core_mqtt.c` ou le *freertos*`/demos/ota/ota_demo_core_http/ota_demo_core_http.c` fichier.

L'application de démonstration OTA effectue les opérations suivantes :

1. Initialise la pile de réseau FreeRTOS et le pool de tampons MQTT.
2. Crée une tâche à utiliser par la bibliothèque OTA`vRunOTAUpdateDemo()`.
3. Crée un client MQTT avec `_establishMqttConnection()`.
4. Se connecte au broker AWS IoT MQTT en utilisant `IotMqtt_Connect()` et enregistre un rappel de déconnexion MQTT :. `prvNetworkDisconnectCallback`
5. Elle appelle `OTA_AgentInit()` pour créer la tâche OTA et enregistrer un rappel à utiliser une fois la tâche OTA terminée.
6. Réutilise la connexion MQTT avec `xOTAConnectionCtx.pvControlClient = _mqttConnection;`
7. Si MQTT se déconnecte, l'application suspend l'agent OTA, tente de se reconnecter en utilisant un délai exponentiel avec instabilité, puis reprend l'agent OTA.

Avant de pouvoir utiliser les mises à jour OTA, remplissez toutes les conditions requises dans Mises à jour de FreeRTOS Over-the-Air

Une fois la configuration des mises à jour OTA terminée, téléchargez, compilez, flashez et exécutez la démo de FreeRTOS OTA sur une plateforme compatible avec les fonctionnalités OTA. Device-specific des instructions de démonstration sont disponibles pour les FreeRTOS-qualified appareils suivants :

- Instruments texans CC3220SF-LAUNCHXL
- Microchip Curiosity PIC32MZEF
- Espressif ESP32
- Téléchargez, compilez, flashez et exécutez la démo FreeRTOS OTA sur le Renesas RX65N

Après avoir créé, flashé et exécuté l'application de démonstration OTA sur votre appareil, vous pouvez utiliser la AWS IoT console ou créer une tâche de mise à jour OTA. AWS CLI Une fois que vous avez créé une tâche de mise à jour OTA, connectez un émulateur de terminal pour afficher la progression de la mise à jour OTA. Notez les erreurs générées au cours du processus.

Une tâche de mise à jour OTA réussie affiche une sortie similaire à la suivante. Quelques lignes de cet exemple ont été supprimées de la liste par souci de concision.

```
    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 configurations de démonstration

Les configurations de démonstration OTA sont des options de configuration spécifiques aux démos fournies dans. `aws_iot_ota_update_demo.c` Ces configurations sont différentes des configurations de bibliothèque OTA fournies dans le fichier de configuration de bibliothèque OTA.

OTA_DEMO_KEEP_ALIVE_SECONDS

Pour le client MQTT, cette configuration correspond à l'intervalle de temps maximal qui peut s'écouler entre la fin de la transmission d'un paquet de contrôle et le début de l'envoi du suivant. En l'absence de paquet de contrôle, un PINGREQ est envoyé. Le courtier doit déconnecter un client qui n'envoie pas de message ou de paquet PINGREQ pendant une fois et demie pendant cet intervalle de maintien en vie. Cette configuration doit être ajustée en fonction des exigences de l'application.

OTA_DEMO_CONN_RETRY_BASE_INTERVAL_SECONDS

Intervalle de base, en secondes, avant une nouvelle tentative de connexion réseau. La démo OTA essaiera de se reconnecter après cet intervalle de temps de base. L'intervalle est doublé après chaque tentative infructueuse. Un délai aléatoire, jusqu'à un maximum de ce délai de base, est également ajouté à l'intervalle.

OTA_DEMO_CONN_RETRY_MAX_INTERVAL EN SECONDES

Intervalle maximal, en secondes, avant une nouvelle tentative de connexion réseau. Le délai de reconnexion est doublé à chaque tentative infructueuse, mais il ne peut atteindre que cette valeur maximale, plus une instabilité correspondant au même intervalle.

Téléchargez, compilez, flashez et exécutez la démo de FreeRTOS OTA sur Texas Instruments CC3220SF-LAUNCHXL

> **Important**
>
> Cette intégration de référence est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Pour télécharger FreeRTOS et le code de démonstration OTA

- Vous pouvez télécharger le code source sur le GitHub site à l'adresse https://github.com/FreeRTOS/FreeRTOS.

Pour créer l'application de démonstration

1. Suivez les instructions ci-dessous Commencez avec FreeRTOS pour importer le `aws_demos` projet dans Code Composer Studio, configurer votre AWS IoT point de terminaison, votre Wi-Fi SSID et votre mot de passe, ainsi qu'une clé privée et un certificat pour votre tableau.
2. Ouvrez*`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/aws_demo_config.h`, commentez `#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED` et définissez `CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` ou`CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`.
3. Générez la solution et assurez-vous qu'elle se génère sans erreurs.
4. Démarrez un émulateur de terminal et utilisez les paramètres suivants pour vous connecter à votre carte :
   - Vitesse de transmission : 115200
   - Bits de données : 8
   - Parité : aucune
   - Bits d'arrêt : 1

5. Exécutez le projet sur votre tableau pour vérifier qu'il peut se connecter au courtier de messages AWS IoT MQTT Wi-Fi et qu'il peut être connecté.

Lorsqu'il s'exécute, l'émulateur de terminal doit afficher un texte semblable au suivant :

```
    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
```

Téléchargez, compilez, flashez et exécutez la démo de FreeRTOS OTA sur le Microchip Curiosity PIC32MZEF

> **Important**
>
> Cette intégration de référence est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

> **Note**
>
> En accord avec Microchip, nous retirons le Curiosity PIC32MZEF (DM320104) de la branche principale du référentiel FreeRTOS Reference Integration et ne le proposerons plus dans les nouvelles versions. Microchip a publié un avis officiel indiquant que le PIC32MZEF (DM320104) n'est plus recommandé pour les nouveaux modèles. Les projets et le code source PIC32MZEF sont toujours accessibles via les balises de version précédentes. Microchip recommande aux clients d'utiliser la carte de PIC32MZ-EF-2.0 développement Curiosity (DM320209) pour les nouvelles conceptions. La plate-forme Pic32MZv1 se trouve toujours dans la version v202012.00 du référentiel FreeRTOS Reference Integration. Cependant, la plateforme n'est plus prise en charge par la version v202107.00 du FreeRTOS Reference.

Pour télécharger le code de démonstration FreeRTOS OTA

- Vous pouvez télécharger le code source sur le GitHub site à l'adresse https://github.com/FreeRTOS/FreeRTOS.

Pour créer l'application de démonstration de la mise à jour OTA

1. Suivez les instructions ci-dessous Commencez avec FreeRTOS pour importer le `aws_demos` projet dans l'IDE MPLAB X, configurer votre AWS IoT point de terminaison, votre Wi-Fi SSID et votre mot de passe, ainsi qu'une clé privée et un certificat pour votre carte.

2. Ouvrez le `vendors/vendor/boards/board/aws_demos/config_files/ota_demo_config.h` fichier et entrez votre certificat.

```
[] = "your-certificate-key";
```

3. Collez le contenu de votre certificat de signature de code ici :

```
#define otapalconfigCODE_SIGNING_CERTIFICATE [] = "your-certificate-key";
```

   Suivez le même format que `aws_clientcredential_keys.h` : chaque ligne doit se terminer par le nouveau caractère de ligne («\n») et être placée entre guillemets.

   Par exemple, votre certificat doit se présenter comme suit :

```
"-----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. Installez Python 3 ou version ultérieure.
5. Installez pyOpenSSL en exécutant `pip install pyopenssl`.
6. Copiez votre certificat de signature de code au format .pem dans le chemin d'accès `demos/ota/bootloader/utility/codesigner_cert_utility/`. Renommez le fichier de certificat `aws_ota_codesigner_certificate.pem`.
7. Ouvrez`freertos`/vendors/`vendor`/boards/`board`/aws_demos/config_files/aws_demo_config.h, commentez `#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED` et définissez `CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` ou`CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`.
8. Générez la solution et assurez-vous qu'elle se génère sans erreurs.
9. Démarrez un émulateur de terminal et utilisez les paramètres suivants pour vous connecter à votre carte :

- Vitesse de transmission : 115200
- Bits de données : 8
- Parité : aucune
- Bits d'arrêt : 1

10. Débranchez le débogueur de votre carte et lancez le projet sur votre carte pour vérifier qu'il peut se connecter au courtier de messages MQTT Wi-Fi et au logiciel de messagerie AWS IoT MQTT.

Lorsque vous exécutez le projet, l'IDE MPLAB X doit ouvrir une fenêtre de sortie. Assurez-vous que l'onglet ICD4 est sélectionné. Le résultat suivant doit s'afficher.

```
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
```

L'émulateur de terminal doit afficher un texte semblable au suivant :

```
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
```

Cette sortie montre que le Microchip Curiosity PIC32MZEF peut se connecter AWS IoT et s'abonner aux rubriques MQTT requises pour les mises à jour OTA. Les messages `Missing job parameters`sont attendus, car il n'y a pas de tâches de mise à jour OTA en attente.

Téléchargez, compilez, flashez et exécutez la démo de FreeRTOS OTA sur l'Espressif ESP32

> **Important**
>
> Cette intégration de référence est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

1. Téléchargez le code source FreeRTOS depuis. GitHub Consultez le README.mdfichier pour obtenir des instructions. Créez un projet dans votre environnement IDE qui inclut toutes les sources et bibliothèques requises.
2. Suivez les instructions de la section Getting Started with Espressif pour configurer la chaîne d'outils requise. GCC-based
3. Ouvrez*freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h, commentez `#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED` et définissez `CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` ou`CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`.
4. Générez le projet de démonstration en exécutant `make` dans le répertoire `vendors/espressif/boards/esp32/aws_demos`. Vous pouvez flasher le programme de démonstration et vérifier sa sortie en exécutant `make flash monitor`, comme décrit dans Mise en route avec Espressif.
5. Avant d'exécuter la démonstration de mise à jour OTA :

   - Ouvrez*freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h, commentez

`#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED`
et définissez `CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED`
ou`CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`.

- Ouvrez `vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/ota_demo_config.h` et copiez votre certificat de SHA-256/ECDSA signature de code dans :

```
#define otapalconfigCODE_SIGNING_CERTIFICATE [] = "your-certificate-key";
```

Téléchargez, compilez, flashez et exécutez la démo FreeRTOS OTA sur le Renesas RX65N

> **Important**
>
> Cette intégration de référence est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Ce chapitre explique comment télécharger, créer, flasher et exécuter les applications de démonstration FreeRTOS OTA sur le Renesas RX65N.

Rubriques

- Configurez votre environnement d'exploitation
- Configurez votre AWS resources
- Importez, configurez le fichier d'en-tête et compilez aws_demos et boot_loader

Configurez votre environnement d'exploitation

Les procédures décrites dans cette section utilisent les environnements suivants :

- IDE : e ^{2} studio 7.8.0, e ^{2} studio 2020-07
- Chaînes d'outils : compilateur CCRX v3.0.1
- Appareils cibles : RSKRX65N-2MB
- Débogueurs : émulateur E ^{2}, E $^{2\ \text{Lite}}$

- Logiciel : Renesas Flash Programmer, Renesas Secure Flash, Tera Term Programmer.exe

Pour configurer votre matériel

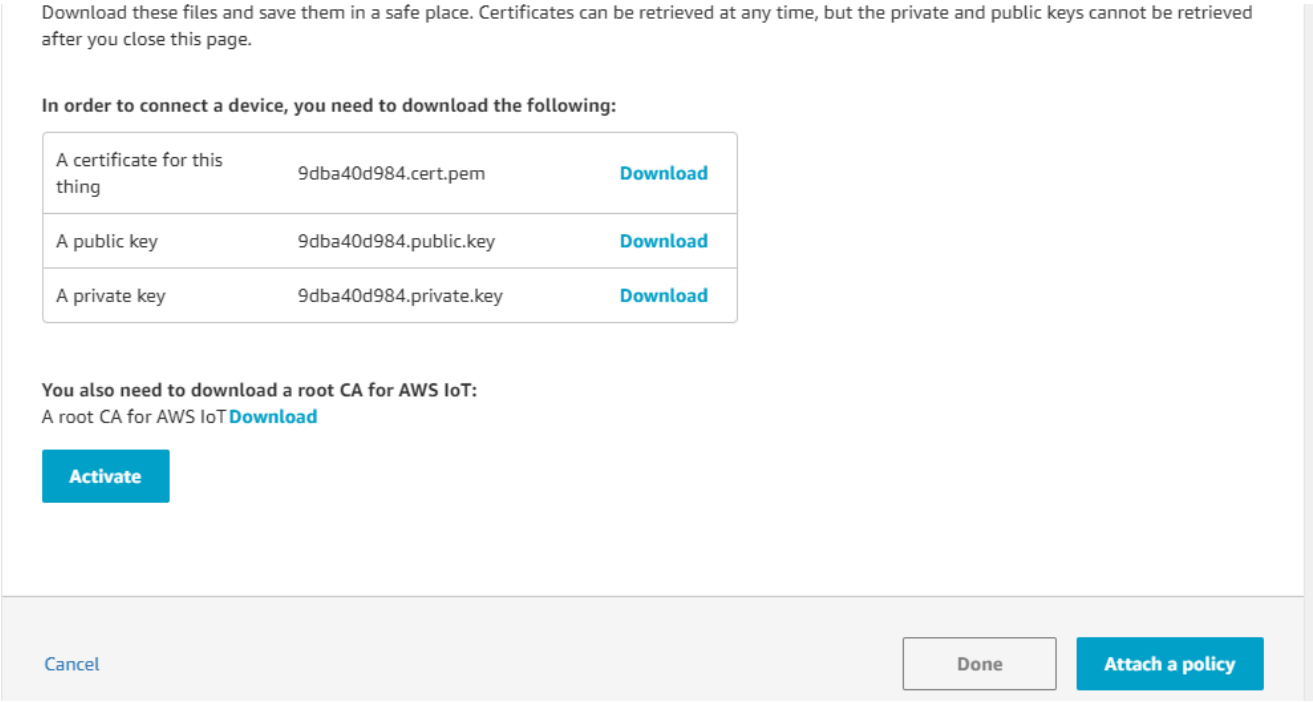
1. Connectez l'émulateur E 2 Lite et le port série USB à votre carte RX65N et à votre PC.
2. Connectez la source d'alimentation au RX65N.

Configurez votre AWS resources

1. Pour exécuter les démos de FreeRTOS, vous devez disposer d' AWS un compte auprès d'un utilisateur IAM autorisé à accéder aux services. AWS IoT Si ce n'est pas déjà fait, suivez les étapes décrites dansConfiguration de votre AWS compte et autorisations.
2. Pour configurer les mises à jour OTA, suivez les étapes décrites dansConditions préalables aux mises à jour OTA. En particulier, suivez les étapes décrites dansConditions préalables aux mises à jour OTA via MQTT.
3. Ouvrez la AWS IoT console.
4. Dans le volet de navigation de gauche, choisissez Gérer, puis choisissez Objets pour créer un objet.

   Un objet est une représentation d'un périphérique ou d'une entité logique dans AWS IoT. Il peut s'agir d'un appareil physique ou d'un capteur (par exemple, une ampoule ou un interrupteur sur un mur). Il peut également s'agir d'une entité logique, telle qu'une instance d'une application ou d'une entité physique qui ne se connecte pas AWS IoT, mais qui est associée à des appareils qui le font (par exemple, une voiture équipée de capteurs de moteur ou d'un panneau de commande). AWS IoT fournit un registre d'objets qui vous aide à gérer vos objets.

   a. Choisissez Create, puis Create a single thing.
   b. Entrez un nom pour votre objet, puis choisissez Next.
   c. Choisissez Create certificate (Créer un certificat).
   d. Téléchargez les trois fichiers créés, puis choisissez Activer.
   e. Choisissez Attacher une stratégie.



f. Sélectionnez la politique que vous avez créée dansPolitique en matière d'appareils.

Chaque appareil qui reçoit une mise à jour OTA à l'aide de MQTT doit être enregistré en tant qu'objet AWS IoT et doit être associé à une politique similaire à celle répertoriée. Vous trouverez plus d'informations sur les éléments dans les objets `"Action"` et `"Resource"` dans Actions de stratégie AWS IoT et Ressources d'action AWS IoT Core.

Remarques

- Les `iot:Connect` autorisations permettent à votre appareil de se connecter AWS IoT via MQTT.
- Les autorisations `iot:Publish` et `iot:Subscribe` sur les rubriques des tâches AWS IoT (`.../jobs/*`) permettent au périphérique connecté de recevoir des notifications de travail et des documents de travail, et de publier l'état d'achèvement d'une exécution de travail.
- Les `iot:Publish` autorisations `iot:Subscribe` et sur le thème des flux AWS IoT OTA (`.../streams/*`) permettent à l'appareil connecté de récupérer les données de mise à jour OTA à partir de AWS IoT. Ces autorisations sont requises pour effectuer des mises à jour du firmware sur MQTT.
- Les `iot:Receive` autorisations permettent AWS IoT Core de publier des messages sur ces sujets sur l'appareil connecté. Cette autorisation est vérifiée à chaque remise d'un

message MQTT. Vous pouvez utiliser cette autorisation pour révoquer l'accès aux clients actuellement abonnés à une rubrique.

5. Pour créer un profil de signature de code et enregistrer un certificat de signature de code sur. AWS

   a. Pour créer les clés et la certification, consultez la section 7.3 « Génération de paires de ECDSA-SHA256 clés avec OpenSSL » dans la politique de conception de mise à jour du microprogramme du microcontrôleur Renesas.

   b. Ouvrez la AWS IoT console. Dans le volet de navigation de gauche, sélectionnez Gérer, puis Tâches. Sélectionnez Créer une tâche, puis Créer une tâche de mise à jour OTA.

   c. Sous Sélectionner les appareils à mettre à jour, choisissez Sélectionner, puis choisissez l'élément que vous avez créé précédemment. Sélectionnez Suivant.

   d. Sur la page Créer une tâche de mise à jour FreeRTOS OTA, procédez comme suit :

      i. Pour sélectionner le protocole de transfert d'image du microprogramme, choisissez MQTT.

      ii. Pour Sélectionner et signer l'image de votre microprogramme, choisissez Signer une nouvelle image du microprogramme pour moi.

      iii. Pour le profil de signature de code, choisissez Create.

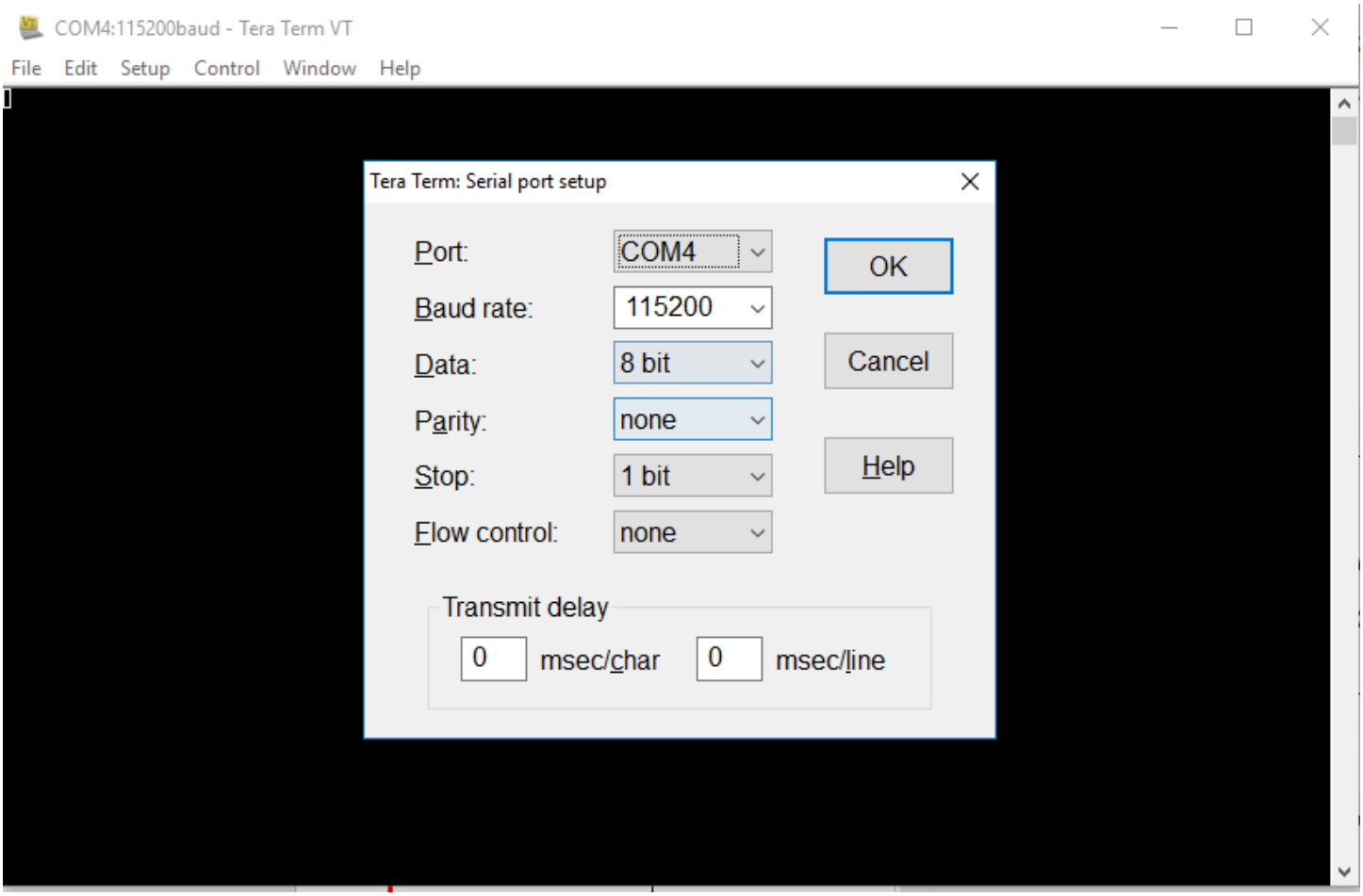
      iv. Dans la fenêtre Créer un profil de signature de code, entrez un nom de profil. Pour la plate-forme matérielle de l'appareil, sélectionnez Windows Simulator. Pour le certificat de signature de code, choisissez Importer.

      v. Naviguez pour sélectionner le certificat (`secp256r1.crt`), la clé privée du certificat (`secp256r1.key`) et la chaîne de certificats (`ca.crt`).

      vi. Entrez le nom de chemin du certificat de signature de code sur l'appareil. Ensuite, choisissez Créer.

6. Pour autoriser l'accès à la signature de code pour AWS IoT, suivez les étapes décrites dansAccorder l'accès à la signature de code pour AWS IoT.

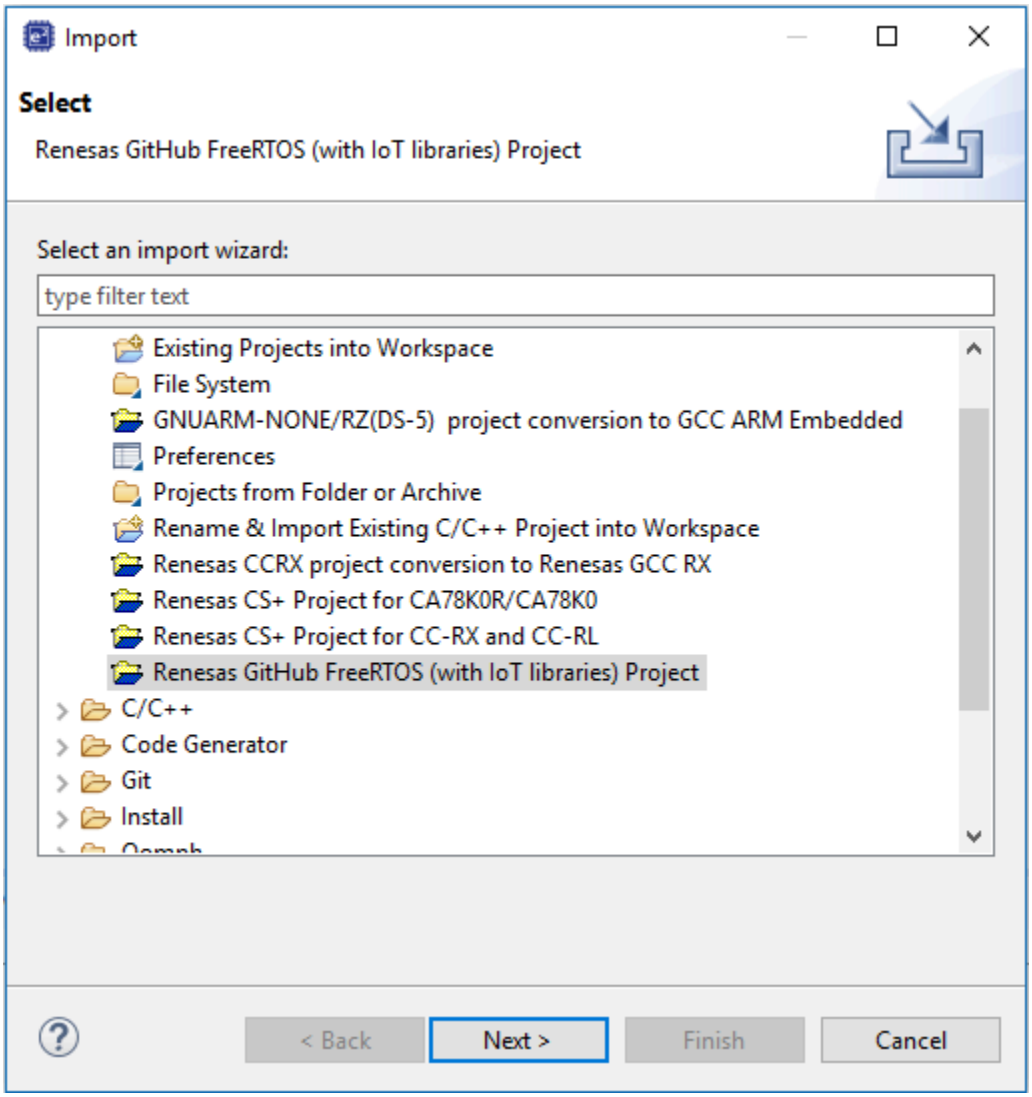
Si Tera Term n'est pas installé sur votre PC, vous pouvez le télécharger https://ttssh2.osdn.jp/index.html.enet le configurer comme indiqué ici. Assurez-vous de brancher le port USB série entre votre appareil et votre PC.



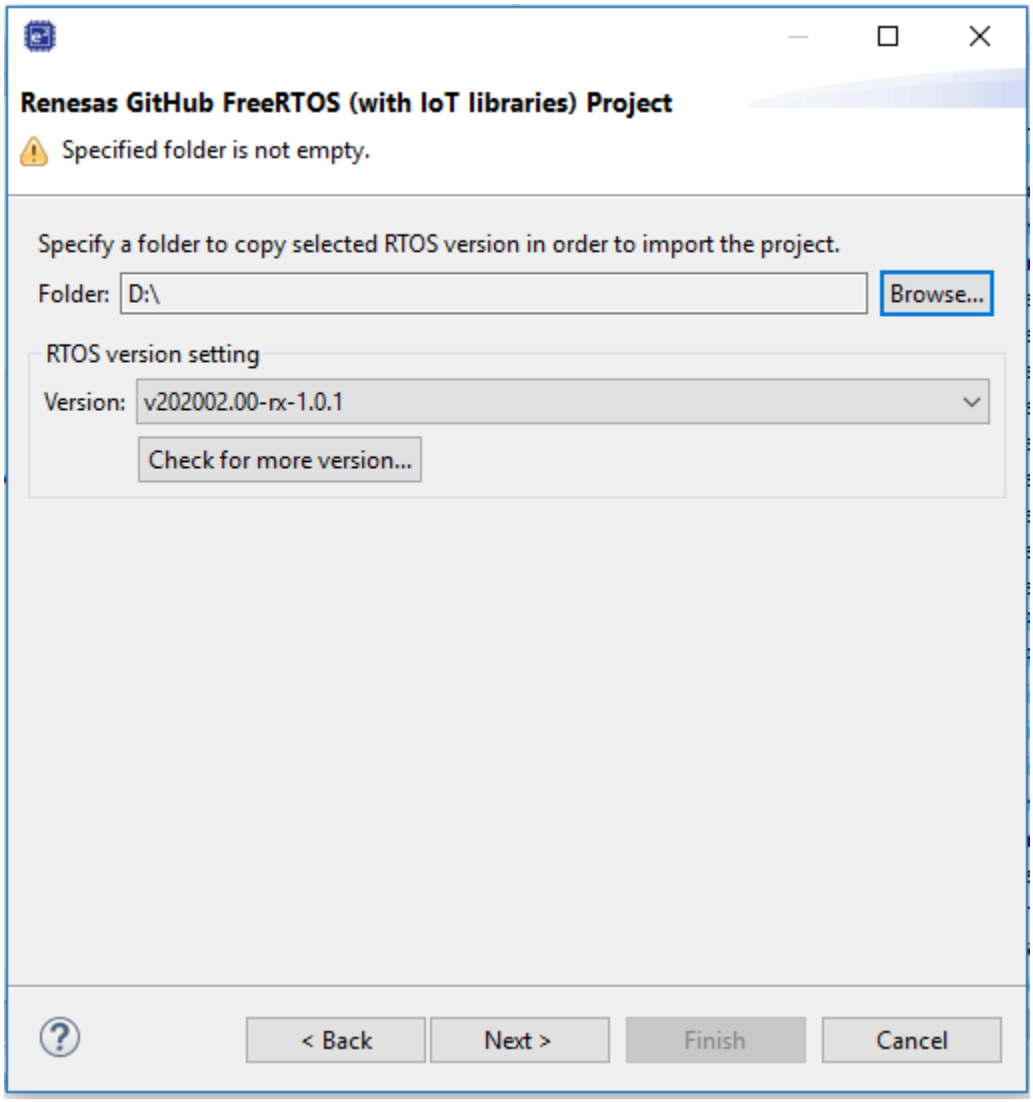
Importez, configurez le fichier d'en-tête et compilez aws_demos et boot_loader

Pour commencer, vous sélectionnez la dernière version du package FreeRTOS, qui sera téléchargée GitHub et importée automatiquement dans le projet. De cette façon, vous pouvez vous concentrer sur la configuration de FreeRTOS et sur l'écriture du code de l'application.

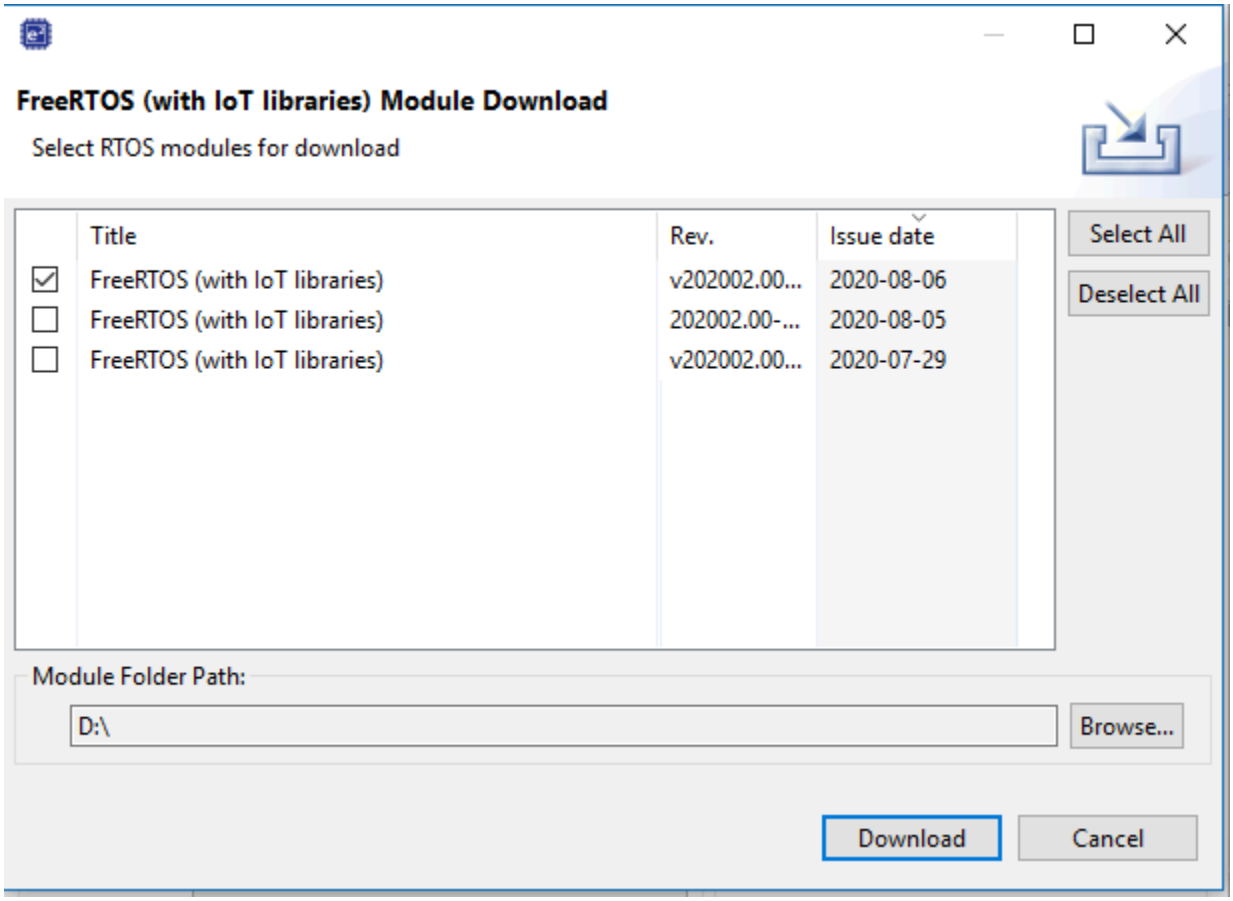
1. Lancez le studio e [2].
2. Choisissez File (Fichier), puis Import (Importer).
3. Sélectionnez le projet GitHub FreeRTOS (avec bibliothèques IoT) de Renesas.



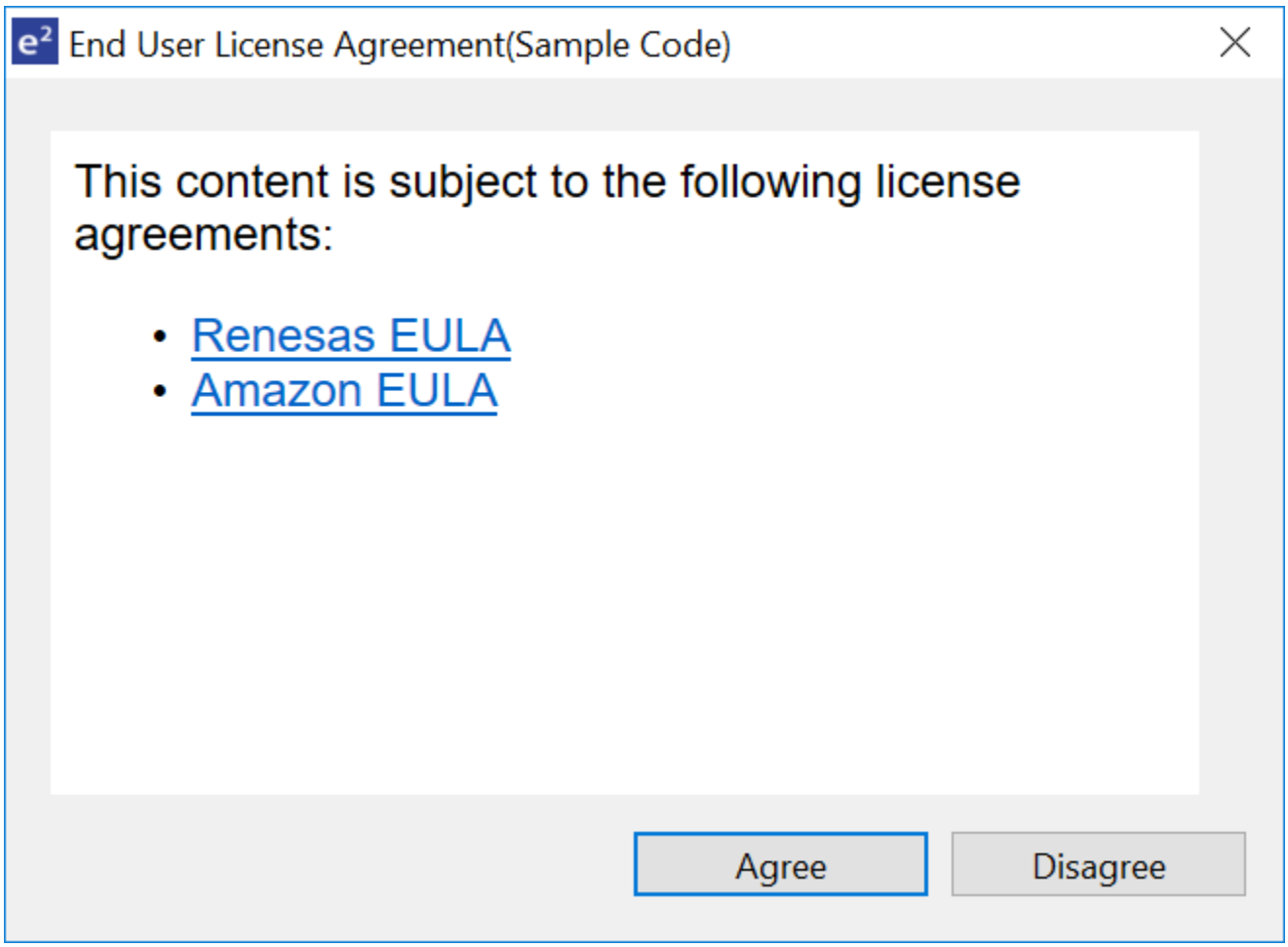
4. Choisissez Rechercher une autre version... pour afficher la boîte de dialogue de téléchargement.



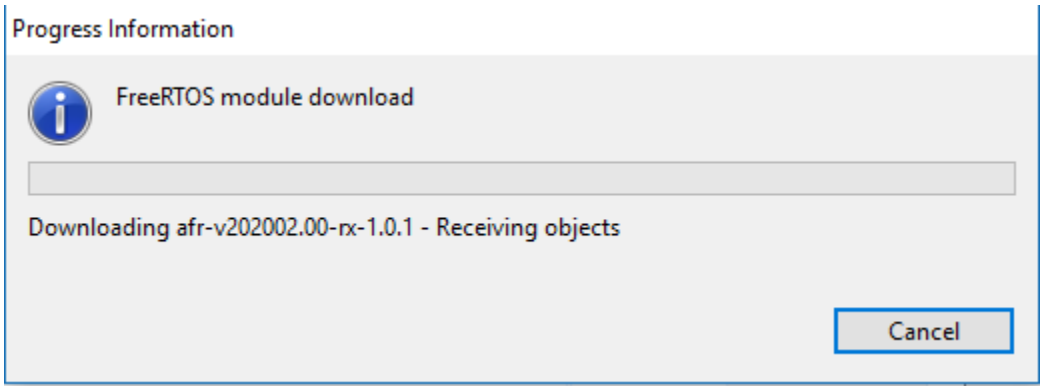
5. Sélectionnez le dernier package.



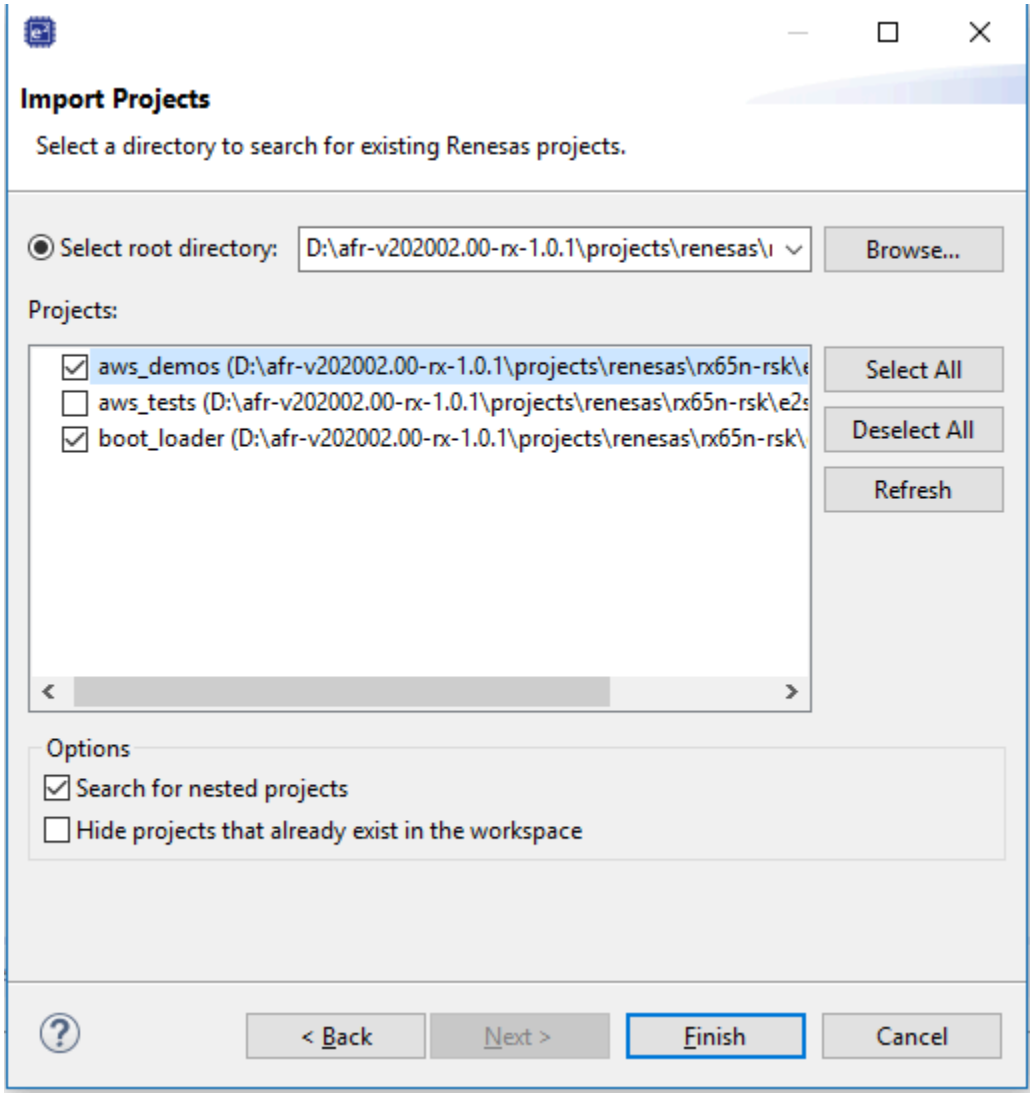
6. Choisissez Accepter pour accepter le contrat de licence de l'utilisateur final.



7. Attendez que le téléchargement soit terminé.



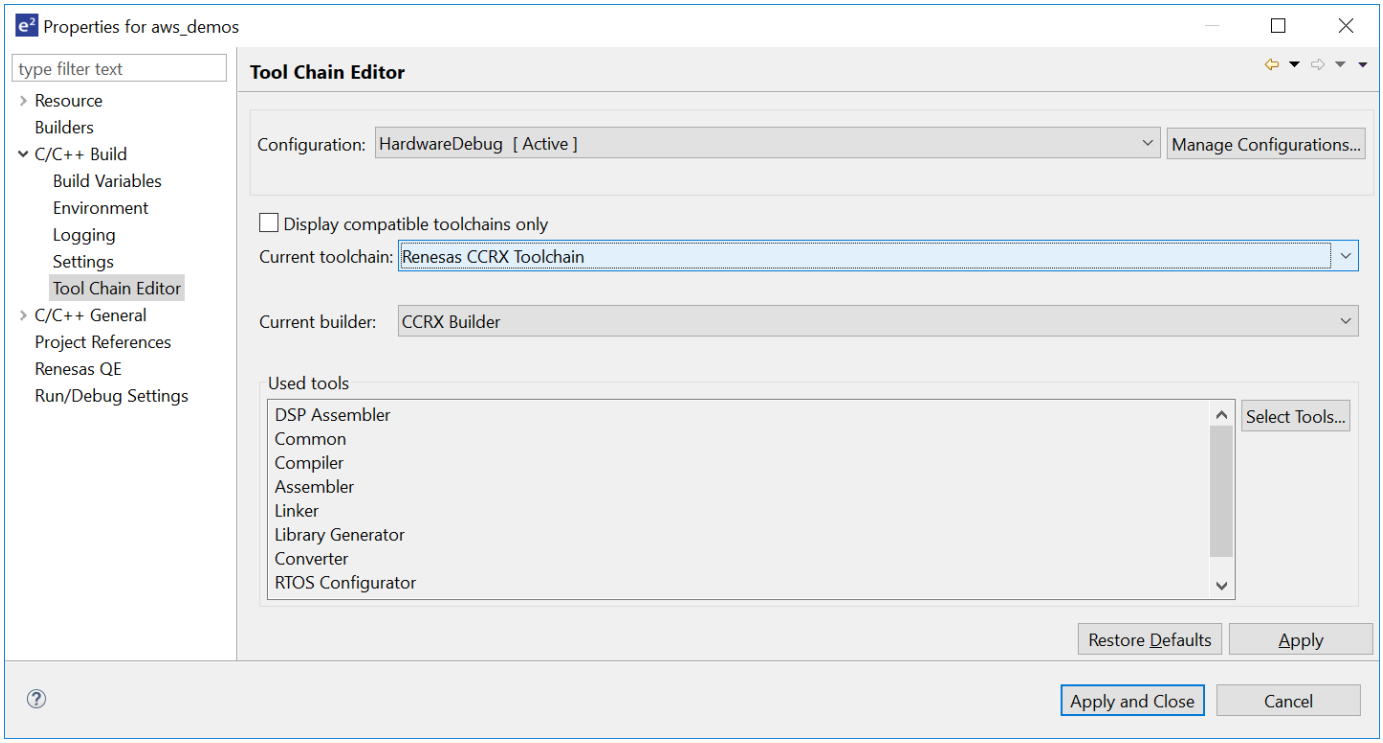
8. Sélectionnez les projets aws_demos et boot_loader, puis choisissez Terminer pour les importer.



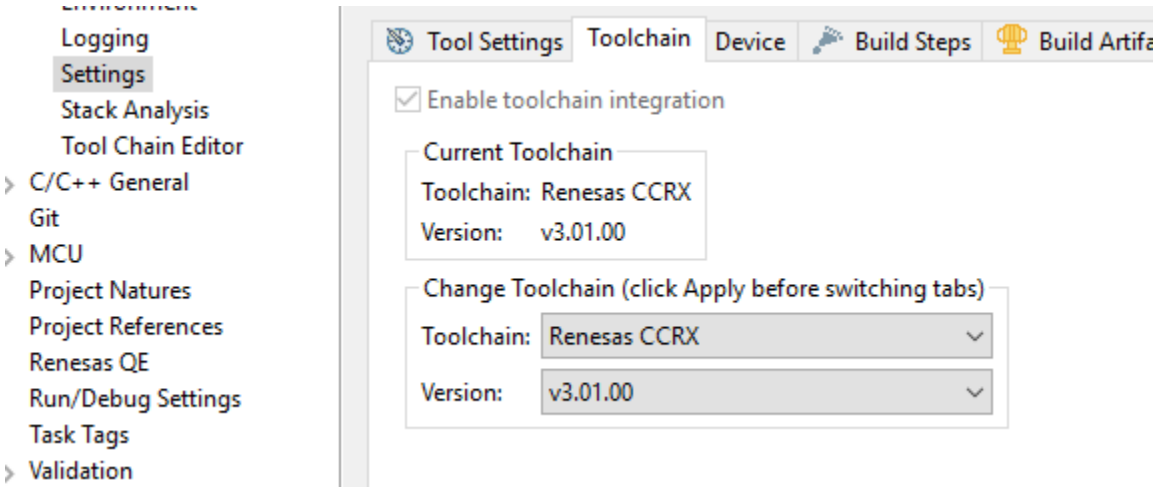
9. Pour les deux projets, ouvrez les propriétés du projet. Dans le volet de navigation, choisissez Tool Chain Editor.

    a. Choisissez la chaîne d'outils actuelle.

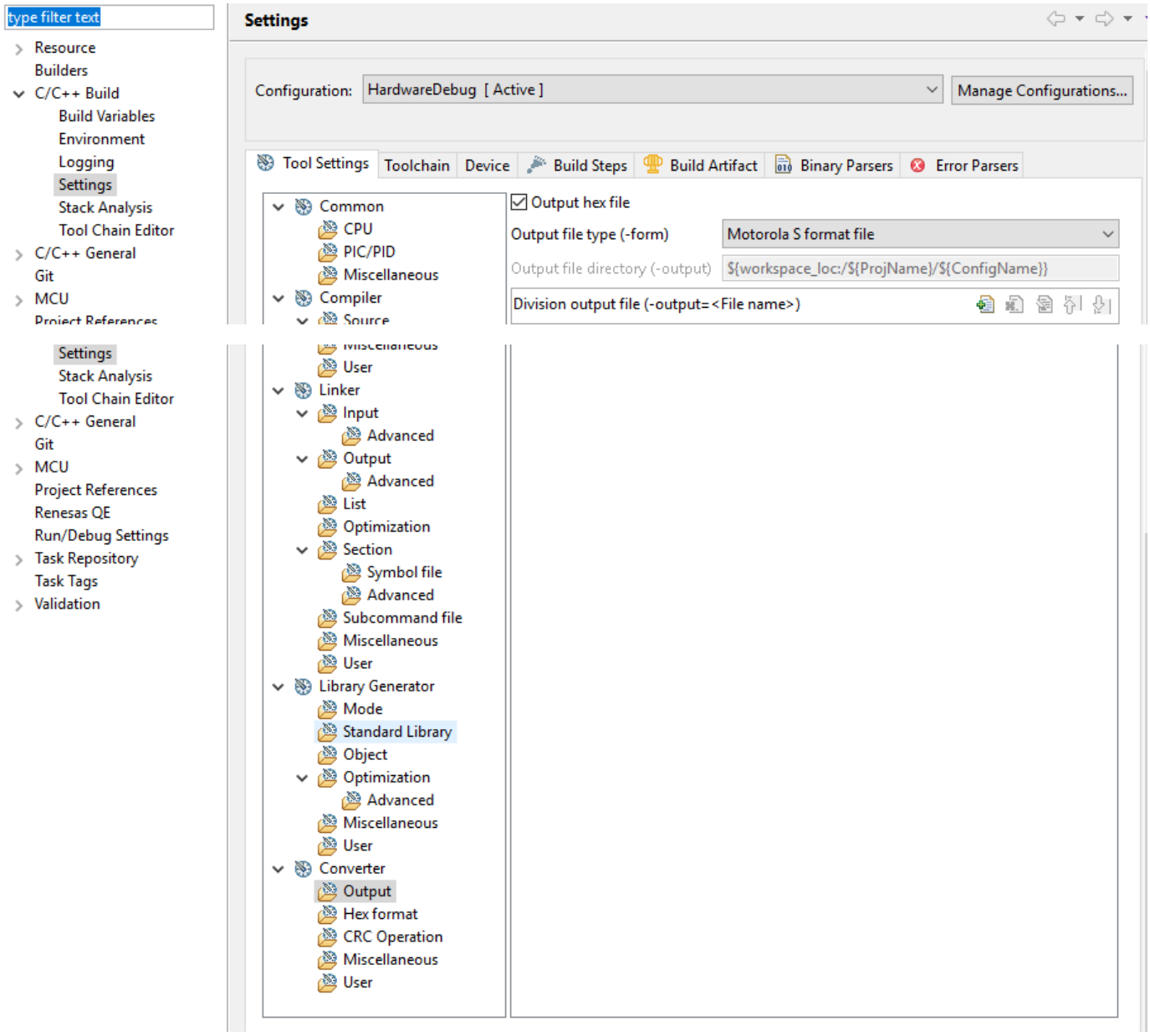
    b. Choisissez le générateur actuel.



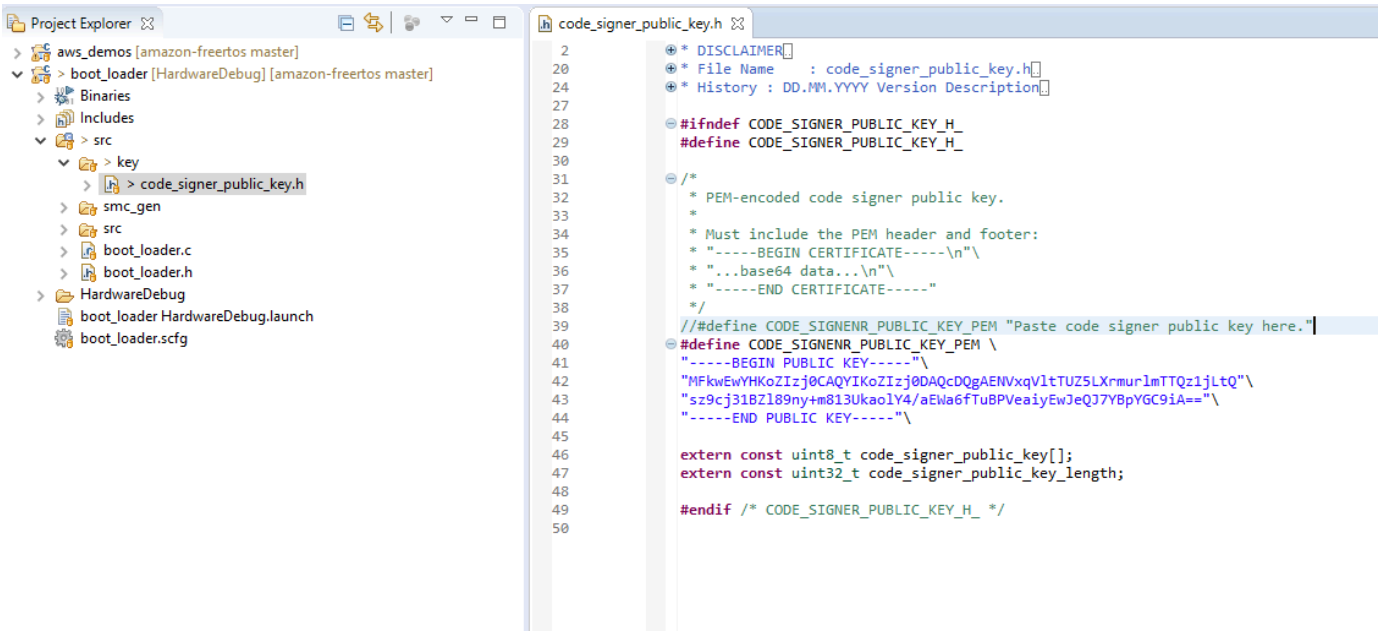
10. Dans le panneau de navigation, sélectionnez Settings (Paramètres). Choisissez l'onglet Chaîne d'outils, puis choisissez la version de la chaîne d'outils.



Choisissez l'onglet Paramètres de l'outil, développez le convertisseur, puis choisissez Sortie. Dans la fenêtre principale, assurez-vous que le fichier hexadécimal de sortie est sélectionné, puis choisissez le type de fichier de sortie.



11. Dans le projet Bootloader, ouvrez `projects\renesas\rx65n-rsk\e2studio\boot_loader\src\key\code_signer_public_key.h` et saisissez la clé publique. Pour plus d'informations sur la création d'une clé publique, consultez Comment implémenter FreeRTOS OTA à l'aide d'Amazon Web Services sur le RX65N et la section 7.3 « Génération de paires de ECDSA-SHA256 clés avec OpenSSL » dans la politique de conception de mise à jour du microprogramme des microcontrôleurs de Renesas.



Construisez ensuite le projet à créer`boot_loader.mot`.

12. Ouvrez le `aws_demos` projet.

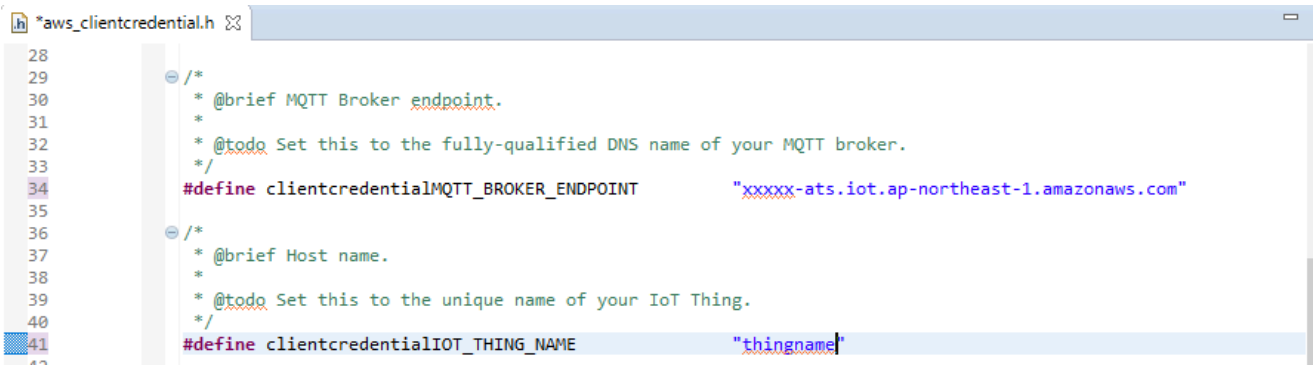
    a. Ouvrez la AWS IoT console.

    b. Dans le panneau de navigation de gauche, choisissez Paramètres. Notez votre point de terminaison personnalisé dans la zone de texte Terminal de données de l'appareil.

    c. Choisissez Gérer, puis choisissez Objets. Notez le nom de l' AWS IoT objet de votre tableau.

    d. Dans le `aws_demos` projet, ouvrez `demos/include/aws_clientcredential.h` et spécifiez les valeurs suivantes.

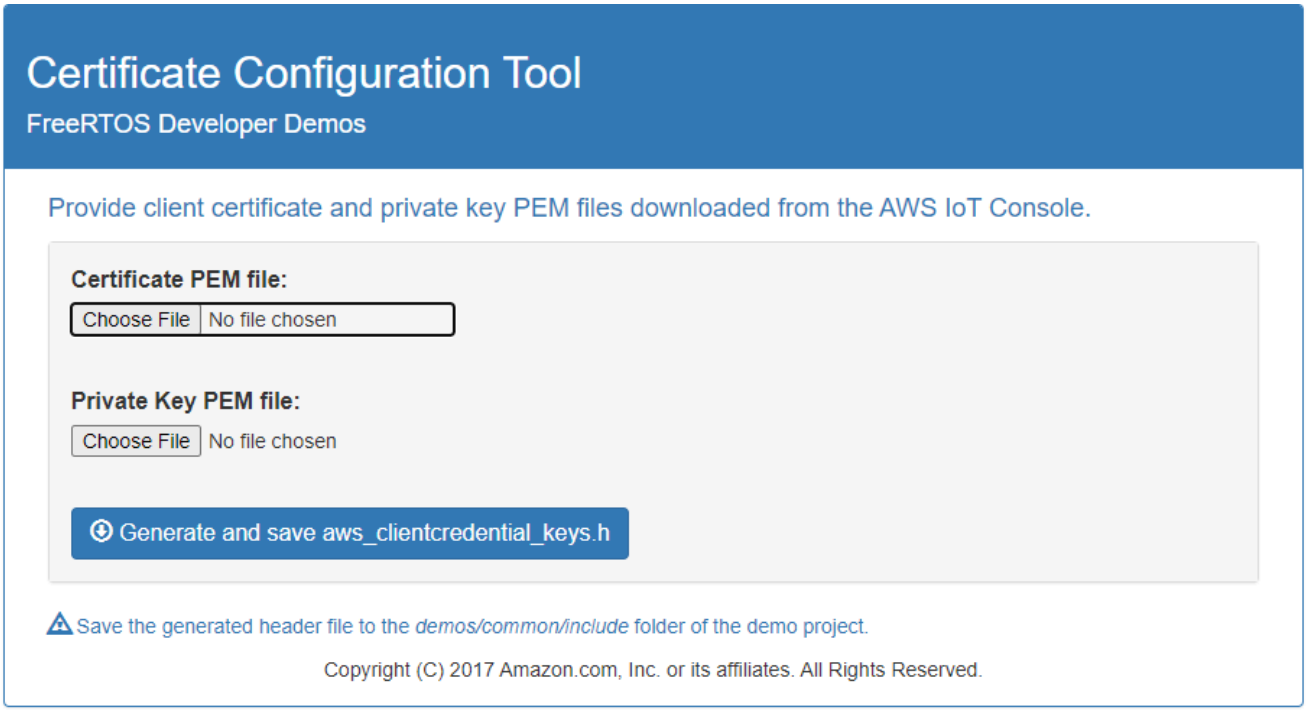
```
#define clientcredentialMQTT_BROKER_ENDPOINT[] = "Your AWS IoT endpoint";
#define clientcredentialIOT_THING_NAME "The AWS IoT thing name of your board"
```



e. Ouvrez le fichier `tools/certificate_configuration/CertificateConfigurator.html`.

f. Importez le fichier PEM de certificat et le fichier PEM de clé privée que vous avez téléchargés précédemment.

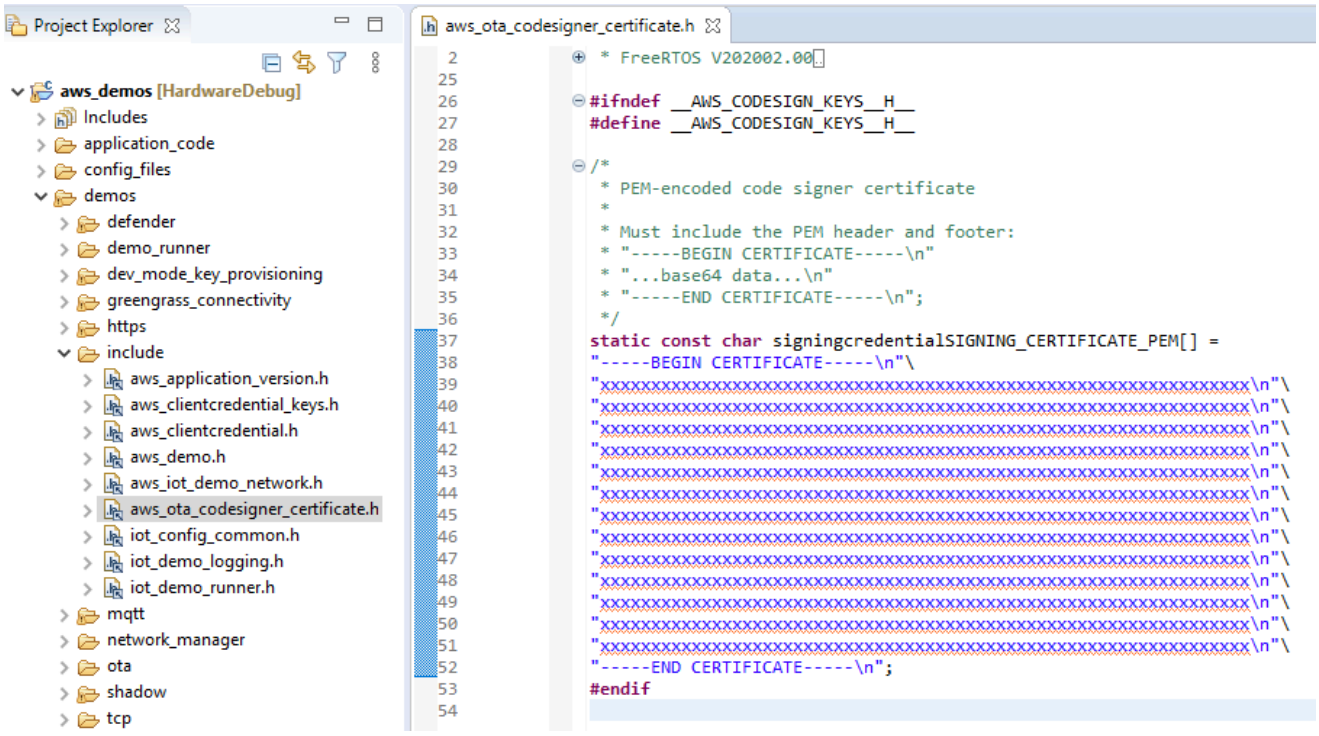
g. Choisissez Generate, enregistrez aws_clientcredential_keys.h et remplacez ce fichier dans le répertoire. `demos/include/`



h. Ouvrez le `vendors/renesas/boards/rx65n-rsk/aws_demos/config_files/ota_demo_config.h` fichier et spécifiez ces valeurs.

```
#define otapalconfigCODE_SIGNING_CERTIFICATE [] = "your-certificate-key";
```

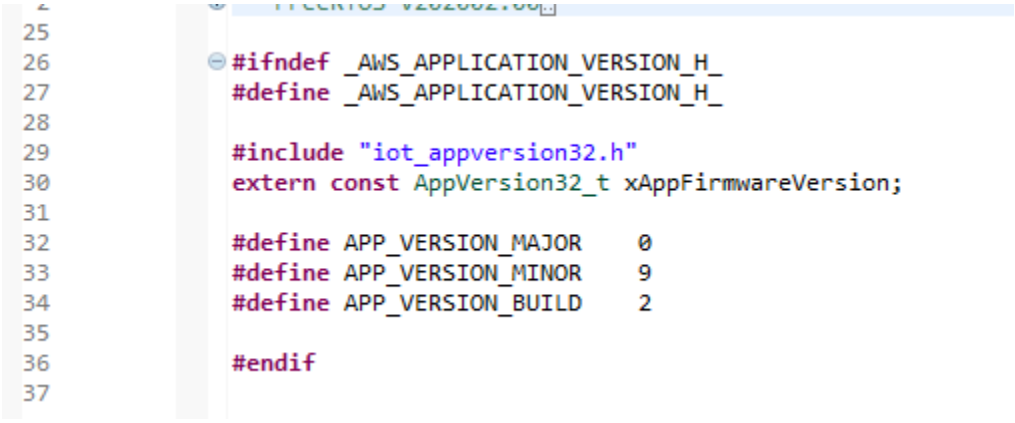
Où *`your-certificate-key`* se trouve la valeur du fichier`secp256r1.crt`. N'oubliez pas d'ajouter « \ » après chaque ligne de la certification. Pour plus d'informations sur la création du `secp256r1.crt` fichier, consultez Comment implémenter FreeRTOS OTA à l'aide d'Amazon Web Services sur le RX65N et la section 7.3 « Génération de paires de ECDSA-SHA256 clés avec OpenSSL » dans la politique de conception des mises à jour du microprogramme des microcontrôleurs de Renesas.



13. Tâche A : Installation de la version initiale du microprogramme

    a. Ouvrez le `vendors/renesas/boards/board/aws_demos/config_files/aws_demo_config.h` fichier, commentez `#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED` et définissez l'un `CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` ou l'autre`CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`.

    b. Ouvrez le `demos/include/ aws_application_version.h` fichier et définissez la version initiale du microprogramme sur`0.9.2`.



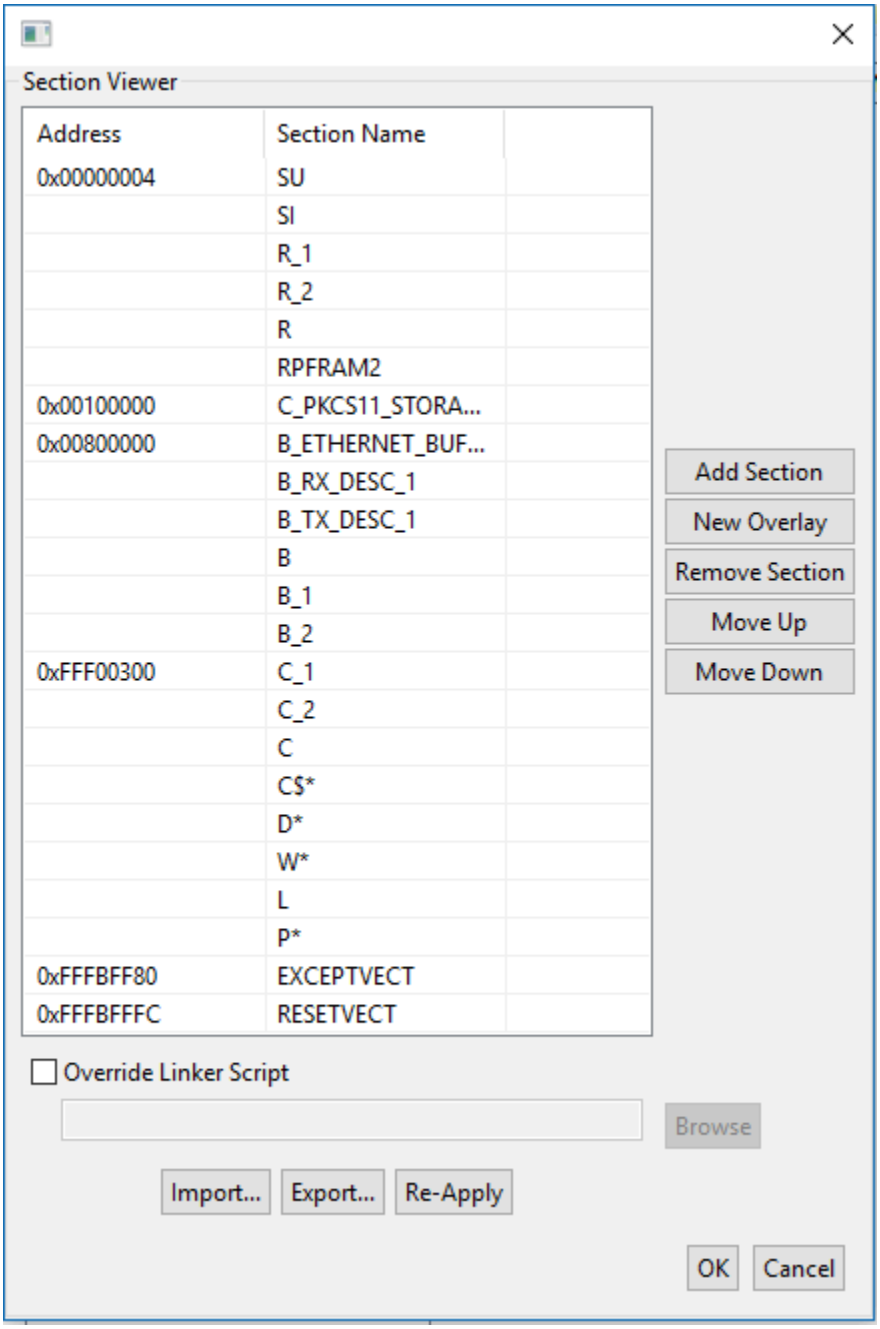
```
#ifndef _AWS_APPLICATION_VERSION_H_
#define _AWS_APPLICATION_VERSION_H_

#include "iot_appversion32.h"
extern const AppVersion32_t xAppFirmwareVersion;

#define APP_VERSION_MAJOR    0
#define APP_VERSION_MINOR    9
#define APP_VERSION_BUILD    2

#endif
```

    c. Modifiez les paramètres suivants dans l'afficheur de sections.



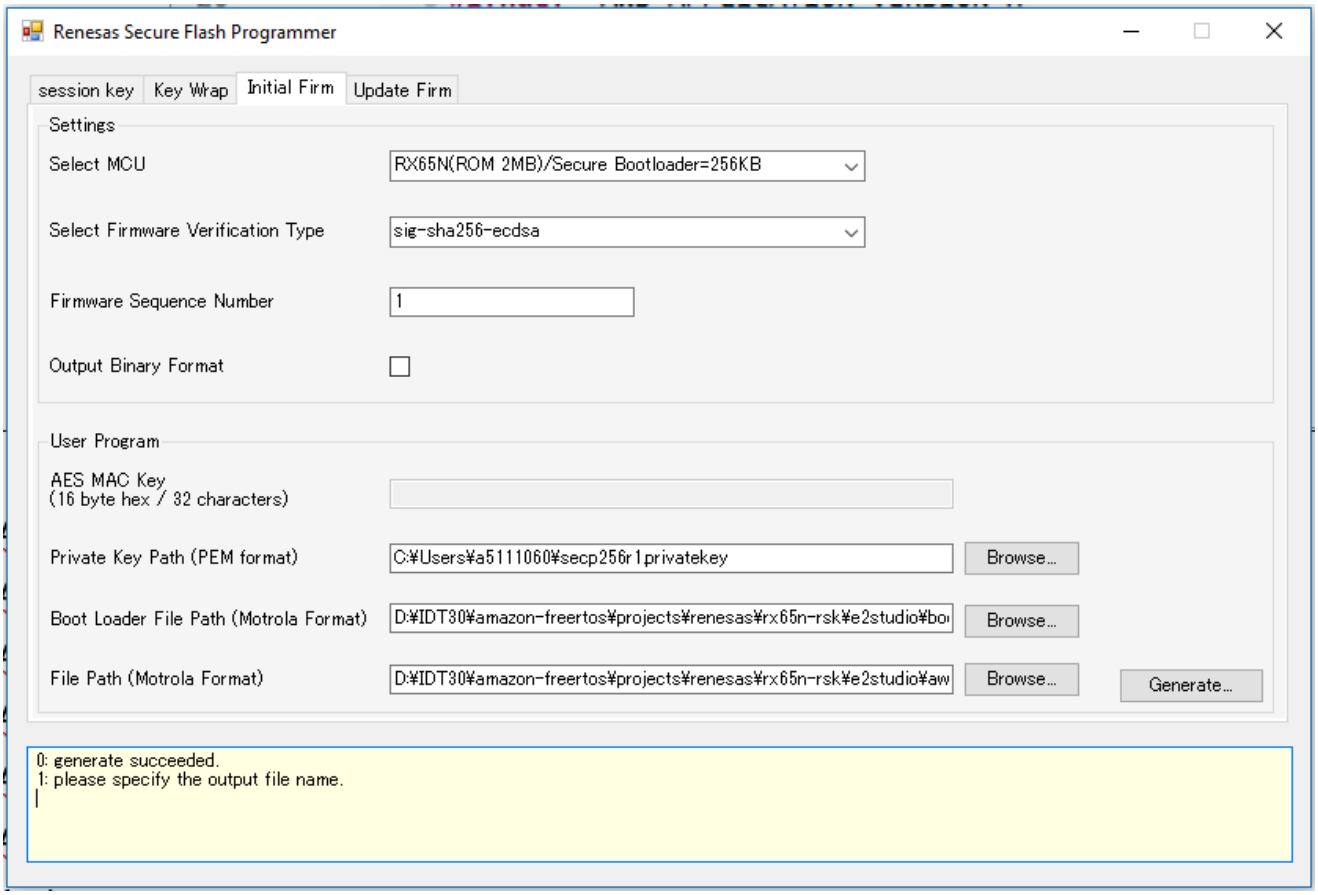
d. Choisissez Build pour créer le `aws_demos.mot` fichier.

14. Créez le fichier `userprog.mot` avec le Renesas Secure Flash Programmer. `userprog.mot`est une combinaison de `aws_demos.mot` et`boot_loader.mot`. Vous pouvez flasher ce fichier sur le RX65N-RSK pour installer le microprogramme initial.

    a. Téléchargez https://github.com/renesas/Amazon-FreeRTOS-Toolset ouvrez`Renesas Secure Flash Programmer.exe`.

    b. Choisissez l'onglet Firme initiale, puis définissez les paramètres suivants :

- Chemin de clé privée : emplacement de `secp256r1.privatekey`.
- Chemin du fichier Boot Loader : emplacement de `boot_loader.mot` (`projects\renesas\rx65n-rsk\e2studio\boot_loader\HardwareDebug`).
- Chemin du fichier : emplacement du `aws_demos.mot` (`projects\renesas\rx65n-rsk\e2studio\aws_demos\HardwareDebug`).



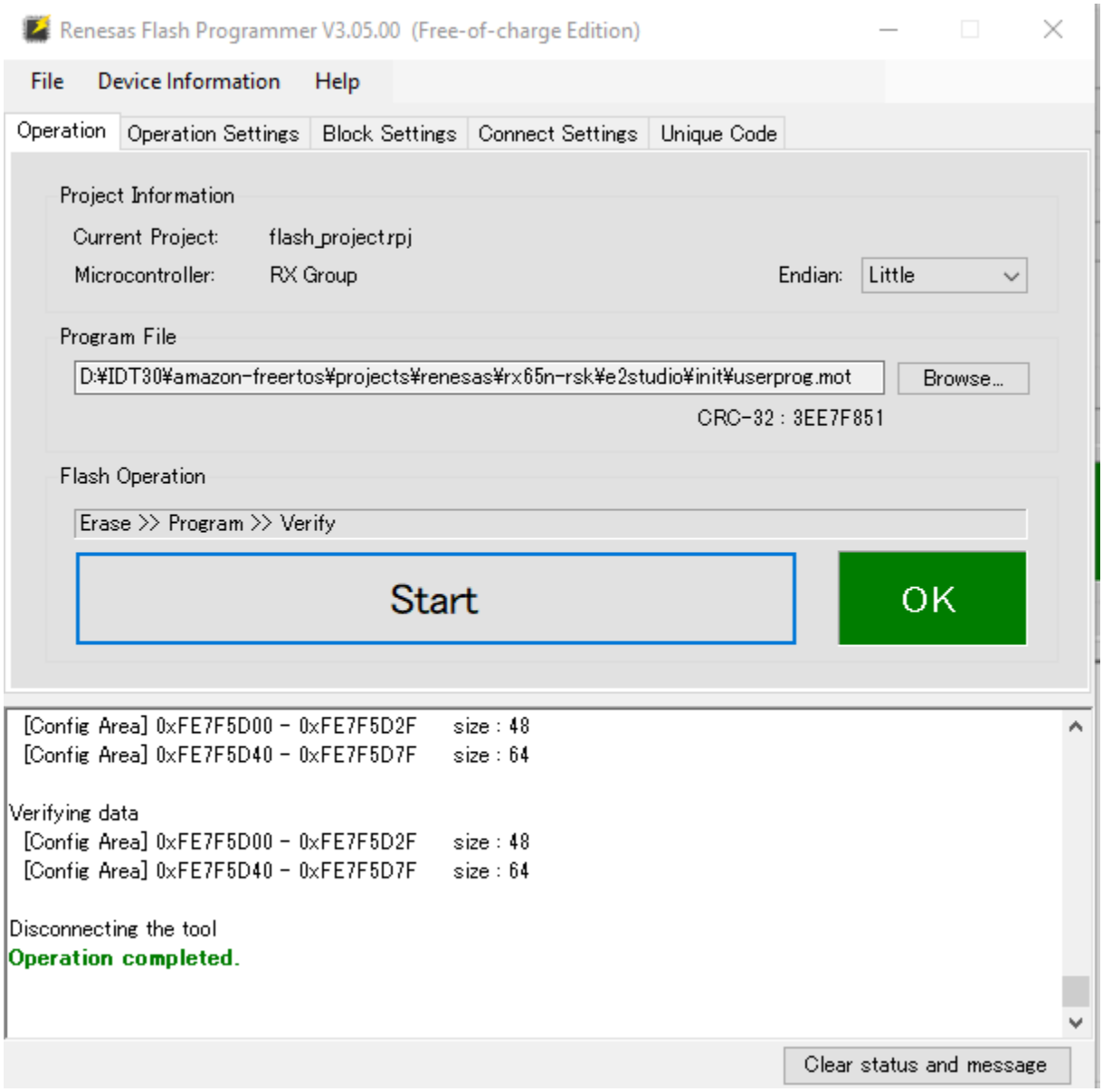
c. Créez un répertoire nommé `init_firmware` `userprog.mot` Generate et enregistrez-le `init_firmware` dans le répertoire. Vérifiez que la génération a réussi.

15. Flashez le microprogramme initial sur le RX65N-RSK.

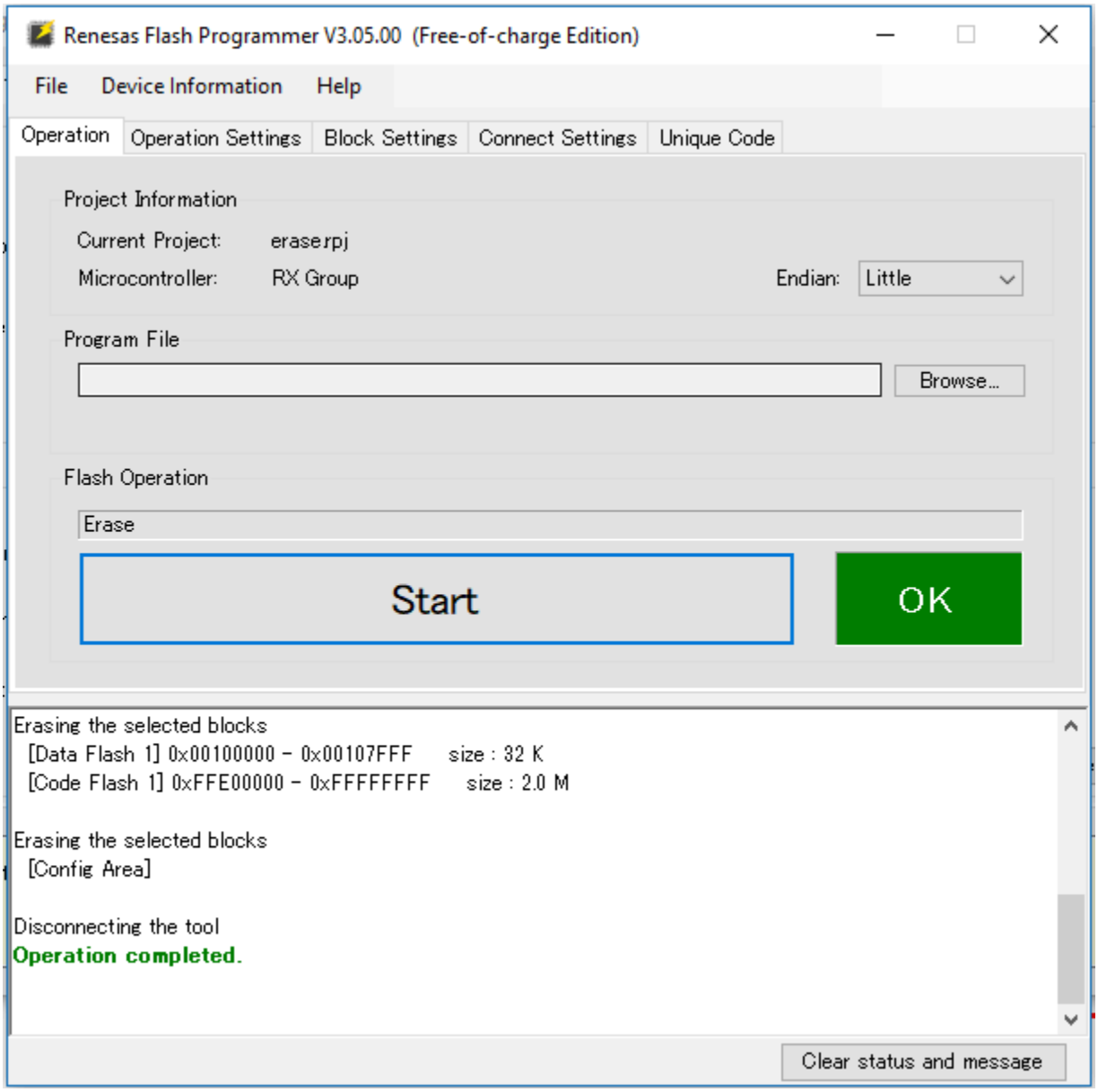
a. Téléchargez la dernière version du programmateur Flash Renesas (interface graphique de programmation) à partir de. https://www.renesas.com/tw/en/products/software-tools/tools/programmer/renesas-flash-programmer-programming-gui.html

b. Ouvrez le `vendors\renesas\rx_mcu_boards\boards\rx65n-rsk\aws_demos\flash_project\erase_from_bank\ erase.rpj` fichier pour effacer les données de la banque.

c. Choisissez Démarrer pour effacer la banque.



d. Pour flasher`userprog.mot`, choisissez Parcourir... et naviguez jusqu'au `init_firmware` répertoire, sélectionnez le `userprog.mot` fichier et choisissez Démarrer.



16. La version 0.9.2 (version initiale) du microprogramme a été installée sur votre RX65N-RSK. Le RX65N-RSK conseil d'administration écoute actuellement les mises à jour de l'OTA. Si vous avez ouvert Tera Term sur votre PC, vous verrez quelque chose comme ceci lorsque le microprogramme initial s'exécute.

```
------------------------------------------------
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. Tâche B : mettre à jour la version de votre microprogramme

    a. Ouvrez le `demos/include/aws_application_version.h` fichier et augmentez la valeur du `APP_VERSION_BUILD` jeton à`0.9.3`.

b. Regénérez le projet.

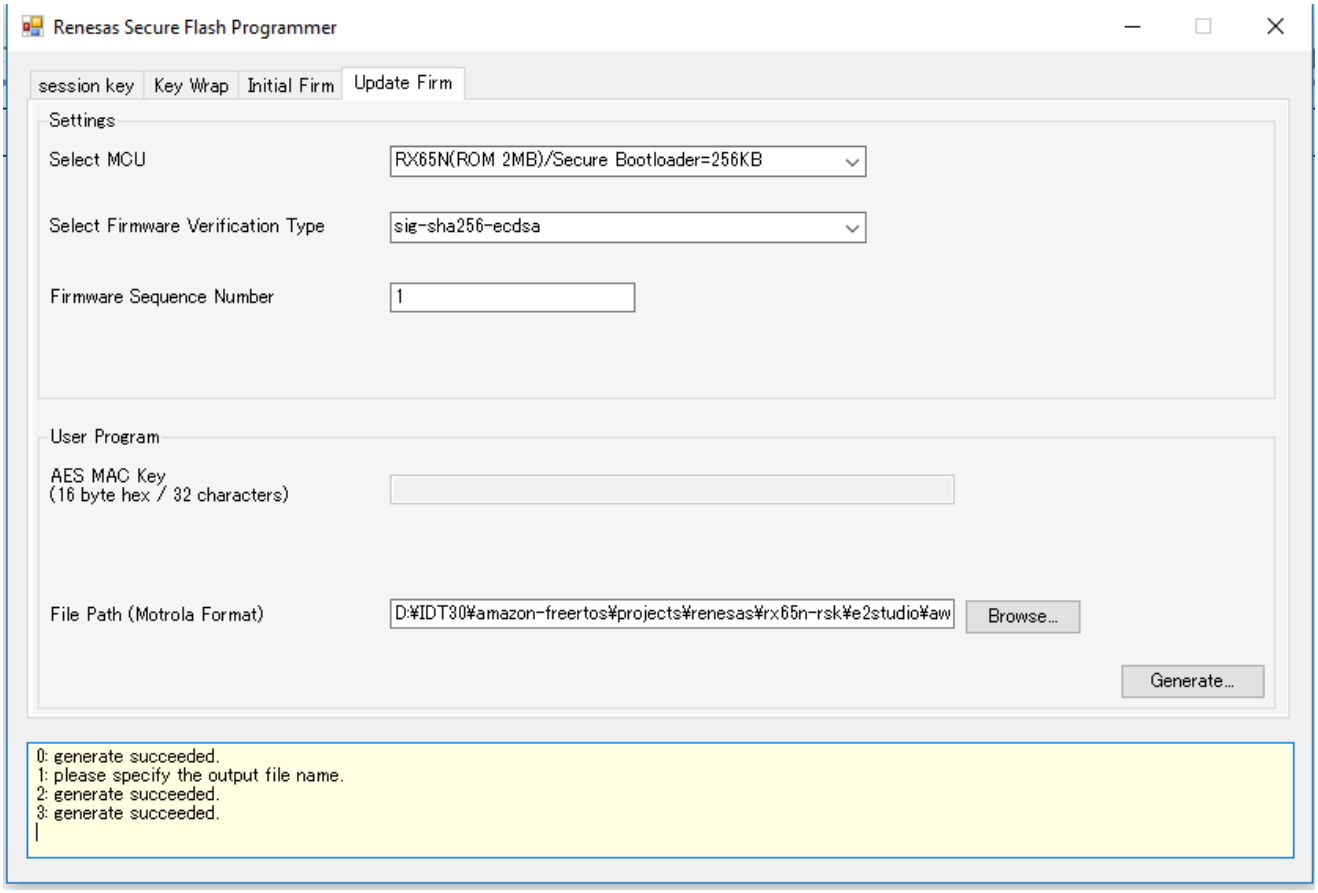
18. Créez le `userprog.rsu` fichier avec le Renesas Secure Flash Programmer pour mettre à jour la version de votre firmware.

    a. Ouvrez le fichier `Amazon-FreeRTOS-Tools\Renesas Secure Flash Programmer.exe`.

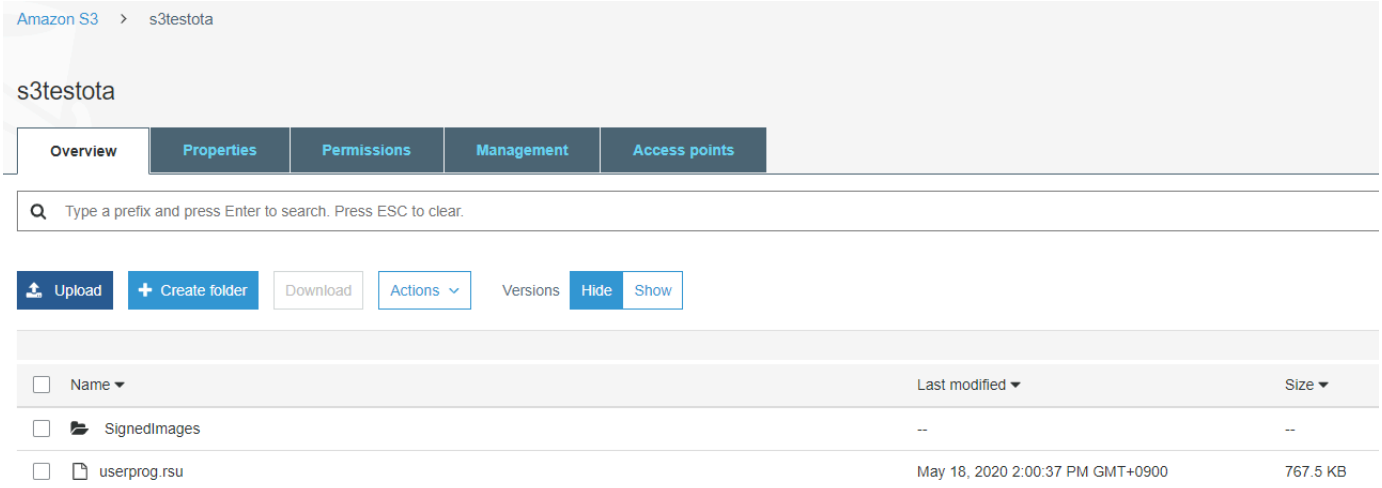
    b. Choisissez l'onglet Update Firm et définissez les paramètres suivants :

    - Chemin du fichier : emplacement du `aws_demos.mot` fichier (`projects\renesas\rx65n-rsk\e2studio\aws_demos\HardwareDebug`).

    c. Créez un répertoire nommé `update _firmware`. Générez-le `userprog.rsu` et enregistrez-le `update_firmware` dans le répertoire. Vérifiez que la génération a réussi.



19. Téléchargez la mise à jour du microprogramme dans un compartiment Amazon S3`userproj.rsu`, comme décrit dans[Créez un compartiment Amazon S3 pour stocker votre mise à jour](#).



20. Créez une tâche pour mettre à jour le microprogramme sur le RX65N-RSK.

    AWS IoT Jobs est un service qui avertit un ou plusieurs appareils connectés d'un Job en attente. Une tâche peut être utilisée pour gérer un parc d'appareils, mettre à jour le microprogramme et les certificats de sécurité des appareils, ou effectuer des tâches administratives telles que le redémarrage des appareils et l'exécution de diagnostics.

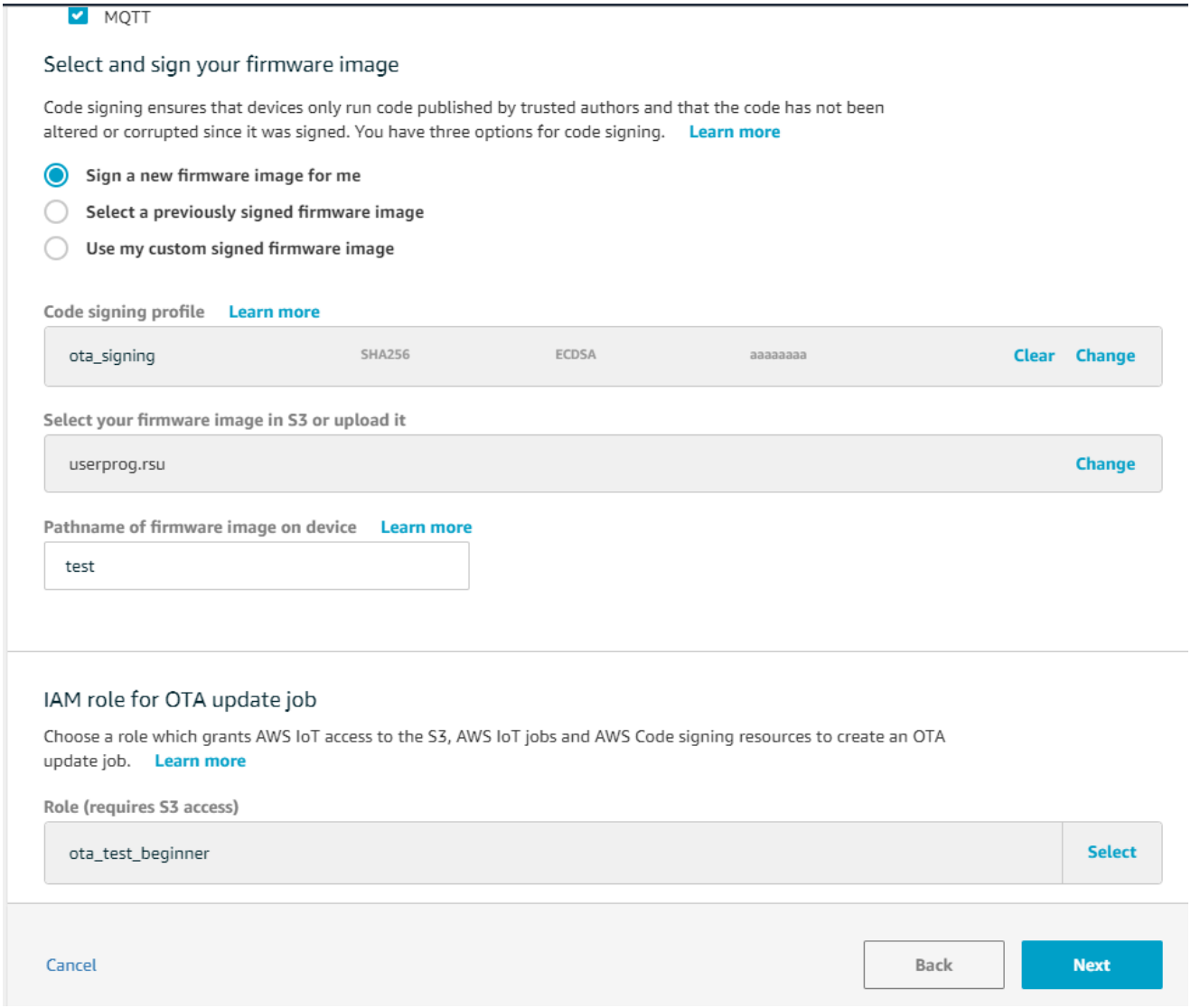
    a. Connectez-vous à la console AWS IoT. Dans le volet de navigation, choisissez Manage, puis Jobs.

    b. Choisissez Create a job, puis Create OTA Update job. Sélectionnez un élément, puis cliquez sur Suivant.

    c. Créez une tâche de mise à jour FreeRTOS OTA comme suit :

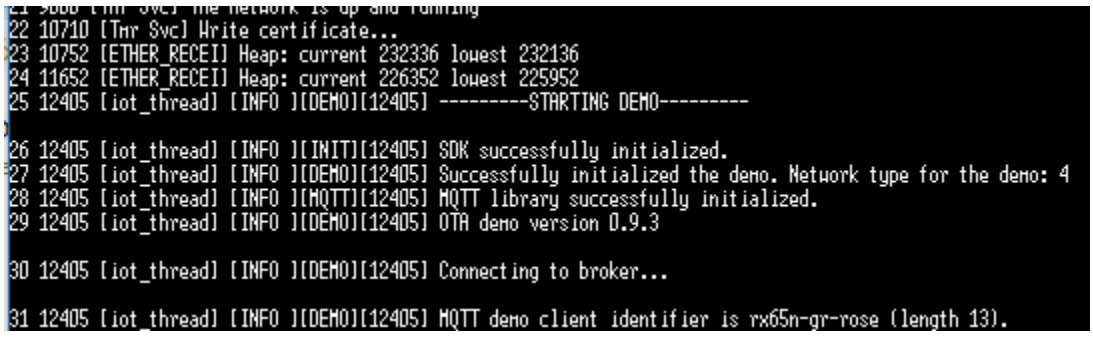
    - Choisissez MQTT.
    - Sélectionnez le profil de signature de code que vous avez créé dans la section précédente.
    - Sélectionnez l'image du microprogramme que vous avez chargée dans un compartiment Amazon S3.
    - Pour le chemin d'accès de l'image du microprogramme sur l'appareil, entrez**`test`**.
    - Choisissez le rôle IAM que vous avez créé dans la section précédente.

    d. Choisissez Suivant.

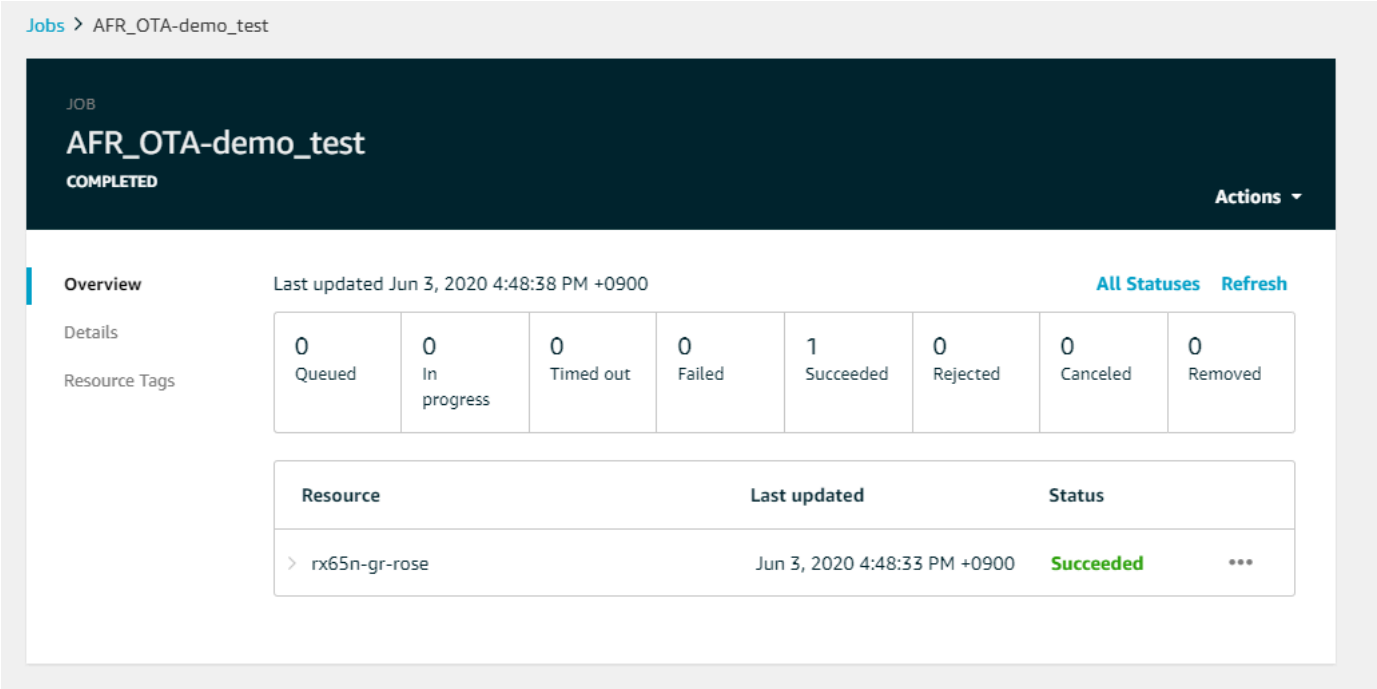


e. Entrez un identifiant, puis choisissez Créer.

21. Rouvrez Tera Term pour vérifier que le microprogramme a été correctement mis à jour vers la version de démonstration OTA 0.9.3.



22. Sur la AWS IoT console, vérifiez que le statut de la tâche est Réussi.



Tutoriel : Effectuez des mises à jour OTA sur Espressif ESP32 à l'aide de FreeRTOS Bluetooth Low Energy

> **Important**
>
> Cette intégration de référence est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Ce didacticiel explique comment mettre à jour un microcontrôleur Espressif ESP32 connecté à un proxy Bluetooth Low Energy MQTT sur un appareil Android. Il met à jour l'appareil à l'aide de tâches de mise à jour AWS IoT Over-the-air (OTA). L'appareil se connecte à AWS IoT l'aide des informations d'identification Amazon Cognito saisies dans l'application de démonstration Android. Un opérateur autorisé lance la mise à jour OTA depuis le cloud. Lorsque l'appareil se connecte via l'application de démonstration Android, la mise à jour OTA est lancée et le micrologiciel est mis à jour sur l'appareil.

Les versions majeures de FreeRTOS 2019.06.00 et ultérieures incluent le support du proxy Bluetooth Low Energy MQTT qui peut être utilisé pour Wi-Fi le provisionnement et la sécurisation

des connexions aux services. AWS IoT En utilisant la fonction Bluetooth Low Energy, vous pouvez créer des appareils à faible consommation qui peuvent être couplés à un appareil mobile pour la connectivité sans en avoir besoin Wi-Fi. Les appareils peuvent communiquer à l'aide de MQTT en se connectant via des SDK Bluetooth Low Energy Android ou iOS qui utilisent des profils de profil d'accès générique (GAP) et d'attributs génériques (GATT).

Voici les étapes que nous allons suivre pour autoriser les mises à jour OTA via Bluetooth Low Energy :

1. Configuration du stockage : créez un compartiment et des politiques Amazon S3 et configurez un utilisateur capable d'effectuer des mises à jour.
2. Création d'un certificat de signature de code : créez un certificat de signature et autorisez l'utilisateur à signer les mises à jour du microprogramme.
3. Configurez l'authentification Amazon Cognito : créez un fournisseur d'informations d'identification, un groupe d'utilisateurs et un accès des applications au groupe d'utilisateurs.
4. Configurer FreeRTOS : configurez Bluetooth Low Energy, les informations d'identification du client et le certificat public de signature de code.
5. Configuration d'une application Android : configurez le fournisseur d'informations d'identification, le groupe d'utilisateurs et déployez l'application sur un appareil Android.
6. Exécutez le script de mise à jour OTA : pour lancer une mise à jour OTA, utilisez le script de mise à jour OTA.

Pour plus d'informations sur le fonctionnement des mises à jour, consultezMises à jour de FreeRTOS Over-the-Air. Pour plus d'informations sur la configuration de la fonctionnalité du proxy Bluetooth Low Energy MQTT, consultez l'article Utilisation du Bluetooth Low Energy avec FreeRTOS sur Espressif ESP32 de Richard Kang.

Conditions préalables

Pour effectuer les étapes de ce didacticiel, vous avez besoin des ressources suivantes :

- Une carte de développement ESP32.
- Un câble microUSB vers USB A.
- Un AWS compte (le niveau gratuit est suffisant).
- Un téléphone Android doté d'Android version 6.0 ou ultérieure et de Bluetooth version 4.2 ou ultérieure.

Sur votre ordinateur de développement, vous devez :

- Espace disque suffisant (~500 Mo) pour la chaîne d'outils Xtensa, le code source FreeRTOS et des exemples.
- Android Studio installé.
- L'AWS CLIinstallé.
- Python3 installé.
- Le kit de développement AWS logiciel (SDK) boto3 pour Python.

Les étapes de ce didacticiel supposent que la chaîne d'outils Xtensa ESP-IDF et le code FreeRTOS sont installés dans le répertoire de votre répertoire personnel. `/esp` Vous devez ajouter des `~/esp/xtensa-esp32-elf/bin` éléments à votre `$PATH` variable.

Étape 1 : Configuration du stockage

1. Créez un compartiment Amazon S3 pour stocker votre mise à jouravec la gestion des versions activée pour conserver les images du microprogramme.
2. Créer un rôle de service de mise à jour OTAet ajoutez les politiques gérées suivantes au rôle :
   - AWSIotLogging
   - AWSIotRuleActions
   - AWSIotThingsRegistration
   - AWSFreeRTOSOTAUpdate
3. Créez un utilisateur capable d'effectuer des mises à jour OTA. Cet utilisateur peut signer et déployer des mises à jour du microprogramme sur les appareils IoT du compte, et a accès aux mises à jour OTA sur tous les appareils. L'accès doit être limité aux entités de confiance.
4. Suivez les étapes pour Créer une stratégie utilisateur OTA le joindre à votre utilisateur.

Étape 2 : Création du certificat de signature de code

1. Créez un compartiment Amazon S3 avec la gestion des versions activée pour contenir les images du microprogramme.
2. Créez un certificat de signature de code qui peut être utilisé pour signer le microprogramme. Notez le certificat Amazon Resource Name (ARN) lorsque le certificat est importé.

```
aws acm import-certificate --profile=ota-update-user --certificate file://
ecdsasigner.crt --private-key file://ecdsasigner.key
```

Exemple de sortie :

```
{
"CertificateArn": "arn:aws:acm:us-east-1:<account>:certificate/<certid>"
}
```

Vous utiliserez l'ARN ultérieurement pour créer un profil de signature. Si vous le souhaitez, vous pouvez créer le profil dès maintenant à l'aide de la commande suivante :

```
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
```

Exemple de sortie :

```
{
"arn": "arn:aws:signer::<account>:/signing-profiles/esp32Profile"
}
```

Étape 3 : configuration de l'authentification Amazon Cognito

Créez un AWS IoT policy

1. Connectez-vous à la console AWS IoT.
2. Dans le coin supérieur droit de la console, sélectionnez Mon compte. Dans Paramètres du compte, notez votre identifiant de compte à 12 chiffres.
3. Dans le panneau de navigation de gauche, choisissez Paramètres. Dans Point de terminaison des données de l'appareil, notez la valeur du point de terminaison. Le point final doit être quelque chose comme`xxxxxxxxxxxxxx.iot.us-west-2.amazonaws.com`. Dans cet exemple, la AWS région est « us-west-2 ».

4. Dans le volet de navigation de gauche, choisissez Secure, Policies, puis Create. Si aucune politique n'est associée à votre compte, le message « Vous n'avez pas encore de politique » s'affichera et vous pourrez choisir Créer une politique.
5. Entrez un nom pour votre politique, par exemple « esp32_mqtt_proxy_iot_policy ».
6. Dans la section Ajouter des instructions, choisissez Mode avancé. Copiez et collez le code JSON suivant dans la fenêtre de l'éditeur de stratégie. `aws-account-id`Remplacez-le par votre numéro de compte et `aws-region` par votre région (par exemple, « 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. Choisissez Créer.

Créez un AWS IoT thing

1. Connectez-vous à la console AWS IoT.
2. Dans le volet de navigation de gauche, choisissez Gérer, puis Objets.
3. Dans le coin supérieur droit, choisissez Create. Si aucun élément n'est enregistré dans votre compte, le message « Vous n'avez rien encore » s'affiche et vous pouvez choisir Enregistrer un objet.
4. Sur la page Création d' AWS IoT objets, choisissez Créer un objet unique.
5. Sur la page Ajouter votre appareil au registre des objets, saisissez un nom pour votre objet (par exemple, « esp32-ble »). Seuls les caractères alphanumériques, les tirets (-) et les traits de soulignement (_) sont autorisés. Choisissez Suivant.
6. Sur la page Ajouter un certificat pour votre objet, sous Ignorer le certificat et créer un objet, choisissez Créer un objet sans certificat. Comme nous utilisons l'application mobile proxy BLE qui utilise un identifiant Amazon Cognito pour l'authentification et l'autorisation, aucun certificat d'appareil n'est requis.

Création d'un client d'application Amazon Cognito

1. Connectez-vous à la console Amazon Cognito.
2. Dans le bandeau de navigation en haut à droite, choisissez Créer un groupe d'utilisateurs.
3. Entrez le nom du pool (par exemple, « esp32_mqtt_proxy_user_pool »).
4. Sélectionnez Review defaults.
5. Dans App Clients, choisissez Ajouter un client d'application, puis choisissez Ajouter un client d'application.
6. Entrez le nom du client de l'application (par exemple « mqtt_app_client »).
7. Assurez-vous que l'option Générer le secret du client est sélectionnée.
8. Choisissez Créer un client d'application.
9. Sélectionnez Revenir aux détails du groupe.
10. Sur la page de révision du groupe d'utilisateurs, choisissez Create pool. Vous devriez voir un message indiquant « Votre groupe d'utilisateurs a été créé avec succès ». Notez l'ID du pool.
11. Dans le volet de navigation, sélectionnez App clients.
12. Choisissez Afficher les détails. Notez l'ID du client de l'application et le secret du client de l'application.

Créer un groupe d'identités Amazon Cognito

1. Connectez-vous à la console Amazon Cognito.
2. Sélectionnez Create new identity pool.
3. Entrez un nom pour le pool d'identités (par exemple, « mqtt_proxy_identity_pool »).
4. Développez les fournisseurs d'authentification.
5. Choisissez l'onglet Cognito.
6. Entrez l'ID du groupe d'utilisateurs et l'ID du client de l'application que vous avez indiqués lors des étapes précédentes.
7. Sélectionnez Créer une réserve.
8. Sur la page suivante, pour créer de nouveaux rôles pour les identités authentifiées et non authentifiées, choisissez Autoriser.
9. Notez l'ID du pool d'identités, qui est au format`us-east-1:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`.

Associer une politique IAM à l'identité authentifiée

1. Ouvrez la console Amazon Cognito.
2. Sélectionnez le pool d'identités que vous venez de créer (par exemple, « mqtt_proxy_identity_pool »).
3. Choisissez Edit identity groupe (Modifier le groupe d'identités).
4. Notez le rôle IAM attribué au rôle authentifié (par exemple, Cognito_mqtt_proxy_identity_poolAuth_Role « »).
5. Ouvrez la console IAM.
6. Dans le panneau de navigation, choisissez Rôles.
7. Recherchez le rôle attribué (par exemple, Cognito_mqtt_proxy_identity_poolAuth_Role « »), puis sélectionnez-le.
8. Choisissez Ajouter une politique intégrée, puis choisissez JSON.
9. Saisissez la politique suivante :

   JSON

```
{
        "Version":"2012-10-17",
```

```
    "Statement": [
    {
       "Effect": "Allow",
       "Action": [
          "iot:AttachPolicy",
          "iot:AttachPrincipalPolicy",
          "iot:Connect",
          "iot:Publish",
          "iot:Subscribe"
       ],
       "Resource": "*"
    }]
}
```

10. Choisissez Review Policy (Examiner une stratégie).
11. Entrez un nom de politique (par exemple, « mqtt ProxyCognitoPolicy »).
12. Choisissez Create Policy (Créer une politique).

Étape 4 : Configuration d'Amazon FreeRTOS

1. Téléchargez la dernière version du code Amazon FreeRTOS depuis le dépôt FreeRTOS. GitHub
2. Pour activer la démo de mise à jour OTA, suivez les étapes décrites dansCommencer à utiliser l'Espressif ESP32-DevKitC et le ESP-WROVER-KIT.
3. Apportez ces modifications supplémentaires dans les fichiers suivants :

   a. Ouvrez `vendors/espressif/boards/esp32/aws_demos/config_files/aws_demo_config.h` et définissez`CONFIG_OTA_UPDATE_DEMO_ENABLED`.

   b. Ouvrez `vendors/espressif/boards/esp32/aws_demos/common/config_files/aws_demo_config.h` et remplacez `democonfigNETWORK_TYPES` par`AWSIOT_NETWORK_TYPE_BLE`.

   c. Ouvrez `demos/include/aws_clientcredential.h` et entrez l'URL de votre point de terminaison pour`clientcredentialMQTT_BROKER_ENDPOINT`.

      Entrez le nom de votre objet pour `clientcredentialIOT_THING_NAME` (par exemple, « esp32-ble »). Il n'est pas nécessaire d'ajouter des certificats lorsque vous utilisez les informations d'identification Amazon Cognito.

d. Ouvrir`vendors/espressif/boards/esp32/aws_demos/config_files/aws_iot_network_config.h`, modifier `configSUPPORTED_NETWORKS` et `configENABLED_NETWORKS` inclure uniquement`AWSIOT_NETWORK_TYPE_BLE`.

e. Ouvrez le `vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/ota_demo_config.h` fichier et entrez votre certificat.

```
#define otapalconfigCODE_SIGNING_CERTIFICATE [] = "your-certificate-key";
```

L'application doit démarrer et imprimer la version de démonstration :

```
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...
```

Étape 5 : configurer une application Android

1. Téléchargez le SDK Bluetooth Low Energy pour Android et un exemple d'application depuis le référentiel GitHub amazon-freertos-ble-android-sdk.
2. Ouvrez le fichier `app/src/main/res/raw/awsconfiguration.json` et renseignez l'ID du pool AppClientId, la région et AppClientSecret suivez les instructions de l'exemple JSON suivant.

```
{
  "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. Ouvrez `app/src/main/java/software/amazon/freertos/DemoConstants.java` et entrez le nom de la politique que vous avez créé précédemment (par exemple,*esp32_mqtt_proxy_iot_policy*) ainsi que la région (par exemple,*us-east-1*).
4. Créez et installez l'application de démonstration.
   a. Dans Android Studio, choisissez Build, puis Make Module app.
   b. Choisissez Exécuter, puis Exécuter l'application. Vous pouvez accéder au volet de la fenêtre Logcat d'Android Studio pour surveiller les messages du journal.
   c. Sur l'appareil Android, créez un compte depuis l'écran de connexion.
   d. Créez un utilisateur. Si un utilisateur existe déjà, entrez ses informations d'identification.
   e. Autorisez la démo Amazon FreeRTOS à accéder à la position de l'appareil.
   f. Recherchez les appareils Bluetooth Low Energy.
   g. Déplacez le curseur de l'appareil trouvé sur Activé.
   h. Appuyez sur y sur la console de débogage du port série de l'ESP32.
   i. Choisissez Pair & Connect.
5. Le plus... le lien devient actif une fois la connexion établie. L'état de connexion doit passer à « BLE_CONNECTED » dans le logcat de l'appareil Android lorsque la connexion est terminée :

```
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. Avant que les messages puissent être transmis, l'appareil Amazon FreeRTOS et l'appareil Android négocient le MTU. Vous devriez voir le résultat suivant dans logcat :

```
2019-06-06 20:11:46.720 23484-23497/software.amazon.freertos.demo I/FRD:
 onMTUChanged : 512 status: Success
```

7. L'appareil se connecte à l'application et commence à envoyer des messages MQTT à l'aide du proxy MQTT. Pour confirmer que le périphérique peut communiquer, assurez-vous que la valeur des données caractéristiques MQTT_CONTROL passe à 01 :

```
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. Lorsque les appareils sont couplés, un message s'affiche sur la console ESP32. Pour activer BLE, appuyez sur y. La démo ne fonctionnera pas tant que vous n'aurez pas effectué cette étape.

```
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.
```

Étape 6 : Exécuter le script de mise à jour OTA

1. Pour installer les prérequis, exécutez les commandes suivantes :

```
pip3 install boto3
```

```
pip3 install pathlib
```

2. Incrémentez la version de l'application FreeRTOS en. `demos/include/aws_application_version.h`
3. Créez un nouveau fichier .bin.
4. Téléchargez le script python start_ota.py. Pour consulter le contenu de l'aide du script, exécutez la commande suivante dans une fenêtre de terminal :

```
python3 start_ota.py -h
```

Vous devriez voir quelque chose comme ceci :

```
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. Si vous avez utilisé le CloudFormation modèle fourni pour créer des ressources, exécutez la commande suivante :

```
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
```

Vous devriez voir la mise à jour démarrer dans la console de débogage de l'ESP32 :

```
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. Lorsque la mise à jour OTA est terminée, l'appareil redémarre conformément au processus de mise à jour OTA. Il essaie ensuite de se connecter à l'aide du firmware mis à jour. Si la mise à niveau est réussie, le microprogramme mis à jour est marqué comme actif et vous devriez voir la version mise à jour dans la console :

```
13 13498 [iot_thread] OTA demo version 0.9.21
```

AWS IoT Application de démonstration Device Shadow

> **Important**
>
> Cette démo est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

Introduction

Cette démo montre comment utiliser la bibliothèque AWS IoT Device Shadow pour se connecter au service AWS Device Shadow. Il utilise le bibliothèque CoreMQTT pour établir une connexion MQTT avec TLS (authentification mutuelle) avec le courtier AWS IoT MQTT et l'analyseur de bibliothèque CoreJSON pour analyser les documents cachés reçus du service Shadow. AWS La démo montre les opérations de base du fantôme, telles que la mise à jour d'un document fantôme et la procédure de suppression d'un document fantôme. La démo montre également comment enregistrer une fonction de rappel dans la bibliothèque CoreMQTT pour gérer des messages tels que le shadow `/update` et les `/update/delta` messages envoyés depuis le service Device Shadow AWS IoT .

Cette démonstration est conçue comme un exercice d'apprentissage uniquement car la demande de mise à jour du document fantôme (état) et la réponse de mise à jour sont effectuées par la même application. Dans un scénario de production réaliste, une application externe demanderait une mise à jour de l'état de l'appareil à distance, même si celui-ci n'est pas connecté actuellement. L'appareil accusera réception de la demande de mise à jour lorsqu'il sera connecté.

> **Note**
>
> Pour configurer et exécuter les démos de FreeRTOS, suivez les étapes décrites dans. Commencez avec FreeRTOS

Fonctionnalité

La démonstration crée une tâche d'application unique qui passe en revue un ensemble d'exemples illustrant le shadow `/update` et les `/update/delta` rappels pour simuler le basculement de l'état d'un appareil distant. Il envoie une mise à jour instantanée avec le nouvel `desired` état et attend que l'appareil change d'`reported`état en réponse au nouvel `desired` état. De plus, un `/update` rappel des ombres est utilisé pour imprimer les états changeants des ombres. Cette démonstration utilise également une connexion MQTT sécurisée avec le courtier AWS IoT MQTT et suppose qu'il existe un `power0n` état dans l'ombre du périphérique.

La démo effectue les opérations suivantes :

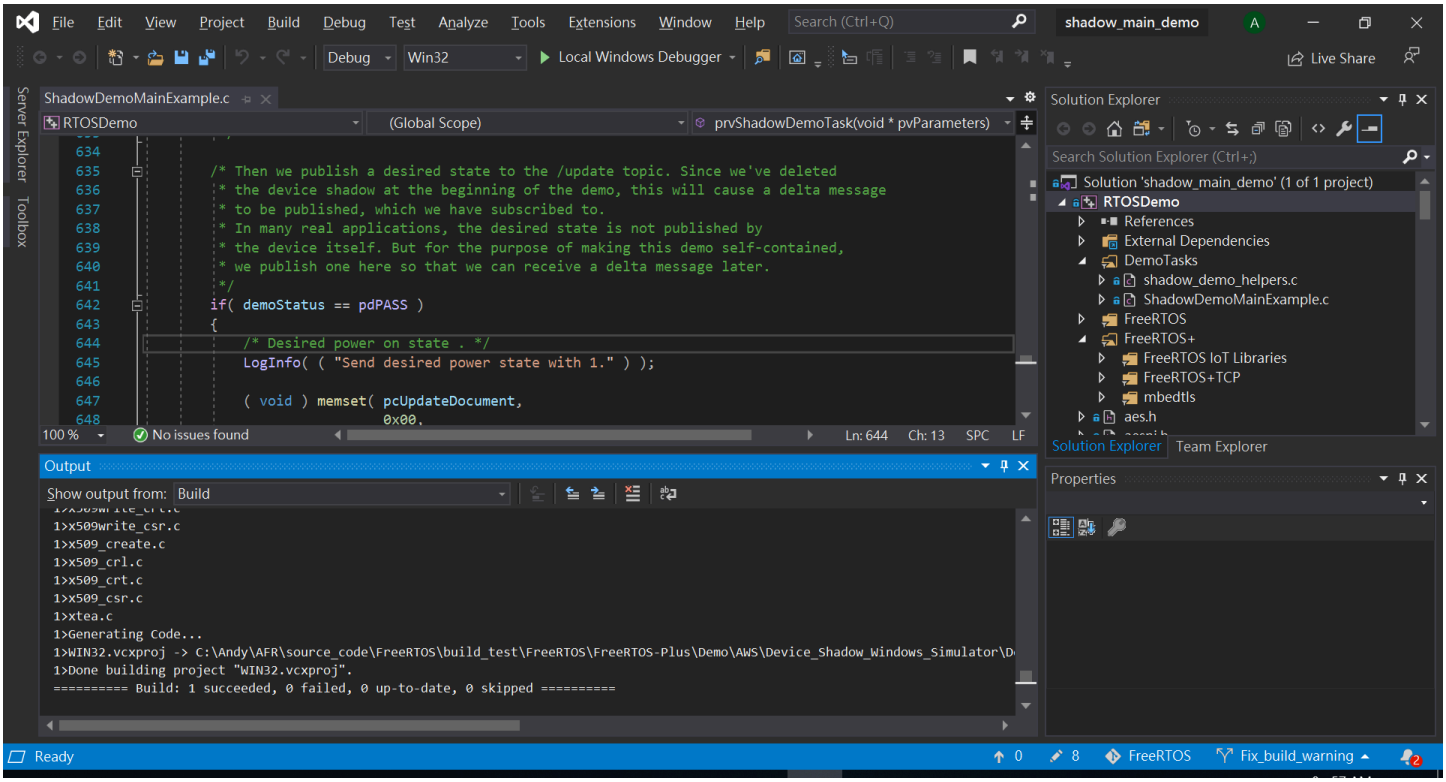
1. Établissez une connexion MQTT à l'aide des fonctions d'assistance dans. `shadow_demo_helpers.c`
2. Assemblez des chaînes thématiques MQTT pour les opérations d'ombrage des appareils, à l'aide de macros définies par la bibliothèque AWS IoT Device Shadow.

3. Publiez dans la rubrique MQTT utilisée pour supprimer une ombre d'appareil afin de supprimer toute ombre de périphérique existante.
4. Abonnez-vous aux rubriques MQTT relatives aux fonctions `/update/delta` d'assistance `/update/accepted` et `/update/rejected` à leur utilisation dans. `shadow_demo_helpers.c`
5. Publiez l'état souhaité d'`power0`nutilisation des fonctions d'assistance dans`shadow_demo_helpers.c`. Cela provoquera l'envoi d'un `/update/delta` message à l'appareil.
6. Gérez les messages MQTT entrants et déterminez s'ils sont liés à l'ombre du périphérique à l'aide d'une fonction définie par la bibliothèque AWS IoT Device Shadow (`Shadow_MatchTopic`). `prvEventCallback` S'il s'agit d'un `/update/delta` message invisible sur l'appareil, la fonction de démonstration principale publiera un deuxième message pour mettre à jour l'état signalé`power0`n. Si un `/update/accepted` message est reçu, vérifiez qu'il contient le même `clientToken` que celui publié précédemment dans le message de mise à jour. Cela marquera la fin de la démo.

```
82 9136 [ShadowDemo] [INFO] [SHADOW] [prvShadowDemoTask:641] 83 9136 [ShadowDemo] Send desired power state with 1.84 9136 [ShadowDemo]
85 9296 [ShadowDemo] [INFO] [SHADOW] [prvEventCallback:482] 86 9296 [ShadowDemo] pPublishInfo->pTopicName:$aws/things/testClient16:34:41/shadow/update/delta.87 9296 [Shad
owDemo]
88 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateDeltaHandler:256] 89 9296 [ShadowDemo] /update/delta json payload:{"version":1,"timestamp":1602751002,"state":{"powerOn":1}
,"metadata":{"powerOn":{"timestamp":1602751002}},"clientToken":"009136"}.90 9296 [ShadowDemo]
91 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateDeltaHandler:298] 92 9296 [ShadowDemo] version: 193 9296 [ShadowDemo]
94 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateDeltaHandler:308] 95 9296 [ShadowDemo] version:1, ulCurrentVersion:0
96 9296 [ShadowDemo]
97 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateDeltaHandler:342] 98 9296 [ShadowDemo] The new power on state newState:1, ulCurrentPowerOnState:0
99 9296 [ShadowDemo]
100 9296 [ShadowDemo] [INFO] [SHADOW] [prvEventCallback:482] 101 9296 [ShadowDemo] pPublishInfo->pTopicName:$aws/things/testClient16:34:41/shadow/update/accepted.102 9296
 [ShadowDemo]
103 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:376] 104 9296 [ShadowDemo] /update/accepted json payload:{"state":{"desired":{"powerOn":1}},"metadata":{"d
esired":{"powerOn":{"timestamp":1602751002}}},"version":1,"timestamp":1602751002,"clientToken":"009136"}.105 9296 [ShadowDemo]
106 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:424] 107 9296 [ShadowDemo] clientToken: 009136108 9296 [ShadowDemo]
109 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:429] 110 9296 [ShadowDemo] receivedToken:9136, clientToken:0
111 9296 [ShadowDemo]
112 9296 [ShadowDemo] [WARN] [SHADOW] [prvUpdateAcceptedHandler:442] 113 9296 [ShadowDemo] The received clientToken=9136 is not identical with the one=0 we sent 114 9296
[ShadowDemo]
115 9696 [ShadowDemo] [INFO] [SHADOW] [prvShadowDemoTask:670] 116 9696 [ShadowDemo] Report to the state change: 1117 9696 [ShadowDemo]
118 9856 [ShadowDemo] [INFO] [SHADOW] [prvEventCallback:482] 119 9856 [ShadowDemo] pPublishInfo->pTopicName:$aws/things/testClient16:34:41/shadow/update/accepted.120 9856
 [ShadowDemo]
121 9856 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:376] 122 9856 [ShadowDemo] /update/accepted json payload:{"state":{"reported":{"powerOn":1}},"metadata":{"
reported":{"powerOn":{"timestamp":1602751003}}},"version":2,"timestamp":1602751003,"clientToken":"009696"}.123 9856 [ShadowDemo]
124 9856 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:424] 125 9856 [ShadowDemo] clientToken: 009696126 9856 [ShadowDemo]
127 9856 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:429] 128 9856 [ShadowDemo] receivedToken:9696, clientToken:9696
129 9856 [ShadowDemo]
130 9856 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:437] 131 9856 [ShadowDemo] Received response from the device shadow. Previously published update with clie
ntToken=9696 has been accepted. 132 9856 [ShadowDemo]
133 10256 [ShadowDemo] [INFO] [SHADOW] [prvShadowDemoTask:698] 134 10256 [ShadowDemo] Start to unsubscribe shadow topics and disconnect from MQTT.
135 10256 [ShadowDemo]
136 12036 [ShadowDemo] [INFO] [SHADOW] [prvShadowDemoTask:747] 137 12036 [ShadowDemo] Demo completed successfully.138 12036 [ShadowDemo]
139 12036 [ShadowDemo] [INFO] [SHADOW] [prvShadowDemoTask:756] 140 12036 [ShadowDemo] Deleting Shadow Demo task.141 12036 [ShadowDemo]
```

La démo se trouve dans le fichier *`freertos`*`/demos/device_shadow_for_aws/shadow_demo_main.c` ou sur GitHub.

La capture d'écran suivante montre le résultat attendu en cas de réussite de la démonstration.



Connectez-vous au AWS IoT courtier MQTT

Pour nous connecter au broker AWS IoT MQTT, nous utilisons la même méthode que `MQTT_Connect()` dans leDémo de l'authentification mutuelle CoreMQTT.

Supprimer le document fantôme

Pour supprimer le document fantôme, appelez `xPublishToTopic` avec un message vide, en utilisant les macros définies par la bibliothèque AWS IoT Device Shadow. Cela permet `MQTT_Publish` de publier sur le `/delete` sujet. La section de code suivante montre comment cela est effectué dans la fonction`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 );
```

Abonnez-vous aux sujets parallèles

Abonnez-vous aux rubriques Device Shadow pour recevoir des notifications du AWS IoT broker concernant les modifications apportées aux Shadow. Les rubriques Device Shadow sont assemblées par des macros définies dans la bibliothèque Device Shadow. La section de code suivante montre comment cela est effectué dans la `prvShadowDemoTask` fonction.

```
/* 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 ) );
}
```

Envoyer des mises à jour sur Sh

Pour envoyer une mise à jour instantanée, la démo appelle `xPublishToTopic` avec un message au format JSON, à l'aide de macros définies par la bibliothèque Device Shadow. Cela permet `MQTT_Publish` de publier sur le `/delete` sujet. La section de code suivante montre comment cela est effectué dans la `prvShadowDemoTask` fonction.

```
#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 ) );
```

Gérer les messages Shadow Delta et les messages Shadow Update

La fonction de rappel utilisateur, qui a été enregistrée dans la bibliothèque cliente CoreMQTT à l'aide de cette `MQTT_Init` fonction, nous informera d'un événement de paquet entrant. Voir la fonction de rappel prv EventCallback on. GitHub

La fonction de rappel confirme le type `MQTT_PACKET_TYPE_PUBLISH` du paquet entrant et utilise l'API Device Shadow Library `Shadow_MatchTopic` pour confirmer que le message entrant est un message fantôme.

Si le message entrant est un message fantôme de type`ShadowMessageTypeUpdateDelta`, nous appelons prv UpdateDeltaHandler pour gérer ce message. Le gestionnaire `prvUpdateDeltaHandler` utilise la bibliothèque CoreJSON pour analyser le message afin d'obtenir la valeur delta de l'`powerOn`état et la compare à l'état actuel du périphérique maintenu localement. S'ils sont différents, l'état du périphérique local est mis à jour pour refléter la nouvelle valeur de l'`powerOn`état figurant dans le document fantôme.

Si le message entrant est un message fantôme de type`ShadowMessageTypeUpdateAccepted`, nous appelons prv UpdateAcceptedHandler pour gérer ce message. Le gestionnaire `prvUpdateAcceptedHandler` analyse le message à l'aide de la bibliothèque CoreJSON pour en extraire le `clientToken` contenu. Cette fonction de gestion vérifie que le jeton client du message JSON correspond au jeton client utilisé par l'application. S'il ne correspond pas, la fonction enregistre un message d'avertissement.

Démonstration du client Secure Sockets Echo

> **Important**
>
> Cette démo est hébergée sur le Amazon-FreeRTOS référentiel qui est obsolète. Nous vous recommandons de commencer ici lorsque vous créez un nouveau projet. Si vous possédez déjà un projet FreeRTOS basé sur le référentiel désormais Amazon-FreeRTOS obsolète, consultez le. Amazon-FreeRTOS Guide de migration du référentiel Github

L'exemple suivant utilise une seule tâche RTOS. Le code source de cet exemple se trouve sur `demos/tcp/aws_tcp_echo_client_single_task.c`.

Avant de commencer, vérifiez que vous avez téléchargé FreeRTOS sur votre microcontrôleur et que vous avez créé et exécuté les projets de démonstration de FreeRTOS. Vous pouvez cloner ou télécharger FreeRTOS depuis. GitHub Consultez le README.mdfichier pour obtenir des instructions.

Pour exécuter la démonstration

> **Note**
>
> Pour configurer et exécuter les démos de FreeRTOS, suivez les étapes décrites dans. Commencez avec FreeRTOS
> Les démonstrations de serveur et client TCP ne sont pas prises en charge actuellement sur le Cypress CYW954907AEVAL1F et les kits de développement CYW943907AEVAL1F.

1. Suivez les instructions de la section Configuration du serveur TLS Echo dans le guide de portage de FreeRTOS.

   Un TLS Echo Server doit être en cours d'exécution et à l'écoute sur le port 9000.

   Pendant la configuration, vous devez avoir généré quatre fichiers :

   - `client.pem` (certificat de client)
   - `client.key` (clé privée du client)
   - `server.pem` (certificat de serveur)
   - `server.key` (clé privée du serveur)

2. Utilisez l'outil `tools/certificate_configuration/CertificateConfigurator.html` pour copier le certificat client (`client.pem`) et la clé privée du client (`client.key`) dans `aws_clientcredential_keys.h`.
3. Ouvrez le fichier `FreeRTOSConfig.h`.
4. Définissez les variables `configECHO_SERVER_ADDR0`, `configECHO_SERVER_ADDR1`, `configECHO_SERVER_ADDR2` et `configECHO_SERVER_ADDR3` pour les quatre nombres entiers qui constituent l'adresse IP à laquelle l'instance TLS Echo Server est exécutée.
5. Définissez la variable `configTCP_ECHO_CLIENT_PORT` sur `9000`, qui correspond au port où l'instance TLS Echo Server est à l'écoute.
6. Définissez la variable `configTCP_ECHO_TASKS_SINGLE_TASK_TLS_ENABLED` sur `1`.
7. Utilisez l'outil `tools/certificate_configuration/PEMfileToCString.html` pour copier le certificat de serveur (`server.pem`) dans `cTlsECHO_SERVER_CERTIFICATE_PEM` dans le fichier `aws_tcp_echo_client_single_task.c`.
8. Ouvrez*`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/aws_demo_config.h`, commentez `#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED` et définissez `CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` ou`CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`.

Le microcontrôleur et l'instance TLS Echo Server doivent être sur le même réseau. Lorsque la démonstration démarre (`main.c`), vous devez voir un message de journal qui indique `Received correct string from echo server`.

Les traductions sont fournies par des outils de traduction automatique. En cas de conflit entre le contenu d'une traduction et celui de la version originale en anglais, la version anglaise prévaudra.