aws

사용자 가이드

FreeRTOS



Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

FreeRTOS: 사용자 가이드

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon의 상표 및 트레이드 드레스는 Amazon 외 제품 또는 서비스와 함께, Amazon 브랜드 이미지를 떨어뜨리거나 고객에게 혼동을 일으킬 수 있는 방식으로 사용할 수 없습니다. Amazon이 소유하지 않은 기타 모든 상표는 Amazon과 제휴 관계이거나 관련이 있거나 후원 관계와 관계없이 해당 소유자의 자산입니다.

Table of Contents

FreerTOS란 무엇인가요? 1
FreeRTOS 소스 코드 다운로드 1
FreeRTOS 버전 관리 1
FreerTOS 아키텍처 2
FreeRTOS 적격 하드웨어 플랫폼 2
개발 워크플로우 3
추가 리소스 4
FreeRTOS 커널 기초 5
FreeRTOS 커널 스케줄러 5
메모리 관리 6
커널 메모리 할당 6
애플리케이션 메모리 관리 6
작업 간 조정 7
대기열 7
세마포어 및 뮤텍스 8
DTT 알림 8
스트림 버퍼 8
메시지 버퍼 10
소프트웨어 타이머 11
저전력 지원 12
FreeRTOSConfig.h 12
AWS IoT 임베디드 C용 디바이스 SDK 13
FreeRTOS 시작하기 14
FreeRTOS 데모 애플리케이션 14
첫 번째 단계 15
AWS 계정 및 권한 설정 16
에 MCU 보드 등록 AWS IoT 18
FreeRTOS 다운로드 21
FreeRTOS 데모 구성 21
문제 해결 23
일반 시작하기 문제 해결 팁 24
터미널 에뮬레이터 설치 24
FreeRTOS에서 CMake 사용 25
사전 조건 26

타사 코드 편집기 및 디버깅 도구를 사용하여 FreeRTOS 애플리케이션 개발 27
CMake를 사용하여 FreeRTOS 빌드 28
개발자 모드 키 프로비저닝 34
소개 35
옵션 #1:에서 프라이빗 키 가져오기 AWS IoT 35
옵션 #2: 온보드 프라이빗 키 생성 35
보드별 시작 안내서 37
Cypress CYW943907AEVAL1F 개발 키트 38
Cypress CYW954907AEVAL1F 개발 키트 43
Cypress CY8CKIT-064S0S2-4343W 48
Windows 시뮬레이터를 사용하는 Microchip ATECC608A 보안 요소 54
Espressif ESP32-DevKitC 및 ESP-WROVER-KIT 61
Espressif ESP32-WROOM-32SE 103
Infineon XMC4800 IoT 연결 키트 119
Infineon OPTIGA Trust X 및 XMC4800 IoT 연결 키트 125
Marvell MW32x AWS IoT 스타터 키트 132
MediaTek MT7697Hx 개발 키트 154
Microchip Curiosity PIC32MZ EF 159
Nordic nRF52840-DK 165
Nuvoton NuMaker-IoT-M487 170
NXP LPC54018 IoT Module 178
Renesas Starter Kit+ for RX65N-2MB 183
STMicroelectronics STM32L4 Discovery Kit IoT 노드 188
Texas Instruments CC3220SF-LAUNCHXL 191
Windows Device Simulator 196
Xilinx Avnet MicroZed Industrial IoT Kit 201
OTA(Over-the-Air) 업데이트 210
OTA 리소스에 태그 지정 211
OTA 업데이트 사전 조건 211
업데이트를 저장할 Amazon S3 버킷 생성 211
OTA 업데이트 서비스 역할 생성 212
OTA 사용자 정책 생성 215
코드 서명 인증서 생성 216
에 대한 코드 서명에 대한 액세스 권한 부여 AWS IoT 226
OTA 라이브러리와 함께 FreeRTOS 다운로드 227
MQTT를 사용한 OTA 업데이트 사전 조건 227

HTTP를 사용한 OTA 업데이트 사전 조건 231
OTA 자습서 236
초기 펌웨어 설치 237
펌웨어 버전 업데이트 250
OTA 업데이트 생성(AWS IoT 콘솔) 251
를 사용하여 OTA 업데이트 생성 AWS CLI 255
OTA 업데이트 관리자 서비스 279
애플리케이션에 OTA 에이전트 통합 281
연결 관리 281
간단한 OTA 데모 282
OTA 완료 이벤트에 대해 사용자 지정 콜백 사용 285
OTA 보안 286
에 대한 코드 서명 AWS IoT 287
OTA 문제 해결 288
OTA 업데이트를 위한 Cloudwatch Logs 설정 288
를 사용하여 AWS IoT OTA API 호출 로깅 AWS CloudTrail 293
를 사용하여 CreateOTAUpdate 실패 세부 정보 가져오기 AWS CLI 297
를 사용하여 OTA 실패 코드 가져오기 AWS CLI 299
여러 디바이스의 OTA 업데이트 문제 해결 300
Texas Instruments CC3220SF Launchpad를 사용하여 OTA 업데이트 문제 해결 300
FreeRTOS 라이브러리 302
FreeRTOS 이식 라이브러리 302
FreeRTOS 애플리케이션 라이브러리 304
FreERTOS 공용 라이브러리 306
FreeRTOS 라이브러리 구성 306
공통 라이브러리 307
원자성 작업 307
선형 컨테이너 308
로깅 309
정적 메모리 309
작업 풀 310
backoffAlgorithm 314
소개 314
Bluetooth Low Energy 315
개요 315
아키텍처 316

종속성 및 요구 사항 318
라이브러리 구성 파일 318
최적화 319
사용 제한 319
Initialization(초기화) 320
API 참조 322
사용 예 322
이식 326
FreeRTOS Bluetooth 디바이스용 Mobile SDK 328
공통 I/O 330
AWS IoT Device Defender 331
소개 331
AWS IoT Greengrass 332
개요 332
종속성 및 요구 사항 332
API 참조 333
사용 예 333
coreHTTP 335
소개 335
coreJSON 336
소개 336
메모리 사용 336
coreMQTT 337
소개 337
OTA 에이전트 338
개요 338
Features 339
API 참조 340
사용 예 340
이식 340
메모리 사용 341
corePKCS11 341
개요 341
Features 342
비대칭 암호화 지원 343
이식 345

메모리 사용 346
보안 소켓 346
개요 346
종속성 및 요구 사항 347
Features 347
문제 해결 348
개발자 지원 348
사용 제한 348
Initialization(초기화) 348
API 참조 349
사용 예 349
이식 351
AWS IoT 디바이스 섀도우 351
소개 352
AWS IoT 작업 352
소개 352
전송 계층 보안 353
Wi-Fi 354
개요 354
종속성 및 요구 사항 354
Features 354
구성 356
Initialization(초기화) 356
API 참조 357
사용 예 357
이식 360
FreeRTOS 데모 361
FreeRTOS 데모 실행 361
데모 구성 362
Bluetooth Low Energy 362
개요 362
사전 조건 363
공통 구성 요소 366
MQTT over Bluetooth Low Energy 373
Wi-Fi 프로비저닝 374
일반 속성 서버 380

Microchip Curiosity PIC32MZEF용 부트로더 382
부트로더 상태 383
플래시 디바이스 384
애플리케이션 이미지 구조 385
이미지 헤더 385
이미지 설명자 387
이미지 트레일러 388
부트로더 구성 389
부트로더 빌드 389
AWS IoT Device Defender 390
소개 390
기능 390
AWS IoT Greengrass 395
Amazon EC2 397
coreHTTP 399
coreHTTP 상호 인증 399
coreHTTP Amazon S3 업로드 401
coreHTTP S3 다운로드 404
coreHTTP 다중 스레드 407
AWS IoT 작업 411
소개 411
소스 코드 구성 412
AWS IoT MQTT 브로커 연결 구성 412
기능 412
coreMQTT 414
coreMQTT 상호 인증 414
coreMQTT 연결 공유 419
OTA(Over-the-Air) 업데이트 424
무선(OTA) 데모 구성 429
Texas Instruments CC3220SF-LAUNCHXL 430
Microchip Curiosity PIC32MZEF 432
Espressif ESP32 438
Renesas RX65N 438
AWS IoT 디바이스 섀도우 463
소개 463
기능 464

AWS IoT MQTT 브로커에 연결 466
섀도우 문서 삭제 466
섀도우 주제 구독 466
섀도우 업데이트 전송 467
섀도우 델타 메시지 및 섀도우 업데이트 메시지 처리 467
보안 소켓 468
FreeRTOS AWS IoT Device Tester 용 사용 470
지원되는 FreeRTOS용 IDT 버전 471
FreeRTOS용 IDT의 최신 버전 472
이전 IDT 버전 472
지원되지 않는 IDT 버전 475
사전 조건 478
FreeRTOS 다운로드 479
LTS 검증(LTS 라이브러리를 사용하는 FreeRTOS에 대한 검증) 479
FreeRTOS용 IDT 다운로드 479
..... 480
AWS IoT Device Tester 관리형 정책 481
(선택 사항) 설치 AWS Command Line Interface 482
처음으로 마이크로 컨트롤러 보드의 테스트 준비 482
라이브러리 이식 계층 추가 483
자격 AWS 증명 구성 483
FreeRTOS용 IDT에서 디바이스 풀 생성 483
빌드, 플래시 및 테스트 설정 구성 488
Bluetooth Low Energy 테스트 실행 501
사전 조건 501
Raspberry Pi 설정 501
FreeRTOS 파일 설정 506
BLE 테스트 실행 506
BLE 테스트 문제 해결 506
FreeRTOS 검증 테스트 제품군 실행 506
FreeRTOS용 IDT 명령 510
재검증을 위한 테스트 512
테스트 제품군 버전 512
결과 및 로그 이해 513
결과 보기 514
IDT를 사용하여 자체 테스트 제품군 개발 및 실행 518

최신 버전의 FreeRTOS용 IDT 다운로드 518
테스트 도구 모음 생성 워크플로 519
자습서: 샘플 IDT 테스트 제품군 빌드 및 실행 519
자습서: 간단한 IDT 테스트 제품군 개발 525
IDT 테스트 제품군 구성 파일 생성 534
IDT 상태 시스템 구성 541
IDT 테스트 사례 실행 파일 생성 564
IDT 컨텍스트 사용 571
테스트 실행기 설정 구성 576
사용자 지정 테스트 제품군 디버그 및 실행 587
IDT 테스트 결과 및 로그 검토 590
IDT 사용량 지표 596
문제 해결 602
디바이스 구성 문제 해결 604
제한 시간 오류 해결 613
셀룰러 기능 및 AWS 요금 613
검증 보고서 생성 정책 613
지원 정책 614
의 보안 AWS 615
ID 및 액세스 관리 615
대상 616
ID를 통한 인증 617
정책을 사용하여 액세스 관리 620
FreeRTOS에서 IAM을 사용하는 방법 622
자격 증명 기반 정책 예제 628
문제 해결 631
규정 준수 확인 632
복원성 634
인프라 보안 634
..... dcxxxv

FreerTOS란 무엇인가요?

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

15년간 세계 유수의 칩 회사와 협력하여 개발되어 현재 175초마다 다운로드되는 FreeRTOS는 마이크로컨트롤러 및 소형 마이크로프로세서를 위한 시장을 선도하는 실시간 운영 체제입니다. MIT 오픈 소스 라이선스에 따라 자유롭게 배포되는 FreeRTOS는 모든 산업 분야에서 사용하기에 적합한 커널과 증가하는 라이브러리 세트가 포함되어 있습니다. FreeRTOS는 신뢰성과 사용 편의성에 중점을 두고 빌드되었습니다.

FreeRTOS에는 연결, 보안 및 무선 업데이트(OTA)를 위한 라이브러리가 포함되어 있습니다. 또한 FreeRTOS에는 적격 보드에서 FreeRTOS 기능을 보여주는 데모 애플리케이션이 포함되어 있습니다.

FreeRTOS는 오픈 소스 프로젝트입니다. GitHub 사이트 https://github.com/aws/amazon-freertos에서 소스 코드를 다운로드하거나 변경 사항 또는 기능 향상에 기여하거나 문제를 보고할 수 있습니다. MIT 오픈 소스 라이선스 하에서 FreeRTOS 코드를 릴리스하므로, 상용 및 개인 프로젝트에서 코드를 사용할 수 있습니다.

또한 FreeRTOS 설명서(FreeRTOS 사용 설명서, FreeRTOS 이식 안내서 및 FreeRTOS 검증 안내서)에 대한 기여도 환영합니다. https://github.com/awsdocs/aws-freertos-docs에서 설명서에 대한 마크다운 소스를 사용할 수 있습니다. 이 소스는 Creative Commons(CC BY-ND) 라이선스 하에서 릴리스됩니다.

FreeRTOS 소스 코드 다운로드

GitHub에서 FreeRTOS를 복제하거나 다운로드할 수 있습니다. 자세한 내용은 README.md 파일을 참조하십시오.

FreeRTOS 버전 관리

FreeRTOS 커널 및 구성 요소는 개별적으로 릴리스되며 의미 체계 버전 관리를 사용합니다. 통합 FreeRTOS 릴리스는 주기적으로 이루어집니다. 모든 릴리스는 YYYYMM.NN 형식의 날짜 기반 버전 관리를 사용합니다. 여기서

- Y는 연도를 나타냅니다.
- M은 월을 나타냅니다.
- N은 지정된 월 내의 릴리스 순서를 나타냅니다(00은 첫 번째 릴리스).

예를 들어, 2021년 7월의 두 번째 릴리스는 202107.01이 됩니다.

이전에 FreeRTOS는 의미 체계 버전 관리를 메이저 릴리스에 사용했습니다. 날짜 기반 버전 관리로 이동했지만(FreeRTOS 1.4.8이 FreeRTOS AWS 참조 통합 201906.00로 업데이트됨), FreeRTOS 커널과 각 개별 FreeRTOS 라이브러리는 여전히 의미 체계 버전 관리를 유지합니다. 의미 체계 버전 관리에서는 버전 번호(X.Y.Z) 자체가 릴리스가 메이저 릴리스인지, 마이너 릴리스인지, 또는 포인트 릴리스인지를 나타냅니다. 라이브러리의 의미 체계 버전을 사용하여 새 릴리스의 범위와 애플리케이션에 미치는 새 릴리스의 영향을 평가할 수 있습니다.

LTS 릴리스는 다른 릴리스 유형과 다르게 유지 관리됩니다. 메이저 릴리스와 마이너 릴리스는 결함 해결뿐 아니라 새 기능으로 자주 업데이트됩니다. LTS 릴리스는 중요한 결함과 보안 취약성을 해결하기 위한 변경 사항으로만 업데이트됩니다. 시작 후 특정 LTS 릴리스에는 새로운 기능이 도입되지 않습니다. 이 유형의 릴리스는 릴리스 후 최소 3역년 동안 유지 관리 되며, 메이저 릴리스와 마이너 릴리스가 나타내는 동적 기준과는 대조적으로 안정적인 기준을 사용할 수 있는 옵션을 디바이스 제조업체에게 제공합니다.

FreerTOS 아키텍처

FreeRTOS는 일반적으로 디바이스 애플리케이션에 필요한 모든 구성 요소와 함께 디바이스의 플래시 메모리에 단일 컴파일 이미지로 저장됩니다. 이 이미지는 내장형 개발자에 의해 작성되는 애플리케이션을 위한 기능, Amazon에서 제공되는 소프트웨어 라이브러리, FreeRTOS 커널, 하드웨어 플랫폼용 드라이버와 BSP(Board Support Package) 등을 결합합니다. 사용 중인 개별 마이크로컨트롤러에 상관없이 내장형 애플리케이션 개발자는 FreeRTOS 커널과 모든 FreeRTOS 소프트웨어 라이브러리에 대해 동일한 표준 인터페이스를 기대할 수 있습니다.

FreeRTOS 적격 하드웨어 플랫폼

다음 하드웨어 플랫폼은 FreeRTOS에 대해 인증되었습니다.

- ATECC608A 용 제로 터치 프로비저닝 키트 AWS IoT
- Cypress CYW943907AEVAL1F 개발 키트
- Cypress CYW954907AEVAL1F 개발 키트

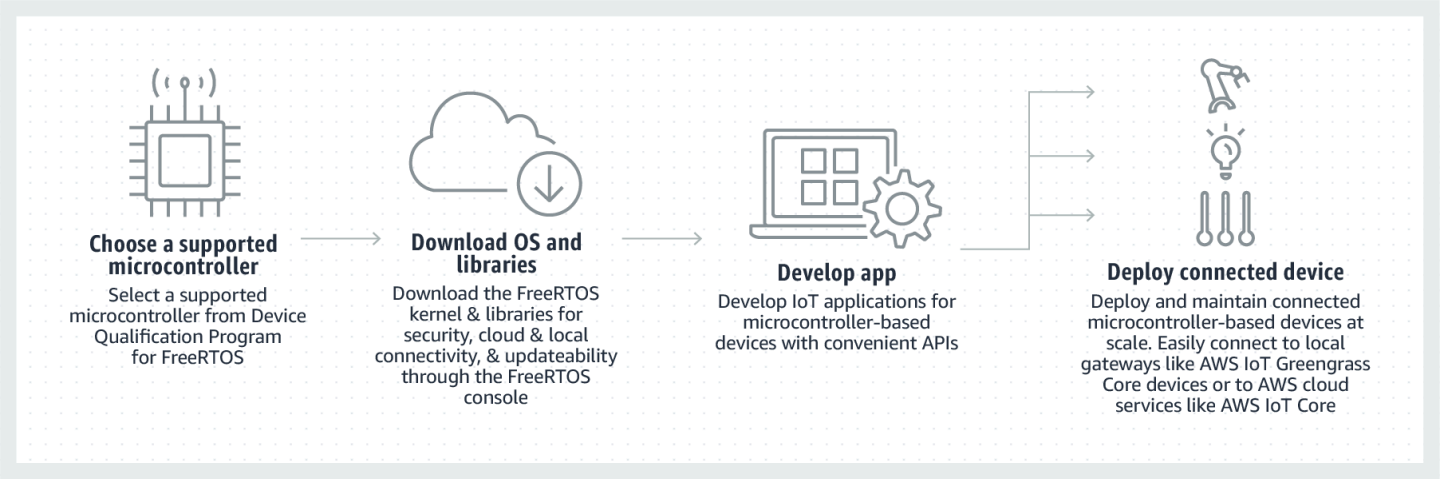
- Espressif ESP32-DevKitC
- Espressif ESP-WROVER-KIT
- Infineon XMC4800 IoT 연결 키트
- Marvell MW320 AWS IoT Starter 키트
- Marvell MW322 AWS IoT Starter 키트
- MediaTek MT7697Hx 개발 키트
- Microchip Curiosity PIC32MZEF 번들
- Nordic nRF52840-DK
- NuMaker-IoT-M487
- NXP LPC54018 IoT Module
- OPTIGA Trust X Security Solution
- Renesas RX65N RSK IoT Module
- STMicroelectronicsSTM32L4 Discovery Kit IoT 노드
- Texas Instruments CC3220SF-LAUNCHXL
- 최소 듀얼 코드 및 유선 이더넷 연결이 있는 Microsoft Windows 7 이상
- Xilinx Avnet MicroZed Industrial IoT Kit

적격 디바이스 목록은 AWS Partner Device Catalog에도 나와 있습니다.

새로운 디바이스 검증에 대한 자세한 내용은 FreeRTOS 검증 안내서를 참조하세요.

개발 워크플로우

FreeRTOS를 다운로드하여 개발을 시작합니다. 패키지의 압축을 풀고 IDE로 가져옵니다. 이제 선택한 하드웨어 플랫폼에서 애플리케이션을 개발하고, 해당 디바이스에 적합한 개발 프로세스를 사용하여 디바이스를 제작하고 배포할 수 있습니다. 배포된 디바이스는 AWS IoT 서비스에 연결하거나 AWS IoT Greengrass 전체 IoT 솔루션의 일부로 연결할 수 있습니다.



추가 리소스

다음은 유용한 리소스입니다.

- FreeRTOS v10.0.0 Reference Manual을 포함한 추가 FreeRTOS 문서는 freertos.org에서 사용할 수 있습니다.
- FreeRTOS 엔지니어링 팀에 FreeRTOS에 대한 질문이 있는 경우 FreeRTOS GitHub 페이지에서 문제를 개설할 수 있습니다.
- FreeRTOS에 대한 기술 관련 질문이 있는 경우 FreeRTOS Community Forums를 방문하세요.
- 디바이스를에 연결하는 방법에 대한 자세한 내용은 AWS IoT Core 개발자 안내서와 해당 안내서의 디바이스 프로비저닝 장을 AWS IoT참조하세요.
- 에 대한 기술 지원을 받으려면 AWS 지원 센터를 AWS방문하세요.
- AWS 결제, 계정 서비스, 이벤트, 남용 또는 기타 관련 문제에 대한 질문은 문의처 페이지를 AWS참조하세요. https://aws.amazon.com/contact-us/

FreeRTOS 커널 기초

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

FreeRTOS 커널은 다양한 아키텍처를 지원하는 실시간 운영 체제이며, 내장형 마이크로 컨트롤러 애플리케이션을 빌드하는 데 적합합니다. 이 커널은 다음을 제공합니다.

- 멀티태스킹 스케줄러
- 다중 메모리 할당 옵션(완전히 정적으로 할당되는 시스템 생성 기능 포함)
- 작업 간 조정 프리미티브(작업 알림, 메시지 대기열, 다양한 유형의 세마포어, 스트림 및 메시지 버퍼 포함)

FreeRTOS 커널에서는 중요 섹션 또는 인터럽트 내에서 연결된 목록 검색과 같은 비결정적 작업을 수행하지 않습니다. FreeRTOS 커널에는 타이머에 서비스가 필요한 경우에만 CPU 시간을 사용하는 효과적인 소프트웨어 타이머 구현이 포함되어 있습니다. 차단된 작업에는 시간이 많이 소요되는 주기적 서비스가 필요하지 않습니다. DTT(Direct-to-Task) 알림을 사용하면 RAM 오버헤드 없이 작업 신호를 빠르게 전송할 수 있습니다. 대부분의 작업 간 및 작업 중단 신호 전송 시나리오에서 DTT 알림을 사용할 수 있습니다.

FreeRTOS 커널은 작고 간단하고 사용하기 쉽게 설계되었습니다. 일반 RTOS 커널 이진 이미지는 4000~9000바이트 범위 내에 있습니다.

FreeRTOS 커널에 대한 최신 문서는 FreeRTOS.org를 방문하세요. FreeRTOS.org에서는 Quick Start Guide 및 상세한 Mastering the FreeRTOS Real Time Kernel을 비롯하여, FreeRTOS 커널 사용에 대한 자세한 자습서와 안내서를 다양하게 제공합니다.

FreeRTOS 커널 스케줄러

RTOS를 사용하는 내장형 애플리케이션을 독립된 작업 세트로 구성할 수 있습니다. 각 작업은 다른 작업에 종속되지 않고 자체 컨텍스트 내에서 실행됩니다. 애플리케이션에서는 한 번에 하나의 작업만 실행합니다. 실시간 RTOS 스케줄러에 따라 각 작업이 실행되는 시간이 결정됩니다. 각 작업에는 자체 스택이 제공됩니다. 다른 작업을 실행할 수 있도록 작업을 스왑하면 작업의 실행 컨텍스트가 작업 스택에 저장됩니다. 따라서 나중에 해당 작업을 재개하기 위해 다시 스왑하면 실행 컨텍스트가 복원됩니다.

결정적인 실시간 동작을 제공하려면 FreeRTOS 작업 스케줄러를 사용하여 작업에 엄격한 우선 순위를 할당할 수 있습니다. RTOS를 사용하면 실행 가능한 작업 중 우선 순위가 가장 높은 작업에 처리 시간이 제공됩니다. 따라서 우선 순위가 동일한 여러 작업이 동시에 실행 준비가 될 경우 작업 간에 처리 시간을 공유해야 합니다. 또한 FreeRTOS에서는 실행 준비된 다른 작업이 없는 경우에만 실행되는 유휴 작업을 생성합니다.

메모리 관리

이 단원에서는 커널 메모리 할당 및 애플리케이션 메모리 관리에 대한 정보를 제공합니다.

커널 메모리 할당

RTOS 커널에서는 작업, 대기열 또는 다른 RTOS 객체가 생성될 때마다 RAM이 필요합니다. RAM을 다음과 같이 할당할 수 있습니다.

- 컴파일 시간에 정적으로
- RTOS API 객체 생성 함수를 사용하여 RTOS 힙에서 동적으로

RTOS 객체를 동적으로 생성하는 경우 다음과 같은 여러 가지 이유로 인해 표준 C 라이브러리 `malloc()` 및 `free()` 함수를 사용하는 것이 적절하지 않은 경우도 있습니다.

- 내장형 시스템에서 사용할 수 없습니다.
- 중요한 코드 공간을 차지합니다.
- 일반적으로 스레드 세이프가 아닙니다.
- 결정적이지 않습니다.

따라서 FreeRTOS는 이동형 계층에 메모리 할당 API를 보관합니다. 이동형 계층은 코어 RTOS 기능을 구현하는 소스 파일의 외부에 있으므로, 개발 중인 실시간 시스템에 적절한 애플리케이션별 구현을 제공할 수 있습니다. RTOS 커널은 RAM이 필요한 경우 `malloc()` 대신 `pvPortMalloc()`를 호출합니다. RAM을 비우고 있는 경우 RTOS 커널은 `free()` 대신 `vPortFree()`를 호출합니다.

애플리케이션 메모리 관리

애플리케이션에 메모리가 필요한 경우 FreeRTOS 힙에서 메모리를 할당할 수 있습니다. FreeRTOS는 복잡성과 기능이 요구되는 다양한 힙 관리 스키마를 제공합니다. 사용자가 고유한 힙 구현을 제공할 수도 있습니다.

FreeRTOS 커널에는 5가지 힙 구현이 포함되어 있습니다.

`heap_1`

가장 간단한 구현입니다. 메모리를 비울 수 없습니다.

`heap_2`

메모리를 비울 수 있지만, 사용 가능한 인접 블록을 결합하지 않습니다.

`heap_3`

스레드 안전을 위해 표준 `malloc()` 및 `free()`를 래핑합니다.

`heap_4`

사용 가능한 인접 블록을 결합하여 조각화 현상을 방지합니다. 절대 주소 배치 옵션을 포함합니다.

`heap_5`

heap_4와 유사합니다. 힙이 인접하지 않은 여러 메모리 영역에 걸쳐서 존재할 수 있습니다.

작업 간 조정

이 단원에는 FreeRTOS 프리미티브에 대한 정보가 포함되어 있습니다.

대기열

대기열은 기본적인 형태의 작업 간 통신입니다. 대기열을 사용하여 작업 간이나 인터럽트와 작업 간에 메시지를 전송할 수 있습니다. 대부분의 경우 대기열은 스레드 세이프(thread-safe) FIFO(First In First Out) 버퍼로 사용되며, 새로운 데이터는 대기열의 뒤쪽으로 전송됩니다. 데이터가 대기열의 앞쪽으로 전송될 수도 있습니다. 메시지는 대기열을 통해 복사본으로 전송됩니다. 즉, 단순히 데이터에 대한 참조를 저장하지 않고 데이터(더 큰 버퍼에 대한 포인터일 수 있음)를 대기열에 복사합니다.

대기열 API는 블록 시간 지정을 허용합니다. 특정 작업이 빈 대기열에서 읽으려고 시도할 경우 대기열에서 데이터를 사용할 수 있거나 블록 시간이 경과할 때까지 해당 작업은 차단 상태로 전환됩니다. 다른 작업이 실행될 수 있도록 차단 상태인 작업은 CPU 시간을 사용하지 않습니다. 마찬가지로, 특정 작업이 전체 대기열에서 쓰려고 시도할 경우 대기열의 공간이 사용 가능해지거나 블록 시간이 경과할 때까지 해당 작업은 차단 상태로 전환됩니다. 동일한 대기열에 차단된 작업이 여러 개 있는 경우 우선 순위가 가장 높은 작업이 먼저 차단 해제됩니다.

다른 FreeRTOS 프리미티브(예: DTT(Direct-to-Task) 알림, 스트림 및 메시지 버퍼)는 다양한 일반 설계 시나리오에서 대기열에 대한 간단한 대안을 제공합니다.

세마포어 및 뮤텍스

FreeRTOS 커널은 상호 제외와 동기화의 목적으로 이진 세마포어, 계수 세마포어 및 뮤텍스를 제공합니다.

이진 세마포어는 두 개의 값만 가질 수 있습니다. 이 세마포어는 작업 간이나 작업과 인터럽트 간에 동기화를 구현하는 데 적합합니다. 계수 세마포어는 세 개 이상의 값을 가집니다. 계수 세마포어를 사용하면 여러 작업에서 리소스를 공유하거나 보다 복잡한 동기화 작업을 수행할 수 있습니다.

뮤텍스는 우선 순위 상속 메커니즘을 포함하는 이진 세마포어입니다. 즉, 우선 순위가 높은 작업이 현재 우선 순위가 낮은 작업에서 보유한 뮤텍스를 가져오려고 시도하는 동안 차단될 경우 토큰을 보유한 작업의 우선 순위가 차단된 작업의 우선 순위로 일시적으로 상승됩니다. 이 메커니즘은 우선 순위가 더 높은 작업이 차단 상태로 유지되는 시간을 최대한 단축하여 우선 순위 반전이 발생하는 것을 최소화하기 위해 설계되었습니다.

DTT 알림

작업 알림을 사용하면 세마포어와 같은 개별 통신 객체 없이 작업 간에 상호 작용하고, 인터럽트 서비스 루틴(ISR)과 동기화할 수 있습니다. 각 RTOS 작업에는 알림 내용(있는 경우)을 저장하는 데 사용되는 32비트 알림 값이 있습니다. RTOS 작업 알림은 수신 작업을 차단 해제하고 수신 작업의 알림 값을 선택적으로 업데이트할 수 있는 작업에 직접 전송되는 이벤트입니다.

RTOS 작업 알림을 이진 세마포어, 계수 세마포어 및 대기열(해당하는 경우)에 대한 더 빠르고 간단한 대안으로 사용할 수 있습니다. 작업 알림은 동일한 기능을 수행하는 데 사용될 수 있는 다른 FreeRTOS 기능에 비해 속도와 RAM 공간의 측면에서 이점이 있습니다. 하지만 작업 알림은 이벤트 수신자가 될 수 있는 작업이 하나뿐인 경우에만 사용될 수 있습니다.

스트림 버퍼

스트림 버퍼를 사용하면 바이트 스트림을 인터럽트 서비스 루틴에서 작업으로 또는 한 작업에서 다른 작업으로 전달할 수 있습니다. 바이트 스트림은 임의 길이를 가질 수 있으며 시작 또는 끝이 없어도 됩니다. 한 번에 쓰고 읽을 수 있는 바이트 수에 제한이 없습니다. 프로젝트에서 `stream_buffer.c` 소스 파일을 포함하여 스트림 버퍼 기능을 활성화합니다.

스트림 버퍼를 사용하면 버퍼에 쓰는 작업 또는 인터럽트 하나(라이터)와 버퍼에서 읽는 작업 또는 인터럽트 하나(리더)만 존재합니다. 라이터와 리더의 작업 또는 인터럽트 서비스 루틴이 달라도 되지만 여러 라이터 또는 리더가 존재하면 안 됩니다.

스트림 버퍼 구현에서는 DTT 알림을 사용합니다. 따라서 호출 작업을 차단 상태로 전환하는 스트림 버퍼 API를 호출하면 호출 작업의 알림 상태와 값이 변경될 수 있습니다.

데이터 전송

xStreamBufferSend()는 데이터를 작업 내 스트림 버퍼로 전송하는 데 사용되고, xStreamBufferSendFromISR()은 데이터를 인터럽트 서비스 루틴(ISR) 내 스트림 버퍼로 전송하는 데 사용됩니다.

xStreamBufferSend()는 블록 시간 지정을 허용합니다. xStreamBufferSend()가 스트림 버퍼에 쓰기 위해 0이 아닌 블록 시간으로 호출되고 버퍼가 꽉 찬 경우 공간을 사용할 수 있거나 블록 시간이 만료될 때까지 작업은 차단 상태로 전환됩니다.

sbSEND_COMPLETED() 및 sbSEND_COMPLETED_FROM_ISR()은 스트림 버퍼에 데이터를 쓸 때 FreeRTOS API에 의해 내부적으로 호출되는 매크로입니다. 이 매크로는 업데이트된 스트림 버퍼의 핸들을 사용합니다. 두 매크로는 모두 스트림 버퍼에 데이터를 대기 중인 차단된 작업이 있는지 확인한 후, 있으면 해당 작업을 차단 상태에서 제거합니다.

FreeRTOSConfig.h에서 sbSEND_COMPLETED()의 자체 구현을 제공하여 이 기본 동작을 변경할 수 있습니다. 이 기능은 스트림 버퍼를 사용하여 멀티 코어 프로세서의 코어 간에 데이터를 전달할 때 유용합니다. 그러한 시나리오에서는 sbSEND_COMPLETED()를 구현하여 다른 CPU 코어에서 인터럽트를 생성한 다음 인터럽트의 서비스 루틴에서 xStreamBufferSendCompletedFromISR() API를 사용하여 데이터를 대기 중인 작업이 있는지 확인하고 필요한 경우 해당 작업을 차단 해제할 수 있습니다.

데이터 수신

xStreamBufferReceive()는 작업 내 스트림 버퍼에서 데이터를 읽는 데 사용되고, xStreamBufferReceiveFromISR()은 인터럽트 서비스 루틴(ISR) 내 스트림 버퍼에서 데이터를 읽는 데 사용됩니다.

xStreamBufferReceive()는 블록 시간 지정을 허용합니다. xStreamBufferReceive()가 스트림 버퍼에서 읽기 위해 0이 아닌 블록 시간으로 호출되고 버퍼가 비어 있는 경우 스트림 버퍼에서 지정된 양의 데이터를 사용할 수 있거나 블록 시간이 만료될 때까지 작업은 차단 상태로 전환됩니다.

작업이 차단 해제되기 전에 스트림 버퍼에 있어야 하는 데이터의 양을 스트림 버퍼의 트리거 수준이라고 합니다. 트리거 수준 10으로 차단된 작업은 버퍼에 10바이트 이상을 쓰거나 작업의 블록 시간이 만료될 때 차단 해제됩니다. 트리거 수준에 도달하기 전에 읽기 작업의 블록 시간이 만료될 경우 해당 작업은 버퍼에 기록된 데이터를 수신합니다. 작업의 트리거 수준을 1과 스트림 버퍼 크기 사이의 값으로 설정해야 합니다. 스트림 버퍼의 트리거 수준은 xStreamBufferCreate()를 호출할 때 설정됩니다. xStreamBufferSetTriggerLevel()를 호출하여 트리거 수준을 변경할 수 있습니다.

sbRECEIVE_COMPLETED() 및 sbRECEIVE_COMPLETED_FROM_ISR()은 스트림 버퍼에서 데이터를 읽을 때 (FreeRTOS API에 의해 내부적으로) 호출되는 매크로입니다. 이 매크로는 버퍼 내에서 공간을 사용할 수 있을 때까지 대기하는 스트림 버퍼에 차단된 작업이 있는지 확인한 후 있는 경우 해당 작업을 차단 상태에서 제거합니다. FreeRTOSConfig.h에서 대체 구현을 제공하여 sbRECEIVE_COMPLETED()의 기본 동작을 변경할 수 있습니다.

메시지 버퍼

메시지 버퍼를 사용하면 인터럽트 서비스 루틴에서 작업으로 또는 한 작업에서 다른 작업으로 가변 길이의 별도 메시지를 전달할 수 있습니다. 예를 들어, 길이가 10, 20 및 123바이트인 메시지를 모두 동일한 메시지 버퍼에서 쓰고 읽을 수 있습니다. 10바이트 메시지는 개별 바이트가 아닌 10바이트 메시지로만 읽을 수 있습니다. 메시지 버퍼는 스트림 버퍼 구현을 기반으로 빌드됩니다. 프로젝트에서 stream_buffer.c 소스 파일을 포함하여 메시지 버퍼 기능을 활성화할 수 있습니다.

메시지 버퍼를 사용하면 버퍼에 쓰는 작업 또는 인터럽트 하나(라이터)와 버퍼에서 읽는 작업 또는 인터럽트 하나(리더)만 존재합니다. 라이터와 리더의 작업 또는 인터럽트 서비스 루틴이 달라도 되지만 여러 라이터 또는 리더가 존재하면 안 됩니다.

메시지 버퍼 구현에서는 DTT 알림을 사용합니다. 따라서 호출 작업을 차단 상태로 전환하는 스트림 버퍼 API를 호출하면 호출 작업의 알림 상태와 값이 변경될 수 있습니다.

메시지 버퍼를 사용하여 가변 크기 메시지를 처리하려면 각 메시지의 길이가 메시지보다 먼저 메시지 버퍼에 기록됩니다. 길이는 size_t 형식의 변수로 저장되며, 32바이트 아키텍처의 경우 일반적으로 4바이트입니다. 따라서 메시지 버퍼에 10바이트 메시지를 쓸 경우 실제로 14바이트의 버퍼 공간이 사용됩니다. 마찬가지로 메시지 버퍼에 100바이트 메시지를 쓸 경우 실제로 104바이트의 버퍼 공간이 사용됩니다.

데이터 전송

xMessageBufferSend()는 작업에서 메시지 버퍼로 전송하는 데 사용되고, xMessageBufferSendFromISR()은 인터럽트 서비스 루틴(ISR)에서 메시지 버퍼로 데이터를 전송하는 데 사용됩니다.

xMessageBufferSend()는 블록 시간 지정을 허용합니다. xMessageBufferSend()가 메시지 버퍼에 쓰기 위해 0이 아닌 블록 시간으로 호출되고 버퍼가 꽉 찬 경우 메시지 버퍼에서 공간을 사용할 수 있거나 블록 시간이 만료될 때까지 작업은 차단 상태로 전환됩니다.

sbSEND_COMPLETED() 및 sbSEND_COMPLETED_FROM_ISR()은 스트림 버퍼에 데이터를 쓸 때 FreeRTOS API에 의해 내부적으로 호출되는 매크로입니다. 이 매크로는 단일 파라미터 즉, 업데이트

된 스트림 버퍼의 핸들을 사용합니다. 두 매크로는 모두 스트림 버퍼에 데이터를 대기 중인 차단된 작업이 있는지 확인한 후, 있으면 해당 작업을 차단 상태에서 제거합니다.

FreeRTOSConfig.h에서 sbSEND_COMPLETED()의 자체 구현을 제공하여 이 기본 동작을 변경할 수 있습니다. 이 기능은 스트림 버퍼를 사용하여 멀티 코어 프로세서의 코어 간에 데이터를 전달할 때 유용합니다. 그러한 시나리오에서는 sbSEND_COMPLETED()를 구현하여 다른 CPU 코어에서 인터럽트를 생성한 다음 인터럽트의 서비스 루틴에서 xStreamBufferSendCompletedFromISR() API를 사용하여 데이터를 대기 중인 작업이 있는지 확인하고 필요한 경우 해당 작업을 차단 해제할 수 있습니다.

데이터 수신

xMessageBufferReceive()는 작업에서 메시지 버퍼의 데이터를 읽는 데 사용되고, xMessageBufferReceiveFromISR()은 인터럽트 서비스 루틴(ISR)에서 메시지 버퍼의 데이터를 읽는 데 사용됩니다. xMessageBufferReceive()는 블록 시간 지정을 허용합니다. xMessageBufferReceive()가 메시지 버퍼에서 읽기 위해 0이 아닌 블록 시간으로 호출되고 버퍼가 비어 있는 경우 데이터를 사용할 수 있거나 블록 시간이 만료될 때까지 작업은 차단 상태로 전환됩니다.

sbRECEIVE_COMPLETED() 및 sbRECEIVE_COMPLETED_FROM_ISR()은 스트림 버퍼에서 데이터를 읽을 때 (FreeRTOS API에 의해 내부적으로) 호출되는 매크로입니다. 이 매크로는 버퍼 내에서 공간을 사용할 수 있을 때까지 대기하는 스트림 버퍼에 차단된 작업이 있는지 확인한 후 있는 경우 해당 작업을 차단 상태에서 제거합니다. FreeRTOSConfig.h에서 대체 구현을 제공하여 sbRECEIVE_COMPLETED()의 기본 동작을 변경할 수 있습니다.

소프트웨어 타이머

소프트웨어 타이머를 사용하여 이후의 설정된 시간에 함수를 실행할 수 있습니다. 타이머에 의해 실행되는 함수를 타이머의 콜백 함수라고 합니다. 타이머가 시작되는 시간부터 콜백 함수가 실행되는 시간 사이를 타이머 기간이라고 합니다. FreeRTOS 커널이 효과적인 소프트웨어 타이머 구현을 제공하는 이유는 다음과 같습니다.

- 인터럽트 컨텍스트에서 타이머 콜백 함수를 실행하지 않습니다.
- 타이머가 실제로 만료되지 않는 한 처리 시간을 사용하지 않습니다.
- 틱 인터럽트에 처리 오버헤드를 추가하지 않습니다.
- 인터럽트가 비활성화된 상태에서 링크 목록 구조를 이동하지 않습니다.

저전력 지원

대부분의 임베디드 운영 체제와 마찬가지로 FreeRTOS 커널은 하드웨어 타이머를 사용하여 시간을 측정하는 데 사용되는 주기적 틱 인터럽트를 생성합니다. 틱 인터럽트를 처리하기 위해 저전력 상태를 주기적으로 종료했다가 다시 시작해야 하므로 정기적인 하드웨어 타이머 구현의 절전 효과는 제한됩니다. 틱 인터럽트의 빈도가 너무 높을 경우 모든 틱에 대한 저전력 상태를 시작하고 종료하는 데 사용되는 에너지 및 시간이 최대 절전 모드를 제외한 모든 모드의 잠재적 절전 효과를 능가합니다.

이 제한을 해결하기 위해 FreeRTOS에는 저전력 애플리케이션을 위한 타이머 미작동 모드가 포함되어 있습니다. FreeRTOS 타이머 미작동 유휴 모드에서는 유휴 기간(실행 가능한 애플리케이션 작업이 없는 기간) 동안 주기적으로 틱 인터럽트를 중지한 후, 틱 인터럽트를 다시 시작할 때 RTOS 틱 카운트 값을 알맞게 조정합니다. 틱 인터럽트를 중지하면 인터럽트가 발생하거나 RTOS 커널에서 작업을 준비 상태로 전환할 때까지 마이크로 컨트롤러가 딥 절전 상태로 유지될 수 있습니다.

커널 구성

`FreeRTOSConfig.h` 헤더 파일을 사용하여 특정 보드 및 애플리케이션에 대해 FreeRTOS 커널을 구성할 수 있습니다. 커널을 기반으로 빌드되는 모든 애플리케이션은 프리프로세서 포함 경로에 `FreeRTOSConfig.h` 헤더 파일이 있어야 합니다. `FreeRTOSConfig.h`는 애플리케이션 고유의 파일이며, FreeRTOS 커널 소스 코드 디렉터리 중 하나가 아니라 애플리케이션 디렉터리 아래에 위치해야 합니다.

FreeRTOS 데모 및 테스트 애플리케이션용 `FreeRTOSConfig.h` 파일은 *`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/FreeRTOSConfig.h` 및 *`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_tests/config_files/FreeRTOSConfig.h`에 위치합니다.

`FreeRTOSConfig.h`에서 지정할 수 있는 구성 파라미터의 목록은 FreeRTOS.org를 참조하십시오.

AWS IoT 임베디드 C용 디바이스 SDK

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

> **Note**
>
> 이 SDK는 숙련된 임베디드 소프트웨어 개발자가 사용하기 위한 것입니다.

AWS IoT Device SDK for Embedded C (C-SDK)는 임베디드 애플리케이션에서 IoT 디바이스를 안전하게 연결하는 데 사용할 수 있는 MIT 오픈 소스 라이선스에 따른 C 소스 파일 모음입니다 AWS IoT Core. 여기에는 MQTT 클라이언트, HTTP 클라이언트, JSON Parser 및 AWS IoT Device Shadow, AWS IoT Jobs 및 AWS IoT Device Defender 라이브러리가 포함됩니다. 이 SDK는 소스 형식으로 배포되며 애플리케이션 코드, 기타 라이브러리 및 선택한 운영 체제(OS)와 함께 고객 펌웨어에 구축될 수 있습니다.

AWS IoT Device SDK for Embedded C (C-SDK)는 임베디드 애플리케이션에서 IoT 디바이스를 안전하게 연결하는 데 사용할 수 있는 MIT 오픈 소스 라이선스에 따른 C 소스 파일 모음입니다 AWS IoT Core. 여기에는 MQTT 클라이언트, HTTP 클라이언트, JSON Parser 및 AWS IoT Device Shadow, AWS IoT Jobs, AWS IoT Fleet Provisioning 및 AWS IoT Device Defender 라이브러리가 포함됩니다. 이 SDK는 소스 형식으로 배포되며 애플리케이션 코드, 기타 라이브러리 및 선택한 운영 체제(OS)와 함께 고객 펌웨어에 구축될 수 있습니다.

AWS IoT Device SDK for Embedded C 는 일반적으로 최적화된 C 언어 런타임이 필요한 리소스 제한 디바이스를 대상으로 합니다. 모든 운영 체제에서 SDK를 사용하고, 모든 프로세서 유형(예: MCU 및 MPU)에서 호스팅할 수 있습니다. 하지만 디바이스에 메모리 및 처리 리소스가 충분한 경우, 상위 순위의 AWS IoT 디바이스 SDK 중 하나를 사용하는 것이 좋습니다.

자세한 내용은 다음 자료를 참조하세요.

- AWS IoT GitHub의 임베디드 C용 디바이스 SDK
- AWS IoT 임베디드 C Readme용 디바이스 SDK
- AWS IoT 임베디드 C 샘플용 디바이스 SDK

FreeRTOS 시작하기

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

이 FreeRTOS 시작하기 자습서에서는 호스트 시스템에 FreeRTOS를 다운로드하고 구성한 후 적격 마이크로컨트롤러 보드에서 단일 데모 애플리케이션을 컴파일하여 실행하는 방법을 보여 줍니다.

이 자습서에서는 AWS IoT 및 AWS IoT 콘솔에 익숙하다고 가정합니다. 그렇지 않은 경우 AWS IoT 시작하기 자습서를 완료한 후 진행하는 것이 좋습니다.

주제:

- FreeRTOS 데모 애플리케이션
- 첫 번째 단계
- 시작하기 문제 해결
- FreeRTOS에서 CMake 사용
- 개발자 모드 키 프로비저닝
- 보드별 시작 안내서

FreeRTOS 데모 애플리케이션

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

이 자습서의 데모 애플리케이션은 *freertos*/demos/coreMQTT/mqtt_demo_mutual_auth.c 파일에 정의된 coreMQTT 상호 인증 데모입니다. coreMQTT 라이브러리를 사용하여 AWS 클라우드에 연결한 다음 MQTT AWS IoT 브로커가 호스팅하는 MQTT 주제에 메시지를 주기적으로 게시합니다.

FreeRTOS 데모 애플리케이션은 한 번에 하나만 실행할 수 있습니다. FreeRTOS 데모 프로젝트를 빌드하는 경우 *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/

`aws_demo_config.h` 헤더 파일에서 활성화된 첫 번째 데모 애플리케이션이 실행됩니다. 이 자습서에서는 어떤 데모도 활성화 또는 비활성화할 필요가 없습니다. coreMQTT 상호 인증 데모는 기본적으로 활성화되어 있습니다.

FreeRTOS에 포함된 데모 애플리케이션에 대한 자세한 내용은 FreeRTOS 데모 섹션을 참조하세요.

첫 번째 단계

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

에서 FreeRTOS 사용을 시작하려면 AWS IoT 및 FreeRTOS 클라우드 서비스에 액세스할 수 있는 권한이 있는 IAM 사용자인 AWS 계정이 있어야 AWS IoT합니다. 또한 FreeRTOS를 다운로드하고 보드의 FreeRTOS 데모 프로젝트가 함께 작동하도록 구성해야 합니다 AWS IoT. 다음 단원에서는 이러한 요구 사항을 소개합니다.

> **Note**
>
> - Espressif ESP32-DevKitC, ESP-WROVER-KIT 또는 ESP32-WROOM-32SE를 사용하는 경우 이 단계를 건너뛰고 Espressif ESP32-DevKitC 및 ESP-WROVER-KIT 시작하기로 이동하십시오.
> - Nordic nRF52840-DK를 사용하는 경우 이러한 단계들을 건너뛰고 Nordic nRF52840-DK 시작하기로 이동하십시오.

1. AWS 계정 및 권한 설정
2. 에 MCU 보드 등록 AWS IoT
3. FreeRTOS 다운로드
4. FreeRTOS 데모 구성

AWS 계정 및 권한 설정

에 가입 AWS 계정

이 없는 경우 다음 단계를 AWS 계정완료하여 생성합니다.

에 가입하려면 AWS 계정

1. https://portal.aws.amazon.com/billing/signup을 엽니다.
2. 온라인 지시 사항을 따릅니다.

   가입 절차의 일부로는 전화 또는 문자 메시지를 수신하고 전화 키패드에 확인 코드를 입력하는 것이 포함됩니다.

   에 가입하면 AWS 계정AWS 계정 루트 사용자이 생성됩니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스에 액세스할 권한이 있습니다. 보안 모범 사례는 사용자에게 관리 액세스 권한을 할당하고, 루트 사용자만 사용하여 루트 사용자 액세스 권한이 필요한 작업을 수행하는 것입니다.

AWS 는 가입 프로세스가 완료된 후 확인 이메일을 보냅니다. 언제든지 https://aws.amazon.com/으로 이동하고 내 계정을 선택하여 현재 계정 활동을 보고 계정을 관리할 수 있습니다.

관리자 액세스 권한이 있는 사용자 생성

에 가입한 후 일상적인 작업에 루트 사용자를 사용하지 않도록 관리 사용자를 AWS 계정보호 AWS IAM Identity Center, AWS 계정 루트 사용자활성화 및 생성합니다.

보안 AWS 계정 루트 사용자

1. 루트 사용자를 선택하고 AWS 계정 이메일 주소를 입력하여 계정 소유자AWS Management Console로에 로그인합니다. 다음 페이지에서 비밀번호를 입력합니다.

   루트 사용자를 사용하여 로그인하는 데 도움이 필요하면 AWS 로그인 User Guide의 루트 사용자로 로그인을 참조하세요.
2. 루트 사용자의 다중 인증(MFA)을 활성화합니다.

   지침은 IAM 사용 설명서의 AWS 계정 루트 사용자(콘솔)에 대한 가상 MFA 디바이스 활성화를 참조하세요.

관리자 액세스 권한이 있는 사용자 생성

1. IAM Identity Center를 활성화합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 AWS IAM Identity Center설정을 참조하세요.
2. IAM Identity Center에서 사용자에게 관리 액세스 권한을 부여합니다.

   를 자격 증명 소스 IAM Identity Center 디렉터리 로 사용하는 방법에 대한 자습서는 사용 AWS IAM Identity Center 설명서의 기본값으로 사용자 액세스 구성을 IAM Identity Center 디렉터리 참조하세요.

관리 액세스 권한이 있는 사용자로 로그인

- IAM IDentity Center 사용자로 로그인하려면 IAM Identity Center 사용자를 생성할 때 이메일 주소로 전송된 로그인 URL을 사용합니다.

  IAM Identity Center 사용자를 사용하여 로그인하는 데 도움이 필요하면 AWS 로그인 사용 설명서의 AWS 액세스 포털에 로그인을 참조하세요.

추가 사용자에게 액세스 권한 할당

1. IAM Identity Center에서 최소 권한 적용 모범 사례를 따르는 권한 세트를 생성합니다.

   지침은AWS IAM Identity Center 사용 설명서의 Create a permission set를 참조하세요.
2. 사용자를 그룹에 할당하고, 그룹에 Single Sign-On 액세스 권한을 할당합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 Add groups를 참조하세요.

액세스 권한을 제공하려면 사용자, 그룹 또는 역할에 권한을 추가하세요:

- 의 사용자 및 그룹 AWS IAM Identity Center:

  권한 세트를 생성합니다. AWS IAM Identity Center 사용 설명서의 권한 세트 생성의 지침을 따릅니다.
- 보안 인증 공급자를 통해 IAM에서 관리되는 사용자:

  ID 페더레이션을 위한 역할을 생성합니다. IAM 사용 설명서의 Create a role for a third-party identity provider (federation)의 지침을 따릅니다.

- IAM 사용자:
  - 사용자가 맡을 수 있는 역할을 생성합니다. IAM 사용 설명서에서 Create a role for an IAM user의 지침을 따릅니다.
  - (권장되지 않음)정책을 사용자에게 직접 연결하거나 사용자를 사용자 그룹에 추가합니다. IAM 사용 설명서에서 사용자(콘솔)에 권한 추가의 지침을 따르세요.

AWS 계정 및 권한을 설정한 후 로 계속 진행합니다에 MCU 보드 등록 AWS IoT.

에 MCU 보드 등록 AWS IoT

AWS 클라우드와 통신 AWS IoT 하려면 보드를에 등록해야 합니다. 보드를에 등록하려면 다음이 AWS IoT필요합니다.

AWS IoT 정책

이 AWS IoT 정책은 디바이스에 AWS IoT 리소스에 액세스할 수 있는 권한을 부여합니다. 클라우드에 저장됩니다 AWS .

AWS IoT 사물

AWS IoT 사물을 사용하면에서 디바이스를 관리할 수 있습니다 AWS IoT. 클라우드에 저장됩니다 AWS .

프라이빗 키 및 X.509 인증서

프라이빗 키와 인증서를 사용하면 디바이스가 로 인증할 수 있습니다 AWS IoT.

보드를 등록하려면 아래 절차를 수행합니다.

AWS IoT 정책을 생성하려면

1. IAM 정책을 생성하려면 AWS 리전 및 AWS 계정 번호를 알아야 합니다.

   AWS 계정 번호를 찾으려면 AWS 관리 콘솔을 열고 오른쪽 상단의 계정 이름 아래에 있는 메뉴를 찾아 확장한 다음 내 계정을 선택합니다. 계정 ID는 Account Settings(계정 설정) 아래에 표시됩니다.

   AWS 계정의 AWS 리전을 찾으려면를 사용합니다 AWS Command Line Interface. 를 설치하려면 AWS Command Line Interface 사용 설명서의 지침을 AWS CLI따릅니다. 를 설치한 후 명령 프롬프트 창을 AWS CLI열고 다음 명령을 입력합니다.

```
aws iot describe-endpoint
```

출력은 다음과 같아야 합니다.

```
{
    "endpointAddress": "xxxxxxxxxxxxxx.iot.us-west-2.amazonaws.com"
}
```

이 예제에서 리전은 `us-west-2`입니다.

2. AWS IoT 콘솔로 이동합니다.
3. 탐색 창에서 Secure(보안)를 선택하고 Policies(정책)를 선택한 다음 Create(생성)를 선택합니다.
4. 정책을 식별할 이름을 입력합니다.
5. Add statements(설명문 추가) 섹션에서 Advanced mode(고급 모드)를 선택합니다. 다음 JSON을 복사하여 정책 편집기 창에 붙여 넣습니다. *aws-region* 및 *aws-account*를 AWS 리전 및 계정 ID로 바꿉니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
    {
        "Effect": "Allow",
        "Action": "iot:Connect",
        "Resource":"arn:aws:iot:aws-region:aws-account-id:*"
    },
    {
        "Effect": "Allow",
        "Action": "iot:Publish",
        "Resource": "arn:aws:iot:aws-region:aws-account-id:*"
    },
    {
         "Effect": "Allow",
         "Action": "iot:Subscribe",
         "Resource": "arn:aws:iot:aws-region:aws-account-id:*"
    },
    {
         "Effect": "Allow",
         "Action": "iot:Receive",
         "Resource": "arn:aws:iot:aws-region:aws-account-id:*"
    }
```

```
    ]
}
```

이 정책은 다음 권한을 부여합니다.

`iot:Connect`

모든 클라이언트 ID로 AWS IoT 메시지 브로커에 연결할 수 있는 권한을 디바이스에 부여합니다.

`iot:Publish`

모든 MQTT 주제에 MQTT 메시지를 게시할 수 있는 권한을 디바이스에 부여합니다.

`iot:Subscribe`

모든 MQTT 주제 필터를 구독할 수 있는 권한을 디바이스에 부여합니다.

`iot:Receive`

모든 MQTT 주제에 대한 AWS IoT 메시지 브로커에서 메시지를 수신할 수 있는 권한을 디바이스에 부여합니다.

6. 생성(Create)을 선택합니다.

디바이스에 대한 IoT 사물, 프라이빗 키 및 인증서를 생성하려면

1. AWS IoT 콘솔로 이동합니다.
2. 탐색 창에서 Manage(관리)를 선택한 다음 Things(사물)를 선택합니다.
3. 계정에 등록된 IoT 사물이 없는 경우 You don't have any things yet(아직 사물이 없습니다) 페이지가 표시됩니다. 이 페이지가 나타나면 Register a thing(사물 등록)을 선택합니다. 그렇지 않은 경우, 생성을 선택합니다.
4. AWS IoT 사물 생성 페이지에서 단일 사물 생성을 선택합니다.
5. Add your device to the thing registry(디바이스를 사물 등록에 추가) 페이지에 사물의 이름을 입력하고 Next(다음)를 선택합니다.
6. Add a certificate for your thing(사물에 대한 인증서 추가) 페이지의 One-click certificate creation(원클릭 인증서 생성)에서 Create certificate(인증서 생성)를 선택합니다.
7. 각각에 대한 Download(다운로드) 링크를 선택하여 프라이빗 키와 인증서를 다운로드합니다.
8. Activate(활성화)를 선택하여 인증서를 활성화합니다. 인증서는 사용 전에 활성화해야 합니다.

9. 정책 연결을 선택하여 디바이스에 AWS IoT 작업에 대한 액세스 권한을 부여하는 정책을 인증서에 연결합니다.
10. 방금 생성한 정책을 선택하고 Register thing(사물 등록)을 선택합니다.

보드가에 등록된 후 로 계속 AWS IoT진행할 수 있습니다FreeRTOS 다운로드.

FreeRTOS 다운로드

FreeRTOS는 FreeRTOS GitHub 리포지토리에서 다운로드할 수 있습니다.

> **Note**
>
> Cypress CYW954907AEVAL1F 또는 CYW943907AEVAL1F 개발 키트를 시작하는 경우 GitHub에서 FreeRTOS를 다운로드해야 합니다. 자세한 내용은 README.md 파일을 참조하십시오. 현재 이러한 보드에 대한 FreeRTOS 구성은 FreeRTOS 콘솔에서 사용할 수 없습니다.

FreeRTOS 데모 구성

보드에서 데모를 컴파일하고 실행하려면 먼저 FreeRTOS 디렉터리에서 몇 가지 구성 파일을 편집해야 합니다.

AWS IoT 엔드포인트를 구성하려면

보드에서 실행되는 애플리케이션이 올바른 AWS IoT 엔드포인트로 요청을 보낼 수 있도록 FreeRTOS에 엔드포인트를 제공해야 합니다.

1. AWS IoT 콘솔로 이동합니다.
2. 탐색 창에서 설정을 선택합니다.

   AWS IoT 엔드포인트는 엔드포인트에 표시됩니다. URL은 *1234567890123*-ats.iot.*us-east-1*.amazonaws.com와 같아야 합니다. 이 엔드포인트를 기록해 둡니다.
3. 탐색 창에서 Manage(관리)를 선택한 다음 Things(사물)를 선택합니다.

   디바이스에는 AWS IoT 사물 이름이 있어야 합니다. 이 이름을 기록해 둡니다.
4. demos/include/aws_clientcredential.h를 엽니다.
5. 다음 상수의 값을 지정합니다.

   - `#define clientcredentialMQTT_BROKER_ENDPOINT "Your AWS IoT endpoint";`

- `#define clientcredentialIOT_THING_NAME "The AWS IoT thing name of your board"`

Wi-Fi를 구성하려면

보드를 Wi-Fi 연결을 통해 인터넷에 연결하는 경우 FreeRTOS에 네트워크 연결에 필요한 Wi-Fi 보안 인증 정보를 제공해야 합니다. 보드가 Wi-Fi를 지원하지 않는 경우 이 단계를 건너뛰어도 됩니다.

1. `demos/include/aws_clientcredential.h`.
2. 다음 `#define` 상수의 값을 지정합니다.

   - `#define clientcredentialWIFI_SSID "The SSID for your Wi-Fi network"`
   - `#define clientcredentialWIFI_PASSWORD "The password for your Wi-Fi network"`
   - `#define clientcredentialWIFI_SECURITY Wi-Fi ##### ## ##`

     유효한 보안 유형은 다음과 같습니다.

     - `eWiFiSecurityOpen`(열림, 보안 없음)
     - `eWiFiSecurityWEP`(WEP 보안)
     - `eWiFiSecurityWPA`(WPA 보안)
     - `eWiFiSecurityWPA2`(WPA2 보안)

AWS IoT 자격 증명의 형식을 지정하려면

디바이스를 AWS IoT 대신하여와 성공적으로 통신하려면 FreeRTOS에 등록된 사물과 연결된 AWS IoT 인증서 및 프라이빗 키와 권한 정책이 있어야 합니다.

> **Note**
>
> AWS IoT 자격 증명을 구성하려면 디바이스를 등록할 때 AWS IoT 콘솔에서 다운로드한 프라이빗 키와 인증서가 있어야 합니다. 디바이스를 AWS IoT 사물로 등록한 후에는 AWS IoT 콘솔에서 디바이스 인증서를 검색할 수 있지만 프라이빗 키는 검색할 수 없습니다.

FreeRTOS는 C 언어 프로젝트이며, 인증서와 프라이빗 키를 프로젝트에 추가하려면 특수한 형식을 지정해야 합니다.

1. 브라우저 창에서 `tools/certificate_configuration/CertificateConfigurator.html`을 엽니다.
2. Certificate PEM file(인증서 PEM 파일)에서는 AWS IoT 콘솔에서 다운로드한 *ID*-`certificate.pem.crt`를 선택합니다.
3. Private Key PEM file(프라이빗 키 PEM 파일)에서는 AWS IoT 콘솔에서 다운로드한 *ID*-`private.pem.key`를 선택합니다.
4. aws_clientcredential_keys.h 생성 및 저장을 선택한 다음 파일을 `demos/include`에 저장합니다. 이 파일은 디렉터리의 기존 파일을 덮어씁니다.

> **Note**
>
> 인증서와 프라이빗 키는 데모 용도로만 하드 코딩됩니다. 프로덕션 수준 애플리케이션은 이러한 파일을 보안 위치에 저장해야 합니다.

FreeRTOS를 구성한 후에는 이어서 보드에 대한 시작하기 가이드를 진행하여 플랫폼의 하드웨어 및 소프트웨어 개발 환경을 설정한 후 보드에서 데모를 컴파일하고 실행할 수 있습니다. 보드별 지침은 보드별 시작 안내서 단원을 참조하십시오. 시작하기 자습서에서 사용하는 데모 애플리케이션은 coreMQTT 상호 인증 데모이며 `demos/coreMQTT/mqtt_demo_mutual_auth.c`에 있습니다.

시작하기 문제 해결

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

다음 항목은 FreeRTOS를 시작하는 과정에서 발생할 수 있는 문제를 해결하는 데 도움이 될 수 있습니다.

주제

- 일반 시작하기 문제 해결 팁
- 터미널 에뮬레이터 설치

보드별 문제 해결은 해당 보드의 FreeRTOS 시작하기 설명서를 참조하십시오.

일반 시작하기 문제 해결 팁

Hello World 데모 프로젝트를 실행한 후에는 AWS IoT 콘솔에 메시지가 표시되지 않습니다. 어떻게 해야 합니까?

다음을 시도해 보세요.

1. 터미널 창을 열고 샘플을 로깅 출력을 봅니다. 이 단계는 무엇이 잘못되었는지 확인하는 데 도움이 됩니다.
2. 자격 증명이 유효한지 확인합니다.

데모를 실행할 때 터미널에 표시되는 로그가 잘립니다. 길이를 늘리려면 어떻게 해야 하나요?

실행 중인 데모의 `FreeRTOSconfig.h` 파일에서 `configLOGGING_MAX_MESSAGE_LENGTH` 값을 255로 늘리십시오.

```
#define configLOGGING_MAX_MESSAGE_LENGTH    255
```

터미널 에뮬레이터 설치

터미널 에뮬레이터는 문제를 진단하거나 디바이스 코드가 올바르게 실행되는지 확인하는 데 도움이 될 수 있습니다. Windows, macOS 및 Linux에 사용할 수 있는 다양한 터미널 에뮬레이터가 있습니다.

터미널 에뮬레이터로 보드에 직렬 연결을 설정하려고 하기 전에 보드를 컴퓨터에 연결해야 합니다.

다음 설정을 사용하여 터미널 에뮬레이터를 구성합니다.

터미널 설정	값
전송 속도	115200
테스트	8비트
패리티	없음
중지	1비트
흐름 제어	없음

보드의 직렬 포트 찾기

보드의 직렬 포트를 모를 경우 명령줄이나 터미널에서 다음 명령 중 하나를 실행하여 호스트 컴퓨터에 연결된 모든 디바이스의 직렬 포트를 반환할 수 있습니다.

Windows

```
chgport
```

Linux

```
ls /dev/tty*
```

macOS

```
ls /dev/cu.*
```

FreeRTOS에서 CMake 사용

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

CMake를 사용하여 FreeRTOS 애플리케이션 소스 코드에서 프로젝트 빌드 파일을 생성하고 소스 코드를 빌드 및 실행할 수 있습니다.

IDE를 사용하여 FreeRTOS 적격 디바이스에서 코드를 편집, 디버깅, 컴파일, 플래시 및 실행할 수도 있습니다. 각 보드별 시작 안내서에는 특정 플랫폼용 IDE 설정 지침이 포함되어 있습니다. IDE 없이 작업하려면 다른 타사 코드 편집 및 디버깅 도구를 사용하여 코드를 개발 및 디버깅한 다음 CMake를 사용하여 애플리케이션을 빌드하고 실행할 수 있습니다.

CMake를 지원하는 보드는 다음과 같습니다.

- Espressif ESP32-DevKitC
- Espressif ESP-WROVER-KIT
- Infineon XMC4800 IoT 연결 키트

- Marvell MW320 AWS IoT Starter 키트
- Marvell MW322 AWS IoT Starter 키트
- Microchip Curiosity PIC32MZEF 번들
- Nordic nRF52840 DK 개발 키트
- STMicroelectronicsSTM32L4 Discovery Kit IoT 노드
- Texas Instruments CC3220SF-LAUNCHXL
- Microsoft Windows Simulator

FreeRTOS에서 CMake를 사용하는 방법에 대한 자세한 내용은 아래 주제를 참조하세요.

주제

- 사전 조건
- 타사 코드 편집기 및 디버깅 도구를 사용하여 FreeRTOS 애플리케이션 개발
- CMake를 사용하여 FreeRTOS 빌드

사전 조건

계속하기 전에 호스트 시스템이 다음 사전 조건을 충족하는지 확인하십시오.

- 디바이스의 컴파일 도구 체인이 컴퓨터의 운영 체제를 지원해야 합니다. CMake는 모든 버전의 Windows, macOS 및 Linux를 지원합니다.

  Linux용 Windows 하위 시스템(WSL)은 지원되지 않습니다. Windows 시스템에서 네이티브 CMake를 사용합니다.
- CMake 버전 3.13 이상이 설치되어 있어야 합니다.

  CMake.org에서 CMake의 바이너리 배포를 다운로드할 수 있습니다.

  > **Note**
  >
  > CMake 바이너리 배포를 다운로드하는 경우 명령줄에서 CMake를 사용하기 전에 CMake 실행 파일을 PATH 환경 변수에 추가해야 합니다.

  또한 macOS에서 homebrew, Windows에서 scoop이나 chocolatey와 같은 패키지 관리자를 사용하여 CMake를 다운로드하고 설치할 수 있습니다.

> **Note**
>
> 많은 Linux 배포판에 대한 패키지 관리자에 제공되는 CMake 패키지 버전은 최신 버전이 아닙니다. 배포의 패키지 관리자에 최신 버전의 CMake가 포함되어 있지 않으면 `linuxbrew` 또는 `nix`와 같은 다른 패키지 관리자를 사용해 볼 수 있습니다.

- 호환 가능한 네이티브 빌드 시스템이 있어야 합니다.

  CMake는 GNU Make 또는 Ninja 등의 많은 네이티브 빌드 시스템을 대상으로 할 수 있습니다. Make와 Ninja는 모두 Linux, macOS 및 Windows의 패키지 관리자를 사용하여 설치할 수 있습니다. Windows에서 Make를 사용하는 경우 수식에서 독립 실행형 버전을 설치하거나 Make를 번들로 제공하는 MinGW를 설치할 수 있습니다.

  > **Note**
  >
  > MinGW의 Make 실행 파일은 `make.exe`가 아니라 `mingw32-make.exe`입니다.

  Ninja는 Make보다 빠르며 모든 데스크톱 운영 체제에 네이티브 지원을 제공하므로 Ninja를 사용하는 것이 좋습니다.

타사 코드 편집기 및 디버깅 도구를 사용하여 FreeRTOS 애플리케이션 개발

코드 편집기, 디버깅 확장 또는 타사 디버깅 도구를 사용하여 FreeRTOS용 애플리케이션을 개발할 수 있습니다.

예를 들어 Visual Studio Code를 코드 편집기로 사용하는 경우 Cortex-Debug VS Code 확장을 디버거로 설치할 수 있습니다. 애플리케이션 개발을 마치면 CMake 명령줄 도구를 호출하여 VS Code 내에서 프로젝트를 빌드할 수 있습니다. CMake를 사용한 FreeRTOS 애플리케이션 빌드에 대한 자세한 내용은 CMake를 사용하여 FreeRTOS 빌드 섹션을 참조하세요.

디버깅을 위해 다음과 비슷한 디버그 구성으로 VS Code를 제공할 수 있습니다.

```
"configurations": [
    {
        "name": "Cortex Debug",
        "cwd": "${workspaceRoot}",
        "executable": "./build/st/stm32l475_discovery/aws_demos.elf",
```

```
        "request": "launch",
        "type": "cortex-debug",
        "servertype": "stutil"
    }
]
```

CMake를 사용하여 FreeRTOS 빌드

CMake는 기본적으로 호스트 운영 체제를 대상 시스템으로 사용합니다. CMake를 교차 컴파일에 사용하려면 도구 체인 파일이 필요합니다. 이 파일은 사용할 컴파일러를 지정합니다. FreeRTOS에서는 기본 도구 체인 파일이 *freertos*/tools/cmake/toolchains에서 제공됩니다. 이 파일을 CMake에 제공하는 방법은 CMake 명령줄 인터페이스를 사용하는지, 아니면 GUI를 사용하는지에 따라 다릅니다. 추가 세부 정보는 아래 빌드 파일 생성(CMake 명령줄 도구) 지침을 따르십시오. CMake의 교차 컴파일에 대한 자세한 내용은 공식 CMake wiki의 CrossCompiling을 참조하십시오.

CMake 기반의 프로젝트를 빌드하려면

1. CMake를 실행하여 Make 또는 Ninja와 같은 네이티브 빌드 시스템용 빌드 파일을 생성합니다.

   CMake 명령줄 도구 또는 CMake GUI를 사용하여 네이티브 빌드 시스템용 빌드 파일을 생성할 수 있습니다.

   FreeRTOS 빌드 파일 생성에 대한 자세한 내용은 빌드 파일 생성(CMake 명령줄 도구) 및 빌드 파일 생성(CMake GUI) 섹션을 참조하세요.
2. 네이티브 빌드 시스템을 호출하여 프로젝트를 실행 파일로 만듭니다.

   FreeRTOS 빌드 파일 생성에 대한 자세한 내용은 생성된 빌드 파일에서 FreeRTOS 빌드 섹션을 참조하세요.

빌드 파일 생성(CMake 명령줄 도구)

CMake 명령줄 도구(cmake)를 사용하여 FreeRTOS 빌드 파일을 생성할 수 있습니다. 빌드 파일을 생성하려면 대상 보드, 컴파일러, 소스 코드와 빌드 디렉터리의 위치를 지정해야 합니다.

cmake에는 다음 옵션을 사용할 수 있습니다.

- `-DVENDOR` - 대상 보드를 지정합니다.
- `-DCOMPILER` - 컴파일러를 지정합니다.
- `-S` - 소스 코드의 위치를 지정합니다.

- `-B` - 생성된 빌드 파일의 위치를 지정합니다.

> **Note**
>
> 컴파일러는 시스템의 PATH 변수에 있어야 하며 그렇지 않은 경우 컴파일러의 위치를 지정해야 합니다.

예를 들어, 공급업체가 Texas Instruments이고 보드가 CC3220 Launchpad이며 컴파일러가 ARM용 GCC인 경우, 다음 명령을 실행하여 현재 디렉터리에서 *build-directory* 디렉터리로 소스 파일을 빌드할 수 있습니다.

```
cmake -DVENDOR=ti -DBOARD=cc3220_launchpad -DCOMPILER=arm-ti -S . -B build-directory
```

> **Note**
>
> Windows를 사용하는 경우 CMake가 기본적으로 Visual Studio를 사용하기 때문에 네이티브 빌드 시스템을 지정해야 합니다. 예시:
>
> ```
> cmake -DVENDOR=ti -DBOARD=cc3220_launchpad -DCOMPILER=arm-ti -S . -B build-directory -G Ninja
> ```
>
> 또는 다음과 같습니다.
>
> ```
> cmake -DVENDOR=ti -DBOARD=cc3220_launchpad -DCOMPILER=arm-ti -S . -B build-directory -G "MinGW Makefiles"
> ```

정규 표현식 `${VENDOR}.*` 및 `${BOARD}.*`는 일치하는 보드를 검색하는 데 사용되므로 VENDOR 및 BOARD 옵션에 대해 공급업체 및 보드의 전체 이름을 사용할 필요가 없습니다. 일치하는 항목이 하나만 있는 경우 이름의 일부만 사용해도 됩니다. 예를 들어, 다음 명령은 동일한 소스에서 동일한 빌드 파일을 생성합니다.

```
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
```

기본 디렉터리 cmake/toolchains에 없는 도구 체인 파일을 사용하려면 CMAKE_TOOLCHAIN_FILE 옵션을 사용할 수 있습니다. 예시:

```
cmake -DBOARD=cc3220 -DCMAKE_TOOLCHAIN_FILE='/path/to/toolchain_file.cmake' -S . -
B build-directory
```

도구 체인 파일이 컴파일러에 대한 절대 경로를 사용하지 않고 컴파일러를 PATH 환경 변수에 추가하지 않은 경우 CMake가 해당 경로를 찾을 수 없습니다. CMake가 도구 체인 파일을 찾을 수 있게 하려면 AFR_TOOLCHAIN_PATH 옵션을 사용합니다. 이 옵션은 bin 아래에서 지정된 도구 체인 디렉터리 경로와 도구 체인의 하위 폴더를 검색합니다. 예시:

```
cmake -DBOARD=cc3220 -DCMAKE_TOOLCHAIN_FILE='/path/to/toolchain_file.cmake' -
DAFR_TOOLCHAIN_PATH='/path/to/toolchain/' -S . -B build-directory
```

디버깅을 활성화하려면 CMAKE_BUILD_TYPE을 debug로 설정합니다. 이 옵션을 활성화한 경우 CMake는 디버그 플래그를 컴파일 옵션에 추가하고 디버그 기호로 FreeRTOS를 빌드합니다.

```
# Build with debug symbols
cmake -DBOARD=cc3220 -DCOMPILER=arm-ti -DCMAKE_BUILD_TYPE=debug -S . -B build-directory
```

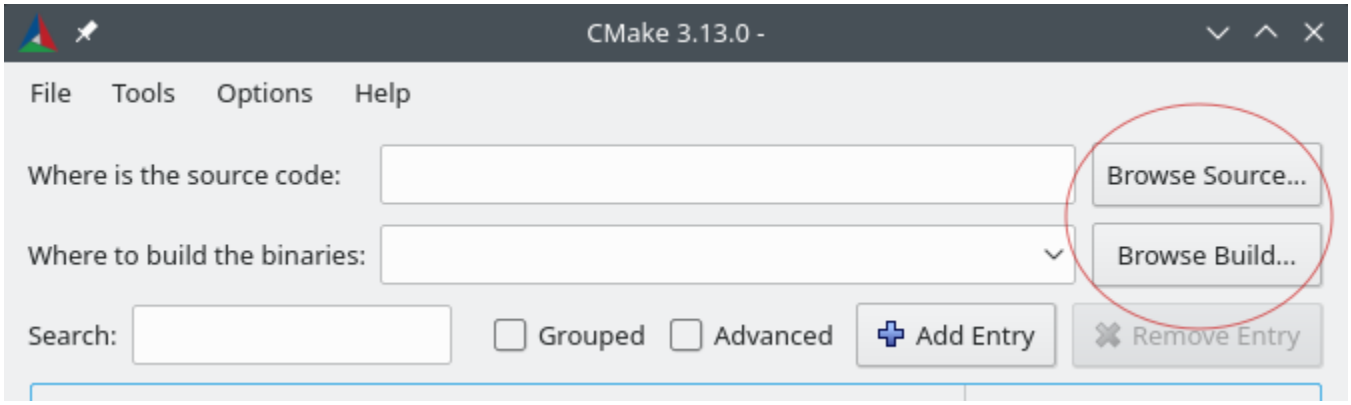
CMAKE_BUILD_TYPE을 release로 설정하여 컴파일 옵션에 최적화 플래그를 추가할 수도 있습니다.

빌드 파일 생성(CMake GUI)

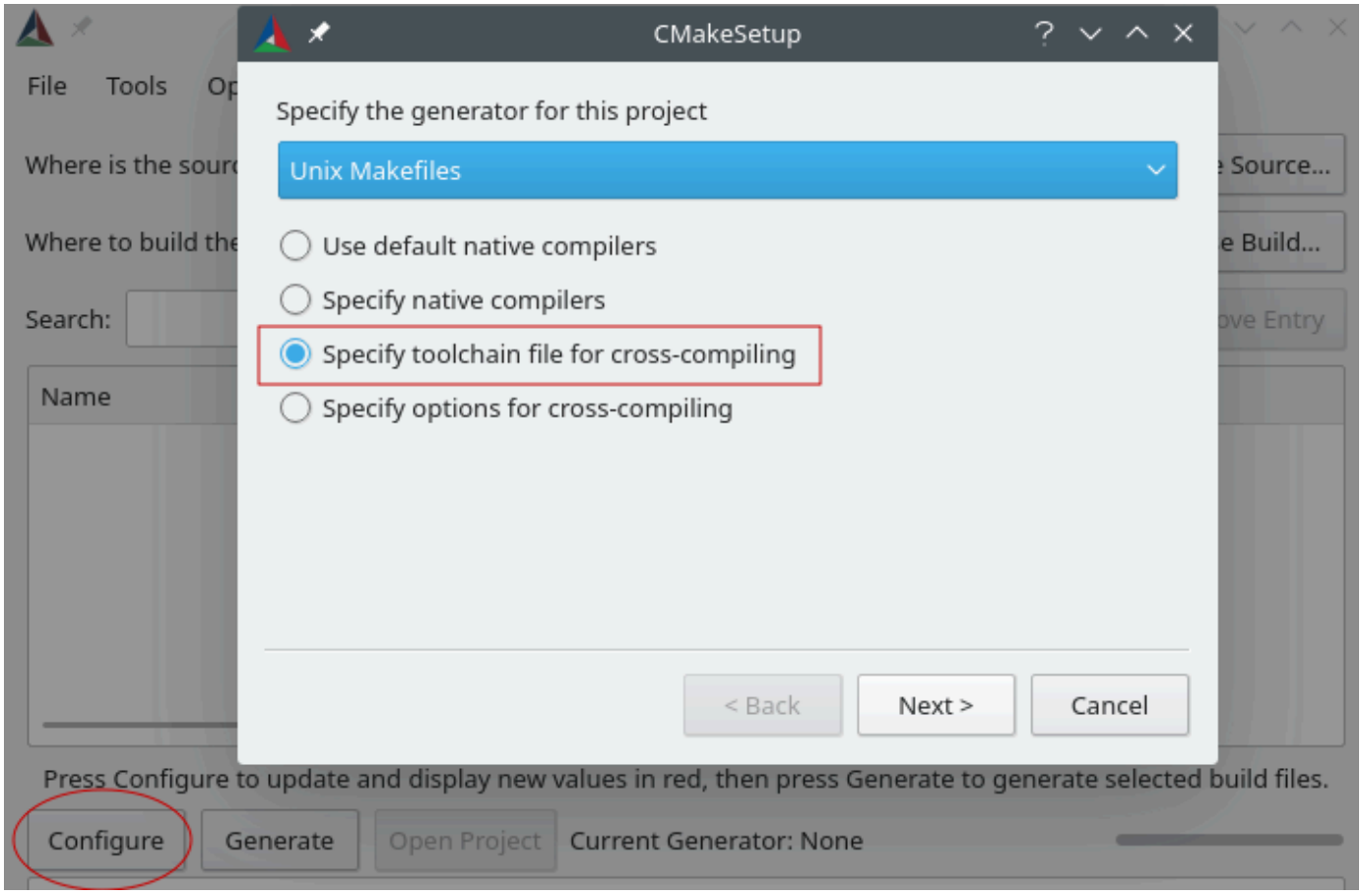
CMake GUI를 사용하여 FreeRTOS 빌드 파일을 생성할 수 있습니다.

CMake GUI를 사용하여 빌드 파일을 생성하려면

1. 명령줄에서 cmake-gui를 실행하여 GUI를 시작합니다.
2. Browse Source(소스 찾아보기)를 선택하고 소스 입력을 지정한 다음 Browse Build(빌드 찾아보기)를 선택하고 빌드 출력을 지정합니다.



3. 구성을 선택하고 Specify the build generator for this project(이 프로젝트에 대한 빌드 생성기 지정)에서 생성된 빌드 파일을 빌드하는 데 사용할 빌드 시스템을 찾아서 선택합니다. 팝업 창이 보이지 않으면 기존 빌드 디렉터리를 재사용하는 것일 수 있습니다. 이 경우 파일 메뉴에서 캐시 삭제를 선택하여 CMake 캐시를 삭제합니다.



4. Specify toolchain file for cross-compiling(교차 컴파일을 위한 도구 체인 파일 지정)을 선택하고 다음을 선택합니다.

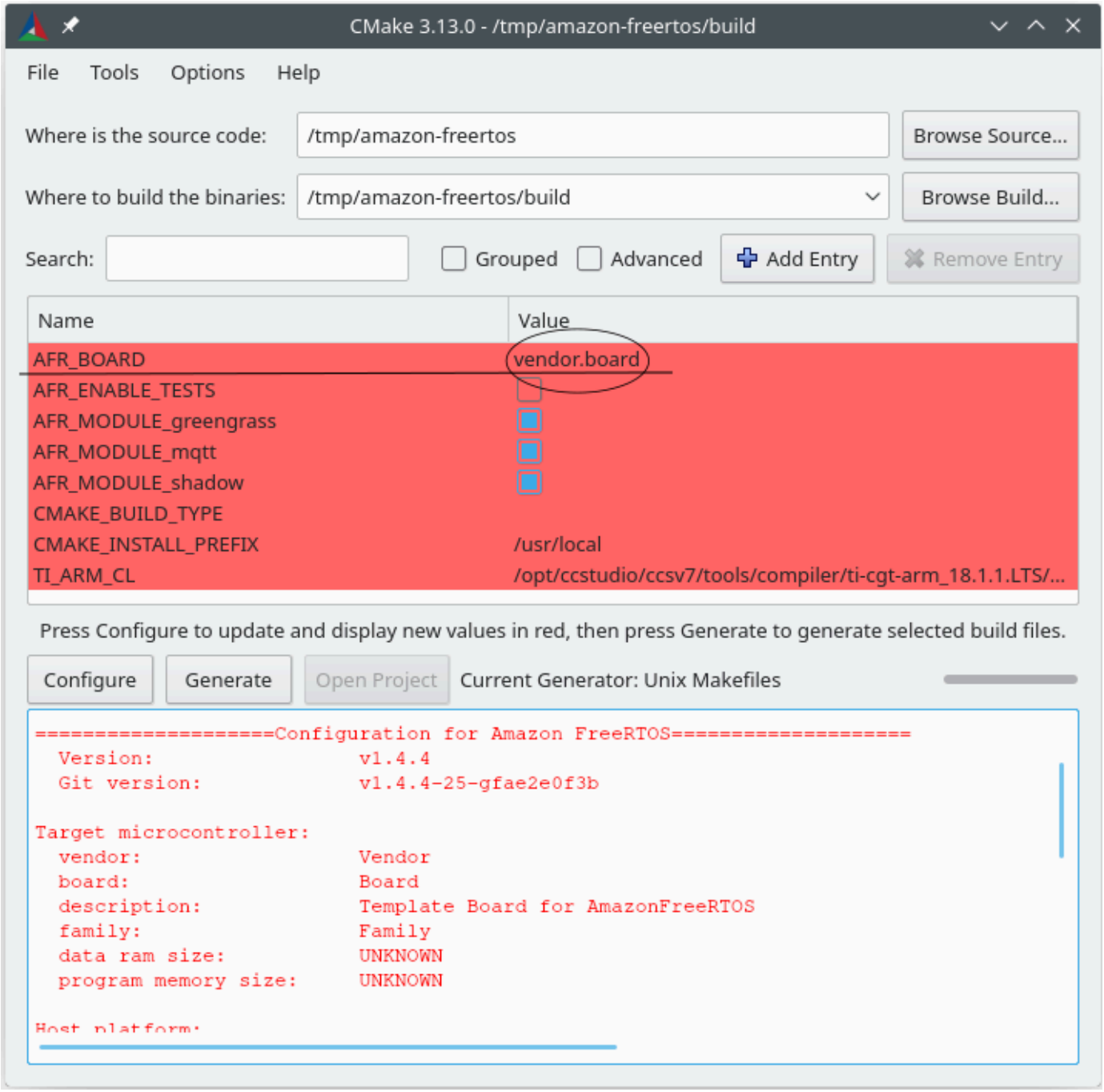
5. 도구 체인 파일(예: *freertos*/tools/cmake/toolchains/arm-ti.cmake)을 선택하고 완료를 선택합니다.

FreeRTOS의 기본 구성은 이식 가능 계층 대상을 제공하지 않는 템플릿 보드입니다. 결과적으로 Error in configuration process 메시지와 함께 창이 나타납니다.

> **Note**
>
> 다음 오류가 발생하는 경우
>
> ```
> CMake Error at tools/cmake/toolchains/find_compiler.cmake:23 (message):
> Compiler not found, you can specify search path with AFR_TOOLCHAIN_PATH.
> ```

컴파일러가 PATH 환경 변수에 없는 것입니다. GUI에서 AFR_TOOLCHAIN_PATH 변수를 설정하여 컴퓨터를 설치한 위치를 CMake에 알립니다. AFR_TOOLCHAIN_PATH 변수가 보이지 않으면 Add Entry(항목 추가)를 선택합니다. 팝업 창의 Name(이름)에 **AFR_TOOLCHAIN_PATH**를 입력합니다. Compiler Path(컴파일러 경로)에 컴파일의 경로를 입력합니다(예: `C:/toolchains/arm-none-eabi-gcc`).

6. GUI는 이제 다음과 같아야 합니다.



AFR_BOARD를 선택하고 보드를 선택한 다음 구성을 다시 선택합니다.

7. 생성을 선택합니다. CMake는 빌드 시스템 파일(예: makefiles 또는 ninja 파일)을 생성하며, 이러한 파일은 첫 번째 단계에서 지정한 빌드 디렉터리에 나타납니다. 다음 단원의 지침에 따라 이진 이미지를 생성합니다.

생성된 빌드 파일에서 FreeRTOS 빌드

네이티브 빌드 시스템을 사용하여 빌드

출력 바이너리 디렉터리에서 빌드 시스템 명령을 호출하여 네이티브 빌드 시스템으로 FreeRTOS를 빌드할 수 있습니다.

예를 들어, 빌드 파일 출력 디렉터리가 <build_dir>이고 네이티브 빌드 시스템으로 Make를 사용하는 경우 다음 명령을 실행하세요.

```
cd <build_dir>
make -j4
```

CMake를 사용하여 빌드

CMake 명령줄 도구를 사용하여 FreeRTOS를 빌드할 수도 있습니다. CMake는 네이티브 빌드 시스템을 호출하기 위한 추상 계층을 제공합니다. 예시:

```
cmake --build build_dir
```

다음은 CMake 명령줄 도구의 빌드 모드를 사용하는 몇 가지 일반적인 사례입니다.

```
# 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
```

CMake 빌드 모드에 대한 자세한 내용은 CMake 설명서를 참조하십시오.

개발자 모드 키 프로비저닝

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

소개

이 섹션에서는 랩 테스트를 위해 IoT 디바이스에 신뢰할 수 있는 X.509 클라이언트 인증서를 가져오는 두 가지 옵션에 대해 설명합니다. 디바이스의 기능에 따라 온보드 ECDSA 키 생성, 프라이빗 키 가져오기 및 X.509 인증서 등록을 비롯한 다양한 프로비저닝 관련 작업이 지원되거나 지원되지 않을 수 있습니다. 또한 다양한 사용 사례는 온보드 플래시 스토리지에서 전용 암호화 하드웨어 사용에 이르기까지 다양한 수준의 키 보호를 요구합니다. 이 섹션에서는 디바이스의 암호화 기능 내에서 작업하기 위한 로직을 제공합니다.

옵션 #1:에서 프라이빗 키 가져오기 AWS IoT

랩 테스트를 위해 디바이스에서 프라이빗 키 가져오기를 허용하는 경우 FreeRTOS 데모 구성의 지침을 따르십시오.

옵션 #2: 온보드 프라이빗 키 생성

디바이스에 보안 요소가 있거나 사용자 고유의 디바이스 키 페어 및 인증서를 생성하려는 경우 여기 나와 있는 지침을 따르십시오.

초기 구성

먼저의 단계를 수행하되 마지막 단계는 건너뜁니다(즉FreeRTOS 데모 구성, AWS IoT 자격 증명 형식을 지정하려면 하지 않음). 최종 결과는 `demos/include/aws_clientcredential.h` 파일이 사용자 설정으로 업데이트되었지만 `demos/include/aws_clientcredential_keys.h` 파일은 업데이트되지 않은 것입니다.

데모 프로젝트 구성

보드별 시작 안내서의 보드 안내서에 설명된 대로 coreMQTT 상호 인증 데모를 엽니다. 프로젝트에서 `aws_dev_mode_key_provisioning.c` 파일을 열고 기본적으로 0으로 설정된 `keyprovisioningFORCE_GENERATE_NEW_KEY_PAIR`의 정의를 1로 변경합니다.

```
#define keyprovisioningFORCE_GENERATE_NEW_KEY_PAIR 1
```

그런 다음 데모 프로젝트를 빌드 및 실행하고 다음 단계를 계속합니다.

퍼블릭 키 추출

디바이스가 아직 프라이빗 키 및 클라이언트 인증서로 프로비저닝되지 않았으므로, 데모는 AWS IoT에 대한 인증이 실패합니다. 그러나 coreMQTT 상호 인증 데모는 개발자 모드 키 프로비저닝을

실행하여 시작되므로 프라이빗 키가 없으면 생성됩니다. 직렬 콘솔 출력 시작 부분에 다음과 같이 표시되어야 합니다.

```
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
```

6줄의 키 바이트를 `DevicePublicKeyAsciiHex.txt` 파일에 복사합니다. 그런 다음 명령줄 도구 “xxd”를 사용하여 16진수 바이트를 바이너리로 구문 분석합니다.

```
xxd -r -ps DevicePublicKeyAsciiHex.txt DevicePublicKeyDer.bin
```

“openssl”을 사용하여 바이너리 인코딩(DER) 디바이스 퍼블릭 키를 PEM으로 포맷합니다.

```
openssl ec -inform der -in DevicePublicKeyDer.bin -pubin -pubout -outform pem -out
 DevicePublicKey.pem
```

위에서 활성화한 임시 키 생성 설정을 비활성화해야 합니다. 그렇지 않으면 디바이스가 또 다른 키 쌍을 생성하며 이전 단계를 반복해야 합니다.

```
#define keyprovisioningFORCE_GENERATE_NEW_KEY_PAIR 0
```

퍼블릭 키 인프라 설정

CA 인증서 등록의 지침에 따라 디바이스 랩 인증서에 대한 인증서 계층 구조를 만듭니다. CA 인증서를 사용하여 디바이스 인증서 생성 섹션에 설명된 시퀀스를 실행하기 전에 중지합니다.

이 경우 ROM 크기를 줄이기 위해 CSR을 생성하고 서명하는 데 필요한 X.509 인코딩 로직이 FreeRTOS 데모 프로젝트에서 제외되었으므로 디바이스가 인증서 요청(즉, 인증서 서비스 요청 또는 CSR)에 서명하지 않습니다. 대신 랩 테스트를 위해 워크스테이션에 프라이빗 키를 만들어 CSR에 서명하는 데 사용합니다.

```
openssl genrsa -out tempCsrSigner.key 2048
openssl req -new -key tempCsrSigner.key -out deviceCert.csr
```

인증 기관이 생성되고에 등록되면 다음 명령을 AWS IoT사용하여 이전 단계에서 서명된 디바이스 CSR을 기반으로 클라이언트 인증서를 발급합니다.

```
openssl x509 -req -in deviceCert.csr -CA rootCA.pem -CAkey rootCA.key
 -CAcreateserial -out deviceCert.pem -days 500 -sha256 -force_pubkey
 DevicePublicKey.pem
```

CSR이 임시 프라이빗 키로 서명되었더라도 발급된 인증서는 실제 디바이스 프라이빗 키에만 사용할 수 있습니다. 별도의 하드웨어에 CSR 서명자 키를 저장하고 해당 특정 키로 서명된 요청에 대해서만 인증서를 발급하도록 인증 기관을 구성하는 경우 동일한 메커니즘을 프로덕션에서 사용할 수 있습니다. 이 키는 지정된 관리자의 통제 하에 있어야 합니다.

인증서 가져오기

인증서를 발급한 후 다음 단계는 인증서를 디바이스로 가져오는 것입니다. 또한 JITP를 사용할 때 최초 인증이 성공하려면 CA(인증 기관) 인증서를 가져와야 AWS IoT 하기 때문입니다. 프로젝트의 `aws_clientcredential_keys.h` 파일에서 `keyCLIENT_CERTIFICATE_PEM` 매크로를 deviceCert.pem의 콘텐츠로 설정하고 `keyJITR_DEVICE_CERTIFICATE_AUTHORITY_PEM` 매크로를 `rootCA.pem`의 콘텐츠로 설정합니다.

디바이스 승인

자체 인증서 사용에 설명된 대로 AWS IoT 레지스트리`deviceCert.pem`로를 가져옵니다. 새 AWS IoT 사물을 생성하고, 보류 중인 인증서와 정책을 사물에 연결한 다음 인증서를 활성으로 표시해야 합니다. 이러한 모든 단계는 AWS IoT 콘솔에서 수동으로 수행할 수 있습니다.

새 클라이언트 인증서가 활성 상태이고 사물 및 정책과 연결되면 coreMQTT 상호 인증 데모를 다시 실행합니다. 이번에는 AWS IoT MQTT 브로커에 대한 연결이 성공합니다.

보드별 시작 안내서

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

첫 번째 단계를 완료한 후 FreeRTOS 시작하기에 대한 보드별 지침은 해당 보드의 안내서를 참조하세요.

- Cypress CYW943907AEVAL1F 개발 키트 시작하기
- Cypress CYW954907AEVAL1F 개발 키트 시작하기
- Cypress CY8CKIT-064S0S2-4343W 키트 시작하기
- Infineon XMC4800 IoT 연결 키트 시작하기
- MW32x AWS IoT 스타터 키트 시작하기
- MediaTek MT7697Hx 개발 키트 시작하기
- Microchip Curiosity PIC32MZ EF 시작하기
- Nuvoton NuMaker-IoT-M487 시작하기
- NXP LPC54018 IoT 모듈 시작하기
- Renesas Starter Kit+ for RX65N-2MB 시작하기
- STMicroelectronics STM32L4 Discovery Kit IoT Node 시작하기
- Texas Instruments CC3220SF-LAUNCHXL 시작하기
- Windows Device Simulator 시작하기
- Xilinx Avnet MicroZed Industrial IoT Kit 시작하기

Note

다음 FreeRTOS 시작하기 자습 안내서에서 첫 번째 단계를 수행할 필요가 없습니다.

- Windows 시뮬레이터를 사용하여 Microchip ATECC608A 보안 요소 시작하기
- Espressif ESP32-DevKitC 및 ESP-WROVER-KIT 시작하기
- Espressif ESP32-WROOM-32SE 시작하기
- Infineon OPTIGA Trust X 및 XMC4800 IoT 연결 키트 시작하기
- Nordic nRF52840-DK 시작하기

Cypress CYW943907AEVAL1F 개발 키트 시작하기

Important

이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

이 자습서에서는 Cypress CYW943907AEVAL1F 개발 키트를 시작하기 위한 지침을 제공합니다. Cypress CYW943907AEVAL1F 개발 키트가 없는 경우 AWS Partner Device Catalog를 방문하여 파트너에서 구입하시기 바랍니다.

> **Note**
>
> 이 자습서는 coreMQTT 상호 인증 데모를 설정하고 실행하는 단계를 안내합니다. 현재, 이 보드의 FreeRTOS 포트는 TCP 서버 및 클라이언트 데모를 지원하지 않습니다.

시작하기 전에 AWS IoT 및 FreeRTOS 다운로드를 구성하여 디바이스를 AWS 클라우드에 연결해야 합니다. 자세한 내용은 첫 번째 단계 섹션을 참조하세요.

> **Important**
>
> - 이 주제에서는 FreeRTOS 다운로드 디렉터리의 경로를 *freertos*이라고 합니다.
> - *freertos* 경로의 공백 문자로 인해 빌드 실패가 발생할 수 있습니다. 리포지토리를 복제하거나 복사할 때 생성하는 경로에 공백 문자가 없어야 합니다.
> - Microsoft Windows에서 파일 경로의 최대 길이는 260자입니다. FreeRTOS 다운로드 디렉터리 경로가 길면 빌드 오류가 발생할 수 있습니다.
> - FreeRTOS 다운로드에서 언급했듯이 Cypress용 FreeRTOS 포트는 현재 GitHub에서만 사용할 수 있습니다.

개요

이 자습서에는 다음의 시작하기 단계에 대한 지침이 포함되어 있습니다.

1. 마이크로 컨트롤러 보드용 내장형 애플리케이션을 개발 및 디버깅하기 위한 소프트웨어를 호스트 시스템에 설치합니다.
2. FreeRTOS 데모 애플리케이션을 바이너리 이미지로 크로스 컴파일합니다.
3. 애플리케이션 바이너리 이미지를 보드에 로드한 후 애플리케이션을 실행합니다.
4. 모니터링 및 디버깅을 위해 직렬 연결로 보드에서 실행되는 애플리케이션과 상호 작용합니다.

개발 환경 설정

WICED Studio SDK 다운로드 및 설치

이 시작 안내서에서는 Cypress WICED Studio SDK를 사용하여 FreeRTOS 데모로 보드를 프로그래밍합니다. WICED Software 웹 사이트를 방문하여 Cypress에서 WICED Studio SDK를 다운로드합니다. 이 소프트웨어를 다운로드하려면 무료 Cypress 계정에 등록해야 합니다. WICED Studio SDK는 Windows, macOS, Linux 운영 체제와 호환됩니다.

> **Note**
>
> 일부 운영 체제에는 추가 설치 단계가 필요합니다. 운영 체제와 설치하려는 WICED Studio 버전에 대한 모든 설치 지침을 읽고 따라야 합니다.

환경 변수 설정

WICED Studio를 사용하여 보드를 프로그래밍하기 전에 WICED Studio SDK 설치 디렉터리의 환경 변수를 만들어야 합니다. 변수를 만드는 동안 WICED Studio를 실행하는 경우, 변수를 설정한 후 애플리케이션을 다시 시작해야 합니다.

> **Note**
>
> WICED Studio 설치 관리자는 시스템에 `WICED-Studio-`*`m`*`.`*`n`*이라는 별도의 두 폴더를 생성합니다. 여기서 `m` 및 `n`은 각각 메이저 및 마이너 버전 번호입니다. 이 문서에서는 `WICED-Studio-6.2`의 폴더 이름을 가정하지만, 설치한 버전의 정확한 이름을 사용해야 합니다. `WICED_STUDIO_SDK_PATH` 환경 변수를 정의할 때 WICED Studio IDE의 설치 경로가 아닌, WICED Studio SDK의 전체 설치 경로를 지정해야 합니다. Windows 및 macOS에서는 SDK의 `WICED-Studio-`*`m`*`.`*`n`* 폴더가 기본적으로 `Documents` 폴더에 생성됩니다.

Windows에서 환경 변수를 만들려면

1. 제어판을 열고 시스템을 선택한 후 고급 시스템 설정을 선택합니다.
2. 고급 탭에서 환경 변수를 선택합니다.
3. 사용자 변수 아래에서 새로 만들기를 선택합니다.
4. 변수 이름에 **`WICED_STUDIO_SDK_PATH`**를 입력합니다. 변수 값에 WICED Studio SDK 설치 디렉터리를 입력합니다.

Linux 또는 macOS에서 환경 변수를 만들려면

1. 시스템에서 `/etc/profile` 파일을 열고, 파일의 마지막 줄에 다음을 추가합니다.

```
export WICED_STUDIO_SDK_PATH=installation-path/WICED-Studio-6.2
```

2. 시스템을 다시 시작합니다.
3. 터미널을 열고 다음 명령을 실행합니다.

```
cd freertos/vendors/cypress/WICED_SDK
```

```
perl platform_adjust_make.pl
```

```
chmod +x make
```

직렬 연결 설정

호스트 시스템과 보드 간의 직렬 연결을 설정하려면

1. USB 스탠다드-A-마이크로-B 케이블을 사용하여 보드를 호스트 컴퓨터에 연결합니다.
2. 호스트 컴퓨터의 보드에 연결할 수 있도록 USB 직렬 포트 번호를 확인합니다.
3. 직렬 터미널을 시작하고 다음 설정으로 연결을 엽니다.
   - 전송 속도: 115200
   - 데이터: 8비트
   - 패리티: 없음
   - 정지 비트: 1
   - 흐름 제어: 없음

터미널 설치 및 직렬 연결 설정에 대한 자세한 내용은 터미널 에뮬레이터 설치를 참조하십시오.

FreeRTOS 데모 프로젝트 빌드 및 실행

보드에 대해 직렬 연결을 설정했으면 FreeRTOS 데모 프로젝트를 빌드하고, 데모를 보드에 전송한 후 데모를 실행합니다.

WICED Studio에서 FreeRTOS 데모 프로젝트를 빌드하고 실행하려면

1. WICED Studio를 시작합니다.
2. 파일 메뉴에서 가져오기를 선택합니다. `General` 폴더를 확장하고 Existing Projects into Workspace(기존 프로젝트를 Workspace로)를 선택한 후 다음을 선택합니다.
3. 루트 디렉터리 선택에서 찾아보기...를 선택하고 *`freertos`*`/projects/cypress/CYW943907AEVAL1F/wicedstudio` 경로로 이동한 후 확인을 선택합니다.
4. Projects(프로젝트)에서 aws_demo 프로젝트의 상자만 선택합니다. Finish(완료)를 선택하여 프로젝트를 가져옵니다. 대상 프로젝트 aws_demo가 Make Target(대상 만들기) 창에 나타나야 합니다.
5. WICED Platform(WICED 플랫폼) 메뉴를 확장하고 WICED Filters off(WICED 필터 끄기)를 선택합니다.
6. Make Target(대상 만들기) 창에서 aws_demo를 확장하고 `demo.aws_demo` 파일을 마우스 오른쪽 버튼으로 누르고 Build Target(대상 빌드)을 선택하여 데모를 빌드하고 보드로 다운로드합니다. 데모를 빌드하여 보드에 다운로드하면 자동으로 실행되어야 합니다.

클라우드에서 MQTT 메시지 모니터링

AWS IoT 콘솔에서 MQTT 클라이언트를 사용하여 디바이스가 AWS 클라우드로 보내는 메시지를 모니터링할 수 있습니다.

MQTT 클라이언트를 사용하여 AWS IoT MQTT 주제를 구독하려면

1. AWS IoT 콘솔에 로그인합니다.
2. 탐색 창에서 Test(테스트)를 선택하여 MQTT 클라이언트를 엽니다.
3. 구독 주제에 ***`your-thing-name`*`/example/topic`**을 입력한 다음 주제 구독을 선택합니다.

문제 해결

- Windows를 사용하는 경우, 데모 프로젝트를 빌드 및 실행할 때 다음 오류가 표시될 수 있습니다.

```
: recipe for target 'download_dct' failed
make.exe[1]: *** [download_dct] Error 1
```

  이 오류 문제를 해결하려면 다음을 수행합니다.

1. *WICED-Studio-SDK-PATH*\WICED-Studio-6.2\43xxx_Wi-Fi\tools\OpenOCD\Win32로 이동하고 openocd-all-brcm-libftdi.exe 파일을 두 번 클릭합니다.
2. *WICED-Studio-SDK-PATH*\WICED-Studio-6.2\43xxx_Wi-Fi\tools\drivers\CYW9WCD1EVAL1로 이동하고 InstallDriver.exe 파일을 두 번 클릭합니다.

- Linux 또는 macOS를 사용하는 경우, 데모 프로젝트를 빌드 및 실행할 때 다음 오류가 표시될 수 있습니다.

```
make[1]: *** [download_dct] Error 127
```

이 오류 문제를 해결하려면 다음 명령을 사용하여 libusb-dev 패키지를 업데이트합니다.

```
sudo apt-get install libusb-dev
```

FreeRTOS 시작하기에 대한 일반 문제 해결 정보는 시작하기 문제 해결 섹션을 참조하세요.

Cypress CYW954907AEVAL1F 개발 키트 시작하기

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

이 자습서에서는 Cypress CYW954907AEVAL1F 개발 키트를 시작하기 위한 지침을 제공합니다. Cypress CYW954907AEVAL1F 개발 키트가 없는 경우 AWS Partner Device Catalog를 방문하여 파트너에서 구입하시기 바랍니다.

> **Note**
>
> 이 자습서는 coreMQTT 상호 인증 데모를 설정하고 실행하는 단계를 안내합니다. 현재, 이 보드의 FreeRTOS 포트는 TCP 서버 및 클라이언트 데모를 지원하지 않습니다.

시작하기 전에 AWS IoT 및 FreeRTOS 다운로드를 구성하여 디바이스를 AWS 클라우드에 연결해야 합니다. 자세한 내용은 첫 번째 단계 섹션을 참조하세요. 이 자습서에서는 FreeRTOS 다운로드 디렉터리의 경로를 *freertos*라고 합니다.

Important

- 이 주제에서는 FreeRTOS 다운로드 디렉터리의 경로를 *freertos*이라고 합니다.
- *freertos* 경로의 공백 문자로 인해 빌드 실패가 발생할 수 있습니다. 리포지토리를 복제하거나 복사할 때 생성하는 경로에 공백 문자가 없어야 합니다.
- Microsoft Windows에서 파일 경로의 최대 길이는 260자입니다. FreeRTOS 다운로드 디렉터리 경로가 길면 빌드 오류가 발생할 수 있습니다.
- FreeRTOS 다운로드에서 언급했듯이 Cypress용 FreeRTOS 포트는 현재 GitHub에서만 사용할 수 있습니다.

개요

이 자습서에는 다음의 시작하기 단계에 대한 지침이 포함되어 있습니다.

1. 마이크로 컨트롤러 보드용 내장형 애플리케이션을 개발 및 디버깅하기 위한 소프트웨어를 호스트 시스템에 설치합니다.
2. FreeRTOS 데모 애플리케이션을 바이너리 이미지로 크로스 컴파일합니다.
3. 애플리케이션 바이너리 이미지를 보드에 로드한 후 애플리케이션을 실행합니다.
4. 모니터링 및 디버깅을 위해 직렬 연결로 보드에서 실행되는 애플리케이션과 상호 작용합니다.

개발 환경 설정

WICED Studio SDK 다운로드 및 설치

이 시작 안내서에서는 Cypress WICED Studio SDK를 사용하여 FreeRTOS 데모로 보드를 프로그래밍합니다. WICED Software 웹 사이트를 방문하여 Cypress에서 WICED Studio SDK를 다운로드합니다. 이 소프트웨어를 다운로드하려면 무료 Cypress 계정에 등록해야 합니다. WICED Studio SDK는 Windows, macOS, Linux 운영 체제와 호환됩니다.

Note

일부 운영 체제에는 추가 설치 단계가 필요합니다. 운영 체제와 설치하려는 WICED Studio 버전에 대한 모든 설치 지침을 읽고 따라야 합니다.

환경 변수 설정

WICED Studio를 사용하여 보드를 프로그래밍하기 전에 WICED Studio SDK 설치 디렉터리의 환경 변수를 만들어야 합니다. 변수를 만드는 동안 WICED Studio를 실행하는 경우, 변수를 설정한 후 애플리케이션을 다시 시작해야 합니다.

> **Note**
>
> WICED Studio 설치 관리자는 시스템에 `WICED-Studio-`*`m`*`.`*`n`*이라는 별도의 두 폴더를 생성합니다. 여기서 `m` 및 `n`은 각각 메이저 및 마이너 버전 번호입니다. 이 문서에서는 `WICED-Studio-6.2`의 폴더 이름을 가정하지만, 설치한 버전의 정확한 이름을 사용해야 합니다. `WICED_STUDIO_SDK_PATH` 환경 변수를 정의할 때 WICED Studio IDE의 설치 경로가 아닌, WICED Studio SDK의 전체 설치 경로를 지정해야 합니다. Windows 및 macOS에서는 SDK의 `WICED-Studio-`*`m`*`.`*`n`* 폴더가 기본적으로 `Documents` 폴더에 생성됩니다.

Windows에서 환경 변수를 만들려면

1. 제어판을 열고 시스템을 선택한 후 고급 시스템 설정을 선택합니다.
2. 고급 탭에서 환경 변수를 선택합니다.
3. 사용자 변수 아래에서 새로 만들기를 선택합니다.
4. 변수 이름에 **`WICED_STUDIO_SDK_PATH`**를 입력합니다. 변수 값에 WICED Studio SDK 설치 디렉터리를 입력합니다.

Linux 또는 macOS에서 환경 변수를 만들려면

1. 시스템에서 `/etc/profile` 파일을 열고, 파일의 마지막 줄에 다음을 추가합니다.

   ```
   export WICED_STUDIO_SDK_PATH=installation-path/WICED-Studio-6.2
   ```

2. 시스템을 다시 시작합니다.
3. 터미널을 열고 다음 명령을 실행합니다.

   ```
   cd freertos/vendors/cypress/WICED_SDK
   ```

   ```
   perl platform_adjust_make.pl
   ```

```
chmod +x make
```

직렬 연결 설정

호스트 시스템과 보드 간의 직렬 연결을 설정하려면

1. USB 스탠다드-A-마이크로-B 케이블을 사용하여 보드를 호스트 컴퓨터에 연결합니다.
2. 호스트 컴퓨터의 보드에 연결할 수 있도록 USB 직렬 포트 번호를 확인합니다.
3. 직렬 터미널을 시작하고 다음 설정으로 연결을 엽니다.

   - 전송 속도: 115200
   - 데이터: 8비트
   - 패리티: 없음
   - 정지 비트: 1
   - 흐름 제어: 없음

터미널 설치 및 직렬 연결 설정에 대한 자세한 내용은 터미널 에뮬레이터 설치를 참조하십시오.

FreeRTOS 데모 프로젝트 빌드 및 실행

보드에 대해 직렬 연결을 설정했으면 FreeRTOS 데모 프로젝트를 빌드하고, 데모를 보드에 전송한 후 데모를 실행합니다.

WICED Studio에서 FreeRTOS 데모 프로젝트를 빌드하고 실행하려면

1. WICED Studio를 시작합니다.
2. 파일 메뉴에서 가져오기를 선택합니다. `General` 폴더를 확장하고 Existing Projects into Workspace(기존 프로젝트를 Workspace로)를 선택한 후 다음을 선택합니다.
3. 루트 디렉터리 선택에서 찾아보기...를 선택하고 *`freertos`*`/projects/cypress/CYW954907AEVAL1F/wicedstudio` 경로로 이동한 후 확인을 선택합니다.
4. Projects(프로젝트)에서 aws_demo 프로젝트의 상자만 선택합니다. Finish(완료)를 선택하여 프로젝트를 가져옵니다. 대상 프로젝트 aws_demo가 Make Target(대상 만들기) 창에 나타나야 합니다.
5. WICED Platform(WICED 플랫폼) 메뉴를 확장하고 WICED Filters off(WICED 필터 끄기)를 선택합니다.

6. Make Target(대상 만들기) 창에서 aws_demo를 확장하고 `demo.aws_demo` 파일을 마우스 오른쪽 버튼으로 누르고 Build Target(대상 빌드)을 선택하여 데모를 빌드하고 보드로 다운로드합니다. 데모를 빌드하여 보드에 다운로드하면 자동으로 실행되어야 합니다.

클라우드에서 MQTT 메시지 모니터링

AWS IoT 콘솔에서 MQTT 클라이언트를 사용하여 디바이스가 AWS 클라우드로 보내는 메시지를 모니터링할 수 있습니다.

MQTT 클라이언트를 사용하여 AWS IoT MQTT 주제를 구독하려면

1. AWS IoT 콘솔에 로그인합니다.
2. 탐색 창에서 Test(테스트)를 선택하여 MQTT 클라이언트를 엽니다.
3. 구독 주제에 ***your-thing-name*/example/topic**을 입력한 다음 주제 구독을 선택합니다.

문제 해결

- Windows를 사용하는 경우, 데모 프로젝트를 빌드 및 실행할 때 다음 오류가 표시될 수 있습니다.

```
: recipe for target 'download_dct' failed
make.exe[1]: *** [download_dct] Error 1
```

  이 오류 문제를 해결하려면 다음을 수행합니다.

  1. *WICED-Studio-SDK-PATH*\WICED-Studio-6.2\43xxx_Wi-Fi\tools\OpenOCD\Win32로 이동하고 openocd-all-brcm-libftdi.exe 파일을 두 번 클릭합니다.
  2. *WICED-Studio-SDK-PATH*\WICED-Studio-6.2\43xxx_Wi-Fi\tools\drivers\CYW9WCD1EVAL1로 이동하고 InstallDriver.exe 파일을 두 번 클릭합니다.

- Linux 또는 macOS를 사용하는 경우, 데모 프로젝트를 빌드 및 실행할 때 다음 오류가 표시될 수 있습니다.

```
make[1]: *** [download_dct] Error 127
```

  이 오류 문제를 해결하려면 다음 명령을 사용하여 libusb-dev 패키지를 업데이트합니다.

```
sudo apt-get install libusb-dev
```

FreeRTOS 시작하기에 대한 일반 문제 해결 정보는 시작하기 문제 해결 섹션을 참조하세요.

Cypress CY8CKIT-064S0S2-4343W 키트 시작하기

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

이 자습서에서는 CY8CKIT-064S0S2-4343W 키트를 시작하기 위한 지침을 제공합니다. 키트가 없는 경우 해당 링크를 사용하여 구입할 수 있습니다. 해당 링크를 사용하여 키트 사용 설명서에 액세스할 수도 있습니다.

시작

시작하기 전에 디바이스를 AWS 클라우드에 연결하도록 AWS IoT 및 FreeRTOS를 구성해야 합니다. 지침은 첫 번째 단계 단원을 참조하십시오. 사전 조건을 완료하면 AWS IoT Core 자격 증명이 포함된 FreeRTOS 패키지가 제공됩니다.

> **Note**
>
> 이 자습서에서는 '첫 번째 단계' 섹션에서 생성한 FreeRTOS 다운로드 디렉터리의 경로를 *freertos*라고 합니다.

개발 환경 설정

FreeRTOS는 CMake 또는 Make 빌드 흐름에서 작동합니다. ModusToolbox를 Make 빌드 흐름에 사용할 수 있습니다. ModusToolbox와 함께 제공되는 Eclipse IDE 또는 IAR EW-Arm, Arm MDK, Microsoft Visual Studio Code와 같은 파트너 IDE를 사용할 수 있습니다. Eclipse IDE는 Windows, macOS, Linux 운영 체제와 호환됩니다.

시작하기 전에 최신 ModusToolbox 소프트웨어를 다운로드하여 설치합니다. 자세한 내용은 ModusToolbox 설치 안내서를 참조하세요.

ModusToolbox 2.1 이상에서 도구 업데이트

ModusToolBox 2.1 Eclipse IDE를 사용하여 이 키트를 프로그래밍하는 경우 OpenOCD 및 Firmware-loader 도구를 업데이트해야 합니다.

다음 단계에서 기본적으로 *ModusToolbox* 경로는 다음과 같습니다.

- Windows는 C:\Users*user_name*\ModusToolbox입니다.
- Linux는 *user_home*/ModusToolbox 또는 사용자가 선택한 아카이브 파일 추출 위치입니다.
- macOS는 마법사에서 선택한 볼륨의 Applications 폴더 아래에 있습니다.

OpenOCD 업데이트

이 키트가 성공적으로 칩을 소거하고 프로그래밍하기 위해 Cypress OpenOCD 4.0.0 이상이 필요합니다.

Cypress OpenOCD를 업데이트하려면

1. Cypress OpenOCD 릴리스 페이지로 이동합니다.
2. 운영 체제(Windows/Mac/Linux)에 해당하는 아카이브 파일을 다운로드합니다.
3. *ModusToolbox*/tools_2.x/openocd에서 기존 파일을 삭제합니다.
4. *ModusToolbox*/tools_2.x/openocd의 파일을 이전 단계에서 다운로드한 아카이브에서 추출된 파일로 바꿉니다.

Firmware-loader 업데이트

이 키트에는 Cypress Firmware-loader 3.0.0 이상이 필요합니다.

Cypress Firmware-loader를 업데이트하려면

1. Cypress Firmware-loader 릴리스 페이지로 이동합니다.
2. 운영 체제(Windows/Mac/Linux)에 해당하는 아카이브 파일을 다운로드합니다.
3. *ModusToolbox*/tools_2.x/fw-loader에서 기존 파일을 삭제합니다.
4. *ModusToolbox*/tools_2.x/fw-loader의 파일을 이전 단계에서 다운로드한 아카이브에서 추출된 파일로 바꿉니다.

또는, CMake를 사용하여 FreeRTOS 애플리케이션 소스 코드에서 프로젝트 빌드 파일을 생성하고 선호하는 빌드 도구를 사용하여 프로젝트를 빌드한 다음 OpenOCD를 사용하여 키트를 프로그래밍할 수 있습니다. CMake 흐름으로 프로그래밍하기 위해 GUI 도구를 사용하고 싶다면 Cypress Programming Solutions 웹 페이지에서 Cypress Programmer를 다운로드하여 설치합니다. 자세한 내용은 FreeRTOS에서 CMake 사용 단원을 참조하십시오.

하드웨어 설정

다음 단계에 따라 키트 하드웨어를 설정합니다.

1. 키트 프로비저닝

   Provisioning Guide for CY8CKIT-064S0S2-4343W Kit 지침에 따라 AWS IoT용 키트를 안전하게 프로비저닝합니다.
2. 직렬 연결 설정

   a. 키트를 호스트 컴퓨터에 연결합니다.

   b. 키트용 USB 직렬 포트는 호스트 컴퓨터에 자동으로 열거됩니다. 포트 번호를 확인합니다. Windows에서는 장치 관리자를 사용하여 포트(COM 및 LPT) 아래에서 확인할 수 있습니다.

   c. 직렬 터미널을 시작하고 다음 설정으로 연결을 엽니다.

      - 전송 속도: 115200
      - 데이터: 8비트
      - 패리티: 없음
      - 정지 비트: 1
      - 흐름 제어: 없음

FreeRTOS 데모 프로젝트 빌드 및 실행

이 섹션에서는 데모를 빌드하고 실행합니다.

1. Provisioning Guide for CY8CKIT-064S0S2-4343W Kit의 단계를 따라야 합니다.
2. FreeRTOS 데모 빌드.

   a. Eclipse IDE for ModusToolbox를 열고 작업 공간을 선택하거나 생성합니다.

   b. 파일 메뉴에서 가져오기를 선택합니다.

      일반을 확장하고 기존 프로젝트를 작업 공간으로를 선택한 다음 다음을 선택합니다.

   c. 루트 디렉터리에 aws_demos를 입력하고 프로젝트 이름 *freertos*/projects/cypress/CY8CKIT-064S0S2-4343W/mtb/aws_demos를 선택합니다. 이 이름이 기본적으로 선택되어 있을 것입니다.

   d. 완료를 선택하여 프로젝트를 작업 공간으로 가져옵니다.

   e. 다음 중 하나를 수행하여 애플리케이션을 빌드합니다.

- 퀵 패널에서 aws_demos 애플리케이션 빌드를 선택합니다.
- 프로젝트를 선택하고 모두 빌드를 선택합니다.

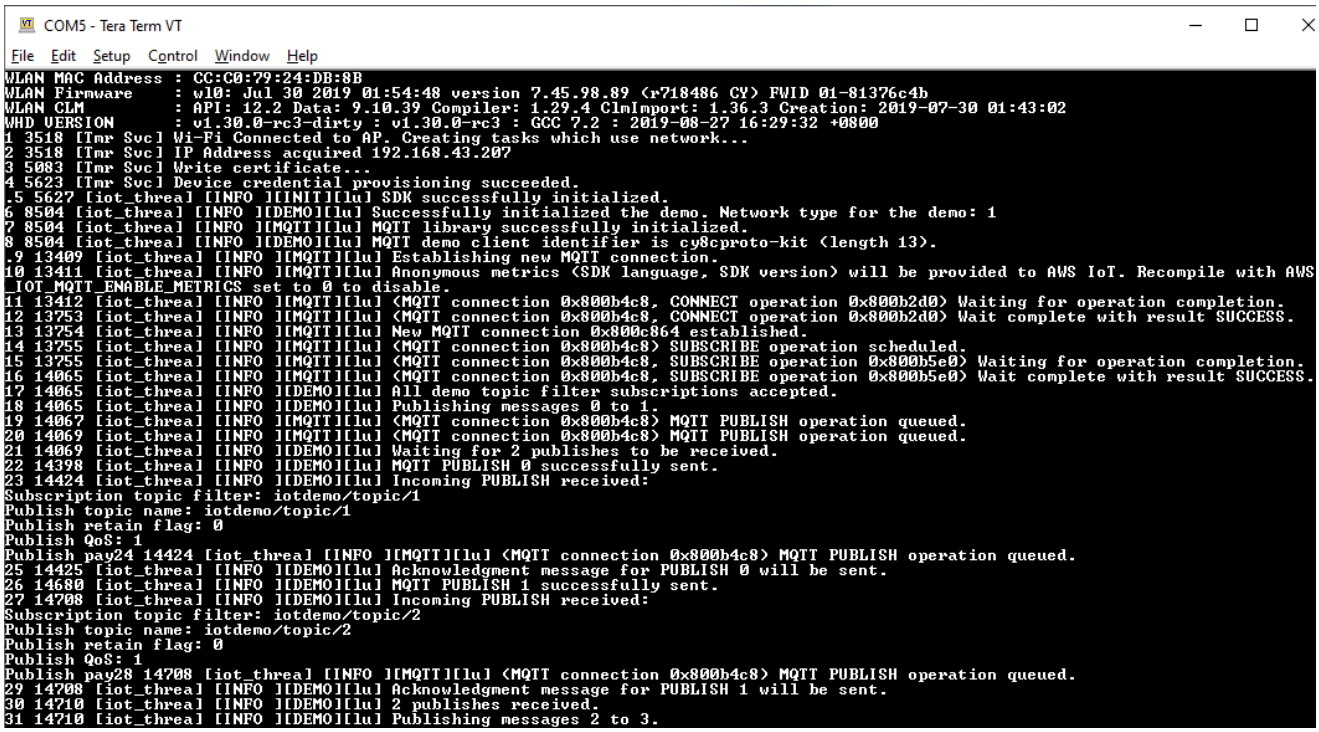
프로젝트가 오류 없이 컴파일되는지 확인합니다.

3. FreeRTOS 데모 프로젝트 실행

a. 작업 공간에서 프로젝트 aws_demos를 선택합니다.

b. 퀵 패널에서 aws_demos 프로그램(KitProg3)을 선택합니다. 이렇게 하면 보드가 프로그래밍되고 프로그래밍이 끝나면 데모 애플리케이션이 실행을 시작합니다.

c. 직렬 터미널에서 실행 중인 애플리케이션의 상태를 볼 수 있습니다. 다음 그림은 터미널 출력의 일부를 보여줍니다.



MQTT 데모는 네 가지 주제(iotdemo/topic/*n*, 여기서 n=1~4)에 대한 메시지를 게시하고 해당 주제를 모두 구독하여 동일한 메시지를 다시 수신합니다. 메시지가 수신되면 데모는 해당 주제 iotdemo/acknowledgements에 대한 승인 메시지를 게시합니다. 다음 목록은 메시지 일련 번호에 대한 참조와 함께 터미널 출력에 나타나는 디버그 메시지를 설명합니다. 출력에서는 WICED Host Driver(WHD) 드라이버 세부 정보가 일련 번호 지정 없이 먼저 인쇄됩니다.

1. 1~4 - 디바이스가 구성된 액세스 포인트(AP)에 연결하고 구성된 엔드포인트 및 인증서를 사용하여 AWS 서버에 연결하여 프로비저닝됩니다.

2. 5~13 - coreMQTT 라이브러리가 초기화되고 디바이스가 MQTT 연결을 설정합니다.
3. 14~17 - 디바이스가 모든 주제를 구독하여 게시된 메시지를 다시 수신합니다.
4. 18~30 - 디바이스가 두 개의 메시지를 게시하고 메시지가 다시 수신되기를 기다립니다. 각 메시지가 수신되면 디바이스는 승인 메시지를 보냅니다.

모든 메시지가 게시될 때까지 동일한 게시, 수신 및 확인 주기가 계속됩니다. 구성된 수의 주기가 완료될 때까지 주기당 두 개의 메시지가 게시됩니다.

4. FreeRTOS에서 CMake 사용

CMake를 사용하여 데모 애플리케이션을 빌드하고 실행할 수도 있습니다. CMake 및 네이티브 빌드 시스템을 설정하려면 사전 조건 섹션을 참조하세요.

a. 빌드 파일을 생성할 때는 다음 명령을 사용합니다. `-DBOARD` 옵션을 사용하여 대상 보드를 지정합니다.

```
cmake -DVENDOR=cypress -DBOARD=CY8CKIT_064S0S2_4343W -DCOMPILER=arm-gcc -
S freertos -B build_dir
```

Windows를 사용하는 경우 `-G` 옵션을 사용하여 네이티브 빌드 시스템을 지정해야 합니다. CMake가 기본적으로 Visual Studio를 사용하기 때문입니다.

Example

```
cmake -DVENDOR=cypress -DBOARD=CY8CKIT_064S0S2_4343W -DCOMPILER=arm-gcc -
S freertos -B build_dir -G Ninja
```

`arm-none-eabi-gcc`가 셸 경로에 없으면 `AFR_TOOLCHAIN_PATH` CMake 변수도 설정해야 합니다.

Example

```
 -DAFR_TOOLCHAIN_PATH=/home/user/opt/gcc-arm-none-eabi/bin
```

b. CMake를 사용하여 프로젝트를 빌드하려면 다음 명령을 사용합니다.

```
cmake --build build_dir
```

c. 마지막으로 Cypress Programmer를 사용하여 *build_dir*에서 생성된 cm0.hex 및 cm4.hex 파일을 프로그래밍합니다.

5. 클라우드에서 MQTT 메시지 모니터링

AWS IoT 콘솔에서 MQTT 클라이언트를 사용하여 디바이스가 AWS 클라우드로 보내는 메시지를 모니터링할 수 있습니다. MQTT 클라이언트를 사용하여 AWS IoT MQTT 주제를 구독하려면 다음 단계를 따르세요.

a. AWS IoT 콘솔에 로그인합니다.

b. 탐색 창에서 Test(테스트)를 선택하여 MQTT 클라이언트를 엽니다.

c. 구독 주제에 ***your-thing-name*/example/topic**을 입력한 다음 주제 구독을 선택합니다.

d. 키트를 재설정하여 MQTT 메시지를 강제로 게시하면 콘솔 테스트 클라이언트에서 확인할 수 있습니다.

다른 데모 실행

다음 데모 애플리케이션은 현재 릴리스에서 작동하도록 테스트 및 검증되었습니다. *freertos*/demos 디렉터리에서 이러한 데모를 찾을 수 있습니다. 이러한 데모를 실행하는 방법에 대한 자세한 내용은 FreeRTOS 데모 섹션을 참조하세요.

- Bluetooth Low Energy 데모
- 무선(OTA) 업데이트 데모
- 보안 소켓 에코 클라이언트 데모
- AWS IoT 디바이스 섀도우 데모

디버깅

키트의 KitProg3는 SWD 프로토콜을 통한 디버깅을 지원합니다.

- FreeRTOS 애플리케이션을 디버깅하려면 작업 공간에서 aws_demos 프로젝트를 선택한 다음 퀵 패널에서 aws_demos 디버그(KitProg3)를 선택합니다.

OTA 업데이트

PSoC 64 MCU는 필요한 FreeRTOS 검증 테스트를 모두 통과했습니다. 그러나 PSoC 64 표준 보안 AWS 펌웨어 라이브러리에 구현over-the-air(OTA) 기능은 여전히 평가 보류 중입니다. 현재 구현된

OTA 기능은 aws_ota_test_case_rollback_if_unable_to_connect_after_update.py를 제외한 모든 OTA 검증 테스트를 통과했습니다.

PSoC64 Standard Secure – AWS MCU를 사용하여 성공적으로 검증된 OTA 이미지가 디바이스에 적용되고 디바이스와 통신할 수 없는 경우 AWS IoT Core디바이스는 원래 알려진 정상 이미지로 자동으로 롤백할 수 없습니다. 이로 인해 추가 업데이트를 AWS IoT Core 위해에서 디바이스에 연결할 수 없게 될 수 있습니다. 이 기능은 Cypress 팀에서 아직 개발 중입니다.

자세한 내용은 AWS 및 CY8CKIT-064S0S2-4343W 키트를 사용한 OTA 업데이트를 참조하세요. 추가 질문이 있거나 기술 지원이 필요한 경우 Cypress Developer Community에 문의하세요.

Windows 시뮬레이터를 사용하여 Microchip ATECC608A 보안 요소 시작하기

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

이 자습서에서는 Windows 시뮬레이터를 사용하여 Microchip ATECC608A 보안 요소를 시작하기 위한 지침을 제공합니다.

다음 하드웨어가 필요합니다.

- Microchip ATECC608A 보안 요소 클릭보드
- SAMD21 XPlained Pro
- mikroBUS Xplained Pro 어댑터

시작하기 전에 AWS IoT 및 FreeRTOS 다운로드를 구성하여 디바이스를 AWS 클라우드에 연결해야 합니다. 이 자습서에서는 FreeRTOS 다운로드 디렉터리의 경로를 *freertos*라고 합니다.

개요

이 자습서에 포함된 단계는 다음과 같습니다.

1. 보드를 호스트 시스템에 연결합니다.
2. 마이크로 컨트롤러 보드용 내장형 애플리케이션을 개발 및 디버깅하기 위한 소프트웨어를 호스트 시스템에 설치합니다.

3. FreeRTOS 데모 애플리케이션을 바이너리 이미지로 크로스 컴파일합니다.
4. 애플리케이션 이진 이미지를 보드에 로드한 다음 애플리케이션을 실행합니다.

Microchip ATECC608A 하드웨어 설정

Microchip ATECC608A 디바이스와 상호 작용하려면 먼저 SAMD21을 프로그래밍해야 합니다.

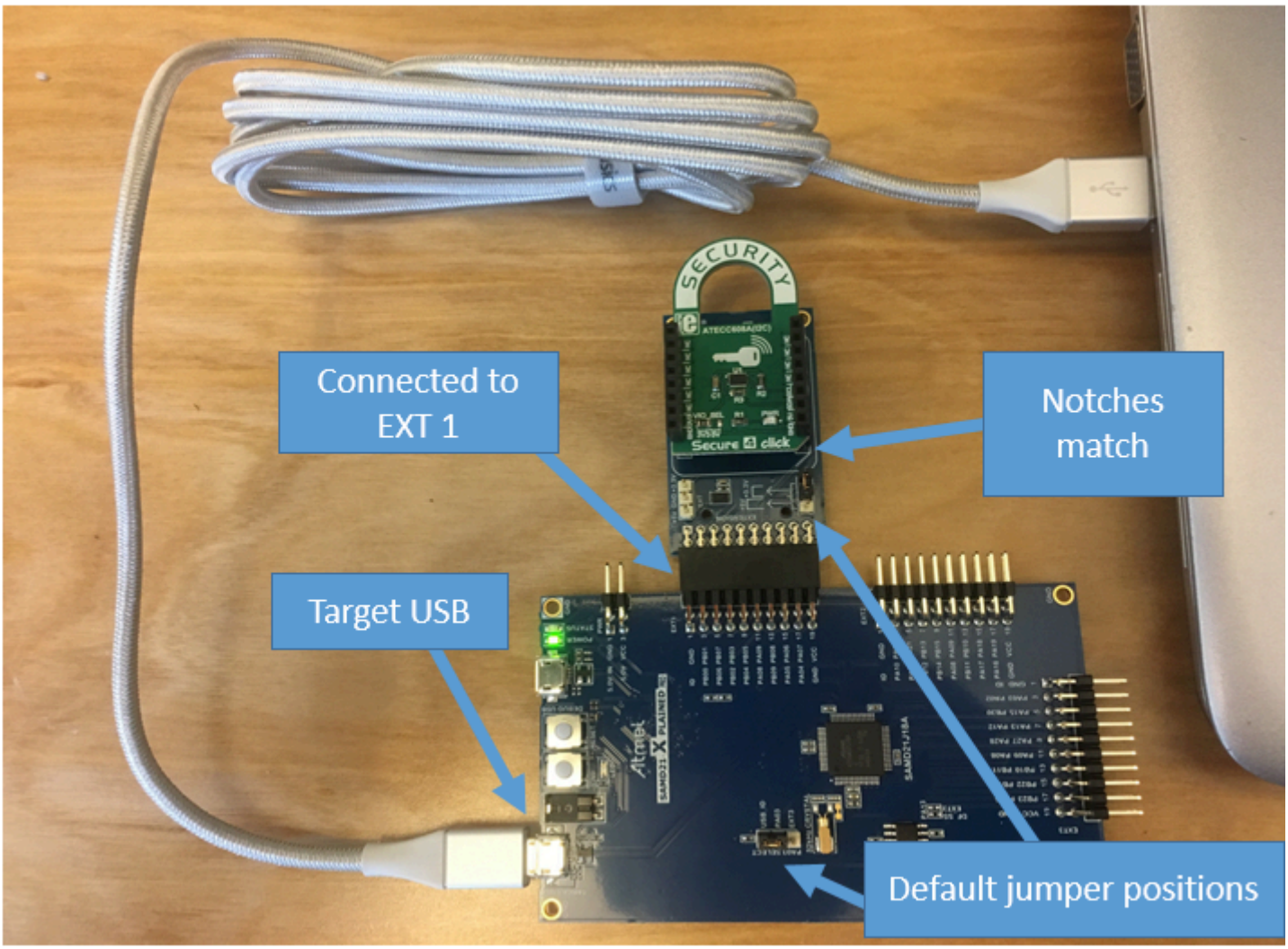
SAMD21 XPlained Pro 보드를 설정하려면

1. CryptoAuthSSH-XSTK(DM320109) - 최신 펌웨어 링크를 따라 지침(PDF)과 D21에 프로그래밍할 수 있는 바이너리가 들어 있는.zip 파일을 다운로드합니다.
2. Amtel Studio 7 IDP를 다운로드하여 설치합니다. 설치 중에 SMART ARM MCU 드라이버 아키텍처를 선택해야 합니다.
3. USB 2.0 마이크로 B 케이블을 사용하여 “디버그 USB” 커넥터를 컴퓨터에 연결하고 PDF의 지침을 따릅니다. (“디버그 USB”커넥터는 전원 LED 및 핀에 가장 가까운 USB 포트입니다.)

하드웨어를 연결하려면

1. 디버그 USB에서 마이크로 USB 케이블을 분리합니다.
2. mikroBUS XPlained Pro 어댑터를 EXT1 위치의 SAMD21 보드에 연결합니다.
3. ATECC608A Secure 4 Click 보드를 mikroBUSX XPlained Pro 어댑터에 연결합니다. 클릭 보드의 노치 모서리가 어댑터 보드의 노치 아이콘과 일치해야 합니다.
4. 마이크로 USB 케이블을 대상 USB에 연결합니다.

설정이 다음과 같아야 합니다.



개발 환경 설정

에 가입 AWS 계정

이 없는 경우 다음 단계를 AWS 계정완료하여 생성합니다.

에 가입하려면 AWS 계정

1. https://portal.aws.amazon.com/billing/signup을 엽니다.
2. 온라인 지시 사항을 따릅니다.

   가입 절차의 일부로는 전화 또는 문자 메시지를 수신하고 전화 키패드에 확인 코드를 입력하는 것이 포함됩니다.

에 가입하면 AWS 계정AWS 계정 루트 사용자이 생성됩니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스에 액세스할 권한이 있습니다. 보안 모범 사례는 사용자에게 관리 액세스 권한을 할당하고, 루트 사용자만 사용하여 루트 사용자 액세스 권한이 필요한 작업을 수행하는 것입니다.

AWS 는 가입 프로세스가 완료된 후 확인 이메일을 보냅니다. 언제든지 https://aws.amazon.com/으로 이동하고 내 계정을 선택하여 현재 계정 활동을 보고 계정을 관리할 수 있습니다.

관리자 액세스 권한이 있는 사용자 생성

에 가입한 후 일상적인 작업에 루트 사용자를 사용하지 않도록 관리 사용자를 AWS 계정보호 AWS IAM Identity Center, AWS 계정 루트 사용자활성화 및 생성합니다.

보안 AWS 계정 루트 사용자

1. 루트 사용자를 선택하고 AWS 계정 이메일 주소를 입력하여 계정 소유자AWS Management Console로에 로그인합니다. 다음 페이지에서 비밀번호를 입력합니다.

   루트 사용자를 사용하여 로그인하는 데 도움이 필요하면 AWS 로그인 User Guide의 루트 사용자로 로그인을 참조하세요.
2. 루트 사용자의 다중 인증(MFA)을 활성화합니다.

   지침은 IAM 사용 설명서의 AWS 계정 루트 사용자(콘솔)에 대한 가상 MFA 디바이스 활성화를 참조하세요.

관리자 액세스 권한이 있는 사용자 생성

1. IAM Identity Center를 활성화합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 AWS IAM Identity Center설정을 참조하세요.
2. IAM Identity Center에서 사용자에게 관리 액세스 권한을 부여합니다.

   를 자격 증명 소스 IAM Identity Center 디렉터리 로 사용하는 방법에 대한 자습서는 사용 AWS IAM Identity Center 설명서의 기본값으로 사용자 액세스 구성을 IAM Identity Center 디렉터리 참조하세요.

관리 액세스 권한이 있는 사용자로 로그인

- IAM IDentity Center 사용자로 로그인하려면 IAM Identity Center 사용자를 생성할 때 이메일 주소로 전송된 로그인 URL을 사용합니다.

  IAM Identity Center 사용자를 사용하여 로그인하는 데 도움이 필요하면 AWS 로그인 사용 설명서의 AWS 액세스 포털에 로그인을 참조하세요.

추가 사용자에게 액세스 권한 할당

1. IAM Identity Center에서 최소 권한 적용 모범 사례를 따르는 권한 세트를 생성합니다.

   지침은AWS IAM Identity Center 사용 설명서의 Create a permission set를 참조하세요.
2. 사용자를 그룹에 할당하고, 그룹에 Single Sign-On 액세스 권한을 할당합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 Add groups를 참조하세요.

액세스 권한을 제공하려면 사용자, 그룹 또는 역할에 권한을 추가하세요:

- 의 사용자 및 그룹 AWS IAM Identity Center:

  권한 세트를 생성합니다. AWS IAM Identity Center 사용 설명서의 권한 세트 생성의 지침을 따릅니다.
- 보안 인증 공급자를 통해 IAM에서 관리되는 사용자:

  ID 페더레이션을 위한 역할을 생성합니다. IAM 사용 설명서의 Create a role for a third-party identity provider (federation)의 지침을 따릅니다.
- IAM 사용자:
  - 사용자가 맡을 수 있는 역할을 생성합니다. IAM 사용 설명서에서 Create a role for an IAM user의 지침을 따릅니다.
  - (권장되지 않음)정책을 사용자에게 직접 연결하거나 사용자를 사용자 그룹에 추가합니다. IAM 사용 설명서에서 사용자(콘솔)에 권한 추가의 지침을 따르세요.

설정

1. GitHub에서 FreeRTOS를 다운로드하려면

   1. FreeRTOS GitHub 리포지토리를 찾습니다.

2. 코드를 선택하고 ZIP 복제 또는 다운로드를 선택합니다.
3. 컴퓨터 명령줄에서 호스트 컴퓨터의 디렉터리로 리포지토리를 복제합니다.

```
git clone https://github.com/aws/amazon-freertos.git --recurse-submodules
```

> **Important**
>
> - 이 주제에서는 FreeRTOS 다운로드 디렉터리의 경로를 *freertos*이라고 합니다.
> - *freertos* 경로의 공백 문자로 인해 빌드 실패가 발생할 수 있습니다. 리포지토리를 복제하거나 복사할 때 생성하는 경로에 공백 문자가 없어야 합니다.
> - Microsoft Windows에서 파일 경로의 최대 길이는 260자입니다. FreeRTOS 다운로드 디렉터리 경로가 길면 빌드 오류가 발생할 수 있습니다.

4. *freertos* 디렉터리에서 사용할 브랜치를 체크 아웃합니다.

2. 개발 환경 설정.

   a. 최신 버전의 WinPCap을 설치합니다.

   b. Microsoft Visual Studio를 설치합니다.

      Visual Studio 2017 및 2019 버전이 지원됩니다. 모든 Visual Studio 버전이 지원됩니다 (Community, Professional 또는 Enterprise).

      IDE 외에도 Desktop development with C++ 구성 요소를 설치합니다. 그런 다음 선택 사항에서 최신 Windows 10 SDK를 설치합니다.

   c. 활성 유선 이더넷 연결이 있는지 확인합니다.

FreeRTOS 데모 프로젝트 빌드 및 실행

> **Important**
>
> Microchip ATECC608A 디바이스에서는 `C_InitToken`을 호출하는 동안 처음 프로젝트가 실행될 때 디바이스에서 초기화가 1회 잠금 상태가 됩니다. 그러나 FreeRTOS 데모 프로젝트와 테스트 프로젝트의 구성은 다릅니다. 데모 프로젝트 구상 중 디바이스가 잠겨 있다면, 테스트 프로젝트의 모든 테스트를 성공할 수 없게 됩니다.

Visual Studio IDE를 사용하여 FreeRTOS 데모 프로젝트를 빌드하고 실행하려면

1. Visual Studio에서 프로젝트를 로드합니다.

   파일 메뉴에서 열기를 선택합니다. File/Solution(파일/솔루션)을 선택하고 *freertos*\projects\microchip\ecc608a_plus_winsim\visual_studio\aws_demos\aws_demos.sln 파일로 이동한 다음 열기를 선택합니다.

2. 데모 프로젝트의 목표를 재설정합니다.

   데모 프로젝트는 Windows SDK에 따라 달라지지만 Windows SDK 버전이 지정되어 있지 않습니다. 기본적으로 IDE에서 컴퓨터에 없는 SDK 버전으로 데모를 빌드하려고 시도할 수 있습니다. Windows SDK 버전을 설정하려면 aws_demos를 마우스 오른쪽 버튼으로 클릭한 후 Retarget Projects(프로젝트 대상 재지정)를 선택합니다. Review Solution Actions(솔루션 작업 검토) 창이 열립니다. 컴퓨터에 있는 Windows SDK 버전을 선택하고(드롭다운 목록에 먼저 나오는 값 사용) 확인을 선택합니다.

3. 프로젝트를 빌드 및 실행합니다.

   빌드 메뉴에서 솔루션 빌드를 선택하고 솔루션이 오류 없이 빌드되는지 확인합니다. Debug(디버깅), Start Debugging(디버깅 시작)을 선택하여 프로젝트를 실행합니다. 첫 번째 실행에서 디바이스 인터페이스를 구성하고 다시 컴파일해야 합니다. 자세한 내용은 네트워크 인터페이스 구성 단원을 참조하십시오.

4. Microchip ATECC608A를 프로비저닝합니다.

   마이크로칩은 ATECC608A 부품 설정에 도움이 되는 여러 스크립팅 도구를 제공합니다. *freertos*\vendors\microchip\secure_elements\app\example_trust_chain_tool로 이동하여 README.md 파일을 엽니다.

   README.md 파일의 지침에 따라 디바이스를 프로비저닝합니다. 이 단계에는 다음 사항이 포함됩니다.

   1. AWS에 인증 기관을 생성하고 등록합니다.
   2. Microchip ATECC608A에서 키를 생성하고 퍼블릭 키와 디바이스 일련 번호를 내보냅니다.
   3. 디바이스에 대한 인증서를 생성하고 해당 인증서를 AWS에 등록합니다.
   4. CA 인증서 및 디바이스 인증서를 디바이스에 로드합니다.

5. FreeRTOS 샘플을 빌드하고 실행합니다.

   데모 프로젝트를 다시 실행합니다. 이제 연결되어야 합니다!

문제 해결

일반적인 문제 해결 정보는 시작하기 문제 해결 단원을 참조하십시오.

Espressif ESP32-DevKitC 및 ESP-WROVER-KIT 시작하기

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

이 자습서에 따라 ESP32-WROOM-32, ESP32-SOLO-1 또는 ESP-WROVER 모듈 및 ESP-WROVER-KIT-VB가 장착된 Espressif ESP32-DevKitC를 시작합니다. AWS 파트너 디바이스 카탈로그의 파트너에서 구입하려면 다음 링크를 사용합니다.

- ESP32-WROOM-32 DevKitC
- ESP32-SOLO-1
- ESP32-WROVER-KIT

이러한 개발 보드 버전이 FreeRTOS에서 지원됩니다.

이러한 보드의 최신 버전에 대한 자세한 내용은 Espressif 웹 사이트에서 ESP32-DevKitC 또는 ESP-WROVER-KIT v4.1를 참조하세요.

> **Note**
>
> 현재 ESP32-WROVER-KIT 및 ESP DevKitC용 FreeRTOS 포트는 SMP(대칭적 다중 처리) 기능을 지원하지 않습니다.

개요

이 자습서에서는 다음과 같은 단계를 안내합니다.

1. 보드를 호스트 시스템에 연결합니다.
2. 마이크로컨트롤러 보드용 내장형 애플리케이션을 개발 및 디버깅하기 위한 소프트웨어를 호스트 시스템에 설치합니다.

3. FreeRTOS 데모 애플리케이션을 바이너리 이미지로 크로스 컴파일합니다.
4. 애플리케이션 바이너리 이미지를 보드에 로드한 후 애플리케이션을 실행합니다.
5. 모니터링 및 디버깅을 위해 직렬 연결로 보드에서 실행되는 애플리케이션과 상호 작용합니다.

사전 조건

Espressif 보드에서 FreeRTOS를 시작하기 전에 AWS 계정과 권한을 설정해야 합니다.

에 가입 AWS 계정

이 없는 경우 다음 단계를 AWS 계정완료하여 생성합니다.

에 가입하려면 AWS 계정

1. https://portal.aws.amazon.com/billing/signup을 엽니다.
2. 온라인 지시 사항을 따릅니다.

   가입 절차의 일부로는 전화 또는 문자 메시지를 수신하고 전화 키패드에 확인 코드를 입력하는 것이 포함됩니다.

   에 가입하면 AWS 계정AWS 계정 루트 사용자이 생성됩니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스에 액세스할 권한이 있습니다. 보안 모범 사례는 사용자에게 관리 액세스 권한을 할당하고, 루트 사용자만 사용하여 루트 사용자 액세스 권한이 필요한 작업을 수행하는 것입니다.

AWS 는 가입 프로세스가 완료된 후 확인 이메일을 보냅니다. 언제든지 https://aws.amazon.com/으로 이동하고 내 계정을 선택하여 현재 계정 활동을 보고 계정을 관리할 수 있습니다.

관리자 액세스 권한이 있는 사용자 생성

에 가입한 후 일상적인 작업에 루트 사용자를 사용하지 않도록 관리 사용자를 AWS 계정보호 AWS IAM Identity Center, AWS 계정 루트 사용자활성화 및 생성합니다.

보안 AWS 계정 루트 사용자

1. 루트 사용자를 선택하고 AWS 계정 이메일 주소를 입력하여 계정 소유자AWS Management Console로에 로그인합니다. 다음 페이지에서 비밀번호를 입력합니다.

   루트 사용자를 사용하여 로그인하는 데 도움이 필요하면 AWS 로그인 User Guide의 루트 사용자로 로그인을 참조하세요.

2. 루트 사용자의 다중 인증(MFA)을 활성화합니다.

   지침은 IAM 사용 설명서의 AWS 계정 루트 사용자(콘솔)에 대한 가상 MFA 디바이스 활성화를 참조하세요.

관리자 액세스 권한이 있는 사용자 생성

1. IAM Identity Center를 활성화합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 AWS IAM Identity Center설정을 참조하세요.

2. IAM Identity Center에서 사용자에게 관리 액세스 권한을 부여합니다.

   를 자격 증명 소스 IAM Identity Center 디렉터리 로 사용하는 방법에 대한 자습서는 사용 AWS IAM Identity Center 설명서의 기본값으로 사용자 액세스 구성을 IAM Identity Center 디렉터리 참조하세요.

관리 액세스 권한이 있는 사용자로 로그인

- IAM IDentity Center 사용자로 로그인하려면 IAM Identity Center 사용자를 생성할 때 이메일 주소로 전송된 로그인 URL을 사용합니다.

  IAM Identity Center 사용자를 사용하여 로그인하는 데 도움이 필요하면 사용 설명서의 AWS 액세스 포털에 로그인을 참조하세요. AWS 로그인

추가 사용자에게 액세스 권한 할당

1. IAM Identity Center에서 최소 권한 적용 모범 사례를 따르는 권한 세트를 생성합니다.

   지침은AWS IAM Identity Center 사용 설명서의 Create a permission set를 참조하세요.

2. 사용자를 그룹에 할당하고, 그룹에 Single Sign-On 액세스 권한을 할당합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 Add groups를 참조하세요.

액세스 권한을 제공하려면 사용자, 그룹 또는 역할에 권한을 추가하세요:

- 의 사용자 및 그룹 AWS IAM Identity Center:

권한 세트를 생성합니다. AWS IAM Identity Center 사용 설명서의 권한 세트 생성의 지침을 따릅니다.

- 보안 인증 공급자를 통해 IAM에서 관리되는 사용자:

  ID 페더레이션을 위한 역할을 생성합니다. IAM 사용 설명서의 Create a role for a third-party identity provider (federation)의 지침을 따릅니다.

- IAM 사용자:
  - 사용자가 맡을 수 있는 역할을 생성합니다. IAM 사용 설명서에서 Create a role for an IAM user의 지침을 따릅니다.
  - (권장되지 않음)정책을 사용자에게 직접 연결하거나 사용자를 사용자 그룹에 추가합니다. IAM 사용 설명서에서 사용자(콘솔)에 권한 추가의 지침을 따르세요.

> **Note**
>
> FreeRTOS 202012.00 릴리스는 ESP-IDF v3.3을 지원합니다. 최신 버전의 FreeRTOS를 사용하는 경우 ESP-IDF v4.2 시작하기에서 IDFv4.2 전용 지침을 따르세요.

ESP-IDF v3.3 시작하기

이 섹션에서는 Espressif 하드웨어에서 ESP-IDF v3.3을 사용하는 방법을 보여줍니다. ESP-IDF v4.2를 사용하려면 ESP-IDF v4.2 시작하기 섹션을 참조하세요.

Espressif 하드웨어 설정

ESP32-DevKitC 개발 보드 하드웨어를 설정하는 방법에 대한 정보는 ESP32-DevKitC V4 Getting Started Guide를 참조하세요.

ESP-WROVER-KIT 개발 보드 하드웨어를 설정하는 방법에 대한 정보는 ESP-WROVER-KIT V4.1 Getting Started Guide를 참조하세요.

> **Important**
>
> Espressif 안내서의 시작하기 섹션에 도달하면 여기서 멈추고 아래 단계를 따릅니다.

개발 환경 설정

보드와 통신하려면 도구 체인을 다운로드하여 설치해야 합니다.

도구 체인 설정

> **Note**
>
> ESP-IDF v3.3은 최신 버전의 ESP32 컴파일러를 지원하지 않습니다. 최신 버전의 ESP32 컴파일러를 이미 설치한 경우 이를 제거하고 도구 체인 설치에 포함된 호환 가능한 버전의 컴파일러를 사용해야 합니다. 이 섹션의 링크를 참조하세요. 컴파일러의 버전을 확인하려면 다음 명령을 실행합니다.
>
> ```
> xtensa-esp32-elf-gcc --version
> ```

도구 체인을 설정하려면 호스트 시스템의 운영 체제별 지침을 따릅니다.

- Standard Setup of Toolchain (ESP-IDF v3.3) for Windows
- Standard Setup of Toolchain (ESP-IDF v3.3) for macOS
- Standard Setup of Toolchain (ESP-IDF v3.3) for Linux

> **Important**
>
> 다음 단계의 'ESP-IDF 가져오기' 지침에 도달하면 여기서 멈추고 이 페이지의 지침으로 돌아옵니다.
> 방금 또는 이전에 다음 단계의 'ESP-IDF 가져오기' 지침을 따랐다면 계속하기 전에 시스템에서 `IDF_PATH` 환경 변수를 지워야 합니다. 'ESP-IDF 가져오기' 지침을 따랐다면 이 환경 변수가 자동으로 설정됩니다.

CMake 설치

FreeRTOS 데모를 빌드하고 이 디바이스에 대해 애플리케이션을 테스트하려면 CMake 빌드 시스템이 필요합니다. FreeRTOS는 버전 3.13 이상을 지원합니다.

CMake.org에서 CMake의 최신 버전을 다운로드할 수 있습니다. 소스 및 이진 배포를 둘 다 사용할 수 있습니다.

FreeRTOS에서 CMake를 사용하는 방법에 대한 자세한 내용은 FreeRTOS에서 CMake 사용 섹션을 참조하세요.

직렬 연결 설정

호스트 시스템과 ESP32-DevKitC 사이에 직렬 연결을 설정하려면 CP210x USB to UART Bridge VCP 드라이버를 설치해야 합니다. Silicon Labs에서 이러한 드라이버를 다운로드할 수 있습니다.

호스트 머신과 ESP32-WROVER-KIT 사이에 직렬 연결을 설정하려면 일부 FTDI 가상 COM 포트 드라이버를 설치해야 합니다. FTDI에서 이러한 드라이버를 다운로드할 수 있습니다.

자세한 내용은 ESP32와 직렬 연결 설정을 참조하십시오. 직렬 연결을 설정한 후 보드의 연결을 위한 직렬 포트를 기록해 두십시오. 데모를 빌드할 때 이 정보가 필요합니다.

FreeRTOS 다운로드 및 구성

환경을 설정한 후에는 GitHub에서 FreeRTOS를 다운로드할 수 있습니다. 자세한 지침은 GitHub 웹 사이트의 README.md 파일을 참조하세요.

FreeRTOS 데모 애플리케이션 구성

1. macOS 또는 Linux를 실행하는 경우 터미널 프롬프트를 엽니다. Windows를 실행하는 경우 `mingw32.exe`를 엽니다. MinGW는 네이티브 Microsoft Windows 애플리케이션을 위한 최소한의 개발 환경입니다.
2. Python 2.7.10 이상이 설치되어 있는지 확인하려면 다음을 실행합니다.

   ```
   python --version
   ```

   설치된 버전이 표시됩니다. Python 2.7.10 이상이 설치되어 있지 않으면 Python 웹 사이트에서 설치할 수 있습니다.
3. AWS IoT 명령을 실행하려면 AWS Command Line Interface (AWS CLI)가 필요합니다. Windows를 실행하는 경우 easy_install awscli를 사용하여 mingw32 환경에 AWS CLI 를 설치합니다.

   macOS 또는 Linux를 실행하는 경우 AWS CLI설치를 참조하세요.
4. 액세스 키 ID, 보안 AWS 액세스 키 aws configure 및 AWS 리전을 AWS CLI 사용하여를 실행하고 구성합니다. 자세한 내용은 AWS CLI구성 섹션을 참조하세요.
5. 다음 명령을 사용하여 AWS SDK for Python(boto3)을 설치합니다.

   - Windows에서는 mingw32 환경에서 다음을 실행합니다.

```
easy_install boto3
```

- macOS 또는 Linux에서는 다음을 실행합니다.

```
pip install tornado nose --user
```

계속해서 다음을 실행합니다.

```
pip install boto3 --user
```

FreeRTOS에는 AWS IoT에 연결하기 위해 Espressif 보드를 더 쉽게 설정할 수 있게 해주는 SetupAWS.py 스크립트가 포함되어 있습니다. 스크립트를 구성하려면 *freertos*/tools/aws_config_quick_start/configure.json을 열고 다음 속성을 설정합니다.

afr_source_dir

컴퓨터의 *freertos* 디렉터리에 대한 전체 경로입니다. 이 경로를 지정하기 위해 슬래시를 사용하고 있는지 확인합니다.

thing_name

보드를 나타내는 AWS IoT 사물에 할당하려는 이름입니다.

wifi_ssid

Wi-Fi 네트워크의 SSID입니다.

wifi_password

Wi-Fi 네트워크의 암호입니다.

wifi_security

Wi-Fi 네트워크의 보안 유형입니다.

유효한 보안 유형은 다음과 같습니다.

- eWiFiSecurityOpen(열림, 보안 없음)
- eWiFiSecurityWEP(WEP 보안)
- eWiFiSecurityWPA(WPA 보안)

- eWiFiSecurityWPA2(WPA2 보안)

구성 스크립트를 실행하려면

1. macOS 또는 Linux를 실행하는 경우 터미널 프롬프트를 엽니다. Windows를 실행하는 경우 mingw32.exe를 엽니다.
2. *freertos*/tools/aws_config_quick_start 디렉터리로 이동하여 다음을 실행합니다.

```
python SetupAWS.py setup
```

스크립트는 다음 작업을 수행합니다.

- IoT 사물, 인증서 및 정책을 생성합니다.
- 이 스크립트는 IoT 정책을 인증서에 연결하고 인증서를 AWS IoT 사물에 연결합니다.
- AWS IoT 엔드포인트, Wi-Fi SSID 및 자격 증명으로 aws_clientcredential.h 파일을 채웁니다.
- 인증서와 프라이빗 키에 형식을 지정하고 aws_clientcredential_keys.h 헤더 파일에 기록합니다.

> **Note**
>
> 인증서는 데모 용도로만 하드 코딩됩니다. 프로덕션 수준 애플리케이션은 이러한 파일을 보안 위치에 저장해야 합니다.

SetupAWS.py에 대한 자세한 내용은 *freertos*/tools/aws_config_quick_start 디렉터리의 README.md 파일을 참조하세요.

FreeRTOS 데모 프로젝트 빌드, 플래시 및 실행

CMake를 사용하여 빌드 파일을 생성하고, Make를 사용하여 애플리케이션 바이너리를 빌드하고, Espressif의 IDF 유틸리티를 사용하여 보드를 플래시할 수 있습니다.

Linux 및 MacOS에서 FreeRTOS 빌드(ESP-IDF v3.3)

Windows를 사용하는 경우 Windows에서 FreeRTOS 빌드(ESP-IDF v3.3) 섹션을 참조하세요.

CMake를 사용하여 빌드 파일을 생성한 다음 Make를 사용하여 애플리케이션을 빌드합니다.

CMake를 사용하여 데모 애플리케이션의 빌드 파일을 생성하려면

1. FreeRTOS 다운로드 디렉터리의 루트로 디렉터리를 변경합니다.
2. 다음 명령을 사용하여 빌드 파일을 생성합니다.

```
cmake -DVENDOR=espressif -DBOARD=esp32_wrover_kit -DCOMPILER=xtensa-esp32 -S . -
B your-build-directory
```

> **Note**
>
> 디버깅을 위해 애플리케이션을 빌드하려면 이 명령에 `-DCMAKE_BUILD_TYPE=Debug` 플래그를 추가합니다.
> 테스트 애플리케이션 빌드 파일을 생성하려면 `-DAFR_ENABLE_TESTS=1` 플래그를 추가합니다.
> Espressif에서 제공하는 코드는 경량 IP(lwIP) 스택을 기본 네트워킹 스택으로 사용합니다. 대신 FreeRTOS+TCP 네트워킹 스택을 사용하려면 CMake 명령에 `-DAFR_ESP_FREERTOS_TCP` 플래그를 추가합니다.
> 공급업체가 제공하지 않은 코드에 대한 lwIP 종속성을 추가하려면 사용자 지정 Wi-Fi 구성 요소에 대해 다음 행을 CMake 종속성 파일인 `CMakeLists.txt`에 추가합니다.
>
> ```
> # Add a dependency on the bluetooth espressif component to the common
>  component
> set(COMPONENT_REQUIRES lwip)
> ```

make를 사용하여 애플리케이션을 빌드하려면

1. 디렉터리를 `build` 디렉터리로 변경합니다.
2. Make를 사용하여 애플리케이션을 빌드하려면 다음 명령을 사용합니다.

```
make all -j4
```

> **Note**
>
> `aws_demos` 프로젝트와 `aws_tests` 프로젝트 간에 전환할 때마다 cmake 명령으로 빌드 파일을 생성해야 합니다.

Windows에서 FreeRTOS 빌드(ESP-IDF v3.3)

Windows에서는 CMake에 대한 빌드 생성기를 지정해야 합니다. 그렇지 않으면 CMake는 기본적으로 Visual Studio입니다. Espressif는 Windows, Linux 및 MacOS에서 작동하는 Ninja 빌드 시스템을 공식적으로 권장합니다. cmd 또는 PowerShell과 같은 기본 Windows 환경에서 CMake 명령을 실행해야 합니다. MSYS2 또는 WSL과 같은 가상 Linux 환경에서는 CMake 명령 실행이 지원되지 않습니다.

CMake를 사용하여 빌드 파일을 생성한 다음 Make를 사용하여 애플리케이션을 빌드합니다.

CMake를 사용하여 데모 애플리케이션의 빌드 파일을 생성하려면

1. FreeRTOS 다운로드 디렉터리의 루트로 디렉터리를 변경합니다.
2. 다음 명령을 사용하여 빌드 파일을 생성합니다.

```
cmake -DVENDOR=espressif -DBOARD=esp32_wrover_kit -DCOMPILER=xtensa-esp32 -GNinja -
S . -B build-directory
```

> **Note**
>
> 디버깅을 위해 애플리케이션을 빌드하려면 이 명령에 `-DCMAKE_BUILD_TYPE=Debug` 플래그를 추가합니다.
> 테스트 애플리케이션 빌드 파일을 생성하려면 `-DAFR_ENABLE_TESTS=1` 플래그를 추가합니다.
> Espressif에서 제공하는 코드는 경량 IP(lwIP) 스택을 기본 네트워킹 스택으로 사용합니다. 대신 FreeRTOS+TCP 네트워킹 스택을 사용하려면 CMake 명령에 `-DAFR_ESP_FREERTOS_TCP` 플래그를 추가합니다.
> 공급업체가 제공하지 않은 코드에 대한 lwIP 종속성을 추가하려면 사용자 지정 Wi-Fi 구성 요소에 대해 다음 행을 CMake 종속성 파일인 `CMakeLists.txt`에 추가합니다.
>
> ```
> # Add a dependency on the bluetooth espressif component to the common
>  component
> set(COMPONENT_REQUIRES lwip)
> ```

애플리케이션을 빌드하려면

1. 디렉터리를 `build` 디렉터리로 변경합니다.
2. Ninja를 간접 호출하여 애플리케이션을 빌드합니다.

```
ninja
```

또는 일반 CMake 인터페이스를 사용하여 애플리케이션을 빌드합니다.

```
cmake --build build-directory
```

> **Note**
>
> aws_demos 프로젝트와 aws_tests 프로젝트 간에 전환할 때마다 cmake 명령으로 빌드 파일을 생성해야 합니다.

Freertos 플래시 및 실행(ESP-IDF v3.3)

이 섹션에서는 Espressif 하드웨어에서 ESP-IDF v3.3을 사용하는 방법을 보여줍니다. ESP-IDF v4.2를 사용하려면 ESP-IDF v4.2 시작하기 섹션을 참조하세요.

Espressif의 IDF 유틸리티(*freertos*/vendors/espressif/esp-idf/tools/idf.py)를 사용하여 보드를 플래시하고 애플리케이션을 실행하고 로그를 확인합니다.

보드의 플래시를 지우려면 *freertos* 디렉터리로 이동하여 다음 명령을 사용합니다.

```
./vendors/espressif/esp-idf/tools/idf.py erase_flash -B build-directory
```

애플리케이션 바이너리를 보드에 플래시하려면 make를 사용합니다.

```
make flash
```

IDF 스크립트를 사용하여 보드를 플래시할 수도 있습니다.

```
./vendors/espressif/esp-idf/tools/idf.py flash -B build-directory
```

모니터링하려면 다음 명령을 사용합니다.

```
./vendors/espressif/esp-idf/tools/idf.py monitor -p /dev/ttyUSB1 -B build-directory
```

> **Note**
>
> 이러한 명령을 결합할 수 있습니다. 예시:
>
> ```
> ./vendors/espressif/esp-idf/tools/idf.py erase_flash flash monitor -p /dev/
> ttyUSB1 -B build-directory
> ```

ESP-IDF v4.2 시작하기

이 섹션에서는 Espressif 하드웨어에서 ESP-IDF v4.2를 사용하는 방법을 보여줍니다. ESP-IDF v3.3을 사용하려면 ESP-IDF v3.3 시작하기 섹션을 참조하세요.

> **Note**
>
> 이 자습서의 Linux 명령을 사용하려면 Bash 쉘을 사용해야 합니다.

Espressif 하드웨어 설정

ESP32-DevKitC 개발 보드 하드웨어를 설정하는 방법에 대한 정보는 ESP32-DevKitC V4 Getting Started Guide를 참조하세요.

ESP-WROVER-KIT 개발 보드 하드웨어를 설정하는 방법에 대한 정보는 ESP-WROVER-KIT V4.1 Getting Started Guide를 참조하세요.

> **Important**
>
> Espressif 안내서의 다음 단계 섹션에 도달하면 여기서 멈추고 이 페이지의 지침으로 돌아갑니다.

개발 환경 설정

보드와 통신하려면 도구 체인을 다운로드하여 설치해야 합니다.

도구 체인 설정

도구 체인을 설정하려면 호스트 시스템의 운영 체제별 지침을 따릅니다.

- Standard Setup of Toolchain and prerequisites (ESP-IDF v4.2) for Windows

- Standard Setup of Toolchain prerequisites (ESP-IDF v4.2) for macOS
- Standard Setup of Toolchain prerequisites (ESP-IDF v4.2) for Linux

> **Important**
>
> 다음 단계의 'ESP-IDF 가져오기' 지침에 도달하면 여기서 멈추고 이 페이지의 지침으로 돌아옵니다.

설치 완료(Linux/MacOS)

ESP-IDF Windows 설치 관리자는 필요한 도구를 모두 설치합니다. Linux 및 MacOS 플랫폼에서 설치를 완료하려면 추가 단계가 필요합니다.

1. 명령줄 창을 엽니다.
2. FreeRTOS 다운로드 디렉터리로 이동한 후, 다음 스크립트를 실행하여 플랫폼에 해당하는 espressif 도구 체인을 다운로드하고 설치합니다.

```
vendors/espressif/esp-idf/install.sh
```

3. 다음 명령을 사용하여 터미널 경로에 ESP-IDF 도구 체인 도구를 추가합니다.

```
source vendors/espressif/esp-idf/export.sh
```

직렬 연결 설정

호스트 시스템과 ESP32-DevKitC 사이에 직렬 연결을 설정하려면 CP210x USB to UART Bridge VCP 드라이버를 설치해야 합니다. Silicon Labs에서 이러한 드라이버를 다운로드할 수 있습니다.

호스트 시스템과 ESP32-WROVER-KIT 사이에 직렬 연결을 설정하려면 FTDI 가상 COM 포트 드라이버를 설치해야 합니다. FTDI에서 이 드라이버를 다운로드할 수 있습니다.

자세한 내용은 ESP32와 직렬 연결 설정을 참조하십시오. 직렬 연결을 설정한 후 보드의 연결을 위한 직렬 포트를 기록해 두십시오. 데모를 빌드할 때 이 정보가 필요합니다.

FreeRTOS 다운로드 및 구성

환경을 설정한 후에는 GitHub에서 FreeRTOS를 다운로드할 수 있습니다. 자세한 지침은 GitHub 웹 사이트의 README.md 파일을 참조하세요.

FreeRTOS 데모 애플리케이션 구성

1. macOS 또는 Linux를 실행하는 경우 터미널 프롬프트를 엽니다. Windows를 실행하는 경우 mingw32.exe를 엽니다. (MinGW는 네이티브 Microsoft Windows 애플리케이션을 위한 최소한의 개발 환경입니다.)
2. Python3이 설치되어 있는지 확인하려면 다음을 실행합니다.

   ```
   python --version
   ```

   설치된 버전이 표시됩니다. Python 3.0.1 이상이 설치되어 있지 않으면 Python 웹 사이트에서 설치할 수 있습니다.
3. AWS IoT 명령을 실행하려면 AWS Command Line Interface (AWS CLI)가 필요합니다. Windows를 실행하는 경우 easy_install awscli를 사용하여 mingw32 환경에 AWS CLI 를 설치합니다.

   macOS 또는 Linux를 실행하는 경우 AWS CLI설치를 참조하세요.
4. Run

   ```
   aws configure
   ```

   액세스 키 ID, 보안 AWS 액세스 키 및 기본 AWS 리전을 AWS CLI 사용하여를 구성합니다. 자세한 내용은 AWS CLI구성 섹션을 참조하세요.
5. 다음 명령을 사용하여 AWS SDK for Python(boto3)을 설치합니다.

   - Windows에서는 mingw32 환경에서 다음을 실행합니다.

     ```
     easy_install boto3
     ```

   - macOS 또는 Linux에서는 다음을 실행합니다.

     ```
     pip install tornado nose --user
     ```

     계속해서 다음을 실행합니다.

     ```
     pip install boto3 --user
     ```

FreeRTOS에는 AWS IoT에 연결하기 위해 Espressif 보드를 더 쉽게 설정할 수 있게 해주는 SetupAWS.py 스크립트가 포함되어 있습니다. 스크립트를 구성하려면 *freertos*/tools/aws_config_quick_start/configure.json을 열고 다음 속성을 설정합니다.

afr_source_dir

컴퓨터의 *freertos* 디렉터리에 대한 전체 경로입니다. 이 경로를 지정하기 위해 슬래시를 사용하고 있는지 확인합니다.

thing_name

보드를 나타내는 AWS IoT 사물에 할당하려는 이름입니다.

wifi_ssid

Wi-Fi 네트워크의 SSID입니다.

wifi_password

Wi-Fi 네트워크의 암호입니다.

wifi_security

Wi-Fi 네트워크의 보안 유형입니다.

유효한 보안 유형은 다음과 같습니다.

- eWiFiSecurityOpen(열림, 보안 없음)
- eWiFiSecurityWEP(WEP 보안)
- eWiFiSecurityWPA(WPA 보안)
- eWiFiSecurityWPA2(WPA2 보안)

구성 스크립트를 실행하려면

1. macOS 또는 Linux를 실행하는 경우 터미널 프롬프트를 엽니다. Windows를 실행하는 경우 mingw32.exe 파일을 엽니다.
2. *freertos*/tools/aws_config_quick_start 디렉터리로 이동하여 다음을 실행합니다.

```
python SetupAWS.py setup
```

스크립트는 다음 작업을 수행합니다.

- IoT 사물, 인증서 및 정책을 생성합니다.
- 이 스크립트는 IoT 정책을 인증서에 연결하고 인증서를 AWS IoT 사물에 연결합니다.
- AWS IoT 엔드포인트, Wi-Fi SSID 및 자격 증명으로 `aws_clientcredential.h` 파일을 채웁니다.
- 인증서와 프라이빗 키에 형식을 지정하고 `aws_clientcredential_keys.h` 헤더 파일에 기록합니다.

> **Note**
>
> 인증서는 데모 용도로만 하드 코딩됩니다. 프로덕션 수준 애플리케이션은 이러한 파일을 보안 위치에 저장해야 합니다.

`SetupAWS.py`에 대한 자세한 내용은 *`freertos`*`/tools/aws_config_quick_start` 디렉터리의 `README.md` 파일을 참조하세요.

idf.py 스크립트를 사용하여 FreeRTOS 데모 프로젝트 빌드, 플래시 및 실행

Espressif의 IDF 유틸리티를 사용하여 빌드 파일을 생성하고, 애플리케이션 바이너리를 빌드하고, 보드를 플래시할 수 있습니다.

Windows, Linux 및 MacOS에서 FreeRTOS 빌드 및 플래시(ESP-IDF v4.2)

`idf.py` 스크립트를 사용하여 프로젝트를 빌드하고 바이너리를 디바이스에 플래시합니다.

> **Note**
>
> 다음 예제와 같이 일부 설정에서는 `idf.py`에서 포트 옵션 "`-p` *`port-name`*"을 사용하여 올바른 포트를 지정해야 할 수도 있습니다.
>
> ```
> idf.py -p /dev/cu.usbserial-00101301B flash
> ```

프로젝트를 빌드하고 플래시하려면

1. FreeRTOS 다운로드 디렉터리의 루트로 이동합니다.
2. 명령줄 창에서 다음 명령을 입력하여 ESP-IDF 도구를 터미널의 경로에 추가합니다.

   Windows

```
vendors\espressif\esp-idf\export.bat
```

Linux/MacOS

```
source vendors/espressif/esp-idf/export.sh
```

3. build 디렉터리에서 cmake를 구성하고 다음 명령을 사용하여 펌웨어 이미지를 빌드합니다.

```
idf.py -DVENDOR=espressif -DBOARD=esp32_wrover_kit -DCOMPILER=xtensa-esp32 build
```

그러면 다음과 같은 결과가 표시됩니다.

```
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'
```

오류가 없는 경우 빌드는 펌웨어 바이너리 .bin 파일을 생성합니다.

4. 다음 명령을 사용하여 개발 보드의 플래시 메모리를 지웁니다.

```
idf.py erase_flash
```

5. idf.py 스크립트를 사용하여 애플리케이션 바이너리를 보드에 플래시합니다.

```
idf.py flash
```

6. 다음 명령을 사용하여 보드의 직렬 포트 출력을 모니터링합니다.

```
idf.py monitor
```

> **Note**
>
> 다음 예제와 같이 이들 명령을 결합할 수 있습니다.
>
> ```
> idf.py erase_flash flash monitor
> ```

> **Note**
>
> 특정 호스트 시스템 설정의 경우 다음 예제와 같이 보드를 플래시할 때 포트를 지정해야 합니다.
>
> ```
> idf.py erase_flash flash monitor -p /dev/ttyUSB1
> ```

CMake를 사용하여 FreeRTOS 빌드 및 플래시

IDF SDK에서 제공하는 `idf.py` 스크립트를 사용하여 코드를 빌드하고 실행하는 방법 외에도 CMake를 사용하여 프로젝트를 빌드할 수도 있습니다. 현재는 Unix Makefile 또는 Ninja 빌드 시스템을 지원합니다.

프로젝트를 빌드하고 플래시하려면

1. 명령줄 창에서 FreeRTOS 다운로드 디렉터리의 루트로 이동합니다.
2. 다음 스크립트를 실행하여 ESP-IDF 도구를 쉘의 경로에 추가합니다.

   Windows

```
vendors\espressif\esp-idf\export.bat
```

Linux/MacOS

```
source vendors/espressif/esp-idf/export.sh
```

3. 다음 명령을 입력하여 빌드 파일을 생성합니다.

   Unix Makefile 사용

   ```
   cmake -DVENDOR=espressif -DBOARD=esp32_wrover_kit -DCOMPILER=xtensa-esp32 -S . -
   B ./YOUR_BUILD_DIRECTORY -DAFR_ENABLE_ALL_MODULES=1 -DAFR_ENABLE_TESTS=0
   ```

   Ninja 사용

   ```
   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. 플래시를 지운 다음 보드를 플래시합니다.

   Unix Makefile 사용

   ```
   make -C ./YOUR_BUILD_DIRECTORY erase_flash
   ```

   ```
   make -C ./YOUR_BUILD_DIRECTORY flash
   ```

   Ninja 사용

   ```
   ninja -C ./YOUR_BUILD_DIRECTORY erase_flash
   ```

   ```
   ninja -C ./YOUR_BUILD_DIRECTORY flash
   ```

클라우드에서 MQTT 메시지 모니터링

AWS IoT 콘솔에서 MQTT 클라이언트를 사용하여 디바이스가 AWS 클라우드로 보내는 메시지를 모니터링할 수 있습니다.

MQTT 클라이언트를 사용하여 AWS IoT MQTT 주제를 구독하려면

1. AWS IoT 콘솔로 이동합니다.

2. 탐색 창에서 Test(테스트)를 선택하여 MQTT 클라이언트를 엽니다.
3. 구독 주제에 ***your-thing-name*/example/topic**을 입력한 다음 주제 구독을 선택합니다.

Bluetooth Low-Energy 데모 실행

FreeRTOS는 Bluetooth Low Energy 연결을 지원합니다.

Bluetooth Low Energy에서 FreeRTOS 데모 프로젝트를 실행하려면 iOS 또는 Android 모바일 디바이스에서 FreeRTOS Bluetooth Low Energy Mobile SDK 데모 애플리케이션을 실행해야 합니다.

FreeRTOS Bluetooth Low Energy Mobile SDK 데모 애플리케이션을 설정하려면

1. FreeRTOS Bluetooth 디바이스용 Mobile SDK에 있는 지침에 따라 호스트 컴퓨터에 모바일 플랫폼용 SDK를 다운로드하고 설치합니다.
2. FreeRTOS Bluetooth Low Energy Mobile SDK 데모 애플리케이션에 있는 지침에 따라 모바일 디바이스에 데모 모바일 애플리케이션을 설정합니다.

보드에서 MQTT over Bluetooth Low Energy 데모를 실행하는 방법에 대한 자세한 내용은 MQTT over Bluetooth Low Energy 데모 애플리케이션을 참조하십시오.

보드에서 Wi-Fi 프로비저닝 데모를 실행하는 방법에 대한 자세한 내용은 Wi-Fi 프로비저닝 데모 애플리케이션을 참조하십시오.

ESP32를 위한 자체 CMake 프로젝트에서 FreeRTOS 사용

자체 CMake 프로젝트에서 FreeRTOS를 사용하려면 하위 디렉터리로 설정하고 애플리케이션과 함께 빌드할 수 있습니다. 먼저 GitHub에서 FreeRTOS의 사본을 가져옵니다. 다음 명령을 사용하여 Git 하위 모듈로 설정할 수도 있습니다, 그러면 나중에 업데이트하기가 더 쉽습니다.

```
git submodule add -b release https://github.com/aws/amazon-freertos.git freertos
```

최신 버전이 릴리스된 경우 다음 명령을 사용하여 로컬 복사본을 업데이트할 수 있습니다.

```
# 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"
```

프로젝트에 다음과 같은 디렉터리 구조가 있는 경우

```
- freertos (the copy that you obtained from GitHub or the AWS IoT console)
- src
  - main.c (your application code)
- CMakeLists.txt
```

다음은 FreeRTOS와 함께 애플리케이션을 빌드하는 데 사용할 수 있는 최상위 `CMakeLists.txt` 파일의 예제입니다.

ESP-IDF v3.3

```
cmake_minimum_required(VERSION 3.13)

project(freertos_examples)

add_executable(my_app src/main.c)

# Tell IDF build to link against this target.
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::mqtt)
```

ESP-IDF v4.2

```
cmake_minimum_required(VERSION 3.13)

project(freertos_examples)

# Pull in ESP-IDF defined CMake build infrastructure logic to configure the project,
 discover all the components, etc
set(esp_idf_dir "${CMAKE_CURRENT_LIST_DIR}/freertos/vendors/espressif/esp-idf")
```

```
include(${esp_idf_dir}/tools/cmake/idf.cmake)

# Tell IDF build to link against this target.
set(IDF_PROJECT_EXECUTABLE my_app)
get_filename_component(
    IDF_EXECUTABLE_SRCS
    "src/main.c" ABSOLUTE
    )
```

프로젝트를 빌드하려면 다음 CMake 명령을 실행합니다. ESP32 컴파일러가 PATH 환경 변수에 있는지 확인하십시오.

```
cmake -S . -B build-directory -DCMAKE_TOOLCHAIN_FILE=freertos/tools/cmake/toolchains/
xtensa-esp32.cmake -GNinja
```

```
cmake --build build
```

애플리케이션을 보드에 플래시하려면 다음 명령을 실행합니다.

```
cmake --build build-directory --target flash
```

FreeRTOS의 구성 요소 사용

CMake를 실행한 후에는 요약 출력에서 사용 가능한 모든 구성 요소를 찾을 수 있습니다. 다음 예제와 비슷합니다.

```
====================Configuration for FreeRTOS====================
  Version:                  201910.00
  Git version:              201910.00-388-gcb3612cb7

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:         ble, ble_hal, ble_wifi_provisioning, common, crypto,
 defender,
                            dev_mode_key_provisioning, freertos_plus_tcp, greengrass,
                            https, kernel, mqtt, ota, pkcs11, pkcs11_implementation,
                            platform, secure_sockets, serializer, shadow, tls, wifi
  Enabled by user:          ble, ble_hal, ble_wifi_provisioning, defender, greengrass,
                            https, mqtt, ota, pkcs11, pkcs11_implementation, platform,
                            secure_sockets, shadow, wifi
  Enabled by dependency:    common, crypto, demo_base, dev_mode_key_provisioning,
                            freertos, freertos_plus_tcp, kernel, pkcs11_mbedtls,
                            secure_sockets_freertos_plus_tcp, serializer, tls, utils
  3rdparty dependencies:    http_parser, jsmn, mbedtls, pkcs11, tinycbor
  Available demos:          demo_ble, demo_ble_numeric_comparison, demo_defender,
                            demo_greengrass_connectivity, demo_https, demo_mqtt,
 demo_ota,
                            demo_shadow, demo_tcp, demo_wifi_provisioning
  Available tests:
==============================================================================
```

`Modules to build` 목록에서 모든 구성 요소를 참조할 수 있습니다. 애플리케이션에 연결하려면 이름 앞에 `AFR::` 네임스페이스를 추가합니다(예: `AFR::mqtt`, `AFR::ota`, 등).

ESP-IDF에 사용자 지정 구성 요소 추가

ESP-IDF 빌드 환경에 더 많은 구성 요소를 추가할 수 있습니다. 예를 들어, `example_component`라는 구성 요소를 추가하려는 경우 프로젝트는 다음과 같습니다.

```
- freertos
- components
  - example_component
    - include
      - example_component.h
    - src
      - example_component.c
    - CMakeLists.txt
- src
  - main.c
```

```
- CMakeLists.txt
```

다음은 구성 요소에 대한 `CMakeLists.txt` 파일 예제입니다.

ESP-IDF v3.3

```
# include paths of this components.
set(COMPONENT_ADD_INCLUDEDIRS include)

# source files of this components.
set(COMPONENT_SRCDIRS src)
# Alternatively, use COMPONENT_SRCS to specify source files explicitly
# set(COMPONENT_SRCS src/example_component.c)

# add this components, this will define a CMake library target.
register_component()
```

ESP-IDF v4.2

```
# Use idf_component_register to add this component, this will define a CMake library
 target.
# Specify component sources in SRCS.
# Specify component include directories in INCLUDE_DIRS.
idf_component_register(SRCS "src/foo.c"
                       INCLUDE_DIRS "include")
# Standard CMake function can be used to specify dependencies. ${COMPONENT_TARGET}
 is defined
# from esp-idf when you call register_component, by default it's
 idf_component_<folder_name>.
target_link_libraries(${COMPONENT_TARGET} PRIVATE AFR::mqtt)
```

표준 CMake 함수 `target_link_libraries`를 사용하여 종속성을 지정할 수도 있습니다. 구성 요소의 대상 이름은 ESP-IDF에 의해 정의된 `COMPONENT_TARGET` 변수에 저장됩니다.

ESP-IDF v3.3

```
# add this component, this will define a CMake library target.
register_component()

# standard CMake function can be used to specify dependencies. ${COMPONENT_TARGET}
 is defined
```

```
# from esp-idf when you call register_component, by default it's
 idf_component_<folder_name>.
target_link_libraries(${COMPONENT_TARGET} PRIVATE AFR::mqtt)
```

ESP-IDF v4.2

```
# add this component, this will define a CMake library target.
idf_component_register(SRCS "src/foo.c"
                       INCLUDE_DIRS "include")

# standard CMake function can be used to specify dependencies. ${COMPONENT_TARGET}
 is defined
# from esp-idf when you call register_component, by default it's
 idf_component_<folder_name>.
target_link_libraries(${COMPONENT_TARGET} PRIVATE AFR::mqtt)
```

ESP 구성 요소의 경우, COMPONENT_REQUIRES 및 COMPONENT_PRIV_REQUIRES라는 두 개의 변수를 설정하여 수행됩니다. ESP-IDF Programming Guide v4.2의 Build System (CMake)을 참조하세요.

ESP-IDF v3.3

```
# If the dependencies are from ESP-IDF, use these 2 variables. Note these need to be
# set before calling register_component().
set(COMPONENT_REQUIRES log)
set(COMPONENT_PRIV_REQUIRES lwip)
```

ESP-IDF v4.2

```
# If the dependencies are from ESP-IDF, use these 2 variables (REQUIRES,
 PRIV_REQUIRES)
#   in the idf_component_register call: (Refer here for information on the
#   idf_component_register function: https://docs.espressif.com/projects/esp-idf/en/
v4.2/esp32/api-guides/build-system.html#cmake-component-register)
idf_component_register(SRCS "src/foo.c"
                       INCLUDE_DIRS "include"
                       REQUIRES log
                       PRIV_REQUIRES lwip)
```

그런 다음 최상위 CMakeLists.txt 파일에서 ESP-IDF에 이러한 구성 요소를 찾을 위치를 알려줍니다. add_subdirectory(freertos) 앞에 다음 줄을 추가합니다.

ESP-IDF v3.3

```
# 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})
```

ESP-IDF v4.2

```
# 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
)
idf_build_component(${EXTRA_COMPONENT_DIRS})
```

이 구성 요소는 이제 기본적으로 애플리케이션 코드에 자동 연결됩니다. 헤더 파일을 포함하고 정의된 함수를 호출할 수 있어야 합니다.

FreeRTOS 구성 재정의

현재 FreeRTOS 소스 트리 외부에서 구성을 재정의하기 위한 잘 정의된 접근 방식은 없습니다. 기본적으로 CMake는 *freertos*/vendors/espressif/boards/esp32/aws_demos/config_files/ 및 *freertos*/demos/include/ 디렉터리를 찾습니다. 그러나 해결 방법을 사용하여 컴파일러가 다른 디렉터리를 먼저 검색하도록 지시할 수 있습니다. 예를 들어, FreeRTOS 구성에 다른 폴더를 추가할 수 있습니다.

```
- freertos
- freertos-configs
  - aws_clientcredential.h
  - aws_clientcredential_keys.h
  - iot_mqtt_agent_config.h
  - iot_config.h
- components
- src
- CMakeLists.txt
```

freertos-configs의 파일은 *freertos*/vendors/espressif/boards/esp32/aws_demos/config_files/ 및 *freertos*/demos/include/ 디렉터리에서 복사됩니다. 그런 다음 최상위 CMakeLists.txt 파일에서 컴파일러가 이 디렉터리를 먼저 검색하도록 add_subdirectory(freertos) 앞에 이 줄을 추가합니다.

```
include_directories(BEFORE freertos-configs)
```

ESP-IDF를 위한 자체 sdkconfig 제공

자체 sdkconfig.default를 제공하려는 경우 명령줄에서 CMake 변수 IDF_SDKCONFIG_DEFAULTS를 설정할 수 있습니다.

```
cmake -S . -B build-directory -DIDF_SDKCONFIG_DEFAULTS=path_to_your_sdkconfig_defaults
 -DCMAKE_TOOLCHAIN_FILE=freertos/tools/cmake/toolchains/xtensa-esp32.cmake -GNinja
```

자체 sdkconfig.default 파일에 대한 위치를 지정하지 않으면 FreeRTOS가 *freertos*/vendors/espressif/boards/esp32/aws_demos/sdkconfig.defaults에 위치한 기본 파일을 사용합니다.

요약

example_component라는 구성 요소가 있는 프로젝트에서 일부 구성을 재정의하려는 경우 여기에 최상위 CMakeLists.txt 파일의 전체 예는 다음과 같습니다.

ESP-IDF v3.3

```
cmake_minimum_required(VERSION 3.13)

project(freertos_examples)

add_executable(my_app 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::mqtt)
```

ESP-IDF v4.2

```
cmake_minimum_required(VERSION 3.13)

project(freertos_examples)

# Pull in ESP-IDF defined CMake build infrastructure logic to configure the project,
 discover all the components, etc
set(esp_idf_dir "${CMAKE_CURRENT_LIST_DIR}/freertos/vendors/espressif/esp-idf")
include(${esp_idf_dir}/tools/cmake/idf.cmake)
# Tell IDF build to link against this target.
set(IDF_PROJECT_EXECUTABLE my_app)
get_filename_component(
    IDF_EXECUTABLE_SRCS
    "src/main.c" ABSOLUTE
    )
# Add some extra components. IDF_EXTRA_COMPONENT_DIRS is an variable used by ESP-IDF
# to collect extra components.
get_filename_component(
    EXTRA_COMPONENT_DIRS
    "components/foo" ABSOLUTE
)
idf_build_component(${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::mqtt)
```

CMake를 사용한 Espressif 빌드 시스템에 관한 자세한 내용은 Espressif API 안내서의 Build System을 참조하세요.

문제 해결

- macOS를 실행하는데 운영 체제에서 ESP-WROVER-KIT를 인식하지 못하면 D2XX 드라이버가 설치되어 있지 않은지 확인합니다. 이를 제거하려면 macOS X용 FTDI 드라이버 설치 가이드의 지침을 따르십시오.
- ESP-IDF에서 제공하는 모니터 유틸리티(및 make monitor를 사용하여 호출)를 사용하면 주소를 디코딩할 수 있습니다. 이러한 이유로 이 유틸리티는 애플리케이션이 작동을 멈추는 경우 의미 있는 역추적을 얻는 데 도움이 될 수 있습니다. 자세한 내용은 Espressif 웹사이트의 Automatic Address Decoding을 참조하세요.
- 특수 JTAG 하드웨어를 사용할 필요 없이 gdb와 통신하기 위해 GDBstub를 활성화할 수도 있습니다. 자세한 내용은 Espressif 웹사이트의 Launching GDB with GDBStub을 참조하세요.
- JTAG 하드웨어 기반 디버깅이 필요한 경우 OpenOCD 기반 환경 설정에 대한 자세한 내용은 Espressif 웹 사이트의 JTAG Debugging을 참조하세요.
- macOS에서 `pip`를 사용하여 `pyserial`을 설치할 수 없는 경우 pyserial 웹 사이트에서 다운로드합니다.
- 보드가 연속적으로 재설정되는 경우 터미널에 다음 명령을 입력하여 플래시를 지워 봅니다.

```
make erase_flash
```

- `idf_monitor.py`를 실행할 때 오류가 나타나는 경우 Python 2.7을 사용합니다.
- ESP-IDF의 필수 라이브러리는 FreeRTOS에 포함되어 있으므로 외부에서 다운로드할 필요가 없습니다. `IDF_PATH` 환경 변수가 설정된 경우 FreeRTOS를 빌드하기 전에 제거하는 것이 좋습니다.
- Window에서 프로젝트를 빌드하려면 3-4분 정도 걸릴 수 있습니다. make 명령에서 `-j4` 스위치를 사용하여 빌드 시간을 줄일 수 있습니다.

```
make flash monitor -j4
```

- 디바이스에 연결하는 데 문제가 있는 경우 `aws_clientcredential.h` 파일을 AWS IoT열고 구성 변수가 파일에 올바르게 정의되어 있는지 확인합니다.는와 같

아clientcredentialMQTT_BROKER_ENDPOINT[]야 합니다*1234567890123*-ats.iot.*us-east-1*.amazonaws.com.

- ESP32를 위한 자체 CMake 프로젝트에서 FreeRTOS 사용의 단계를 따르고 링커에서 정의되지 않은 참조 오류가 표시되는 경우 일반적으로 종속 라이브러리 또는 데모가 누락되어 발생합니다. 이를 추가하려면 표준 CMake 함수 target_link_libraries를 사용하여 루트 디렉터리 아래의 CMakeLists.txt 파일을 업데이트합니다.

- ESP-IDF v4.2는 xtensa-esp32-elf-gcc 8.2.0. 도구 체인 사용을 지원합니다. 이전 버전의 Xtensa 도구 체인을 사용하는 경우 필요한 버전을 다운로드하세요.

- ESP-IDF v4.2에서 충족되지 않는 python 종속성 때문에 다음과 같은 오류 로그가 표시되는 경우

```
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
```

다음 Python 명령을 사용하여 플랫폼에 python 종속성을 설치합니다.

```
root/vendors/espressif/esp-idf/requirements.txt
```

문제 해결에 대한 자세한 정보는 시작하기 문제 해결 섹션을 참조하세요.

디버깅

Espressif ESP32-DevKitC 및 ESP-WROVER-KIT에서 코드 디버깅(ESP-IDF v3.3)

이 섹션에서는 ESP-IDF v3.3을 사용하여 Espressif 하드웨어를 디버깅하는 방법을 보여줍니다. ESP-IDF v4.2를 사용하여 디버깅하려면 Espressif ESP32-DevKitC 및 ESP-WROVER-KIT에서 코드 디버깅(ESP-IDF v4.2) 섹션을 참조하세요.

JTAG - USB 케이블이 필요합니다. 여기서는 USB-MPSSE 케이블을 사용합니다(예: FTDI C232HM-DDHSL-0).

ESP-DevKitC JTAG 설정

FTDI C232HM-DDHSL-0 케이블의 경우 다음은 ESP32 DevkitC에 대한 연결입니다.

C232HM-DDHSL-0 와이어 색상	ESP32 GPIO 핀	JTAG 신호 이름
갈색(핀 5)	IO14	TMS
노란색(핀 3)	IO12	TDI
검은색(핀 10)	GND	GND
주황색(핀 2)	IO13	TCK
녹색(핀 4)	IO15	TDO

ESP-WROVER-KIT JTAG 설정

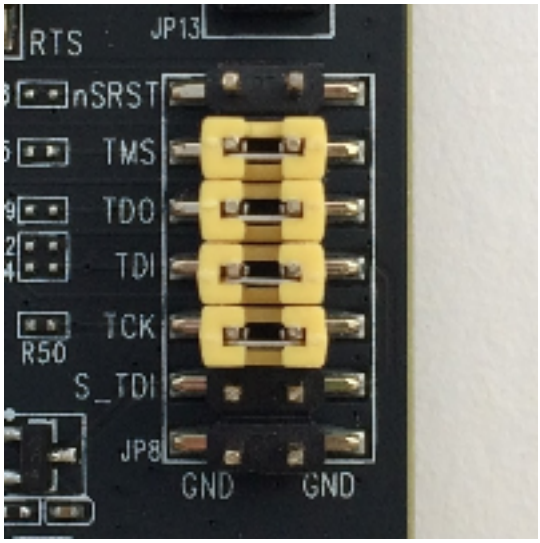
FTDI C232HM-DDHSL-0 케이블의 경우 다음은 ESP32-WROVER-KIT에 대한 연결입니다.

C232HM-DDHSL-0 와이어 색상	ESP32 GPIO 핀	JTAG 신호 이름
갈색(핀 5)	IO14	TMS
노란색(핀 3)	IO12	TDI
주황색(핀 2)	IO13	TCK

C232HM-DDHSL-0 와이어 색상	ESP32 GPIO 핀	JTAG 신호 이름
녹색(핀 4)	IO15	TDO

이러한 테이블은 FTDI C232HM-DDHSL-0 데이터시트에서 개발되었습니다. 자세한 내용은 데이터시트에서 'C232HM MPSSE Cable Connection and Mechanical Details' 섹션을 참조하세요.

ESP-WROVER-KIT에서 JTAG를 활성화하려면 TMS, TDO, TDI, TCK, and S_TDI 핀의 점퍼를 다음과 같이 배치합니다.



Windows에서의 디버깅(ESP-IDF v3.3)

Windows에서 디버깅을 설정하려면

1. Espressif ESP32-DevKitC 및 ESP-WROVER-KIT에서 코드 디버깅(ESP-IDF v3.3)의 설명과 같이 FTDI C232HM-DDHSL-0의 USB 쪽을 컴퓨터 및 다른 쪽에 연결합니다. FTDI C232HM-DDHSL-0 디바이스가 디바이스 관리자의 범용 직렬 버스 컨트롤러에 나타날 것입니다.
2. 범용 직렬 버스 디바이스 목록에서 C232HM-DDHSL-0 디바이스를 마우스 오른쪽 버튼으로 클릭하고 속성을 선택합니다.

> **Note**
> 디바이스가 USB 직렬 포트로 나열될 수 있습니다.

속성 창에서 Details(세부 정보) 탭을 선택하여 디바이스의 속성을 봅니다. 디바이스가 나열되지 않는 경우 FTDI C232HM-DDHSL-0용 Windows 드라이버를 설치합니다.

3. 세부 정보 탭에서 속성을 선택한 후 하드웨어 ID를 선택합니다. 값 필드에 다음과 같이 표시되어야 합니다.

```
FTDIBUS\COMPORT&VID_0403&PID_6014
```

이 예에서 공급업체 ID는 0403이고 제품 ID는 6014입니다.

이러한 ID가 `projects/espressif/esp32/make/aws_demos/esp32_devkitj_v1.cfg`의 ID와 일치하는지 확인합니다. 이 ID는 공급업체 ID 및 제품 ID 뒤에 `ftdi_vid_pid`로 시작하는 줄에서 지정합니다.

```
ftdi_vid_pid 0x0403 0x6014
```

4. Windows용 OpenOCD를 다운로드합니다.
5. `C:\`에 파일의 압축을 풀고 `C:\openocd-esp32\bin`을 시스템 경로에 추가합니다.
6. OpenOCD에는 libusb가 필요하며, 이 프로그램은 Windows에서 기본적으로 설치되지 않습니다.

   libusb를 설치하려면

   a. zadig.exe를 다운로드합니다.
   b. `zadig.exe`을(를) 실행합니다. Options(옵션) 메뉴에서 List All Devices(모든 디바이스 나열)를 선택합니다.
   c. 드롭다운 메뉴에서 C232HM-DDHSL-0를 선택합니다.
   d. 녹색 화살표 오른쪽의 대상 드라이버 필드에서 WinUSB를 선택합니다.
   e. 대상 드라이버 필드 아래의 드롭다운 목록에서 화살표를 선택한 후 드라이버 설치를 선택합니다. Replace Driver(드라이버 바꾸기)를 선택합니다.

7. 명령 프롬프트를 열고 `projects/espressif/esp32/make/aws_demos`로 이동하여 다음 명령을 실행합니다.

   ESP32-WROOM-32 및 ESP32-WROVER:

```
openocd.exe -f esp32_devkitj_v1.cfg -f esp-wroom-32.cfg
```

ESP32-SOLO-1:

```
openocd.exe -f esp32_devkitj_v1.cfg -f esp-solo-1.cfg
```

이 명령 프롬프트를 열어 둡니다.

8. 새 명령 프롬프트를 열고 msys32 디렉터리로 이동한 다음 mingw32.exe를 실행합니다.
9. mingw32 터미널에서 projects/espressif/esp32/make/aws_demos로 이동하여 다음을 실행합니다.

```
make flash monitor
```

10. 다른 mingw32 터미널을 열고 projects/espressif/esp32/make/aws_demos로 이동한 후 해당 보드에서 데모의 실행이 시작될 때까지 기다립니다. 시작되면 xtensa-esp32-elf-gdb -x gdbinit build/aws_demos.elf를 실행합니다. 프로그램이 main 기능에서 중지해야 합니다.

> **Note**
>
> ESP32는 최대 2개의 중단점을 지원합니다.

macOS에서의 디버깅(ESP-IDF v3.3)

1. macOS용 FTDI 드라이버를 다운로드합니다.
2. OpenOCD를 다운로드합니다.
3. 다운로드한 .tar 파일의 압축을 풀고 .bash_profile에서 경로를 *OCD_INSTALL_DIR*/openocd-esp32/bin으로 설정합니다.
4. 다음 명령을 사용하여 libusb를 macOS에 설치합니다.

```
brew install libusb
```

5. 다음 명령을 사용하여 직렬 포트 드라이버를 언로드합니다.

```
sudo kextunload -b com.FTDI.driver.FTDIUSBSerialDriver
```

6. macOS 버전 10.9 이상을 실행하는 경우 다음 명령을 사용하여Apple의 FTDI 드라이버를 언로드합니다.

```
sudo kextunload -b com.apple.driver.AppleUSBFTDI
```

7. 다음 명령을 사용하여 FTDI 케이블의 제품 ID와 공급업체 ID를 가져옵니다. 연결된 USB 디바이스가 나열됩니다.

```
system_profiler SPUSBDataType
```

`system_profiler`의 출력은 다음과 같습니다.

```
DEVICE:

Product ID: product-ID
Vendor ID: vendor-ID (Future Technology Devices International Limited)
```

8. `projects/espressif/esp32/make/aws_demos/esp32_devkitj_v1.cfg` 파일을 엽니다. 디바이스의 공급업체 ID 및 제품 ID는 `ftdi_vid_pid`로 시작하는 줄에서 지정합니다. 이전 단계에서 출력한 `system_profiler`의 ID와 일치하도록 ID를 변경합니다.

9. 터미널 창을 열고 `projects/espressif/esp32/make/aws_demos`로 이동한 후 다음 명령을 사용하여 OpenOCD를 실행합니다.

   ESP32-WROOM-32 및 ESP32-WROVER:

```
openocd -f esp32_devkitj_v1.cfg -f esp-wroom-32.cfg
```

   ESP32-SOLO-1:

```
openocd -f esp32_devkitj_v1.cfg -f esp-solo-1.cfg
```

10. 새 터미널을 열고 다음 명령을 사용하여 FTDI 직렬 포트 드라이버를 로드합니다.

```
sudo kextload -b com.FTDI.driver.FTDIUSBSerialDriver
```

11. `projects/espressif/esp32/make/aws_demos`로 이동하여 다음 명령을 실행합니다.

```
make flash monitor
```

12. 또 다른 터미널을 새로 열고 `projects/espressif/esp32/make/aws_demos`로 이동하여 다음 명령을 실행합니다.

```
xtensa-esp32-elf-gdb -x gdbinit build/aws_demos.elf
```

프로그램이 `main()`에서 중지해야 합니다.

Linux에서의 디버깅(ESP-IDF v3.3)

1. OpenOCD를 다운로드합니다. tarball의 압축을 풀고 readme 파일의 설치 지침을 따릅니다.
2. 다음 명령을 사용하여 libusb를 Linux에 설치합니다.

```
sudo apt-get install libusb-1.0
```

3. 터미널을 열고 **`ls -l /dev/ttyUSB*`**를 입력하여 컴퓨터에 연결된 모든 USB 디바이스를 나열합니다. 이 단계는 보드의 USB 포트가 운영 체제에서 인식되는지 확인하는 데 도움이 됩니다. 그러면 다음과 같은 결과가 표시됩니다.

```
$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. 로그오프했다가 다시 로그인하고 전원을 껐다가 다시 켜서 보드에 변경 사항을 적용합니다. 터미널 프롬프트에서 USB 디바이스를 나열합니다. 그룹 소유자가 `dialout`을 `plugdev`로 변경했는지 확인합니다.

```
$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
```

더 낮은 숫자의 `/dev/ttyUSBn` 인터페이스가 JTAG 통신에 사용됩니다. 다른 인터페이스는 ESP32의 직렬 포트(UART)로 라우팅되며 코드를 ESP32의 플래시 메모리에 업로드하는 데 사용됩니다.

5. 터미널 창에서 `projects/espressif/esp32/make/aws_demos`로 이동한 후 다음 명령을 사용하여 OpenOCD를 실행합니다.

   ESP32-WROOM-32 및 ESP32-WROVER:

   ```
   openocd -f esp32_devkitj_v1.cfg -f esp-wroom-32.cfg
   ```

   ESP32-SOLO-1:

   ```
   openocd -f esp32_devkitj_v1.cfg -f esp-solo-1.cfg
   ```

6. 또 다른 터미널을 열고 `projects/espressif/esp32/make/aws_demos`로 이동하여 다음 명령을 실행합니다.

   ```
   make flash monitor
   ```

7. 또 다른 터미널을 열고 `projects/espressif/esp32/make/aws_demos`로 이동한 후 다음 명령을 실행합니다.

   ```
   xtensa-esp32-elf-gdb -x gdbinit build/aws_demos.elf
   ```

   프로그램이 `main()`에서 중지되어야 합니다.

Espressif ESP32-DevKitC 및 ESP-WROVER-KIT에서 코드 디버깅(ESP-IDF v4.2)

이 섹션에서는 ESP-IDF v4.2를 사용하여 Espressif 하드웨어를 디버깅하는 방법을 보여줍니다. ESP-IDF v3.3을 사용하여 디버깅하려면 Espressif ESP32-DevKitC 및 ESP-WROVER-KIT에서 코드 디버깅(ESP-IDF v3.3) 섹션을 참조하세요.

JTAG - USB 케이블이 필요합니다. 여기서는 USB-MPSSE 케이블을 사용합니다(예: FTDI C232HM-DDHSL-0).

ESP-DevKitC JTAG 설정

FTDI C232HM-DDHSL-0 케이블의 경우 다음은 ESP32 DevkitC에 대한 연결입니다.

C232HM-DDHSL-0 와이어 색상	ESP32 GPIO 핀	JTAG 신호 이름
갈색(핀 5)	IO14	TMS
노란색(핀 3)	IO12	TDI
검은색(핀 10)	GND	GND
주황색(핀 2)	IO13	TCK
녹색(핀 4)	IO15	TDO

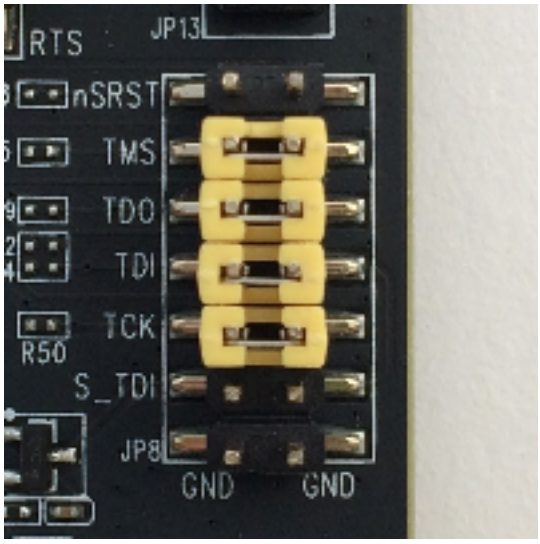
ESP-WROVER-KIT JTAG 설정

FTDI C232HM-DDHSL-0 케이블의 경우 다음은 ESP32-WROVER-KIT에 대한 연결입니다.

C232HM-DDHSL-0 와이어 색상	ESP32 GPIO 핀	JTAG 신호 이름
갈색(핀 5)	IO14	TMS
노란색(핀 3)	IO12	TDI
주황색(핀 2)	IO13	TCK
녹색(핀 4)	IO15	TDO

이러한 테이블은 FTDI C232HM-DDHSL-0 데이터시트에서 개발되었습니다. 자세한 내용은 데이터시트에서 ‘C232HM MPSSE Cable Connection and Mechanical Details’ 섹션을 참조하세요.

ESP-WROVER-KIT에서 JTAG를 활성화하려면 TMS, TDO, TDI, TCK, and S_TDI 핀의 점퍼를 다음과 같이 배치합니다.



Windows에서의 디버깅(ESP-IDF v4.2)

Windows에서 디버깅을 설정하려면

1. Espressif ESP32-DevKitC 및 ESP-WROVER-KIT에서 코드 디버깅(ESP-IDF v4.2)의 설명과 같이 FTDI C232HM-DDHSL-0의 USB 쪽을 컴퓨터 및 다른 쪽에 연결합니다. FTDI C232HM-DDHSL-0 디바이스가 디바이스 관리자의 범용 직렬 버스 컨트롤러에 나타날 것입니다.
2. 범용 직렬 버스 디바이스 목록에서 C232HM-DDHSL-0 디바이스를 마우스 오른쪽 버튼으로 클릭하고 속성을 선택합니다.

   > **Note**
   > 디바이스가 USB 직렬 포트로 나열될 수 있습니다.

   속성 창에서 Details(세부 정보) 탭을 선택하여 디바이스의 속성을 봅니다. 디바이스가 나열되지 않는 경우 FTDI C232HM-DDHSL-0용 Windows 드라이버를 설치합니다.
3. 세부 정보 탭에서 속성을 선택한 후 하드웨어 ID를 선택합니다. 값 필드에 다음과 같이 표시되어야 합니다.

   ```
   FTDIBUS\COMPORT&VID_0403&PID_6014
   ```

   이 예에서 공급업체 ID는 0403이고 제품 ID는 6014입니다.

이러한 ID가 `projects/espressif/esp32/make/aws_demos/esp32_devkitj_v1.cfg`의 ID와 일치하는지 확인합니다. 이 ID는 공급업체 ID 및 제품 ID 뒤에 `ftdi_vid_pid`로 시작하는 줄에서 지정합니다.

```
ftdi_vid_pid 0x0403 0x6014
```

4. Windows용 OpenOCD를 다운로드합니다.
5. `C:\`에 파일의 압축을 풀고 `C:\openocd-esp32\bin`을 시스템 경로에 추가합니다.
6. OpenOCD에는 libusb가 필요하며, 이 프로그램은 Windows에서 기본적으로 설치되지 않습니다.

   libusb를 설치하려면

   a. zadig.exe를 다운로드합니다.
   b. `zadig.exe`을(를) 실행합니다. Options(옵션) 메뉴에서 List All Devices(모든 디바이스 나열)를 선택합니다.
   c. 드롭다운 메뉴에서 C232HM-DDHSL-0를 선택합니다.
   d. 녹색 화살표 오른쪽의 대상 드라이버 필드에서 WinUSB를 선택합니다.
   e. 대상 드라이버 필드 아래의 드롭다운 목록에서 화살표를 선택한 후 드라이버 설치를 선택합니다. Replace Driver(드라이버 바꾸기)를 선택합니다.
7. 명령 프롬프트를 열고 FreeRTOS 다운로드 디렉터리의 루트로 이동하여 다음 명령을 실행합니다.

   ```
   idf.py openocd
   ```

   이 명령 프롬프트를 열어 둡니다.
8. 새 명령 프롬프트를 열고 FreeRTOS 다운로드 디렉터리의 루트로 이동하여 다음을 실행합니다.

   ```
   idf.py flash monitor
   ```
9. 또 다른 명령 프롬프트를 열고 FreeRTOS 다운로드 디렉터리의 루트로 이동한 다음 보드에서 데모가 실행되기 시작할 때까지 기다립니다. 데모가 시작되면 다음을 실행합니다.

   ```
   idf.py gdb
   ```

   프로그램이 `main` 기능에서 중지해야 합니다.

> **Note**
>
> ESP32는 최대 2개의 중단점을 지원합니다.

macOS에서의 디버깅(ESP-IDF v4.2)

1. macOS용 FTDI 드라이버를 다운로드합니다.
2. OpenOCD를 다운로드합니다.
3. 다운로드한 .tar 파일의 압축을 풀고 `.bash_profile`에서 경로를 *`OCD_INSTALL_DIR`*`/openocd-esp32/bin`으로 설정합니다.
4. 다음 명령을 사용하여 `libusb`를 macOS에 설치합니다.

   ```
   brew install libusb
   ```

5. 다음 명령을 사용하여 직렬 포트 드라이버를 언로드합니다.

   ```
   sudo kextunload -b com.FTDI.driver.FTDIUSBSerialDriver
   ```

6. macOS 버전 10.9 이상을 실행하는 경우 다음 명령을 사용하여Apple의 FTDI 드라이버를 언로드합니다.

   ```
   sudo kextunload -b com.apple.driver.AppleUSBFTDI
   ```

7. 다음 명령을 사용하여 FTDI 케이블의 제품 ID와 공급업체 ID를 가져옵니다. 연결된 USB 디바이스가 나열됩니다.

   ```
   system_profiler SPUSBDataType
   ```

   `system_profiler`의 출력은 다음과 같습니다.

   ```
   DEVICE:

   Product ID: product-ID
   Vendor ID: vendor-ID (Future Technology Devices International Limited)
   ```

8. `projects/espressif/esp32/make/aws_demos/esp32_devkitj_v1.cfg` 파일을 엽니다. 디바이스의 공급업체 ID 및 제품 ID는 `ftdi_vid_pid`로 시작하는 줄에서 지정합니다. 이전 단계에서 출력한 `system_profiler`의 ID와 일치하도록 ID를 변경합니다.

9. 터미널 창을 열고 FreeRTOS 다운로드 디렉터리의 루트로 이동한 후 다음 명령을 사용하여 OpenOCD를 실행합니다.

```
idf.py openocd
```

   이 터미널 창을 열어 둡니다.

10. 새 터미널을 열고 다음 명령을 사용하여 FTDI 직렬 포트 드라이버를 로드합니다.

```
sudo kextload -b com.FTDI.driver.FTDIUSBSerialDriver
```

11. FreeRTOS 다운로드 디렉터리의 루트로 이동하여 다음을 실행합니다.

```
idf.py flash monitor
```

12. 또 다른 터미널을 새로 열고 FreeRTOS 다운로드 디렉터리의 루트로 이동하여 다음을 실행합니다.

```
idf.py gdb
```

   프로그램이 `main`에서 중지해야 합니다.

Linux에서의 디버깅(ESP-IDF v4.2)

1. OpenOCD를 다운로드합니다. tarball의 압축을 풀고 readme 파일의 설치 지침을 따릅니다.
2. 다음 명령을 사용하여 libusb를 Linux에 설치합니다.

```
sudo apt-get install libusb-1.0
```

3. 터미널을 열고 **`ls -l /dev/ttyUSB*`**를 입력하여 컴퓨터에 연결된 모든 USB 디바이스를 나열합니다. 이 단계는 보드의 USB 포트가 운영 체제에서 인식되는지 확인하는 데 도움이 됩니다. 그러면 다음과 같은 결과가 표시됩니다.

```
$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. 로그오프했다가 다시 로그인하고 전원을 껐다가 다시 켜서 보드에 변경 사항을 적용합니다. 터미널 프롬프트에서 USB 디바이스를 나열합니다. 그룹 소유자가 `dialout`을 `plugdev`로 변경했는지 확인합니다.

```
$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
```

   더 낮은 숫자의 `/dev/ttyUSBn` 인터페이스가 JTAG 통신에 사용됩니다. 다른 인터페이스는 ESP32의 직렬 포트(UART)로 라우팅되며 코드를 ESP32의 플래시 메모리에 업로드하는 데 사용됩니다.

5. 터미널 창에서 FreeRTOS 다운로드 디렉터리의 루트로 이동하고 다음 명령을 사용하여 OpenOCD를 실행합니다.

```
idf.py openocd
```

6. 또 다른 터미널을 열고 FreeRTOS 다운로드 디렉터리의 루트로 이동하여 다음 명령을 실행합니다.

```
idf.py flash monitor
```

7. 또 다른 터미널을 열고 FreeRTOS 다운로드 디렉터리의 루트로 이동하여 다음 명령을 실행합니다.

```
idf.py gdb
```

   프로그램이 `main()`에서 중지되어야 합니다.

Espressif ESP32-WROOM-32SE 시작하기

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

이 자습서에 따라 Espressif ESP32-WROOM-32SE를 시작하십시오. AWS Partner Device Catalog의 파트너에서 구입하려면 ESP32-WROOM-32SE를 참조하세요.

> **Note**
>
> 현재 ESP32-WROOM-32SE용 FreeRTOS 포트는 SMP(대칭적 다중 처리) 기능을 지원하지 않습니다.

개요

이 자습서에서는 다음과 같은 단계를 안내합니다.

1. 보드를 호스트 시스템에 연결합니다.
2. 마이크로컨트롤러 보드용 내장형 애플리케이션을 개발 및 디버깅하기 위한 소프트웨어를 호스트 시스템에 설치합니다.
3. FreeRTOS 데모 애플리케이션을 바이너리 이미지로 크로스 컴파일합니다.
4. 애플리케이션 바이너리 이미지를 보드에 로드한 다음 애플리케이션을 실행합니다.
5. 직렬 연결을 사용하여 실행 중인 애플리케이션을 모니터링 및 디버깅합니다.

사전 조건

Espressif 보드에서 FreeRTOS를 시작하기 전에 AWS 계정과 권한을 설정해야 합니다.

에 가입 AWS 계정

이 없는 경우 다음 단계를 AWS 계정완료하여 생성합니다.

에 가입하려면 AWS 계정

1. https://portal.aws.amazon.com/billing/signup을 엽니다.
2. 온라인 지시 사항을 따릅니다.

   가입 절차의 일부로는 전화 또는 문자 메시지를 수신하고 전화 키패드에 확인 코드를 입력하는 것이 포함됩니다.

   에 가입하면 AWS 계정AWS 계정 루트 사용자이 생성됩니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스에 액세스할 권한이 있습니다. 보안 모범 사례는 사용자에게 관리 액세스

권한을 할당하고, 루트 사용자만 사용하여 루트 사용자 액세스 권한이 필요한 작업을 수행하는 것입니다.

AWS 는 가입 프로세스가 완료된 후 확인 이메일을 보냅니다. 언제든지 https://aws.amazon.com/으로 이동하고 내 계정을 선택하여 현재 계정 활동을 보고 계정을 관리할 수 있습니다.

관리자 액세스 권한이 있는 사용자 생성

에 가입한 후 일상적인 작업에 루트 사용자를 사용하지 않도록 관리 사용자를 AWS 계정보호 AWS IAM Identity Center, AWS 계정 루트 사용자활성화 및 생성합니다.

보안 AWS 계정 루트 사용자

1. 루트 사용자를 선택하고 AWS 계정 이메일 주소를 입력하여 계정 소유자AWS Management Console로에 로그인합니다. 다음 페이지에서 비밀번호를 입력합니다.

   루트 사용자를 사용하여 로그인하는 데 도움이 필요하면 AWS 로그인 User Guide의 루트 사용자로 로그인을 참조하세요.
2. 루트 사용자의 다중 인증(MFA)을 활성화합니다.

   지침은 IAM 사용 설명서의 AWS 계정 루트 사용자(콘솔)에 대한 가상 MFA 디바이스 활성화를 참조하세요.

관리자 액세스 권한이 있는 사용자 생성

1. IAM Identity Center를 활성화합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 AWS IAM Identity Center설정을 참조하세요.
2. IAM Identity Center에서 사용자에게 관리 액세스 권한을 부여합니다.

   를 자격 증명 소스 IAM Identity Center 디렉터리 로 사용하는 방법에 대한 자습서는 사용 AWS IAM Identity Center 설명서의 기본값으로 사용자 액세스 구성을 IAM Identity Center 디렉터리 참조하세요.

관리 액세스 권한이 있는 사용자로 로그인

- IAM IDentity Center 사용자로 로그인하려면 IAM Identity Center 사용자를 생성할 때 이메일 주소로 전송된 로그인 URL을 사용합니다.

IAM Identity Center 사용자를 사용하여 로그인하는 데 도움이 필요하면 AWS 로그인 사용 설명서의 AWS 액세스 포털에 로그인을 참조하세요.

추가 사용자에게 액세스 권한 할당

1. IAM Identity Center에서 최소 권한 적용 모범 사례를 따르는 권한 세트를 생성합니다.

   지침은AWS IAM Identity Center 사용 설명서의 Create a permission set를 참조하세요.
2. 사용자를 그룹에 할당하고, 그룹에 Single Sign-On 액세스 권한을 할당합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 Add groups를 참조하세요.

액세스 권한을 제공하려면 사용자, 그룹 또는 역할에 권한을 추가하세요:

- 의 사용자 및 그룹 AWS IAM Identity Center:

  권한 세트를 생성합니다. AWS IAM Identity Center 사용 설명서의 권한 세트 생성의 지침을 따릅니다.
- 보안 인증 공급자를 통해 IAM에서 관리되는 사용자:

  ID 페더레이션을 위한 역할을 생성합니다. IAM 사용 설명서의 Create a role for a third-party identity provider (federation)의 지침을 따릅니다.
- IAM 사용자:
  - 사용자가 맡을 수 있는 역할을 생성합니다. IAM 사용 설명서에서 Create a role for an IAM user의 지침을 따릅니다.
  - (권장되지 않음)정책을 사용자에게 직접 연결하거나 사용자를 사용자 그룹에 추가합니다. IAM 사용 설명서에서 사용자(콘솔)에 권한 추가의 지침을 따르세요.

> **Note**
>
> FreeRTOS 202012.00 릴리스는 ESP-IDF v3.3을 지원합니다. 최신 버전의 FreeRTOS를 사용하는 경우 ESP-IDF v4.2 시작하기에서 IDFv4.2 전용 지침을 따르세요.

ESP-IDF v3.3 시작하기

이 섹션에서는 Espressif 하드웨어에서 ESP-IDF v3.3을 사용하는 방법을 보여줍니다. ESP-IDF v4.2를 사용하려면 ESP-IDF v4.2 시작하기 섹션을 참조하세요.

Espressif 하드웨어 설정

ESP32-WROOM-32SE 개발 보드 하드웨어를 설정하는 방법에 대한 정보는 ESP32-DevKitC Getting Started Guide를 참조하세요.

> **Important**
>
> Espressif 안내서의 시작하기 섹션에 도달하면 여기서 멈추고 아래 단계를 따릅니다.

개발 환경 설정

보드와 통신하려면 도구 체인을 다운로드하여 설치해야 합니다.

도구 체인 설정

> **Note**
>
> ESP-IDF v3.3(FreeRTOS에서 사용하는 버전)은 ESP32 컴파일러의 최신 버전을 지원하지 않습니다. ESP-IDF 버전 3.3과 호환되는 컴파일러를 사용해야 합니다. 이전 링크를 참조하십시오. 컴파일러의 버전을 확인하려면 다음 명령을 실행합니다.
>
> ```
> xtensa-esp32-elf-gcc --version
> ```

도구 체인을 설정하려면 호스트 시스템의 운영 체제별 지침을 따릅니다.

- Standard Setup of Toolchain (ESP-IDF v3.3) for Windows
- Standard Setup of Toolchain (ESP-IDF v3.3) for macOS
- Standard Setup of Toolchain (ESP-IDF v3.3) for Linux

> **Important**
>
> 다음 단계에서 "Get ESP-IDF" 지침에 이르면 중지하고 이 페이지의 지침으로 돌아오십시오.

방금 또는 이전에 다음 단계의 'ESP-IDF 가져오기' 지침을 따랐다면 계속하기 전에 시스템에서 `IDF_PATH` 환경 변수를 지워야 합니다. 'ESP-IDF 가져오기' 지침을 따랐다면 이 환경 변수가 자동으로 설정됩니다.

CMake 설치

FreeRTOS 데모를 빌드하고 이 디바이스에 대해 애플리케이션을 테스트하려면 CMake 빌드 시스템이 필요합니다. FreeRTOS는 버전 3.13 이상을 지원합니다.

CMake.org에서 CMake의 최신 버전을 다운로드할 수 있습니다. 소스 및 이진 배포를 사용할 수 있습니다.

FreeRTOS에서 CMake를 사용하는 방법에 대한 자세한 내용은 FreeRTOS에서 CMake 사용 섹션을 참조하세요.

직렬 연결 설정

1. 호스트 시스템과 ESP32-WROOM-32SE 사이에 직렬 연결을 설정하려면 UART Bridge VCP 드라이버에 대한 CP210x USB를 설치합니다. Silicon Labs에서 이러한 드라이버를 다운로드할 수 있습니다.
2. ESP32와 직렬 연결 설정 단계를 따릅니다.
3. 직렬 연결을 설정한 후 보드의 연결을 위한 직렬 포트를 기록해 두십시오. 데모를 빌드할 때 이 정보가 필요합니다.

FreeRTOS 다운로드 및 구성

환경을 설정한 후에는 GitHub에서 FreeRTOS를 다운로드할 수 있습니다. 자세한 지침은 GitHub 웹 사이트의 README.md 파일을 참조하세요.

Important

ATECC608A 디바이스에서는 `C_InitToken`을 직접 호출하는 동안 처음 프로젝트가 실행될 때 디바이스에서 초기화가 1회 잠금 상태가 됩니다. 그러나 FreeRTOS 데모 프로젝트와 테스트 프로젝트의 구성은 다릅니다. 데모 프로젝트 구성 중에 디바이스가 잠겨 있으면 테스트 프로젝트에서 성공하지 못하는 테스트가 발생합니다.

1. FreeRTOS 데모 구성의 단계에 따라 FreeRTOS 데모 프로젝트를 구성합니다. 마지막 단계인 AWS IoT 자격 증명을 포맷하려면을 건너 뛰고 대신에 아래 단계를 수행합니다.
2. 마이크로칩은 ATECC608A 부품 설정에 도움이 되는 여러 스크립팅 도구를 제공합니다. *freertos*/vendors/microchip/secure_elements/app/example_trust_chain_tool 디렉터리로 이동하여 README.md 파일을 엽니다.
3. README.md 파일의 지침에 따라 디바이스를 프로비저닝합니다. 이 단계에는 다음 사항이 포함됩니다.
   a. 에 인증 기관 생성 및 등록 AWS
   b. ATECC608A에서 키를 생성하고 퍼블릭 키 및 디바이스 일련 번호를 내보냅니다.
   c. 디바이스에 대한 인증서를 생성하고에 해당 인증서를 등록합니다. AWS
4. 개발자 모드 키 프로비저닝의 지침에 따라 CA 인증서 및 디바이스 인증서를 디바이스에 로드합니다.

FreeRTOS 데모 프로젝트 빌드, 플래시 및 실행

CMake를 사용하여 빌드 파일을 생성하고, Make를 사용하여 애플리케이션 바이너리를 빌드하고, Espressif의 IDF 유틸리티를 사용하여 보드를 플래시할 수 있습니다.

Linux 또는 MacOS에서 FreeRTOS 빌드(ESP-IDF v3.3)

Windows를 사용하는 경우에는 Windows에서 FreeRTOS 빌드(ESP-IDF v3.3)로 건너뛸 수 있습니다.

CMake를 사용하여 빌드 파일을 생성한 다음 Make를 사용하여 애플리케이션을 빌드합니다.

CMake를 사용하여 데모 애플리케이션의 빌드 파일을 생성하려면

1. FreeRTOS 다운로드 디렉터리의 루트로 이동합니다.
2. 명령줄 창에서 다음 명령을 입력하여 빌드 파일을 생성합니다.

```
cmake -DVENDOR=espressif -DBOARD=esp32_plus_ecc608a_devkitc -DCOMPILER=xtensa-esp32
 -S . -B your-build-directory
```

> **Notes**
>
> 디버깅을 위한 애플리케이션을 빌드하려면 -DCMAKE_BUILD_TYPE=Debug 플래그를 추가합니다.

테스트 애플리케이션 빌드 파일을 생성하려면 -DAFR_ENABLE_TESTS=1 플래그를 추가합니다.
Espressif에서 제공하는 코드는 경량 IP(lwIP) 스택을 기본 네트워킹 스택으로 사용합니다. 대신 FreeRTOS+TCP 네트워킹 스택을 사용하려면 CMake 명령에 -DAFR_ESP_FREERTOS_TCP 플래그를 추가합니다.
공급업체가 제공하지 않은 코드에 대한 lwIP 종속성을 추가하려면 사용자 지정 Wi-Fi 구성 요소에 대해 다음 행을 CMake 종속성 파일인 CMakeLists.txt에 추가합니다.

```
# Add a dependency on the bluetooth espressif component to the common
 component
set(COMPONENT_REQUIRES lwip)
```

Make를 사용하여 애플리케이션을 빌드하려면

1. build 디렉터리로 이동합니다.
2. 명령줄 창에서 다음 명령을 입력하여 Make를 사용해 애플리케이션을 빌드합니다.

```
make all -j4
```

> **Note**
>
> aws_demos 프로젝트와 aws_tests 프로젝트 간에 전환할 때마다 cmake 명령으로 빌드 파일을 생성해야 합니다.

Windows에서 FreeRTOS 빌드(ESP-IDF v3.3)

Windows에서는 CMake에 대한 빌드 생성기를 지정해야 합니다. 그렇지 않으면 CMake는 기본적으로 Visual Studio입니다. Espressif는 Windows, Linux 및 MacOS에서 작동하는 Ninja 빌드 시스템을 공식적으로 권장합니다. cmd 또는 PowerShell과 같은 기본 Windows 환경에서 CMake 명령을 실행해야 합니다. MSYS2 또는 WSL과 같은 가상 Linux 환경에서는 CMake 명령 실행이 지원되지 않습니다.

CMake를 사용하여 빌드 파일을 생성한 다음 Make를 사용하여 애플리케이션을 빌드합니다.

CMake를 사용하여 데모 애플리케이션의 빌드 파일을 생성하려면

1. FreeRTOS 다운로드 디렉터리의 루트로 이동합니다.

2. 명령줄 창에서 다음 명령을 입력하여 빌드 파일을 생성합니다.

```
cmake -DVENDOR=espressif -DBOARD=esp32_plus_ecc608a_devkitc -DCOMPILER=xtensa-esp32
 -GNinja -S . -B your-build-directory
```

> **Note**
>
> 디버깅을 위한 애플리케이션을 빌드하려면 -DCMAKE_BUILD_TYPE=Debug 플래그를 추가합니다.
> 테스트 애플리케이션 빌드 파일을 생성하려면 -DAFR_ENABLE_TESTS=1 플래그를 추가합니다.
> Espressif에서 제공하는 코드는 경량 IP(lwIP) 스택을 기본 네트워킹 스택으로 사용합니다. 대신 FreeRTOS+TCP 네트워킹 스택을 사용하려면 CMake 명령에 -DAFR_ESP_FREERTOS_TCP 플래그를 추가합니다.
> 공급업체가 제공하지 않은 코드에 대한 lwIP 종속성을 추가하려면 사용자 지정 Wi-Fi 구성 요소에 대해 다음 행을 CMake 종속성 파일인 CMakeLists.txt에 추가합니다.
>
> ```
> # Add a dependency on the bluetooth espressif component to the common
>  component
> set(COMPONENT_REQUIRES lwip)
> ```

애플리케이션을 빌드하려면

1. build 디렉터리로 이동합니다.
2. 명령줄 창에서 다음 명령을 입력하여 Ninja를 호출해 애플리케이션을 빌드합니다.

```
ninja
```

또는 일반 CMake 인터페이스를 사용하여 애플리케이션을 빌드합니다.

```
cmake --build your-build-directory
```

> **Note**
>
> aws_demos 프로젝트와 aws_tests 프로젝트 간에 전환할 때마다 cmake 명령으로 빌드 파일을 생성해야 합니다.

Freertos 플래시 및 실행(ESP-IDF v3.3)

Espressif의 IDF 유틸리티(*freertos*/vendors/espressif/esp-idf/tools/idf.py)를 사용하여 보드를 플래시하고 애플리케이션을 실행하고 로그를 확인합니다.

보드의 플래시를 지우려면 *freertos* 디렉터리로 이동하여 다음 명령을 입력합니다.

```
./vendors/espressif/esp-idf/tools/idf.py erase_flash -B build-directory
```

애플리케이션 바이너리를 보드에 플래시하려면 make를 사용하십시오.

```
make flash
```

또한 IDF 스크립트를 사용하여 보드를 플래시할 수도 있습니다.

```
./vendors/espressif/esp-idf/tools/idf.py flash -B build-directory
```

로그를 보려면 다음 명령을 사용합니다.

```
./vendors/espressif/esp-idf/tools/idf.py monitor -p /dev/ttyUSB1 -B build-directory
```

> **Tip**
>
> 이러한 명령들을 결합할 수도 있습니다.
>
> ```
> ./vendors/espressif/esp-idf/tools/idf.py erase_flash flash monitor -p /dev/
> ttyUSB1 -B build-directory
> ```

ESP-IDF v4.2 시작하기

이 섹션에서는 Espressif 하드웨어에서 ESP-IDF v4.2를 사용하는 방법을 보여줍니다. ESP-IDF v3.3을 사용하려면 ESP-IDF v3.3 시작하기 섹션을 참조하세요.

> **Note**
>
> 이 자습서의 Linux 명령을 사용하려면 Bash 쉘을 사용해야 합니다.

Espressif 하드웨어 설정

ESP32-WROOM-32SE 개발 보드 하드웨어를 설정하는 방법에 대한 정보는 ESP32-DevKitC V4 Getting Started Guide를 참조하세요.

> **Important**
>
> Espressif 안내서의 시작하기 섹션에 도달하면 여기서 멈추고 아래 단계를 따릅니다.

개발 환경 설정

보드와 통신하려면 도구 체인을 다운로드하여 설치해야 합니다.

도구 체인 설정

도구 체인을 설정하려면 호스트 시스템의 운영 체제별 지침을 따릅니다.

- Standard Setup of Toolchain and prerequisites (ESP-IDF v4.2) for Windows
- Standard Setup of Toolchain prerequisites (ESP-IDF v4.2) for macOS
- Standard Setup of Toolchain prerequisites (ESP-IDF v4.2) for Linux

> **Important**
>
> 다음 단계의 'ESP-IDF 가져오기' 지침에 도달하면 여기서 멈추고 이 페이지의 지침으로 돌아옵니다.

설치 완료(Linux / Mac)

ESP-IDF Windows 설치 관리자는 필요한 도구를 모두 설치합니다. Linux 및 MAC OSX 플랫폼에서 설치를 완료하려면 추가 단계가 필요합니다.

명령줄 창을 열고 FreeRTOS 다운로드 디렉터리로 이동한 후, 다음 스크립트를 실행하여 플랫폼에 해당하는 espressif 도구 체인을 다운로드하고 설치합니다.

```
vendors/espressif/esp-idf/install.sh
```

다음 명령을 사용하여 터미널 경로에 ESP-IDF 도구 체인 도구를 추가합니다.

```
source vendors/espressif/esp-idf/export.sh
```

직렬 연결 설정

1. 호스트 시스템과 ESP32-WROOM-32SE 사이에 직렬 연결을 설정하려면 UART Bridge VCP 드라이버에 대한 CP210x USB를 설치합니다. Silicon Labs에서 이러한 드라이버를 다운로드할 수 있습니다.
2. ESP32와 직렬 연결 설정 단계를 따릅니다.
3. 직렬 연결을 설정한 후 보드의 연결을 위한 직렬 포트를 기록해 두십시오. 데모를 빌드할 때 이 정보가 필요합니다.

FreeRTOS 다운로드 및 구성

환경을 설정한 후에는 GitHub에서 FreeRTOS를 다운로드할 수 있습니다. 자세한 지침은 GitHub 웹 사이트의 README.md 파일을 참조하세요.

> **Important**
>
> ATECC608A 디바이스에서는 `C_InitToken`을 직접 호출하는 동안 처음 프로젝트가 실행될 때 디바이스에서 초기화가 1회 잠금 상태가 됩니다. 그러나 FreeRTOS 데모 프로젝트와 테스트 프로젝트의 구성은 다릅니다. 데모 프로젝트 구성 중에 디바이스가 잠겨 있으면 테스트 프로젝트에서 성공하지 못하는 테스트가 발생합니다.

1. FreeRTOS 데모 구성의 단계에 따라 FreeRTOS 데모 프로젝트를 구성합니다. 마지막 단계인 AWS IoT 보안 인증 정보의 형식을 지정하려면에 도달하면 여기서 멈추고 대신 아래 단계를 수행합니다.

2. 마이크로칩은 ATECC608A 부품 설정에 도움이 되는 여러 스크립팅 도구를 제공합니다. *freertos*/vendors/microchip/secure_elements/app/example_trust_chain_tool 디렉터리로 이동하여 README.md 파일을 엽니다.
3. README.md 파일의 지침에 따라 디바이스를 프로비저닝합니다. 이 단계에는 다음 사항이 포함됩니다.

   a. 에 인증 기관 생성 및 등록 AWS

   b. ATECC608A에서 키를 생성하고 퍼블릭 키 및 디바이스 일련 번호를 내보냅니다.

   c. 디바이스에 대한 인증서를 생성하고 해당 인증서를 AWS에 등록합니다.
4. 개발자 모드 키 프로비저닝의 지침에 따라 CA 인증서 및 디바이스 인증서를 디바이스에 로드합니다.

idf.py 스크립트를 사용하여 FreeRTOS 데모 프로젝트 빌드, 플래시 및 실행

Espressif의 IDF 유틸리티를 사용하여 빌드 파일을 생성하고, 애플리케이션 바이너리를 빌드하고, 보드를 플래시할 수 있습니다.

Windows, Linux 및 MacOS에서 FreeRTOS 빌드 및 플래시(ESP-IDF v4.2)

idf.py 스크립트를 사용하여 프로젝트를 빌드하고 바이너리를 디바이스에 플래시합니다.

> **Note**
>
> 다음 예제와 같이 일부 설정에서는 idf.py에서 포트 옵션 "-p *port-name*"을 사용하여 올바른 포트를 지정해야 할 수도 있습니다.
>
> ```
> idf.py -p /dev/cu.usbserial-00101301B flash
> ```

프로젝트를 빌드하고 플래시하려면

1. FreeRTOS 다운로드 디렉터리의 루트로 이동합니다.
2. 명령줄 창에서 다음 명령을 입력하여 ESP-IDF 도구를 터미널의 경로에 추가합니다.

   Windows

   ```
   vendors\espressif\esp-idf\export.bat
   ```

Linux/MacOS

```
source vendors/espressif/esp-idf/export.sh
```

3. build 디렉터리에서 cmake를 구성하고 다음 명령을 사용하여 펌웨어 이미지를 빌드합니다.

```
idf.py -DVENDOR=espressif -DBOARD=esp32_ecc608a_devkitc -DCOMPILER=xtensa-esp32
 build
```

다음 예제와 같은 출력이 표시됩니다.

```
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'
```

오류가 없는 경우 빌드는 펌웨어 바이너리 .bin 파일을 생성합니다.

4. 다음 명령을 사용하여 개발 보드의 플래시 메모리를 지웁니다.

```
idf.py erase_flash
```

5. idf.py 스크립트를 사용하여 애플리케이션 바이너리를 보드에 플래시합니다.

```
idf.py flash
```

6. 다음 명령을 사용하여 보드의 직렬 포트 출력을 모니터링합니다.

```
idf.py monitor
```

> **Note**
>
> 다음 예제와 같이 이들 명령을 결합할 수 있습니다.
>
> ```
> idf.py erase_flash flash monitor
> ```

> **Note**
>
> 특정 호스트 시스템 설정의 경우 다음 예제와 같이 보드를 플래시할 때 포트를 지정해야 합니다.
>
> ```
> idf.py erase_flash flash monitor -p /dev/ttyUSB1
> ```

CMake를 사용하여 FreeRTOS 빌드 및 플래시

IDF SDK에서 제공하는 idf.py 스크립트를 사용하여 코드를 빌드하고 실행하는 방법 외에도 CMake를 사용하여 프로젝트를 빌드할 수도 있습니다. 현재는 Unix Makefile 및 Ninja 빌드 시스템을 지원합니다.

프로젝트를 빌드하고 플래시하려면

1. 명령줄 창에서 FreeRTOS 다운로드 디렉터리의 루트로 이동합니다.
2. 다음 스크립트를 실행하여 ESP-IDF 도구를 쉘의 경로에 추가합니다.

   Windows

   ```
   vendors\espressif\esp-idf\export.bat
   ```

   Linux/MacOS

   ```
   source vendors/espressif/esp-idf/export.sh
   ```

3. 다음 명령을 입력하여 빌드 파일을 생성합니다.

   Unix Makefile 사용

   ```
   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
   ```

   Ninja 사용

   ```
   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. 플래시를 지운 다음 보드를 플래시합니다.

   Unix Makefile 사용

   ```
   make -C ./YOUR_BUILD_DIRECTORY erase_flash
   ```

   ```
   make -C ./YOUR_BUILD_DIRECTORY flash
   ```

   Ninja 사용

   ```
   ninja -C ./YOUR_BUILD_DIRECTORY erase_flash
   ```

   ```
   ninja -C ./YOUR_BUILD_DIRECTORY flash
   ```

AWS 클라우드에서 MQTT 메시지 모니터링

AWS IoT 콘솔에서 MQTT 클라이언트를 사용하여 디바이스가 AWS 클라우드로 보내는 메시지를 모니터링할 수 있습니다.

MQTT 클라이언트를 사용하여 AWS IoT MQTT 주제를 구독하려면

1. AWS IoT 콘솔에 로그인합니다.
2. 탐색 창에서 Test(테스트)를 선택하여 MQTT 클라이언트를 엽니다.
3. 구독 주제에 ***your-thing-name*/example/topic**을 입력한 다음 주제 구독을 선택합니다.

추가 정보

Espressif ESP32 보드 사용 및 문제 해결에 대한 추가 정보는 다음 주제를 참조하세요.

- ESP32를 위한 자체 CMake 프로젝트에서 FreeRTOS 사용
- 문제 해결
- 디버깅

Infineon XMC4800 IoT 연결 키트 시작하기

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

이 자습서에서는 Infineon XMC4800 IoT 연결 키트를 시작하기 위한 지침을 제공합니다. Infineon XMC4800 IoT 연결 키트가 없는 경우 AWS 파트너 디바이스 카탈로그를 방문하여 파트너로부터 구매하세요.

보드와 직렬 연결을 열고 로깅 및 디버깅 정보를 보려는 경우 XMC4800 IoT Connectivity Kit 외에 3.3V USB/직렬 변환기도 필요합니다. CP2104는 Adafruit의 CP2104 Friend와 같은 보드에서 널리 사용되는 일반적인 USB/직렬 변환기입니다.

시작하기 전에 AWS IoT 및 FreeRTOS 다운로드를 구성하여 디바이스를 AWS 클라우드에 연결해야 합니다. 자세한 내용은 첫 번째 단계 섹션을 참조하세요. 이 자습서에서는 FreeRTOS 다운로드 디렉터리의 경로를 *freertos*라고 합니다.

개요

이 자습서에는 다음의 시작하기 단계에 대한 지침이 포함되어 있습니다.

1. 마이크로 컨트롤러 보드용 내장형 애플리케이션을 개발 및 디버깅하기 위한 소프트웨어를 호스트 시스템에 설치합니다.
2. FreeRTOS 데모 애플리케이션을 바이너리 이미지로 크로스 컴파일합니다.
3. 애플리케이션 바이너리 이미지를 보드에 로드한 후 애플리케이션을 실행합니다.
4. 모니터링 및 디버깅을 위해 직렬 연결로 보드에서 실행되는 애플리케이션과 상호 작용합니다.

개발 환경 설정

FreeRTOS는 Infineon의 DAVE 개발 환경을 사용하여 XMC4800을 프로그래밍합니다. 온보드 디버거와 통신하려면 시작하기 전에 DAVE와 몇 가지 J-Link 드라이버를 다운로드하여 설치해야 합니다.

DAVE 설치

1. Infineon의 DAVE 소프트웨어 다운로드 페이지로 이동합니다.
2. 운영 체제에 맞는 DAVE 패키지를 선택하고 등록 정보를 제출합니다. Infineon에 등록한 후 .zip 파일의 다운로드 링크가 포함된 확인 이메일을 수신해야 합니다.
3. DAVE 패키지 .zip 파일(`DAVE_version_os_date.zip`)을 다운로드하고 DAVE를 설치하려는 위치(예: `C:\DAVE4`)에 파일의 압축을 풉니다.

   > **Note**
   >
   > 일부 Windows 사용자는 Windows 탐색기를 사용하여 파일의 압축을 푸는 동안 문제를 보고했습니다. 7-Zip과 같은 타사 프로그램을 사용하는 것이 좋습니다.

4. DAVE를 시작하려면 압축을 푼 `DAVE_version_os_date.zip` 폴더에 있는 실행 파일을 실행합니다.

자세한 내용은 DAVE 빠른 시작 가이드를 참조하십시오.

Segger J-Link 드라이버 설치

XMC4800 Relax EtherCAT 보드의 온보드 디버깅 프로브와 통신하려면 J-Link 소프트웨어 및 설명서 팩에 포함된 드라이버가 필요합니다. Segger의 J-Link 소프트웨어 다운로드 페이지에서 J-Link 소프트웨어 및 설명서 팩을 다운로드할 수 있습니다.

직렬 연결 설정

직렬 연결을 설정하는 것은 선택 사항이지만 설정하는 것이 좋습니다. 직렬 연결을 사용하면 보드가 개발 머신에서 볼 수 있는 형식으로 로깅 및 디버깅 정보를 전송할 수 있습니다.

XMC4800 데모 애플리케이션은 XMC4800 Relax EtherCAT 보드의 실크스크린에서 레이블 지정되는 P0.0 및 P0.1 핀의 UART 직렬 연결을 사용합니다. 직렬 연결을 설정하려면:

1. "RX<P0.0" 레이블이 있는 핀을 USB/직렬 변환기의 "TX" 핀에 연결합니다.
2. "TX>P0.1" 레이블이 있는 핀을 USB/직렬 변환기의 "RX" 핀에 연결합니다.

3. 직렬 변환기의 접지 핀을 보드에서 "GND" 레이블이 있는 핀 중 하나에 연결합니다. 디바이스는 일반 접지를 공유해야 합니다.

전원은 USB 디버깅 포트에서 공급되므로 직렬 어댑터의 양전압 핀을 보드에 연결하지 마십시오.

> **Note**
>
> 일부 직렬 케이블은 5V 신호 전달 수준을 사용합니다. XMC4800 보드와 Wi-Fi Click 모듈에는 3.3V가 필요합니다. 보드의 IOREF 점퍼를 사용하여 보드의 신호를 5V로 변경하지 마십시오.

케이블이 연결된 상태로 GNU Screen과 같은 터미널 에뮬레이터에서 직렬 연결을 엽니다. 보드 속도는 기본적으로 115200로 설정되며 8 데이터 비트, 패리티 없음, 1 정지 비트가 설정됩니다.

FreeRTOS 데모 프로젝트 빌드 및 실행

FreeRTOS 데모를 DAVE로 가져오기

1. DAVE를 시작합니다.
2. DAVE에서 File(파일), Import(가져오기)를 선택합니다. Import(가져오기) 창에서 Infineon 폴더를 확장하고 DAVE Project(DAVE 프로젝트)를 선택한 다음 Next(다음)를 선택합니다.
3. Import DAVE Projects(DAVE 프로젝트 가져오기) 창에서 Select Root Directory(루트 디렉터리 선택)를 선택하고 Browse(가져오기)를 선택한 다음 XMC4800 데모 프로젝트를 선택합니다.

   FreeRTOS 다운로드의 압축을 푼 디렉터리에서 데모 프로젝트는 `projects/infineon/xmc4800_iotkit/dave4/aws_demos`에 있습니다.

   Copy Projects Into Workspace(프로젝트를 작업 영역에 복사)가 선택 해제된 상태인지 확인합니다.
4. 마침을 클릭합니다.

   `aws_demos` 프로젝트를 작업 공간으로 가져오고 활성화한 상태여야 합니다.
5. Project(프로젝트) 메뉴에서 Build Active Project(활성 프로젝트 빌드)를 선택합니다.

   프로젝트가 오류 없이 빌드되는지 확인합니다.

FreeRTOS 데모 프로젝트 실행

1. USB 케이블을 사용하여 XMC4800 IoT Connectivity Kit를 컴퓨터에 연결합니다. 보드에는 두 개의 microUSB 커넥터가 있습니다. "X101" 레이블이 있는 커넥터를 사용합니다. 보드의 실크스크린에서 디버그는 이 커넥터 옆에 나타납니다.
2. Project(프로젝트) 메뉴에서 Rebuild Active Project(활성 프로젝트 다시 빌드)를 선택하여 `aws_demos`를 다시 빌드하고 구성 변경 사항이 적용되었는지 확인합니다.
3. Project Explorer(프로젝트 탐색기)에서 `aws_demos`를 마우스 오른쪽 버튼으로 클릭하고 Debug As(다른 형식으로 디버그)를 선택한 다음 DAVE C/C++ Application(DAVE C/C++ 애플리케이션)을 선택합니다.
4. GDB SEGGER J-Link Debugging(GDB SEGGER J-Link 디버깅)을 두 번 클릭하여 디버그 확인을 생성합니다. Debug(디버그)를 선택합니다.
5. 디버거가 `main()`의 중단점에서 중지되면 실행 메뉴에서 다시 시작을 선택합니다.

AWS IoT 콘솔에서 4~5단계의 MQTT 클라이언트는 디바이스에서 보낸 MQTT 메시지를 표시해야 합니다. 직렬 연결을 사용하는 경우 UART 출력에 다음과 비슷한 내용이 나타납니다.

```
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----
```

CMake로 FreeRTOS 데모 빌드

FreeRTOS 개발용 IDE를 사용하지 않으려는 경우 CMake를 사용하여 데모 애플리케이션이나 타사 코드 편집기 및 디버깅 도구를 사용하여 개발한 애플리케이션을 빌드하고 실행할 수 있습니다.

> **Note**
>
> 이 단원에서는 MingW를 사용하여 Windows에서 CMake를 네이티브 빌드 시스템으로 사용하는 방법을 다룹니다. 다른 운영 체제 및 옵션과 함께 CMake를 사용하는 방법에 대한 자세한 내용은 FreeRTOS에서 CMake 사용 단원을 참조하십시오. (MinGW는 네이티브 Microsoft Windows 애플리케이션을 위한 최소한의 개발 환경입니다.)

CMake로 FreeRTOS 데모를 빌드하려면

1. GNU Arm Embedded Toolchain을 설정합니다.

   a. Arm Embedded Toolchain 다운로드 페이지에서 Windows 버전의 도구 체인을 다운로드합니다.

   > **Note**
   >
   > "8-2018-q4-major" 버전에는 "objcopy" 유틸리티에 대해 보고된 버그가 있으므로 해당 버전 이외의 버전을 다운로드하는 것이 좋습니다.

b. 다운로드한 도구 체인 설치 프로그램을 열고 설치 마법사의 지시에 따라 도구 체인을 설치합니다.

> **Important**
>
> 설치 마법사의 마지막 페이지에서 Add path to environment variable(환경 변수에 경로 추가)를 선택하여 시스템 경로 환경 변수에 도구 체인 경로를 추가합니다.

2. CMake 및 MingW를 설치합니다.

   지침은 CMake 사전 조건을 참조하십시오.

3. 생성된 빌드 파일을 포함할 폴더(*build-folder*)를 생성합니다.
4. 디렉터리를 FreeRTOS 다운로드 디렉터리(*freertos*)로 변경하고 다음 명령을 사용하여 빌드 파일을 생성합니다.

```
cmake -DVENDOR=infineon -DBOARD=xmc4800_iotkit -DCOMPILER=arm-gcc -S . -B build-folder -G "MinGW Makefiles" -DAFR_ENABLE_TESTS=0
```

5. 디렉터리를 빌드 디렉터리(*build-folder*)로 변경하고 다음 명령을 사용하여 바이너리를 빌드합니다.

```
cmake --build . --parallel 8
```

   이 명령은 출력 바이너리 `aws_demos.hex`를 빌드 디렉터리로 빌드합니다.

6. JLINK를 사용하여 이미지를 플래시 및 실행합니다.

   a. 빌드 디렉터리(*build-folder*)에서 다음 명령을 사용하여 플래시 스크립트를 생성합니다.

```
echo loadfile aws_demos.hex > flash.jlink
```

```
echo r >> flash.jlink
```

```
echo g >> flash.jlink
```

```
echo q >> flash.jlink
```

   b. JLNIK 실행 파일을 사용하여 이미지를 플래시합니다.

```
JLINK_PATH\JLink.exe  -device XMC4800-2048 -if SWD -speed auto -CommanderScript
 flash.jlink
```

애플리케이션 로그는 보드에서 설정한 직렬 연결을 통해 볼 수 있어야 합니다.

클라우드에서 MQTT 메시지 모니터링

AWS IoT 콘솔에서 MQTT 클라이언트를 사용하여 디바이스가 AWS 클라우드로 보내는 메시지를 모니터링할 수 있습니다.

MQTT 클라이언트를 사용하여 AWS IoT MQTT 주제를 구독하려면

1. AWS IoT 콘솔에 로그인합니다.
2. 탐색 창에서 Test(테스트)를 선택하여 MQTT 클라이언트를 엽니다.
3. 구독 주제에 ***your-thing-name*/example/topic**을 입력한 다음 주제 구독을 선택합니다.

문제 해결

아직 하지 않았다면 디바이스를 AWS 클라우드에 연결하도록 AWS IoT 및 FreeRTOS 다운로드를 구성해야 합니다. 자세한 내용은 첫 번째 단계 섹션을 참조하세요.

FreeRTOS 시작하기에 대한 일반 문제 해결 정보는 시작하기 문제 해결 섹션을 참조하세요.

Infineon OPTIGA Trust X 및 XMC4800 IoT 연결 키트 시작하기

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

이 자습서에서는 Infineon OPTIGA Trust X 보안 요소 및 XMC4800 IoT 연결 키트를 시작하기 위한 지침을 제공합니다. Infineon XMC4800 IoT 연결 키트 시작하기 자습서와 비교하여, 이 가이드는 Infineon OPTIGA Trust X 보안 요소를 사용하여 보안 자격 증명을 제공하는 방법을 보여줍니다.

다음 하드웨어가 필요합니다.

1. 호스트 MCU - Infineon XMC4800 IoT Connectivity Kit, AWS 파트너 디바이스 카탈로그를 방문하여 파트너로부터 구매하세요.
2. 보안 확장 팩:
   - 보안 요소 - Infineon OPTIGA Trust X.

     AWS 파트너 디바이스 카탈로그를 방문하여 파트너로부터 구매하세요.
   - 개인화 보드 - Infineon OPTIGA 개인화 보드.
   - 어댑터 보드 - Infineon MyIoT 어댑터.

여기 나와 있는 단계를 따르려면 보드와의 직렬 연결을 열어서 로깅 및 디버깅 정보를 확인해야 합니다. (한 단계에서는 보드의 직렬 디버깅 출력에서 퍼블릭 키를 복사하여 파일에 붙여 넣어야 합니다.) 이를 위해서는 XMC4800 IoT 연결 키트에 추가로 3.3V USB/직렬 변환기가 필요합니다. JBtek EL - PN-47310126 USB/직렬 변환기는 이 데모에서 작동하는 것으로 알려져 있습니다. 또한 Infineon MyIoT 어댑터 보드에 직렬 케이블을 연결하려면 3개의 수-수 점퍼 와이어(수신(RX), 전송(TX) 및 접지(GND))가 필요합니다.

시작하기 전에 AWS IoT 및 FreeRTOS 다운로드를 구성하여 디바이스를 AWS 클라우드에 연결해야 합니다. 지침은 옵션 #2: 온보드 프라이빗 키 생성 단원을 참조하십시오. 이 자습서에서는 FreeRTOS 다운로드 디렉터리의 경로를 *freertos*라고 합니다.

개요

이 자습서에 포함된 단계는 다음과 같습니다.

1. 마이크로컨트롤러 보드용 내장형 애플리케이션을 개발 및 디버깅하기 위한 소프트웨어를 호스트 머신에 설치합니다.
2. FreeRTOS 데모 애플리케이션을 바이너리 이미지로 크로스 컴파일합니다.
3. 애플리케이션 이진 이미지를 보드에 로드한 다음 애플리케이션을 실행합니다.
4. 모니터링 및 디버깅을 위해 직렬 연결로 보드에서 실행되는 애플리케이션과 상호 작용합니다.

개발 환경 설정

FreeRTOS는 Infineon의 DAVE 개발 환경을 사용하여 XMC4800을 프로그래밍합니다. 온보드 디버거와 통신하려면 시작하기 전에 DAVE와 몇 가지 J-Link 드라이버를 다운로드하여 설치합니다.

DAVE 설치

1. Infineon의 DAVE 소프트웨어 다운로드 페이지로 이동합니다.
2. 운영 체제에 맞는 DAVE 패키지를 선택하고 등록 정보를 제출합니다. 등록한 후 .zip 파일의 다운로드 링크가 포함된 확인 이메일을 수신해야 합니다.
3. DAVE 패키지 .zip 파일(DAVE_*version_os_date*.zip)을 다운로드하고 DAVE를 설치하려는 위치(예: C:\DAVE4)에 파일의 압축을 풉니다.

   > **Note**
   >
   > 일부 Windows 사용자는 Windows 탐색기를 사용하여 파일의 압축을 푸는 동안 문제를 보고했습니다. 7-Zip과 같은 타사 프로그램을 사용하는 것이 좋습니다.

4. DAVE를 시작하려면 압축을 푼 DAVE_*version_os_date*.zip 폴더에 있는 실행 파일을 실행합니다.

자세한 내용은 DAVE 빠른 시작 가이드를 참조하십시오.

Segger J-Link 드라이버 설치

XMC4800 IoT 연결 키트의 온보드 디버깅 프로브와 통신하려면 J-Link 소프트웨어 및 설명서 팩에 포함된 드라이버가 필요합니다. Segger의 J-Link 소프트웨어 다운로드 페이지에서 J-Link 소프트웨어 및 설명서 팩을 다운로드할 수 있습니다.

직렬 연결 설정

USB/직렬 변환기 케이블을 Infineon Shield2Go 어댑터에 연결합니다. 이렇게 하면 보드가 개발 머신에서 볼 수 있는 형식으로 로깅 및 디버깅 정보를 전송할 수 있습니다. 직렬 연결을 설정하려면:

1. RX 핀을 USB/직렬 변환기의 TX 핀에 연결합니다.
2. TX 핀을 USB/직렬 변환기의 RX 핀에 연결합니다.
3. 직렬 변환기의 접지 핀을 보드에 있는 GND 핀 중 하나에 연결합니다. 디바이스는 일반 접지를 공유해야 합니다.

전원은 USB 디버깅 포트에서 공급되므로 직렬 어댑터의 양전압 핀을 보드에 연결하지 마십시오.

> **Note**
>
> 일부 직렬 케이블은 5V 신호 전달 수준을 사용합니다. XMC4800 보드와 Wi-Fi Click 모듈에는 3.3V가 필요합니다. 보드의 IOREF 점퍼를 사용하여 보드의 신호를 5V로 변경하지 마십시오.

케이블이 연결된 상태로 GNU Screen과 같은 터미널 에뮬레이터에서 직렬 연결을 엽니다. 보드 속도는 기본적으로 115200로 설정되며 8 데이터 비트, 패리티 없음, 1 정지 비트가 설정됩니다.

클라우드에서 MQTT 메시지 모니터링

AWS IoT 콘솔에서 MQTT 클라이언트를 사용하여 디바이스가 AWS 클라우드로 보내는 메시지를 모니터링할 수 있습니다. 디바이스가 데모 프로젝트를 실행하기 전에 설정할 수 있습니다.

MQTT 클라이언트를 사용하여 AWS IoT MQTT 주제를 구독하려면

1. AWS IoT 콘솔에 로그인합니다.
2. 탐색 창에서 Test(테스트)를 선택하여 MQTT 클라이언트를 엽니다.
3. 구독 주제에 ***your-thing-name*/example/topic**을 입력한 다음 주제 구독을 선택합니다.

FreeRTOS 데모 프로젝트 빌드 및 실행

FreeRTOS 데모를 DAVE로 가져오기

1. DAVE를 시작합니다.
2. DAVE에서 파일을 선택한 후 가져오기를 선택합니다. Infineon 폴더를 확장하고 DAVE Project(DAVE 프로젝트)를 선택한 후다음을 선택합니다.
3. Import DAVE Projects(DAVE 프로젝트 가져오기) 창에서 Select Root Directory(루트 디렉터리 선택)를 선택하고 Browse(가져오기)를 선택한 다음 XMC4800 데모 프로젝트를 선택합니다.

   FreeRTOS 다운로드의 압축을 푼 디렉터리에서 데모 프로젝트는 `projects/infineon/xmc4800_plus_optiga_trust_x/dave4/aws_demos/dave4`에 있습니다.

   Copy Projects Into Workspace(Workspace에 프로젝트 복사)가 선택 해제된 상태인지 확인합니다.
4. 마침을 클릭합니다.

   `aws_demos` 프로젝트를 작업 공간으로 가져오고 활성화한 상태여야 합니다.

5. Project(프로젝트) 메뉴에서 Build Active Project(활성 프로젝트 빌드)를 선택합니다.

   프로젝트가 오류 없이 빌드되는지 확인합니다.

FreeRTOS 데모 프로젝트 실행

1. 프로젝트 메뉴에서 Rebuild Active Project(활성 프로젝트 다시 빌드)를 선택하여 aws_demos를 다시 빌드하고 구성 변경 사항이 적용되었는지 확인합니다.
2. Project Explorer(프로젝트 탐색기)에서 aws_demos를 마우스 오른쪽 버튼으로 클릭하고 Debug As(다른 형식으로 디버그)를 선택한 다음 DAVE C/C++ Application(DAVE C/C++ 애플리케이션)을 선택합니다.
3. GDB SEGGER J-Link Debugging(GDB SEGGER J-Link 디버깅)을 두 번 클릭하여 디버그 확인을 생성합니다. Debug(디버그)를 선택합니다.
4. 디버거가 main()의 중단점에서 중지되면 실행 메뉴에서 다시 시작을 선택합니다.

여기서 옵션 #2: 온보드 프라이빗 키 생성의 퍼블릭 키 추출 단계를 계속 진행합니다. 모든 단계가 완료되면 AWS IoT 콘솔로 이동합니다. 디바이스에서 보낸 MQTT 메시지가 이전에 설정한 MQTT 클라이언트에 표시되어야 합니다. 디바이스의 직렬 연결을 통해 UART 출력에 다음과 같은 내용이 표시되어야 합니다.

```
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----
```

CMake로 FreeRTOS 데모 빌드

이 단원에서는 MingW를 사용하여 Windows에서 CMake를 네이티브 빌드 시스템으로 사용하는 방법을 다룹니다. 다른 운영 체제 및 옵션과 함께 CMake를 사용하는 방법에 대한 자세한 내용은 FreeRTOS에서 CMake 사용 단원을 참조하십시오. (MinGW는 네이티브 Microsoft Windows 애플리케이션을 위한 최소한의 개발 환경입니다.)

FreeRTOS 개발용 IDE를 사용하지 않으려는 경우 CMake를 사용하여 데모 애플리케이션이나 타사 코드 편집기 및 디버깅 도구를 사용하여 개발한 애플리케이션을 빌드하고 실행할 수 있습니다.

CMake로 FreeRTOS 데모를 빌드하려면

1. GNU Arm Embedded Toolchain을 설정합니다.

   a. Arm Embedded Toolchain 다운로드 페이지에서 Windows 버전의 도구 체인을 다운로드합니다.

> **Note**
>
> objcopy 유틸리티에 보고된 버그가 있으므로 "8-2018-q4-major" 이외의 버전을 다운로드하는 것이 좋습니다.

b. 다운로드한 도구 체인 설치 관리자를 열고 마법사의 지침에 따릅니다.

c. 설치 마법사의 마지막 페이지에서 Add path to environment variable(환경 변수에 경로 추가)를 선택하여 시스템 경로 환경 변수에 도구 체인 경로를 추가합니다.

2. CMake 및 MingW를 설치합니다.

   지침은 CMake 사전 조건을 참조하십시오.

3. 생성된 빌드 파일을 포함할 폴더(*build-folder*)를 생성합니다.

4. 디렉터리를 FreeRTOS 다운로드 디렉터리(*freertos*)로 변경하고 다음 명령을 사용하여 빌드 파일을 생성합니다.

```
cmake -DVENDOR=infineon -DBOARD=xmc4800_plus_optiga_trust_x -DCOMPILER=arm-gcc -S .
 -B build-folder -G "MinGW Makefiles" -DAFR_ENABLE_TESTS=0
```

5. 디렉터리를 빌드 디렉터리(*build-folder*)로 변경하고 다음 명령을 사용하여 바이너리를 빌드합니다.

```
cmake --build . --parallel 8
```

   이 명령은 출력 바이너리 `aws_demos.hex`를 빌드 디렉터리로 빌드합니다.

6. JLINK를 사용하여 이미지를 플래시 및 실행합니다.

   a. 빌드 디렉터리(*build-folder*)에서 다음 명령을 사용하여 플래시 스크립트를 생성합니다.

```
echo loadfile aws_demos.hex > flash.jlink
echo r >> flash.jlink
echo g >> flash.jlink
echo q >> flash.jlink
```

   b. JLNIK 실행 파일을 사용하여 이미지를 플래시합니다.

```
JLINK_PATH\JLink.exe  -device XMC4800-2048 -if SWD -speed auto -CommanderScript
 flash.jlink
```

애플리케이션 로그는 보드에서 설정한 직렬 연결을 통해 볼 수 있어야 합니다. 옵션 #2: 온보드 프라이빗 키 생성의 퍼블릭 키 추출 단계로 계속 진행합니다. 모든 단계가 완료되면 AWS IoT 콘솔로 이동합니다. 디바이스에서 보낸 MQTT 메시지가 이전에 설정한 MQTT 클라이언트에 표시되어야 합니다.

문제 해결

일반적인 문제 해결 정보는 시작하기 문제 해결 단원을 참조하십시오.

MW32x AWS IoT 스타터 키트 시작하기

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

AWS IoT 스타터 키트는 단일 마이크로컨트롤러 칩에 88MW320.11b/g/n Wi-Fi를 통합하는 NXP의 최신 통합 Cortex M4 마이크로컨트롤러인 88MW320/88MW322를 기반으로 하는 개발 키트입니다. 이 개발 키트는 FCC 인증을 받았습니다. 자세한 내용은 AWS Partner Device Catalog를 참조하여 파트너에서 구입하시기 바랍니다. 또한 88MW320/88MW322 모듈은 FCC 인증을 받았으며 사용자 지정 및 대량 판매가 가능합니다.

이 시작 안내서는 호스트 컴퓨터에서 애플리케이션을 SDK와 함께 크로스 컴파일한 다음 SDK와 함께 제공된 도구를 사용하여 생성된 바이너리 파일을 보드에 로드하는 방법을 보여줍니다. 애플리케이션이 보드에서 실행되기 시작하면 호스트 컴퓨터의 직렬 콘솔에서 애플리케이션을 디버깅하거나 상호작용할 수 있습니다.

Ubuntu 16.04는 개발 및 디버깅을 지원하는 호스트 플랫폼입니다. 다른 플랫폼을 사용할 수도 있지만 공식적으로 지원되지는 않습니다. 호스트 플랫폼에 소프트웨어를 설치할 수 있는 권한이 있어야 합니다. 다음은 SDK를 빌드하는 데 필요한 외부 도구입니다.

- Ubuntu 16.04 호스트 플랫폼
- ARM 도구 체인 버전 4_9_2015q3
- Eclipse 4.9.0 IDE

애플리케이션과 SDK를 크로스 컴파일하려면 ARM 도구 체인이 필요합니다. SDK는 최신 버전의 도구 체인을 활용하여 이미지 풋프린트를 최적화하고 더 적은 공간에서 더 많은 기능을 제공합니다. 이 안내서에서는 도구 체인 버전 4_9_2015q3을 사용한다고 가정합니다. 이전 버전의 도구 체인은 사용하지 않는 것이 좋습니다. 개발 키트에는 무선 마이크로컨트롤러 데모 프로젝트 펌웨어가 미리 플래시되어 있습니다.

주제

- 하드웨어 설정
- 개발 환경 설정
- FreeRTOS 데모 프로젝트 빌드 및 실행
- 디버깅
- 문제 해결

하드웨어 설정

미니 USB-USB 케이블을 사용하여 MW32x 보드를 노트북에 연결합니다. 미니 USB 케이블을 보드에 있는 유일한 미니 USB 커넥터에 연결합니다. 점퍼를 변경할 필요는 없습니다.

보드가 노트북 또는 데스크톱 컴퓨터에 연결되어 있으면 외부 전원 공급 장치가 필요하지 않습니다.

이 USB 연결은 다음 기능을 제공합니다.

- 보드에 대한 콘솔 액세스. 콘솔에 액세스하는 데 사용할 수 있는 가상 tty/com 포트가 개발 호스트에 등록되어 있습니다.
- 보드에 대한 JTAG 액세스. 펌웨어 이미지를 보드의 RAM 또는 플래시에 로드하거나 언로드하거나 디버깅 목적으로 사용할 수 있습니다.

개발 환경 설정

개발을 위한 최소 요구 사항은 ARM 도구 체인과 SDK와 함께 제공되는 도구입니다. 다음 섹션에서는 ARM 도구 체인 설정에 대해 자세히 설명합니다.

GNU 도구 체인

SDK는 GCC 컴파일러 도구 체인을 공식적으로 지원합니다. GNU ARM용 크로스 컴파일러 도구 체인은 GNU Arm Embedded Toolchain 4.9-2015-q3-update에서 사용할 수 있습니다.

빌드 시스템은 기본적으로 GNU 도구 체인을 사용하도록 구성되어 있습니다. Makefile은 GNU 컴파일러 도구 체인 바이너리를 사용자의 경로에서 사용할 수 있고 Makefile에서 호출할 수 있다고 가정합니다. 또한 Makefile은 GNU 도구 체인 바이너리의 파일 이름에 접두사 `arm-none-eabi-`가 붙는다고 가정합니다.

GCC 도구 체인은 GDB와 함께 사용하여 OpenOCD(SDK와 함께 제공)로 디버그할 수 있습니다. 이는 JTAG와 인터페이스하는 소프트웨어를 제공합니다.

gcc-arm-embedded toolchain 버전 4_9_2015q3을 사용하는 것이 좋습니다.

Linux 도구 체인 설정 절차

Linux에서 GCC 도구 체인을 설정하려면 다음 단계를 따릅니다.

1. GNU Arm Embedded Toolchain 4.9-2015-q3-update에서 도구 체인 tarball을 다운로드합니다. 파일은 `gcc-arm-none-eabi-4_9-2015q3-20150921-linux.tar.bz2`입니다.
2. 선택한 디렉터리에 파일을 복사합니다. 디렉터리 이름에 공백이 없어야 합니다.
3. 다음 명령을 실행하여 파일 압축을 해제합니다.

   ```
   tar -vxf filename
   ```

4. 설치된 도구 체인의 경로를 시스템 경로에 추가합니다. 예를 들어, /home/*user-name* 디렉터리에 있는 `.profile` 파일 끝에 다음 줄을 추가합니다.

   ```
   PATH=$PATH:path to gcc-arm-none-eabit-4_9_2015_q3/bin
   ```

> **Note**
>
> 최신 Ubuntu 배포에는 Debian 버전의 GCC 크로스 컴파일러가 포함됩니다. 그렇다면 네이티브 크로스 컴파일러를 제거하고 위의 설치 절차를 따라야 합니다.

Linux 개발 호스트 작업

Ubuntu 또는 Fedora와 같은 모든 최신 Linux 데스크톱 배포판을 사용할 수 있습니다. 그러나 가장 최신 버전으로 업그레이드하는 것이 좋습니다. 다음 단계는 Ubuntu 16.04에서 작동하는 것으로 확인되었으며 해당 버전을 사용하고 있다고 가정합니다.

패키지 설치

SDK에는 새로 설정한 Linux 시스템에서 개발 환경을 빠르게 설정할 수 있는 스크립트가 포함되어 있습니다. 이 스크립트는 시스템 유형을 자동으로 감지하고 C 라이브러리, USB 라이브러리, FTDI 라이브러리, ncurses, python, LaTeX 등 적절한 소프트웨어를 설치하려고 시도합니다. 이 단원에서는 일반 디렉터리 이름이 AWS SDK 루트 디렉터리를 *amzsdk_bundle-x.y.z* 나타냅니다. 실제 디렉터리 이름은 다를 수 있습니다. 루트 권한이 있어야 합니다.

- *amzsdk_bundle-x.y.z/* 디렉터리로 이동하여 이 명령을 실행합니다.

```
./lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/bin/installpkgs.sh
```

sudo 피하기

이 안내서의 `flashprog` 작업은 아래 설명과 같이 `flashprog.py` 스크립트를 사용하여 보드의 NAND를 플래시합니다. 마찬가지로, `ramload` 작업은 `ramload.py` 스크립트를 사용하여 NAND를 플래시하지 않고 호스트의 펌웨어 이미지를 마이크로컨트롤러의 RAM으로 직접 복사합니다.

매번 `sudo` 명령을 실행하지 않고도 `flashprog` 및 `ramload` 작업을 수행하도록 Linux 개발 호스트를 구성할 수 있습니다. 다음 명령으로 실행하세요.

```
./lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/bin/perm_fix.sh
```

> **Note**
>
> 원활한 Eclipse IDE 환경을 보장하려면 이러한 방식으로 Linux 개발 호스트 권한을 구성해야 합니다.

직렬 콘솔 설정

USB 케이블을 Linux 호스트 USB 슬롯에 삽입합니다. 그러면 디바이스 감지가 트리거됩니다. `/var/log/messages` 파일에서 또는 `dmesg` 명령을 실행한 후에 다음과 같은 메시지가 표시될 것입니다.

```
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
```

ttyUSB 디바이스가 두 개 생성되었는지 확인합니다. 두 번째 ttyUSB가 직렬 콘솔입니다. 위 예제에서는 이름이 'ttyUsB1'입니다.

이 안내서에서는 minicom을 사용하여 직렬 콘솔 출력을 확인합니다. putty와 같은 다른 직렬 프로그램을 사용할 수도 있습니다. 다음 명령을 실행하여 설치 모드에서 minicom을 실행합니다.

```
minicom -s
```

minicom에서 직렬 포트 설정으로 이동하여 다음 설정을 캡처합니다.

```
| 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
```

나중에 사용하기 위해 이러한 설정을 minicom에 저장할 수 있습니다. 이제 minicom 창에 직렬 콘솔의 메시지가 표시됩니다.

직렬 콘솔 창을 선택하고 Enter 키를 누릅니다. 그러면 화면에 해시(#)가 표시됩니다.

> **Note**
>
> 개발 보드에는 FTDI 실리콘 디바이스가 포함되어 있습니다. FTDI 디바이스는 호스트용 USB 인터페이스 두 개를 제공합니다. 첫 번째 인터페이스는 MCU의 JTAG 기능과 연결되고 두 번째 인터페이스는 MCU의 물리적 uArtX 포트와 연결됩니다.

OpenOCD 설치

OpenOCD는 임베디드 대상 디바이스에 대한 디버깅, 시스템 내 프로그래밍 및 경계 스캔 테스트를 제공하는 소프트웨어입니다.

OpenOCD 버전 0.9가 필요합니다. Eclipse 기능에도 필요합니다. Linux 호스트에 이전 버전(예: 버전 0.7)이 설치된 경우 현재 사용 중인 Linux 배포판에 적합한 명령을 사용하여 해당 리포지토리를 제거합니다.

표준 리눅스 명령을 실행하여 OpenOCD를 설치합니다.

```
apt-get install openocd
```

위 명령이 버전 0.9 이상을 설치하지 않는 경우 다음 절차를 사용하여 openocd 소스 코드를 다운로드하고 컴파일합니다.

OpenOCD를 설치하려면

1. 다음 명령을 실행하여 libusb-1.0을 실행합니다.

   ```
   sudo apt-get install libusb-1.0
   ```

2. http://openocd.org/에서 openocd 0.9.0 소스 코드를 다운로드합니다.
3. openocd를 추출한 후 해당 디렉터리로 이동합니다.
4. 다음 명령을 사용하여 openocd를 구성합니다.

   ```
   ./configure --enable-ftdi --enable-jlink
   ```

5. make 유틸리티를 실행하여 opencd를 컴파일합니다.

   ```
   make install
   ```

Eclipse 설정

> **Note**
>
> 이 섹션에서는 sudo 피하기의 단계를 완료했다고 가정합니다.

Eclipse는 애플리케이션 개발 및 디버깅에 선호되는 IDE입니다. 스레드 인식 디버깅을 비롯하여 통합 디버깅 지원을 갖춘 풍부하고 사용자 친화적인 IDE를 제공합니다. 이 섹션에서는 지원되는 모든 개발 호스트의 일반적인 Eclipse 설정에 대해 설명합니다.

Eclipse를 설정하려면

1. JRE(Java 런타임 환경)를 다운로드하고 설치합니다.

   Eclipse를 사용하려면 JRE를 설치해야 합니다. Eclipse를 먼저 설치할 수도 있지만 JRE부터 설치하는 것이 좋습니다. JRE 버전(32/64비트)이 Eclipse 버전(32/64비트)과 일치해야 합니다. Oracle 웹 사이트의 Java SE Runtime Environment 8 Downloads에서 JRE를 다운로드할 수 있습니다.

2. http://www.eclipse.org에서 'C/C++ 개발자용 Eclipse IDE'를 다운로드하여 설치합니다. Eclipse 버전 4.9.0 이상이 지원됩니다. 설치 시에는 다운로드한 아카이브를 추출하기만 하면 됩니다. 플랫폼별 Eclipse 실행 파일을 실행하여 애플리케이션을 시작합니다.

FreeRTOS 데모 프로젝트 빌드 및 실행

FreeRTOS 데모 프로젝트를 실행하는 방법은 두 가지입니다.

- 명령줄을 사용합니다.
- Eclipse IDE를 사용합니다.

여기서는 두 가지 옵션을 모두 다룹니다.

프로비저닝

- 테스트 애플리케이션 또는 데모 애플리케이션을 사용하는지에 따라 다음 파일 중 하나에서 프로비저닝 데이터를 설정합니다.

  - `./tests/common/include/aws_clientcredential.h`
  - `./demos/common/include/aws_clientcredential.h`

  예시:

```
#define clientcredentialWIFI_SSID "Wi-Fi SSID"
#define clientcredentialWIFI_PASSWORD "Wi-Fi password"
#define clientcredentialWIFI_SECURITY "Wi-Fi security"
```

> **Note**
>
> 다음 Wi-Fi 보안 값을 입력할 수 있습니다.
>
> - `eWiFiSecurityOpen`
> - `eWiFiSecurityWEP`
> - `eWiFiSecurityWPA`
> - `eWiFiSecurityWPA2`
>
> SSID 및 암호는 큰따옴표로 묶여 있어야 합니다.

명령줄을 사용하여 FreeRTOS 데모 빌드 및 실행

1. 데모 애플리케이션을 빌드하려면 다음 명령을 사용합니다.

```
cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S .  -Bbuild -
DAFR_ENABLE_TESTS=0
```

   다음 예제와 동일한 출력이 표시되어야 합니다.

```
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. 빌드 디렉터리로 이동합니다.

```
cd build
```

3. make 유틸리티를 실행하여 애플리케이션을 빌드합니다.

```
make all -j4
```

다음 그림과 같이 출력되어야 합니다.

```
[ 92%] Building C object CMakeFiles/afr_ota.dir/lib/third_party/tinycbor/cborencoder.c.obj
[ 93%] Building C object CMakeFiles/afr_ota.dir/lib/third_party/tinycbor/cborencoder_close
_container_checked.c.obj
[ 93%] Building C object CMakeFiles/afr_ota.dir/lib/third_party/tinycbor/cborerrorstrings.
c.obj
[ 93%] Building C object CMakeFiles/afr_ota.dir/lib/third_party/tinycbor/cborparser.c.obj
[ 94%] Building C object CMakeFiles/afr_ota.dir/lib/third_party/tinycbor/cborparser_dup_st
ring.c.obj
[ 94%] Building C object CMakeFiles/afr_ota.dir/lib/third_party/tinycbor/cborpretty.c.obj
[ 94%] Building C object CMakeFiles/afr_ota.dir/lib/third_party/jsmn/jsmn.c.obj
[ 95%] Building C object CMakeFiles/afr_ota.dir/demos/common/logging/aws_logging_task_dyna
mic_buffers.c.obj
[ 95%] Building C object CMakeFiles/afr_ota.dir/demos/common/demo_runner/aws_demo_runner.c
.obj
[ 95%] Linking C static library afr_ota.a
[ 95%] Built target afr_ota
[ 96%] Linking C executable aws_demos.axf
[100%] Built target aws_demos
marvell@pe-lt586:build$
```

4. 테스트 애플리케이션을 빌드하려면 다음 명령을 사용합니다.

```
cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S .  -Bbuild -
DAFR_ENABLE_TESTS=1
cd build
make all -j4
```

`aws_demos project`와 `aws_tests project` 사이를 전환할 때마다 `cmake` 명령을 실행합니다.

5. 개발 보드의 플래시에 펌웨어 이미지를 기록합니다. 개발 보드가 재설정된 후 펌웨어가 실행됩니다. 이미지를 마이크로컨트롤러에 플래시하기 전에 SDK를 빌드해야 합니다.

   a. 펌웨어 이미지를 플래시하기 전에 공통 구성 요소인 Layout 및 Boot2를 사용하여 개발 보드의 플래시를 준비하세요. 다음 명령을 사용합니다.

   ```
   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
   ```

   `flashprog` 명령은 다음을 시작합니다.

   - 레이아웃 - flashprog 유틸리티는 먼저 플래시에 레이아웃을 쓰라는 명령을 받습니다. 레이아웃은 플래시의 파티션 정보와 비슷합니다. 기본 레이아웃은 `/lib/third_party/`

mcu_vendor/marvell/WMSDK/mw320/sdk/tools/OpenOCD/mw300/layout.txt에 있습니다.

- Boot2 - WMSDK에서 사용하는 부트 로더입니다. 또한 flashprog 명령은 플래시에 부트 로더를 씁니다. 마이크로컨트롤러의 펌웨어 이미지가 플래시되면 이를 로드하는 것이 부트 로더의 역할입니다. 아래 그림에 표시된 것과 동일한 출력이 표시되는지 확인하세요.

```
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. 펌웨어는 기능을 위해 Wi-Fi 칩셋을 사용하며, Wi-Fi 칩셋에는 자체 펌웨어가 있는데 플래시에도 이 펌웨어가 있어야 합니다. Boot2 부트 로더 및 MCU 펌웨어를 플래시할 때와 동일한 방식으로 flashprog.py 유틸리티를 사용하여 Wi-Fi 펌웨어를 플래시합니다. 다음 명령을 사용하여 Wi-Fi 펌웨어를 플래시합니다.

```
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
```

명령 출력이 아래 그림과 비슷해야 합니다.

```
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. 다음 명령을 사용하여 MCU 펌웨어를 플래시합니다.

```
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. 보드를 재설정합니다. 데모 앱의 로그가 표시될 것입니다.

e. 테스트 앱을 실행하려면 동일한 디렉터리에 있는 `aws_tests.bin` 바이너리를 플래시합니다.

```
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
```

명령 출력이 아래 그림과 비슷하게 출력되어야 합니다.

```
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. 펌웨어를 플래시하고 보드를 재설정하면 아래 그림과 같이 데모 앱이 시작됩니다.

```
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----
```

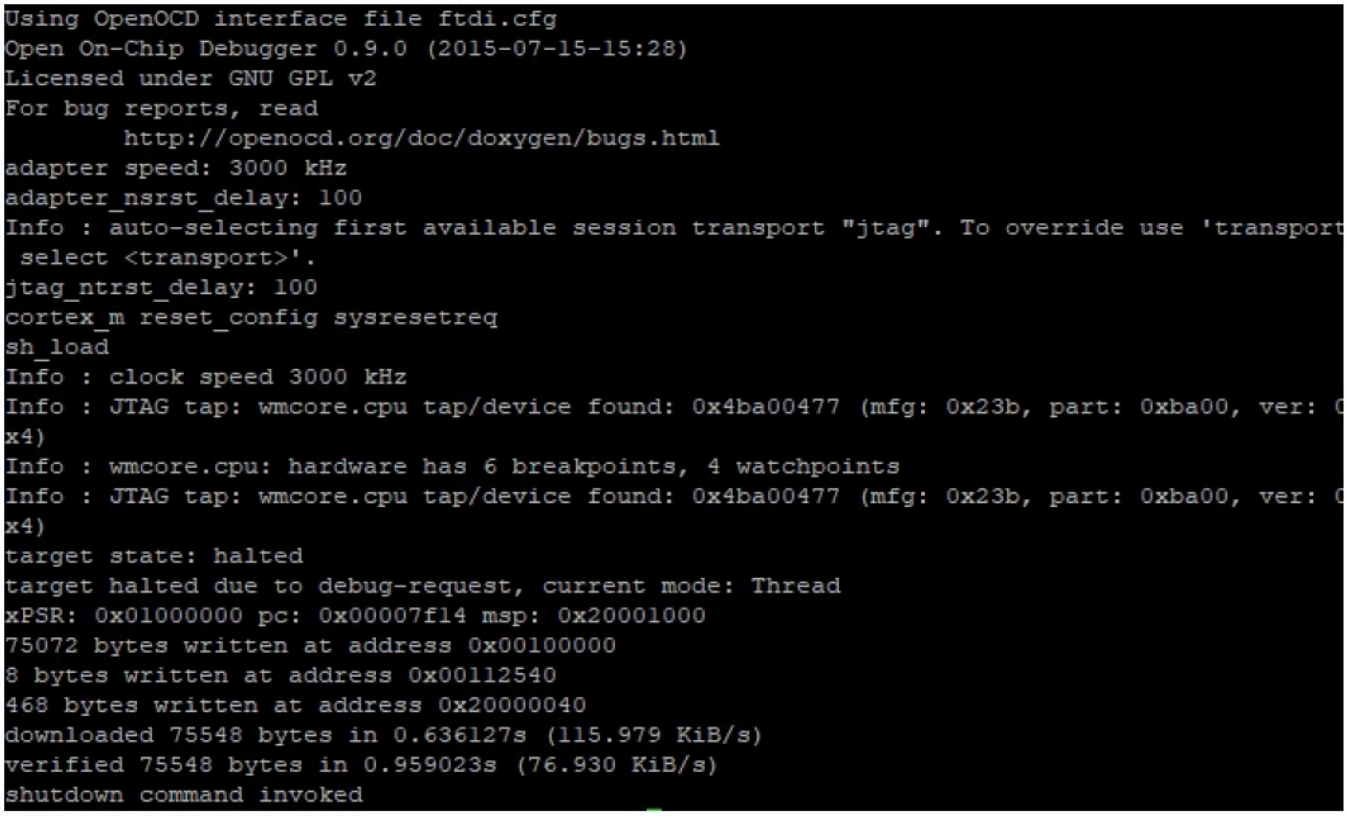
7. (선택 사항) 이미지를 테스트하는 다른 방법으로, flashprog 유틸리티를 사용하여 호스트의 마이크로컨트롤러 이미지를 마이크로컨트롤러 RAM으로 직접 복사할 수 있습니다. 이미지는 플래시에 복사되지 않으므로 마이크로컨트롤러를 재부팅하면 이미지가 손실됩니다.

   펌웨어 이미지를 SRAM으로 로드하면 실행 파일이 즉시 실행되므로 작업 속도가 더 빠릅니다. 이 방법은 주로 반복 개발에 사용됩니다.

   다음 명령을 사용하여 펌웨어를 SRAM에 로드합니다.

   ```
   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
   ```

   명령 출력은 아래 그림과 같습니다.



명령 실행이 완료되면 데모 앱의 로그가 표시됩니다.

Eclipse IDE를 사용하여 FreeRTOS 데모 빌드 및 실행

1. Eclipse 작업 공간을 설정하기 전에 cmake 명령을 실행해야 합니다.

   aws_demos Eclipse 프로젝트에서 작업을 수행하려면 다음 명령을 실행합니다.

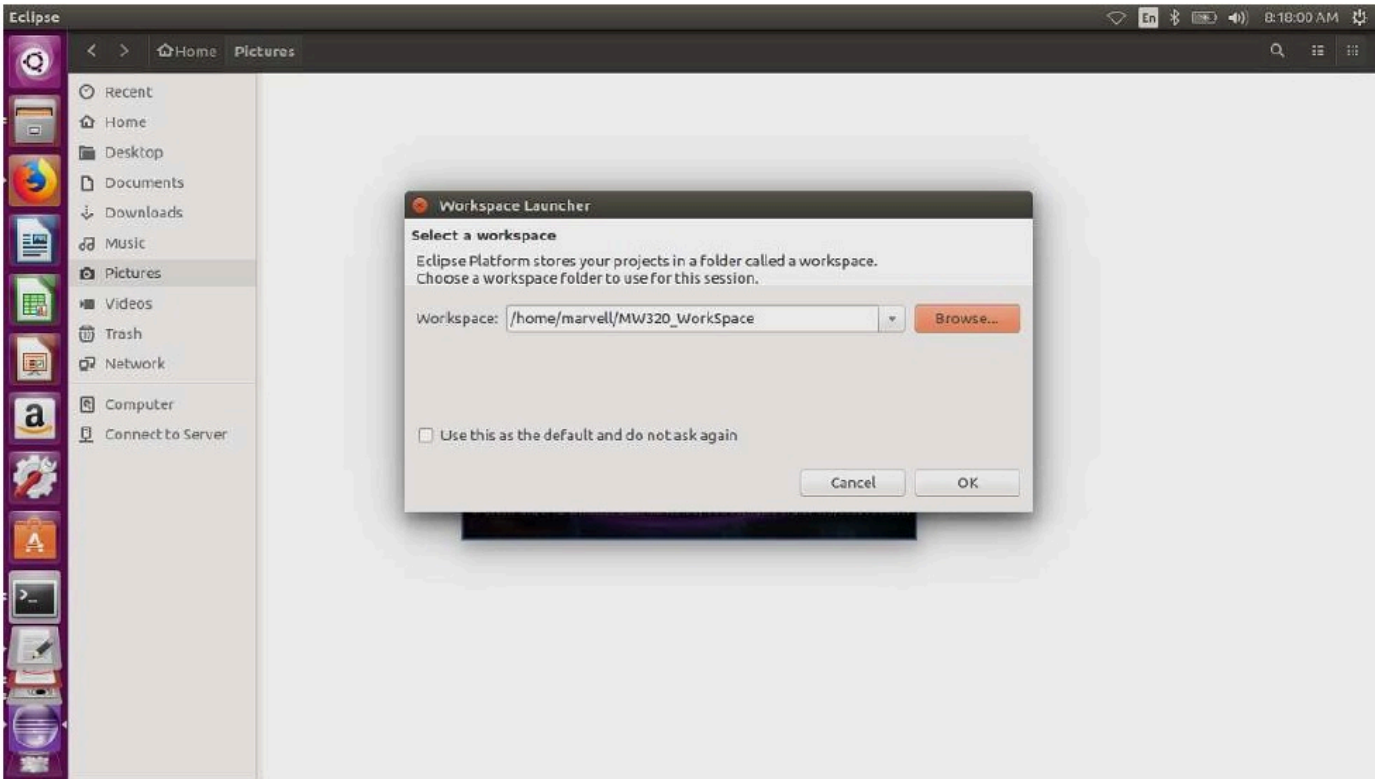
   ```
   cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S .  -Bbuild -
   DAFR_ENABLE_TESTS=0
   ```

   aws_tests Eclipse 프로젝트에서 작업을 수행하려면 다음 명령을 실행합니다.

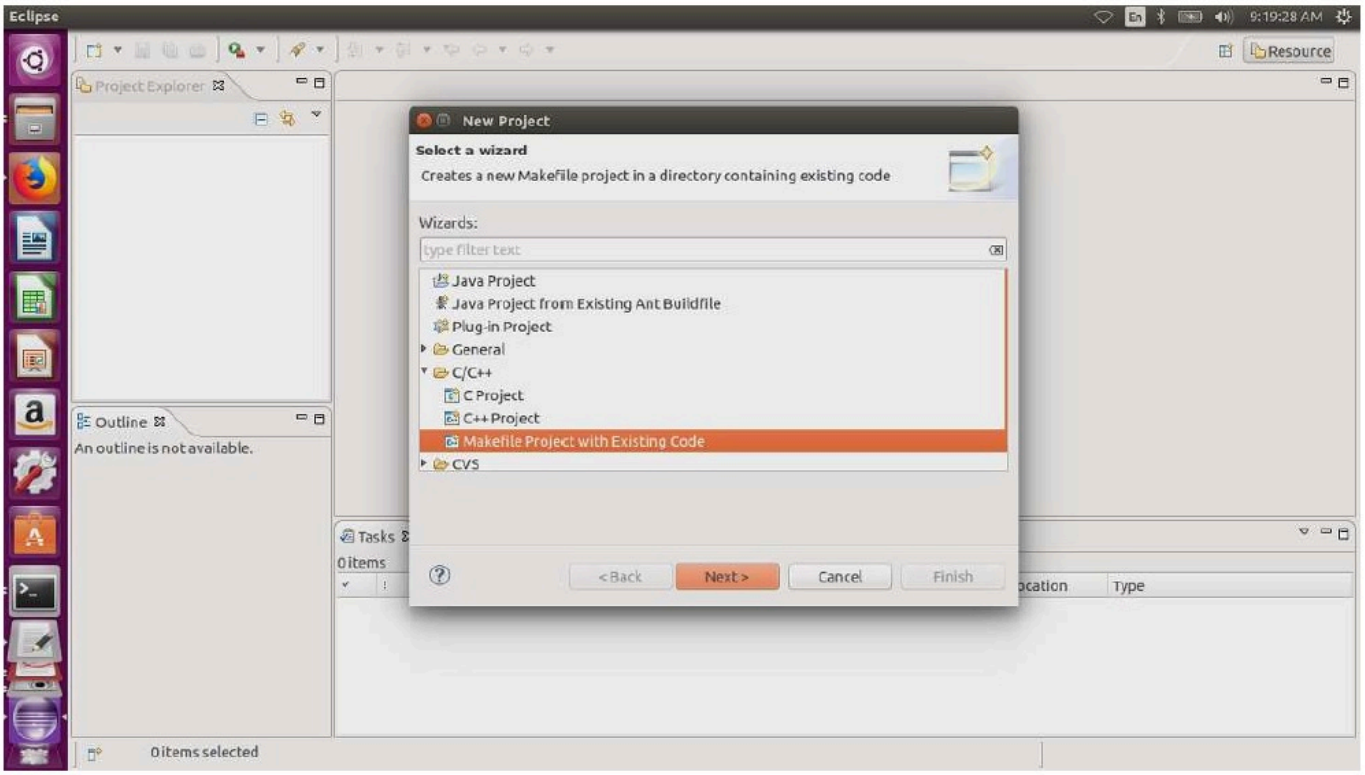
   ```
   cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S .  -Bbuild -
   DAFR_ENABLE_TESTS=1
   ```

> **Tip**
>
> aws_demos 프로젝트와 aws_tests 프로젝트 사이를 전환할 때마다 cmake 명령을 실행합니다.

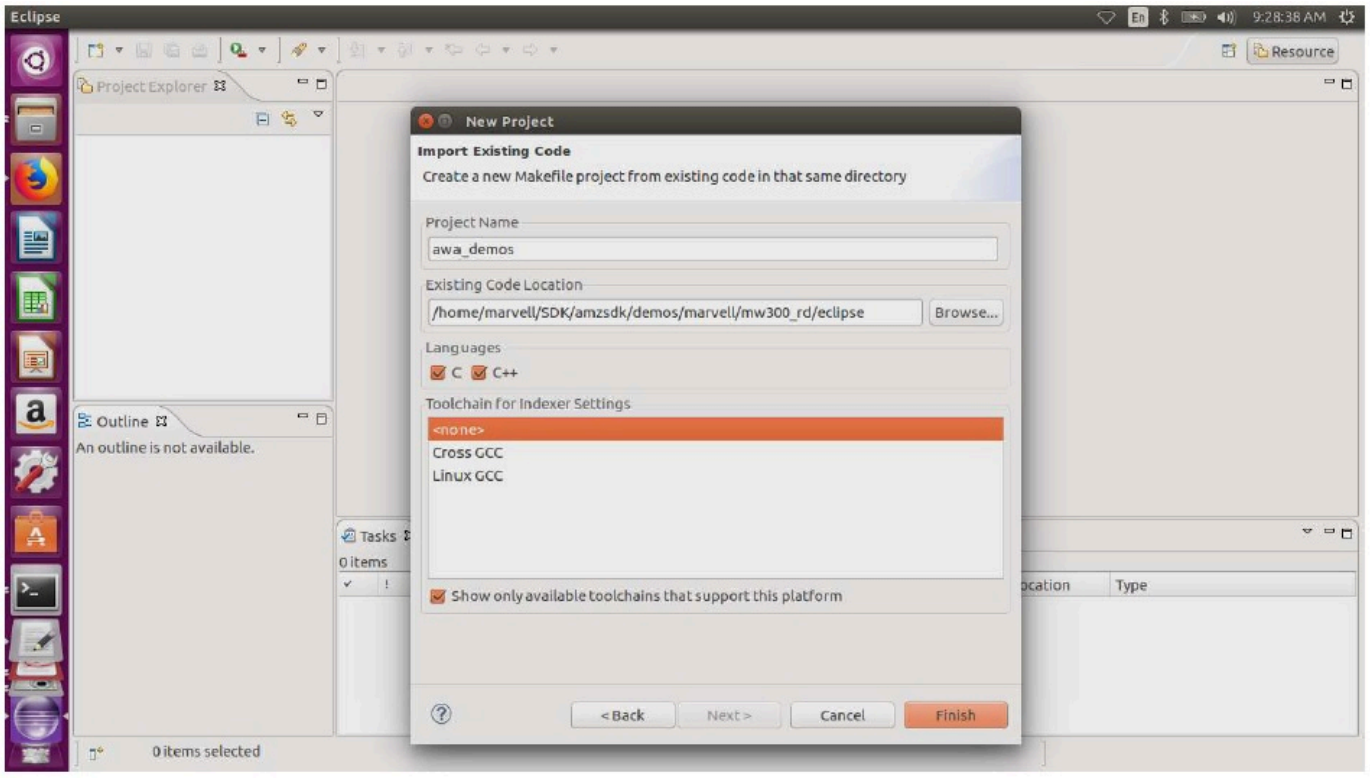
2. Eclipse를 열고 메시지가 표시되면 아래 그림과 같이 Eclipse 작업 공간을 선택합니다.



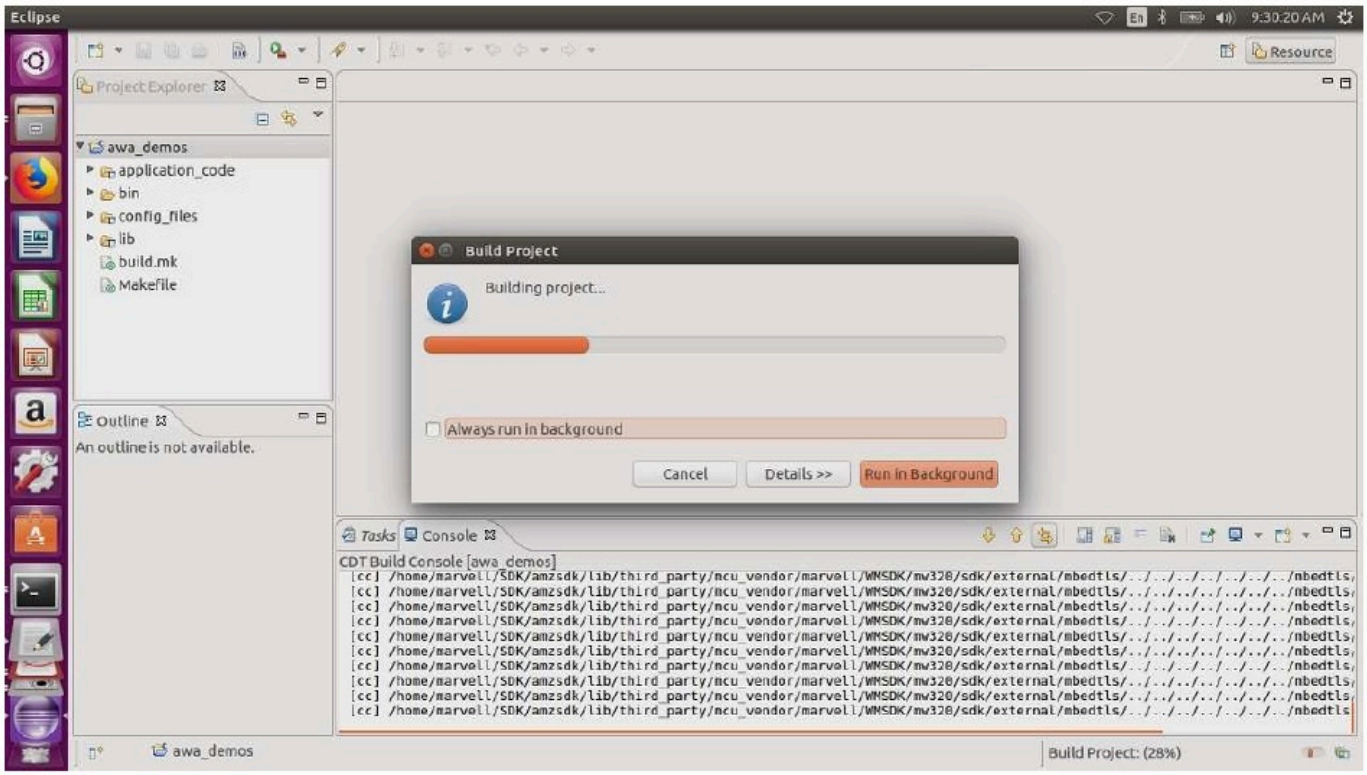
3. 아래 그림과 같이 Makefile 프로젝트: 기존 코드 사용 생성 옵션을 선택합니다.



4. 찾아보기를 선택하고 기존 코드의 디렉터리를 지정한 다음 마침을 선택합니다.



5. 탐색 창의 프로젝트 탐색기에서 aws_demos를 선택합니다. aws_demos를 마우스 오른쪽 버튼으로 클릭하여 메뉴를 열고 빌드를 선택합니다.



빌드가 성공하면 `build/cmake/vendors/marvell/mw300_rd/aws_demos.bin` 파일이 생성됩니다.

6. 명령줄 도구를 사용하여 Layout 파일(`layout.txt`), Boot2 바이너리(`boot2.bin`), MCU 펌웨어 바이너리(`aws_demos.bin`) 및 Wi-Fi 펌웨어를 플래시합니다.

   a. 펌웨어 이미지를 플래시하기 전에 공통 구성 요소인 Layout 및 Boot2를 사용하여 개발 보드의 플래시를 준비하세요. 다음 명령을 사용합니다.

   ```
   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
   ```

   `flashprog` 명령은 다음을 시작합니다.

   - 레이아웃 - flashprog 유틸리티는 먼저 플래시에 레이아웃을 쓰라는 명령을 받습니다. 레이아웃은 플래시의 파티션 정보와 비슷합니다. 기본 레이아웃은 `/lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/OpenOCD/mw300/layout.txt`에 있습니다.

- Boot2 - WMSDK에서 사용하는 부트 로더입니다. 또한 flashprog 명령은 플래시에 부트 로더를 씁니다. 마이크로컨트롤러의 펌웨어 이미지가 플래시되면 이를 로드하는 것이 부트 로더의 역할입니다. 아래 그림에 표시된 것과 동일한 출력이 표시되는지 확인하세요.

```
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. 펌웨어는 기능을 위해 Wi-Fi 칩셋을 사용하며, Wi-Fi 칩셋에는 자체 펌웨어가 있는데 플래시에도 이 펌웨어가 있어야 합니다. boot2 부트 로더 및 MCU 펌웨어를 플래시할 때와 동일한 방식으로 `flashprog.py` 유틸리티를 사용하여 Wi-Fi 펌웨어를 플래시합니다. 다음 명령을 사용하여 Wi-Fi 펌웨어를 플래시합니다.

```
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
```

명령 출력이 아래 그림과 비슷해야 합니다.

```
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. 다음 명령을 사용하여 MCU 펌웨어를 플래시합니다.

```
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. 보드를 재설정합니다. 데모 앱의 로그가 표시될 것입니다.

e. 테스트 앱을 실행하려면 동일한 디렉터리에 있는 `aws_tests.bin` 바이너리를 플래시합니다.

```
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
```

명령 출력이 아래 그림과 비슷하게 출력되어야 합니다.

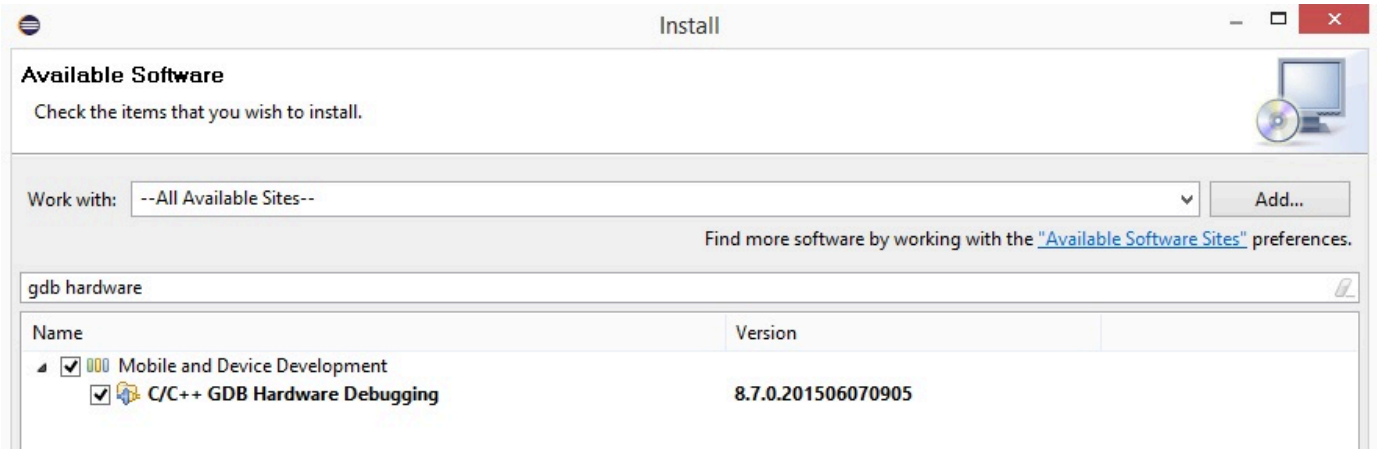
```
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00007f14 msp: 0x20001000
29088 bytes written at address 0x00100000
downloaded 29088 bytes in 0.245499s (115.708 KiB/s)
verified 29088 bytes in 0.350231s (81.107 KiB/s)
semihosting is enabled

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...
```

디버깅

- Eclipse를 시작하고 도움말을 선택한 다음 새 소프트웨어 설치를 선택합니다. 작업 메뉴에서 사용 가능한 모든 사이트를 선택합니다. 필터 텍스트 `GDB Hardware`를 입력합니다. C/C++ GDB 하드웨어 디버깅 옵션을 선택하고 플러그인을 설치합니다.



문제 해결

네트워크 문제

네트워크 보안 인증 정보를 확인합니다. FreeRTOS 데모 프로젝트 빌드 및 실행의 '프로비저닝'을 참조하세요.

추가 로그 활성화

- 보드 관련 로그를 활성화합니다.

  테스트 또는 데모를 위해 `main.c` 파일의 `prvMiscInitialization` 함수에서 `wmstdio_init(UART0_ID, 0)`에 대한 호출을 활성화합니다.

- Wi-Fi 로그 활성화

  freertos`/vendors/marvell/WMSDK/mw320/sdk/src/incl/autoconf.h` 파일에서 매크로 `CONFIG_WLCMGR_DEBUG`를 활성화합니다.

GDB 사용

SDK와 함께 패키징된 `arm-none-eabi-gdb` 및 gdb 명령 파일을 사용하는 것이 좋습니다. 디렉터리로 이동합니다.

```
cd freertos/lib/third_party/mcu_vendor/marvell/WMSDK/mw320
```

다음 명령(한 줄)을 실행하여 GDB에 연결합니다.

```
arm-none-eabi-gdb -x ./sdk/tools/OpenOCD/gdbinit ../../../../../../build/cmake/
vendors/marvell/mw300 _rd/aws_demos.axf
```

MediaTek MT7697Hx 개발 키트 시작하기

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

이 자습서에서는 MediaTek MT7697Hx 개발 키트를 시작하기 위한 지침을 제공합니다. MediaTek MT7697Hx 개발 키트가 없는 경우 AWS 파트너 디바이스 카탈로그를 방문하여 파트너로부터 구매하세요.

시작하기 전에 AWS IoT 및 FreeRTOS 다운로드를 구성하여 디바이스를 클라우드에 AWS 연결해야 합니다. 자세한 내용은 첫 번째 단계 섹션을 참조하세요. 이 자습서에서는 FreeRTOS 다운로드 디렉터리의 경로를 *freertos*라고 합니다.

개요

이 자습서에는 다음의 시작하기 단계에 대한 지침이 포함되어 있습니다.

1. 마이크로 컨트롤러 보드용 내장형 애플리케이션을 개발 및 디버깅하기 위한 소프트웨어를 호스트 시스템에 설치합니다.
2. FreeRTOS 데모 애플리케이션을 바이너리 이미지로 크로스 컴파일합니다.
3. 애플리케이션 바이너리 이미지를 보드에 로드한 후 애플리케이션을 실행합니다.
4. 모니터링 및 디버깅을 위해 직렬 연결로 보드에서 실행되는 애플리케이션과 상호 작용합니다.

개발 환경 설정

환경을 설정하기 전에 컴퓨터를 MediaTek MT7697Hx 개발 키트의 USB 포트에 연결하십시오.

Keil MDK 다운로드 및 설치

GUI 기반의 Keil MDK(Microcontroller Development Kit)를 사용하여 보드에 FreeRTOS 프로젝트를 구성, 빌드 및 실행할 수 있습니다. Keil MDK는 µVision IDE 및 µVision Debugger를 포함합니다.

> **Note**
>
> Keil MDK는 Windows 7, Windows 8 및 Windows 10 64비트 컴퓨터에서만 지원됩니다.

Keil MDK를 다운로드하고 설치하려면

1. Keil MDK 시작하기 페이지로 이동하여 Download MDK-Core(MDK 코어 다운로드)를 선택합니다.
2. Keil로 등록하려면 사용자의 정보를 입력하고 제출합니다.
3. MDK 실행 파일을 마우스 오른쪽 버튼으로 클릭하고 Keil MDK 설치 관리자를 컴퓨터에 저장합니다.
4. Keil MDK 설치 관리자를 열고 단계에 따라 완료합니다. MediaTek 디바이스 팩(MT76x7 시리즈)을 설치해야 합니다.

직렬 연결 설정

MediaTek MT7697Hx 개발 키트와 직렬 연결을 설정하려면 Arm Mbed Windows 직렬 포트 드라이버를 설치해야 합니다. Mbed에서 드라이버를 다운로드할 수 있습니다. Windows serial driver(Windows 직렬 드라이버) 페이지의 단계에 따라 MediaTek MT7697Hx 개발 키트의 드라이버를 다운로드하고 설치합니다.

드라이버를 설치한 후에는 COM 포트가 Windows Device Manager에 나타납니다. 디버깅을 위해 HyperTerminal 또는 TeraTerm과 같은 터미널 유틸리티 도구를 사용하여 포트에 대한 세션을 열 수 있습니다.

> **Note**
>
> 드라이버를 설치한 후 보드 연결에 문제가 있을 경우 시스템을 재부팅해야 할 수 있습니다.

Keil MDK로 FreeRTOS 데모 프로젝트 빌드 및 실행

Keil µVision에서 FreeRTOS 데모 프로젝트를 빌드하려면

1. 시작 메뉴에서 Keil µVision 5를 엽니다.
2. `projects/mediatek/mt7697hx-dev-kit/uvision/aws_demos/aws_demos.uvprojx` 프로젝트 파일을 엽니다.
3. 메뉴에서 프로젝트를 선택한 후 Build target(빌드 대상)을 선택합니다.

   코드가 빌드되면 `projects/mediatek/mt7697hx-dev-kit/uvision/aws_demos/out/Objects/aws_demo.axf`에서 데모 실행 파일을 볼 수 있습니다.

FreeRTOS 데모 프로젝트를 실행하려면

1. MediaTek MT7697Hx 개발 키트를 PROGRAM 모드로 설정합니다.

   키트를 PROGRAM 모드로 설정하려면 PROG 버튼을 길게 누릅니다. PROG 버튼을 계속 누른 상태에서 재설정 버튼에서 손을 뗀 후 PROG 버튼에서 손을 뗍니다.
2. 메뉴에서 Flash(플래시)를 선택한 후 Configure Flash Tools(플래시 도구 구성)를 선택합니다.
3. Options for Target(대상 옵션)'**aws_demo**'에서 디버깅 탭을 선택합니다. 사용을 선택하고 디버깅을 CMSIS-DAP Debugger(CMSIS-DAP 디버거)로 설정한 후 확인을 선택합니다.
4. 메뉴에서 Flash(플래시)를 선택한 후 다운로드를 선택합니다.

   다운로드가 완료되면 µVision에서 알립니다.
5. 터미널 유틸리티를 사용하여 직렬 콘솔 창을 엽니다. 직렬 포트를 115200bps, 패리티 없음, 8비트 및 1 정지 비트로 설정합니다.
6. MediaTek MT7697Hx 개발 키트에서 재설정 버튼을 선택합니다.

클라우드에서 MQTT 메시지 모니터링

AWS IoT 콘솔에서 MQTT 클라이언트를 사용하여 디바이스가 AWS 클라우드로 보내는 메시지를 모니터링할 수 있습니다.

MQTT 클라이언트를 사용하여 AWS IoT MQTT 주제를 구독하려면

1. AWS IoT 콘솔에 로그인합니다.
2. 탐색 창에서 Test(테스트)를 선택하여 MQTT 클라이언트를 엽니다.
3. 구독 주제에 ***your-thing-name*example/topic**을 입력한 다음 주제 구독을 선택합니다.

문제 해결

Keil µVision에서 FreeRTOS 프로젝트 디버깅

현재 Keil µVision이 포함된 MediaTek의 FreeRTOS 데모 프로젝트를 디버깅하기 전에 Keil µVision에 포함된 MediaTek 패키지를 편집해야 합니다.

FreeRTOS 프로젝트를 디버깅하기 위해 MediaTek 패키지를 편집하려면

1. Keil MDK 설치 폴더에서 `Keil_v5\ARM\PACK\.Web\MediaTek.MTx.pdsc` 파일을 검색하여 엽니다.

2. `flag = Read32(0x20000000);`의 인스턴스를 `flag = Read32(0x0010FBFC);`로 모두 바꿉니다.
3. `Write32(0x20000000, 0x76877697);`의 인스턴스를 `Write32(0x0010FBFC, 0x76877697);`로 모두 바꿉니다.

프로젝트 디버깅을 시작하려면

1. 메뉴에서 Flash(플래시)를 선택한 후 Configure Flash Tools(플래시 도구 구성)를 선택합니다.
2. 대상 탭을 선택한 후 Read/Write Memory Areas(읽기/쓰기 메모리 영역)를 선택합니다. IRAM1 및 IRAM2가 모두 선택되었는지 확인합니다.
3. 디버깅 탭을 선택한 후 CMSIS-DAP Debugger(CMSIS-DAP 디버거)를 선택합니다.
4. `vendors/mediatek/boards/mt7697hx-dev-kit/aws_demos/application_code/main.c`를 열고 매크로 `MTK_DEBUGGER`를 `1`로 설정합니다.
5. µVision에서 데모 프로젝트를 다시 빌드합니다.
6. MediaTek MT7697Hx 개발 키트를 PROGRAM 모드로 설정합니다.

   키트를 PROGRAM 모드로 설정하려면 PROG 버튼을 길게 누릅니다. PROG 버튼을 계속 누른 상태에서 재설정 버튼에서 손을 뗀 후 PROG 버튼에서 손을 뗍니다.
7. 메뉴에서 Flash(플래시)를 선택한 후 다운로드를 선택합니다.

   다운로드가 완료되면 µVision에서 알립니다.
8. MediaTek MT7697Hx 개발 키트에서 재설정 버튼을 누릅니다.
9. µVision 메뉴에서 디버깅을 선택하고 디버깅 세션 시작/중지를 선택합니다. 디버깅 세션을 시작하면 Call Stack + Locals(스택 + 로컬 호출) 창이 열립니다.
10. 메뉴에서 디버깅을 선택한 후 중지를 선택하여 코드 실행을 일시 중지합니다. 프로그램 카운터는 다음 줄에서 정지합니다.

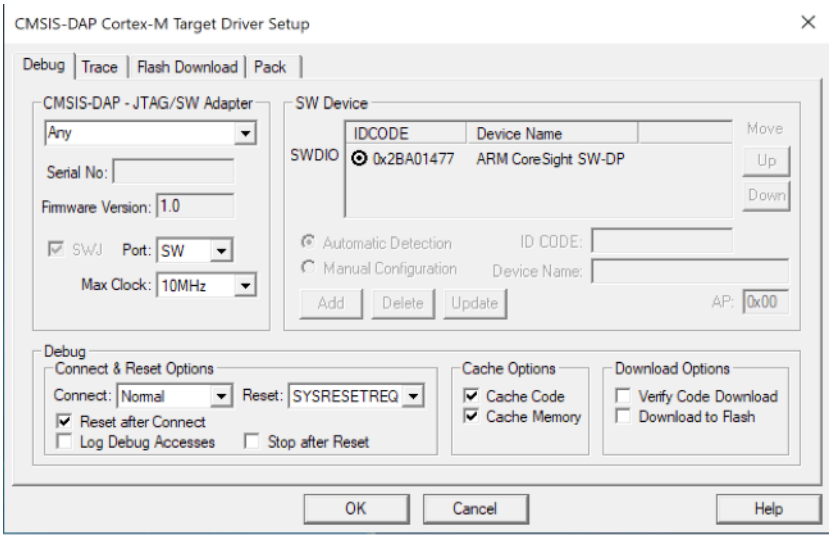
    ```
    { volatile int wait_ice = 1 ; while ( wait_ice ) ; }
    ```
11. Call Stack + Locals(스택 + 로컬 호출) 창에서`wait_ice` 값을 `0`로 변경합니다.
12. 프로젝트의 소스 코드에 중단점을 설정하고 코드를 실행합니다.

IDE 디버거 설정 문제 해결

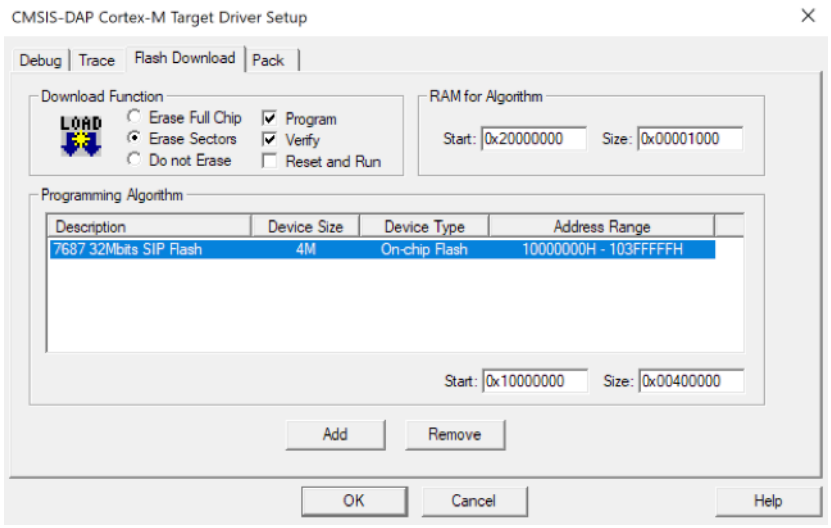
애플리케이션 디버깅에 문제가 있는 경우 디버거 설정이 잘못된 것일 수 있습니다.

디버거 설정이 올바른지 확인하려면

1. Keil µVision을 엽니다.
2. aws_demos 프로젝트를 마우스 오른쪽 버튼으로 클릭하고 옵션 선택한 다음 유틸리티 탭에서 '-디버그 드라이버 사용 -' 옆의 설정을 선택합니다.
3. 디버깅 탭 아래에서 설정이 다음과 같이 표시되는지 확인합니다.



4. Flash Download(플래시 다운로드) 탭 아래에서 설정이 다음과 같이 표시되는지 확인합니다.



FreeRTOS 시작하기에 대한 일반 문제 해결 정보는 시작하기 문제 해결 섹션을 참조하세요.

Microchip Curiosity PIC32MZ EF 시작하기

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

이 자습서에서는 Microchip Curiosity PIC32MZ EF를 시작하기 위한 지침을 제공합니다. Microchip Curiosity PIC32MZ EF 번들이 없는 경우 AWS 파트너 디바이스 카탈로그를 방문하여 파트너로부터 구매하세요.

번들에는 다음 항목이 포함됩니다.

- Curiosity PIC32MZ EF Development Board
- MikroElectronika USB UART Click Board
- MikroElectronika WiFi 7 Click Board
- PIC32 LAN8720 PHY 도터 보드

또한 디버깅을 위해 다음 항목도 필요합니다.

- MPLAB Snap In-Circuit Debugger
- (선택 사항) PICkit 3 Programming Cable Kit

시작하기 전에 AWS IoT 및 FreeRTOS 다운로드를 구성하여 디바이스를 AWS 클라우드에 연결해야 합니다. 자세한 내용은 첫 번째 단계 섹션을 참조하세요.

> **Important**
>
> - 이 주제에서는 FreeRTOS 다운로드 디렉터리의 경로를 *freertos*이라고 합니다.
> - *freertos* 경로의 공백 문자로 인해 빌드 실패가 발생할 수 있습니다. 리포지토리를 복제하거나 복사할 때 생성하는 경로에 공백 문자가 없어야 합니다.
> - Microsoft Windows에서 파일 경로의 최대 길이는 260자입니다. FreeRTOS 다운로드 디렉터리 경로가 길면 빌드 오류가 발생할 수 있습니다.

개요

이 자습서에는 다음의 시작하기 단계에 대한 지침이 포함되어 있습니다.

1. 보드를 호스트 시스템에 연결합니다.
2. 마이크로 컨트롤러 보드용 내장형 애플리케이션을 개발 및 디버깅하기 위한 소프트웨어를 호스트 시스템에 설치합니다.
3. FreeRTOS 데모 애플리케이션을 바이너리 이미지로 크로스 컴파일합니다.
4. 애플리케이션 바이너리 이미지를 보드에 로드한 후 애플리케이션을 실행합니다.
5. 모니터링 및 디버깅을 위해 직렬 연결로 보드에서 실행되는 애플리케이션과 상호 작용합니다.

Microchip Curiosity PIC32MZ EF 하드웨어 설정

1. MikroElectronika USB UART Click Board를 Microchip Curiosity PIC32MZ EF의 microBUS 1 커넥터에 연결합니다.
2. PIC32 LAN8720 PHY 도터 보드를 Microchip Curiosity PIC32MZ EF의 J18 헤더에 연결합니다.
3. USB A - USB 미니 B 케이블을 사용하여 MikroElectronika USB UART Click Board를 컴퓨터에 연결합니다.
4. 보드를 인터넷에 연결하려면 다음 옵션 중 하나를 사용하십시오.

   - Wi-Fi를 사용하려면 MikroElectronika Wi-Fi 7 클릭 보드를 Microchip Curiosity PIC32MZ EF의 microBUS 2 커넥터에 연결합니다. FreeRTOS 데모 구성을 참조하세요.
   - 이더넷을 사용하여 Microchip Curiosity PIC32MZ EF 보드를 인터넷에 연결하려면, PIC32 LAN8720 PHY 부속 보드를 Microchip Curiosity PIC32MZ EF의 J18 헤더에 연결합니다. 이더넷 케이블의 한 쪽 끝을 LAN8720 PHY 도터 보드에 연결합니다. 다른 쪽 끝을 라우터 또는 다른 인터넷 포트에 연결합니다. 또한 프리프로세서 매크로 `PIC32_USE_ETHERNET`을 정의해야 합니다.

5. 아직 연결하지 않은 경우 앵글 커넥터를 Microchip Curiosity PIC32MZ EF의 ICSP 헤더에 납땜합니다.
6. PICkit 3 Programming Cable Kit의 ICSP 케이블 한쪽 끝을 Microchip Curiosity PIC32MZ EF에 연결합니다.

   PICkit 3 Programming Cable Kit가 없는 경우 M-F Dupont 와이어 점퍼를 사용하여 연결합니다. 흰색 원은 Pin 1의 위치를 나타냅니다.

7. ICSP 케이블(또는 점퍼)의 다른쪽 끝을 MPLAB Snap Debugger에 연결합니다. 8핀 SIL Programming Connector의 Pin 1은 보드 오른쪽 하단에 검은색 사각형으로 표시됩니다.

   Microchip Curiosity PIC32MZ EF의 Pin 1에 연결된 케이블(흰색 원으로 표시됨)이 MPLAB Snap Debugger의 Pin 1에 맞아야 합니다.

   MPLAB Snap Debugger에 대한 자세한 내용은 MPLAB Snap In-Circuit Debugger Information Sheet를 참조하세요.

PKOB(PICkit On Board)를 사용하여 Microchip Curiosity PIC32MZ EF 하드웨어 설정

이전 섹션의 설정 절차를 따르는 것이 좋습니다. 그러나 다음 단계에 따라 통합 PKOB(PICkit On Board) 프로그래머/디버거를 사용하여 기본 디버깅으로 FreeRTOS 데모를 평가 및 실행할 수 있습니다.

1. MikroElectronika USB UART Click Board를 Microchip Curiosity PIC32MZ EF의 microBUS 1 커넥터에 연결합니다.
2. 보드를 인터넷에 연결하려면 다음 중 하나를 수행합니다.

   - Wi-Fi를 사용하려면 MikroElectronika Wi-Fi 7 클릭 보드를 Microchip Curiosity PIC32MZ EF의 microBUS 2 커넥터에 연결합니다. (FreeRTOS 데모 구성의 "Wi-Fi를 구성하려면"에 나오는 단계를 따르십시오.)
   - 이더넷을 사용하여 Microchip Curiosity PIC32MZ EF 보드를 인터넷에 연결하려면, PIC32 LAN8720 PHY 부속 보드를 Microchip Curiosity PIC32MZ EF의 J18 헤더에 연결합니다. 이더넷 케이블의 한 쪽 끝을 LAN8720 PHY 도터 보드에 연결합니다. 다른 쪽 끝을 라우터 또는 다른 인터넷 포트에 연결합니다. 또한 프리프로세서 매크로 `PIC32_USE_ETHERNET`을 정의해야 합니다.

3. USB 마이크로 B 케이블에 대한 USB 타입 A를 사용하여 Microchip Curiosity PIC32MZ EF 보드의 "USB DEBUG"라는 USB 마이크로 B 포트를 컴퓨터에 연결합니다.
4. USB A - USB 미니 B 케이블을 사용하여 MikroElectronika USB UART Click Board를 컴퓨터에 연결합니다.

개발 환경 설정

> **Note**
>
> 이 디바이스의 FreeRTOS 프로젝트는 MPLAB Harmony v2를 기반으로 합니다. 프로젝트를 빌드하려면 MPLAB XC32 Compiler 버전 v2.10 및 MHLB Harmony Configurator(MHC) 버전 2.X.X와 같은 Harmony v2와 호환되는 MPLAB 도구 버전을 사용해야 합니다.

1. Python 버전 3.x 이상을 설치합니다.
2. MPLAB X IDE를 설치합니다.

   > **Note**
   >
   > FreeRTOS AWS 참조 통합 v202007.00은 현재 MPLabv5.35에서만 지원됩니다. 이전 버전의 FreeRTOS AWS 참조 통합은 MPLabv5.40.

   MPLABv5.35 다운로드

   - Windows용 MPLAB X 통합 개발 환경
   - macOS용 MPLAB X 통합 개발 환경
   - Linux용 MPLAB X 통합 개발 환경

   최신 MPLab 다운로드(MPLAb v5.40)

   - Windows용 MPLAB X 통합 개발 환경
   - macOS용 MPLAB X 통합 개발 환경
   - Linux용 MPLAB X 통합 개발 환경
3. MPLAB XC32 Compiler를 설치합니다.

   - Windows용 MPLAB XC32/32++ Compiler
   - macOS용 MPLAB XC32/32++ 컴파일러
   - Linux용 MPLAB XC32/32++ Compiler
4. UART 터미널 에뮬레이터를 시작하고 다음 설정으로 연결을 엽니다.

- 전송 속도: 115200
- 데이터: 8비트
- 패리티: 없음
- 정지 비트: 1
- 흐름 제어: 없음

FreeRTOS 데모 프로젝트 빌드 및 실행

MPLAB IDE에서 FreeRTOS 데모 열기

1. MPLAB IDE를 엽니다. 두 개 이상의 컴파일러 버전이 설치된 경우 IDE에서 사용할 컴파일러를 선택해야 합니다.
2. 파일 메뉴에서 Open Project(프로젝트 열기)를 선택합니다.
3. `projects/microchip/curiosity_pic32mzef/mplab/aws_demos`로 이동하여 엽니다.
4. Open project(프로젝트 열기)를 선택합니다.

> **Note**
>
> 프로젝트를 처음 열면 컴파일러에 대한 오류 메시지가 표시될 수 있습니다. IDE에서 도구, 옵션, Embedded(임베디드)로 이동한 다음 프로젝트에 사용 중인 컴파일러를 선택합니다.

이더넷으로 연결하려면 프리프로세서 매크로 `PIC32_USE_ETHERNET`을 정의해야 합니다.

MPLAB IDE를 사용하여 이더넷으로 연결하려면

1. MPLAB IDE에서 프로젝트를 마우스 오른쪽 버튼으로 클릭한 다음 속성을 선택합니다.
2. 프로젝트 속성 대화 상자에서 ***compiler-name***(글로벌 옵션)을 선택하여 확장한 다음 ***compiler-name***-gcc를 선택합니다.
3. 옵션 범주에서 전처리 및 메시지를 선택하고 `PIC32_USE_ETHERNET` 문자열을 프리프로세서 매크로에 추가합니다.

FreeRTOS 데모 프로젝트 실행

1. 프로젝트를 다시 빌드합니다.

2. Projects(프로젝트) 탭에서 aws_demos 최상위 수준 폴더를 마우스 오른쪽 버튼으로 클릭하고 Debug(디버그)를 선택합니다.
3. 디버거가 main()의 중단점에서 중지되면 실행 메뉴에서 다시 시작을 선택합니다.

CMake로 FreeRTOS 데모 빌드

FreeRTOS 개발용 IDE를 사용하지 않으려는 경우 CMake를 사용하여 데모 애플리케이션이나 타사 코드 편집기 및 디버깅 도구를 사용하여 개발한 애플리케이션을 빌드하고 실행할 수 있습니다.

CMake로 FreeRTOS 데모를 빌드하려면

1. 생성된 빌드 파일을 포함하는 디렉터리를 생성합니다(예: *build-directory*).
2. 소스 코드에서 빌드 파일을 생성하려면 다음 명령을 사용합니다.

```
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**
>
> Hexmate 및 도구 체인 바이너리에 대한 올바른 경로를 지정해야 합니다(예: C:\Program Files (x86)\Microchip\MPLABX\v5.35\mplab_platform\bin 및 C:\Program Files\Microchip\xc32\v2.40\bin 경로).

3. 디렉터리를 빌드 디렉터리(*build-directory*)로 변경하고 해당 디렉터리에서 make를 실행합니다.

자세한 내용은 FreeRTOS에서 CMake 사용 단원을 참조하십시오.

이더넷으로 연결하려면 프리프로세서 매크로 PIC32_USE_ETHERNET을 정의해야 합니다.

클라우드에서 MQTT 메시지 모니터링

AWS IoT 콘솔에서 MQTT 클라이언트를 사용하여 디바이스가 AWS 클라우드로 보내는 메시지를 모니터링할 수 있습니다.

MQTT 클라이언트를 사용하여 AWS IoT MQTT 주제를 구독하려면

1. AWS IoT 콘솔에 로그인합니다.

2. 탐색 창에서 Test(테스트)를 선택하여 MQTT 클라이언트를 엽니다.
3. 구독 주제에 ***your-thing-name*/example/topic**을 입력한 다음 주제 구독을 선택합니다.

문제 해결

문제 해결 정보는 시작하기 문제 해결를 참조하세요.

Nordic nRF52840-DK 시작하기

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

이 자습서에서는 Nordic nRF52840-DK를 시작하기 위한 지침을 제공합니다. Nordic nRF52840-DK가 없는 경우 AWS 파트너 디바이스 카탈로그를 방문하여 파트너로부터 구매하세요.

시작하려면 FreeRTOS Bluetooth Low Energy용 AWS IoT 및 Amazon Cognito 설정 단원을 수행해야 합니다.

FreeRTOS Bluetooth Low Energy 데모를 실행하려면 Bluetooth 및 Wi-Fi 기능이 있는 iOS 또는 Android 모바일 디바이스도 필요합니다.

> **Note**
>
> iOS 디바이스를 사용하는 경우 데모 모바일 애플리케이션을 빌드하려면 Xcode가 필요합니다. Android 디바이스를 사용하는 경우 Android Studio를 사용하여 데모 모바일 애플리케이션을 빌드할 수 있습니다.

개요

이 자습서에는 다음의 시작하기 단계에 대한 지침이 포함되어 있습니다.

1. 보드를 호스트 시스템에 연결합니다.
2. 마이크로 컨트롤러 보드용 내장형 애플리케이션을 개발 및 디버깅하기 위한 소프트웨어를 호스트 시스템에 설치합니다.

3. FreeRTOS 데모 애플리케이션을 바이너리 이미지로 크로스 컴파일합니다.
4. 애플리케이션 바이너리 이미지를 보드에 로드한 후 애플리케이션을 실행합니다.
5. 모니터링 및 디버깅을 위해 직렬 연결로 보드에서 실행되는 애플리케이션과 상호 작용합니다.

Nordic 하드웨어 설정

호스트 컴퓨터를 J2라는 레이블이 지정된 USB 포트에 연결합니다. 이 포트는 Nordic nRF52840 보드의 코인 셀 배터리 홀더 바로 위에 있습니다.

Nordic nRF52840-DK 설정에 대한 자세한 내용은 nRF52840 Development Kit 사용 설명서를 참조하십시오.

개발 환경 설정

Segger Embedded Studio 다운로드 및 설치

FreeRTOS는 Segger Embedded Studio를 Nordic nRF52840-DK의 개발 환경으로 지원합니다.

환경을 설정하려면 호스트 컴퓨터에 Segger Embedded Studio를 다운로드하고 설치해야 합니다.

Segger Embedded Studio를 다운로드하고 설치하려면

1. Segger Embedded Studio Downloads 페이지로 이동하고 해당 운영 체제의 Embedded Studio for ARM 옵션을 선택합니다.
2. 설치 관리자를 실행하고 표시되는 메시지에 따라 완료합니다.

FreeRTOS Bluetooth Low Energy Mobile SDK 데모 애플리케이션 설정

Bluetooth Low Energy에서 FreeRTOS 데모 프로젝트를 실행하려면 모바일 디바이스에서 FreeRTOS Bluetooth Low Energy Mobile SDK 데모 애플리케이션을 실행해야 합니다.

FreeRTOS Bluetooth Low Energy Mobile SDK 데모 애플리케이션을 설정하려면

1. FreeRTOS Bluetooth 디바이스용 Mobile SDK의 지침에 따라 모바일 플랫폼용 SDK를 다운로드하여 호스트 컴퓨터에 설치합니다.
2. FreeRTOS Bluetooth Low Energy Mobile SDK 데모 애플리케이션의 지침에 따라 모바일 디바이스에서 데모 모바일 애플리케이션을 설정합니다.

직렬 연결 설정

Segger Embedded Studio에는 보드에 대한 직렬 연결을 통해 로그 메시지를 수신하는 데 사용할 수 있는 터미널 에뮬레이터가 포함되어 있습니다.

Segger Embedded Studio와 직렬 연결을 설정하려면

1. Segger Embedded Studio를 엽니다.
2. 상단 메뉴에서 대상을 선택하고, Connect J-Link(J-Link 연결)를 선택합니다.
3. 상단 메뉴에서 도구, Terminal Emulator(터미널 에뮬레이터), 속성을 선택하고 속성을 터미널 에뮬레이터 설치의 지침에 따라 설정합니다.
4. 상단 메뉴에서 도구, Terminal Emulator(터미널 에뮬레이터), Connect ***port***(포트 연결)(115200,N,8,1)를 선택합니다.

> **Note**
>
> Segger 임베디드 스튜디오 터미널 에뮬레이터는 입력 기능을 지원하지 않습니다. 이 경우 PuTTy, Tera Term 또는 GNU Screen과 같은 터미널 에뮬레이터를 사용하십시오. 터미널 에뮬레이터 설치의 지침에 따라 터미널을 직렬 연결로 보드에 연결하도록 구성합니다.

FreeRTOS 다운로드 및 구성

하드웨어와 환경을 설정한 후 FreeRTOS를 다운로드할 수 있습니다.

FreeRTOS 다운로드

Nordic nRF52840-DK용 FreeRTOS를 다운로드하려면 FreeRTOS GitHub 페이지로 이동하고 리포지토리를 복제합니다. 자세한 내용은 README.md 파일을 참조하십시오.

> **Important**
>
> - 이 주제에서는 FreeRTOS 다운로드 디렉터리의 경로를 *`freertos`*이라고 합니다.
> - *`freertos`* 경로의 공백 문자로 인해 빌드 실패가 발생할 수 있습니다. 리포지토리를 복제하거나 복사할 때 생성하는 경로에 공백 문자가 없어야 합니다.
> - Microsoft Windows에서 파일 경로의 최대 길이는 260자입니다. FreeRTOS 다운로드 디렉터리 경로가 길면 빌드 오류가 발생할 수 있습니다.

프로젝트 구성

데모를 실행하려면 작업할 프로젝트를 구성해야 합니다 AWS IoT. 작업할 프로젝트를 구성하려면 AWS IoT디바이스를 AWS IoT 사물로 등록해야 합니다. FreeRTOS Bluetooth Low Energy용 AWS IoT 및 Amazon Cognito 설정을 수행할 때 디바이스가 등록되어 있어야 합니다.

AWS IoT 엔드포인트를 구성하려면

1. AWS IoT 콘솔에 로그인합니다.
2. 탐색 창에서 설정을 선택합니다.

   AWS IoT 엔드포인트 텍스트 상자에 엔드포인트가 나타납니다. URL은 *1234567890123*-ats.iot.*us-east-1*.amazonaws.com와 같아야 합니다. 이 엔드포인트를 기록해 둡니다.
3. 탐색 창에서 Manage(관리)를 선택한 다음 Things(사물)를 선택합니다. 디바이스의 AWS IoT 사물 이름을 기록해 둡니다.
4. AWS IoT 엔드포인트와 AWS IoT 사물 이름이 있는 상태에서 IDE*freertos*/demos/include/aws_clientcredential.h에서를 열고 다음 #define 상수에 대한 값을 지정합니다.

   - clientcredentialMQTT_BROKER_ENDPOINT *AWS IoT #####*
   - clientcredentialIOT_THING_NAME *### AWS IoT ## ##*

데모를 활성화하려면

1. Bluetooth Low Energy GATT 데모가 활성화되어 있는지 확인합니다. vendors/nordic/boards/nrf52840-dk/aws_demos/config_files/iot_ble_config.h로 이동하여 #define IOT_BLE_ADD_CUSTOM_SERVICES (1)를 정의문 목록에 추가합니다.
2. vendors/nordic/boards/nrf52840-dk/aws_demos/config_files/aws_demos_config.h를 열고 CONFIG_MQTT_DEMO_ENABLED를 정의합니다.
3. Nordic 칩에는 RAM(250KB)이 거의 없으므로 각 속성의 크기에 비해 더 큰 GATT 테이블 항목을 허용하도록 BLE 구성을 변경해야 할 수도 있습니다. 이런 방법으로 애플리케이션이 얻는 메모리의 양을 조정할 수 있습니다. 이렇게 하려면 파일 *freertos*/vendors/nordic/boards/nrf52840-dk/aws_demos/config_files/sdk_config.h에서 다음 속성을 재정의합니다.

   - NRF_SDH_BLE_VS_UUID_COUNT

     공급 업체별 UUID의 수입니다.
   - NRF_SDH_BLE_GATTS_ATTR_TAB_SIZE

속성 테이블 크기(바이트)입니다. 크기는 4의 배수여야 합니다.

(테스트의 경우 파일의 위치는 *freertos*/vendors/nordic/boards/nrf52840-dk/aws_tests/config_files/sdk_config.h입니다.)

FreeRTOS 데모 프로젝트 빌드 및 실행

FreeRTOS를 다운로드하고 데모 프로젝트를 구성하면 보드에서 데모 프로젝트를 빌드하고 실행할 준비가 된 것입니다.

> **Important**
>
> 이 보드에서 처음으로 데모를 실행하는 경우 데모를 실행하려면 먼저 부트로더를 보드로 플래시해야 합니다.
> 부트로더를 빌드 및 플래시하려면 아래 단계를 따르십시오. 그러나 projects/nordic/nrf52840-dk/ses/aws_demos/aws_demos.emProject 프로젝트 파일 대신 projects/nordic/nrf52840-dk/ses/aws_demos/bootloader/bootloader.emProject를 사용합니다.

Segger Embedded Studio에서 FreeRTOS Bluetooth Low Energy 데모를 빌드하고 실행하려면

1. Segger Embedded Studio를 엽니다. 상단 메뉴에서 파일을 선택하고 Open Solution(솔루션 열기)을 선택한 후 projects/nordic/nrf52840-dk/ses/aws_demos/aws_demos.emProject 프로젝트 파일로 이동합니다.
2. Segger Embedded Studio 터미널 에뮬레이터를 사용하는 경우 상단 메뉴에서 도구를 선택한 후 Terminal Emulator(터미널 에뮬레이터), Terminal Emulator(터미널 에뮬레이터)를 선택하여 직렬 연결에서 정보를 표시합니다.

   다른 터미널 도구를 사용하는 경우 직렬 연결에서 출력할 도구를 모니터링할 수 있습니다.
3. Project Explorer에서 aws_demos 데모 프로젝트를 마우스 오른쪽 버튼으로 클릭하고 빌드를 선택합니다.

> **Note**
>
> Segger Embedded Studio를 처음 사용하는 경우 "No license for commercial use"라는 경고가 표시될 수 있습니다. Segger Embedded Studio는 Nordic Semiconductor 디바이스에서 무료로 사용할 수 있습니다. Activate Your Free License를 선택하고 지침을 따릅니다.

4. 디버깅을 선택한 다음 이동을 선택합니다.

   데모가 시작되면 Bluetooth Low Energy를 통해 모바일 장치와 페어링되기를 기다립니다.

5. MQTT over Bluetooth Low Energy 데모 애플리케이션에 대한 지침에 따라 FreeRTOS Bluetooth Low Energy Mobile SDK 데모 애플리케이션을 모바일 MQTT 프록시로 사용하여 데모를 완료합니다.

문제 해결

FreeRTOS 시작하기에 대한 일반 문제 해결 정보는 시작하기 문제 해결 섹션을 참조하세요.

Nuvoton NuMaker-IoT-M487 시작하기

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

이 자습서에서는 Nuvoton NuMaker-IoT-M487 개발 보드를 시작하기 위한 지침을 제공합니다. 시리즈 마이크로컨트롤러로, RJ45 이더넷 및 Wi-Fi 모듈을 내장하고 있습니다. Nuvoton NuMaker-IoT-M487이 없는 경우 AWS Partner Device Catalog를 방문하여 파트너에서 구입하시기 바랍니다.

시작하기 전에 개발 보드를 AWS 클라우드에 연결하도록 AWS IoT 및 FreeRTOS 소프트웨어를 구성해야 합니다. 지침은 첫 번째 단계 단원을 참조하십시오. 이 자습서에서는 FreeRTOS 다운로드 디렉터리의 경로를 *freertos*라고 합니다.

개요

이 자습서에서는 다음과 같은 단계를 안내합니다.

1. 마이크로컨트롤러 보드용 임베디드 애플리케이션을 개발하고 디버깅하기 위한 소프트웨어를 호스트 머신에 설치합니다.
2. FreeRTOS 데모 애플리케이션을 바이너리 이미지로 크로스 컴파일합니다.
3. 애플리케이션 이진 이미지를 보드에 로드한 다음 애플리케이션을 실행합니다.

개발 환경 설정

Keil MDK Nuvoton 에디션은 Nuvoton M487 보드용 애플리케이션을 개발하고 디버깅하기 위해 설계되었습니다. Keil MDK v5 Essential, Plus 또는 Pro 버전은 Nuvoton M487(Cortex-M4 코어) MCU에도 작동합니다. Nuvoton Cortex-M4 시리즈 MCU의 가격 할인을 사용하여 Keil MDK Nuvoton 에디션을 다운로드할 수 있습니다. Keil MDK는 Windows에서만 지원됩니다.

NuMaker-IoT-M487용 개발 도구를 설치하려면

1. Keil MDK 웹 사이트에서 Keil MDK Nuvoton 에디션을 다운로드합니다.
2. 라이선스를 사용하여 호스트 머신에 Keil MDK를 설치합니다. Keil MDK에는 Keil µVision IDE, C/C++ 컴파일 도구 체인, µVision 디버거가 포함되어 있습니다.

   설치 중에 문제가 발생하는 경우 Nuvoton에 지원을 요청하십시오.
3. Nu-Link_Keil_Driver_V3.06.7215r(또는 최신 버전)을 설치합니다. 이 도구는 Nuvoton Development Tool 페이지에 있습니다.

FreeRTOS 데모 프로젝트 빌드 및 실행

FreeRTOS 데모 프로젝트를 빌드하려면

1. Keil µVision IDE를 엽니다.
2. File(파일) 메뉴에서 Open(열기)를 선택합니다. Open file(파일 열기) 대화 상자에서 파일 선택기가 Project Files(프로젝트 파일)로 설정되어 있는지 확인합니다.
3. 구축할 Wi-Fi 또는 이더넷 데모 프로젝트를 선택합니다.

   - Wi-Fi 데모 프로젝트를 열려면 *freertos*\projects\nuvoton\numaker_iot_m487_wifi\uvision\aws_demos 디렉터리에서 대상 프로젝트 aws_demos.uvproj를 선택합니다.
   - 이더넷 데모 프로젝트를 열려면 *freertos*\projects\nuvoton\numaker_iot_m487_wifi\uvision\aws_demos_eth 디렉터리에서 대상 프로젝트 aws_demos_eth.uvproj를 선택합니다.

4. 보드를 플래시하기 위한 설정이 올바른지 확인하려면 aws_demo 프로젝트를 마우스 오른쪽 버튼으로 클릭한 다음 Options(옵션)를 선택합니다. (자세한 내용은 문제 해결 단원을 참조하십시오.)
5. Utilities(유틸리티) 탭에서 Use Target Driver for Flash Programming(플래시 프로그래밍에 대상 드라이버 사용)이 선택되어 있는지 확인하고 Nuvoton Nu-Link Debugger(Nuvoton Nu-Link 디버거)가 대상 드라이버로 설정되어 있는지 확인합니다.
6. Debug(디버깅) 탭에서 Nuvoton Nu-Link Debugger(Nuvoton Nu-Link 디버거) 옆의 Settings(설정)를 선택합니다.
7. Chip Type(칩 유형)이 M480으로 설정되어 있는지 확인합니다.
8. Keil µVision IDE Project(프로젝트) 탐색 창에서 aws_demos 프로젝트를 선택합니다. Project(프로젝트) 메뉴에서 Build Target(대상 빌드)을 선택합니다.

AWS IoT 콘솔에서 MQTT 클라이언트를 사용하여 디바이스가 AWS 클라우드로 보내는 메시지를 모니터링할 수 있습니다.

MQTT 클라이언트를 사용하여 AWS IoT MQTT 주제를 구독하려면

1. AWS IoT 콘솔에 로그인합니다.
2. 탐색 창에서 Test(테스트)를 선택하여 MQTT 클라이언트를 엽니다.
3. 구독 주제에 ***your-thing-name*/example/topic**을 입력한 다음 주제 구독을 선택합니다.

FreeRTOS 데모 프로젝트를 실행하려면

1. Numaker-IoT-M487 보드를 호스트 머신(컴퓨터)에 연결합니다.
2. 프로젝트를 다시 빌드합니다.
3. Keil µVision IDE의 Flash(플래시) 메뉴에서 Download(다운로드)를 선택합니다.
4. Debug(디버깅) 메뉴에서 Start/Stop Debugging(디버깅 시작/중지)을 선택합니다.
5. 디버거가 main()의 중단점에서 중지되면 Run(실행) 메뉴를 열고 Run(실행)(F5)을 선택합니다.

   AWS IoT 콘솔의 MQTT 클라이언트에서 디바이스가 보낸 MQTT 메시지가 표시됩니다.

FreeRTOS에서 CMake 사용

CMake를 사용하여 FreeRTOS 데모 애플리케이션 또는 타사 코드 편집기 및 디버깅 도구를 사용하여 개발한 애플리케이션을 빌드하고 실행할 수 있습니다.

CMake 빌드 시스템을 설치했는지 확인합니다. FreeRTOS에서 CMake 사용의 지침을 따른 다음 이 단원의 단계를 따르십시오.

> **Note**
>
> 컴파일러(Keil)의 위치에 대한 경로가 경로 시스템 변수(예: `C:\Keil_v5\ARM\ARMCC\bin`)에 있어야 합니다.

AWS IoT 콘솔에서 MQTT 클라이언트를 사용하여 디바이스가 AWS 클라우드로 보내는 메시지를 모니터링할 수도 있습니다.

MQTT 클라이언트를 사용하여 AWS IoT MQTT 주제를 구독하려면

1. AWS IoT 콘솔에 로그인합니다.
2. 탐색 창에서 Test(테스트)를 선택하여 MQTT 클라이언트를 엽니다.
3. 구독 주제에 ***your-thing-name*/example/topic**을 입력한 다음 주제 구독을 선택합니다.

소스 파일에서 빌드 파일을 생성하고 데모 프로젝트를 실행하려면

1. 호스트 시스템에서 명령 프롬프트를 열고 *freertos* 폴더로 이동합니다.
2. 생성된 빌드 파일을 포함할 폴더를 만듭니다. 여기서는 이 폴더를 *BUILD_FOLDER*라고 합니다.
3. Wi-Fi 또는 이더넷 데모에 대한 빌드 파일을 생성합니다.

   - Wi-Fi의 경우:

     FreeRTOS 데모 프로젝트의 소스 파일이 포함된 디렉터리로 이동합니다. 다음 명령을 실행하여 빌드 파일을 생성합니다.

     ```
     cmake -DVENDOR=nuvoton -DBOARD=numaker_iot_m487_wifi -DCOMPILER=arm-keil -S . -
     B BUILD_FOLDER -G Ninja
     ```

   - 이더넷의 경우:

     FreeRTOS 데모 프로젝트의 소스 파일이 포함된 디렉터리로 이동합니다. 다음 명령을 실행하여 빌드 파일을 생성합니다.

```
cmake -DVENDOR=nuvoton -DBOARD=numaker_iot_m487_wifi -DCOMPILER=arm-keil -
DAFR_ENABLE_ETH=1 -S . -B BUILD_FOLDER -G Ninja
```

4. 다음 명령을 실행하여 M487로 플래시할 바이너리를 생성합니다.

   ```
   cmake --build BUILD_FOLDER
   ```

   이 시점에서는 이진 파일 `aws_demos.bin`이 *BUILD_FOLDER*`/vendors/Nuvoton/boards/numaker_iot_m487_wifi` 폴더에 있어야 합니다.

5. 플래시 모드에 맞게 보드를 구성하려면 MSG 스위치(ICE에서 ISW1의 4번)가 ON으로 전환되어 있는지 확인합니다. 보드에 플러그인하면 창(드라이브)이 할당됩니다. (문제 해결를 참조하세요.)

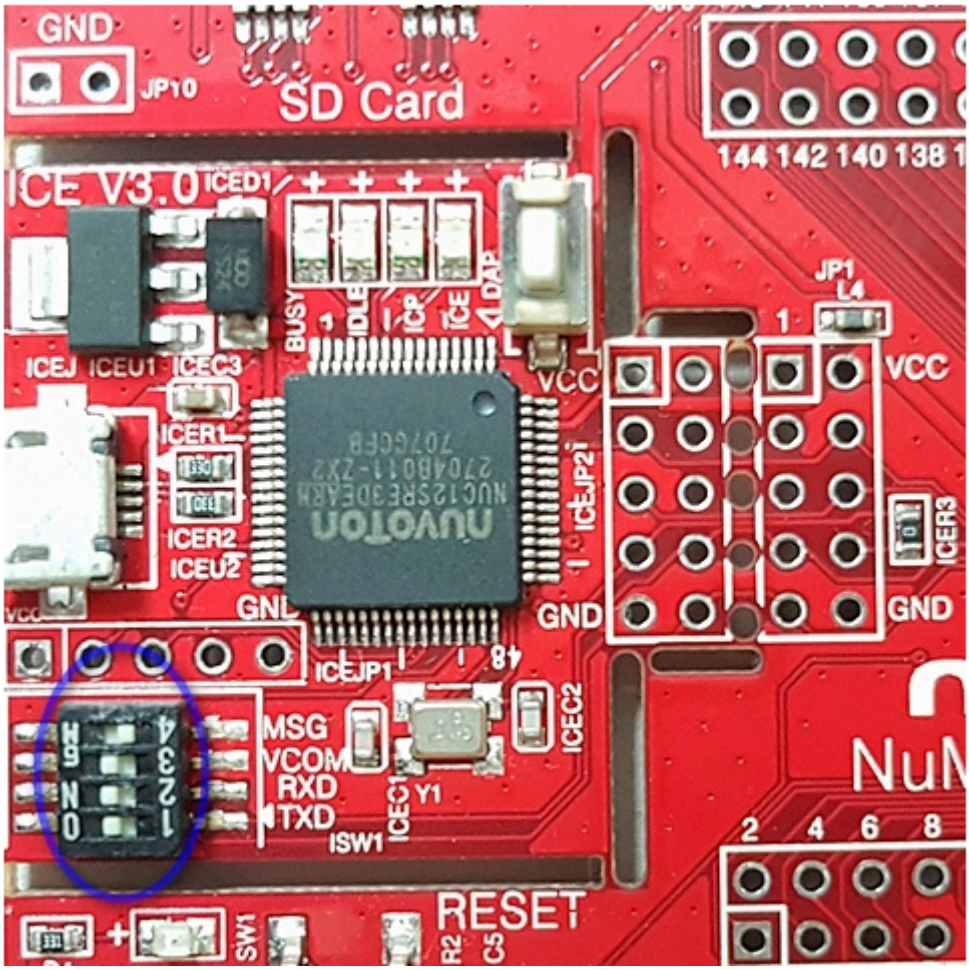
6. 터미널 에뮬레이터를 열어 UART를 통해 메시지를 봅니다. 터미널 에뮬레이터 설치의 지침을 따릅니다.

7. 생성된 이진수를 디바이스에 복사하여 데모 프로젝트를 실행합니다.

   MQTT 클라이언트를 사용하여 AWS IoT MQTT 주제를 구독한 경우 AWS IoT 콘솔에 디바이스에서 보낸 MQTT 메시지가 표시됩니다.

문제 해결

- Windows에서 디바이스 `VCOM`이 인식되지 않는 경우 Nu-Link USB Driver v1.6 링크에서 NuMaker Windows 직렬 포트 드라이버를 설치합니다.
- Nu-Link를 통해 디바이스를 Keil MDK(IDE)에 연결하는 경우 그림과 같이 MSG 스위치(ICE에서 ISW1의 4번)가 OFF로 전환되어 있는지 확인합니다.



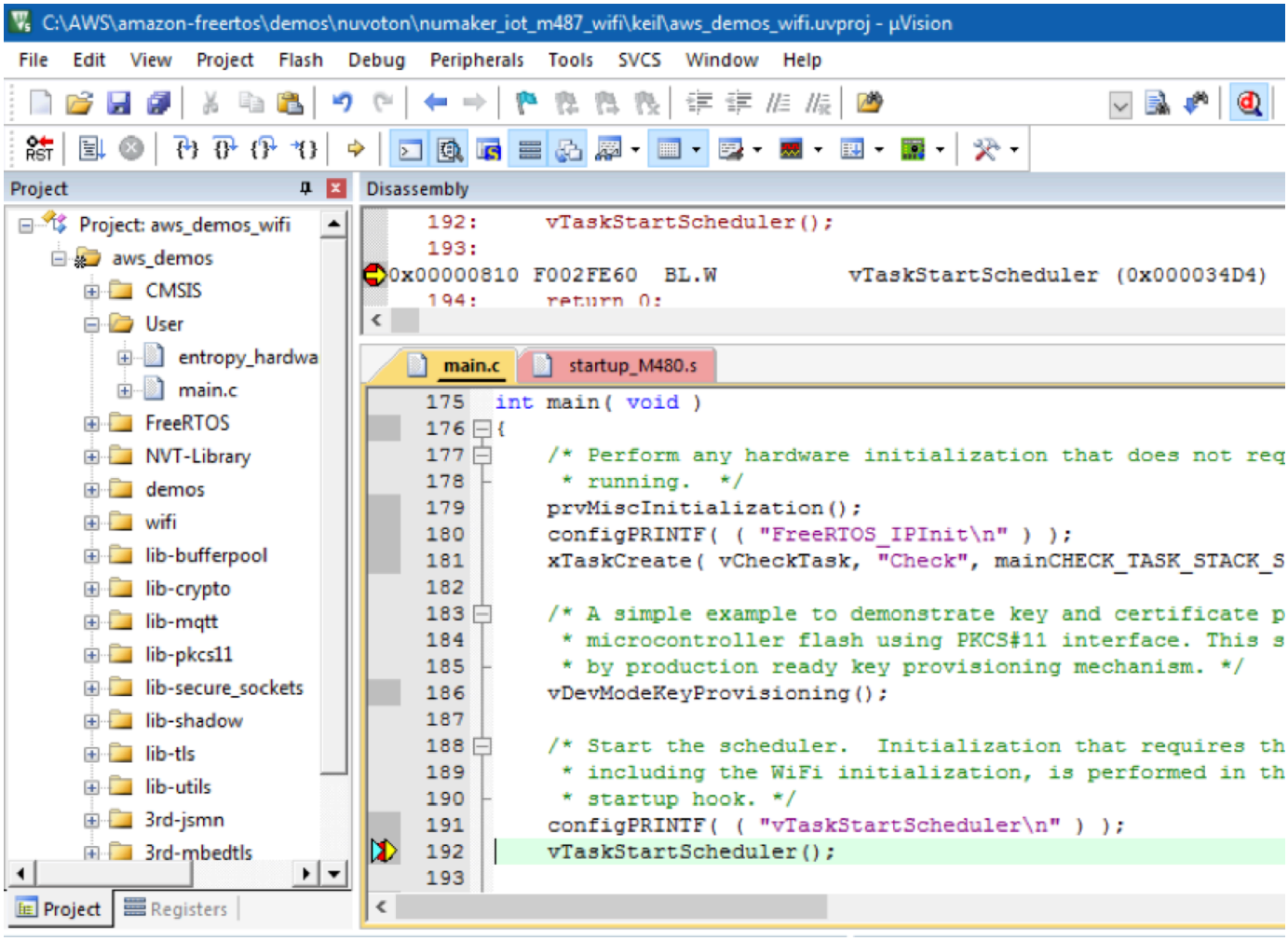
개발 환경을 설정하거나 보드에 연결하는 동안 문제가 발생하는 경우 Nuvoton에 문의하십시오.

Keil µVision에서 FreeRTOS 프로젝트 디버깅

Keil µVision에서 디버깅 세션을 시작하려면

1. Keil µVision을 엽니다.
2. FreeRTOS 데모 프로젝트 빌드 및 실행의 단계에 따라 FreeRTOS 데모 프로젝트를 빌드합니다.
3. Debug(디버깅) 메뉴에서 Start/Stop Debugging(디버깅 시작/중지)을 선택합니다.

   디버깅 세션을 시작하면 Call Stack + Locals(스택 + 로컬 호출) 창이 나타납니다. µVision이 데모를 보드로 플래시하고, 데모를 실행한 다음, `main()` 함수가 시작될 때 중지합니다.
4. 프로젝트의 소스 코드에 중단점을 설정한 다음 코드를 실행합니다. 프로젝트는 다음과 비슷해야 합니다.

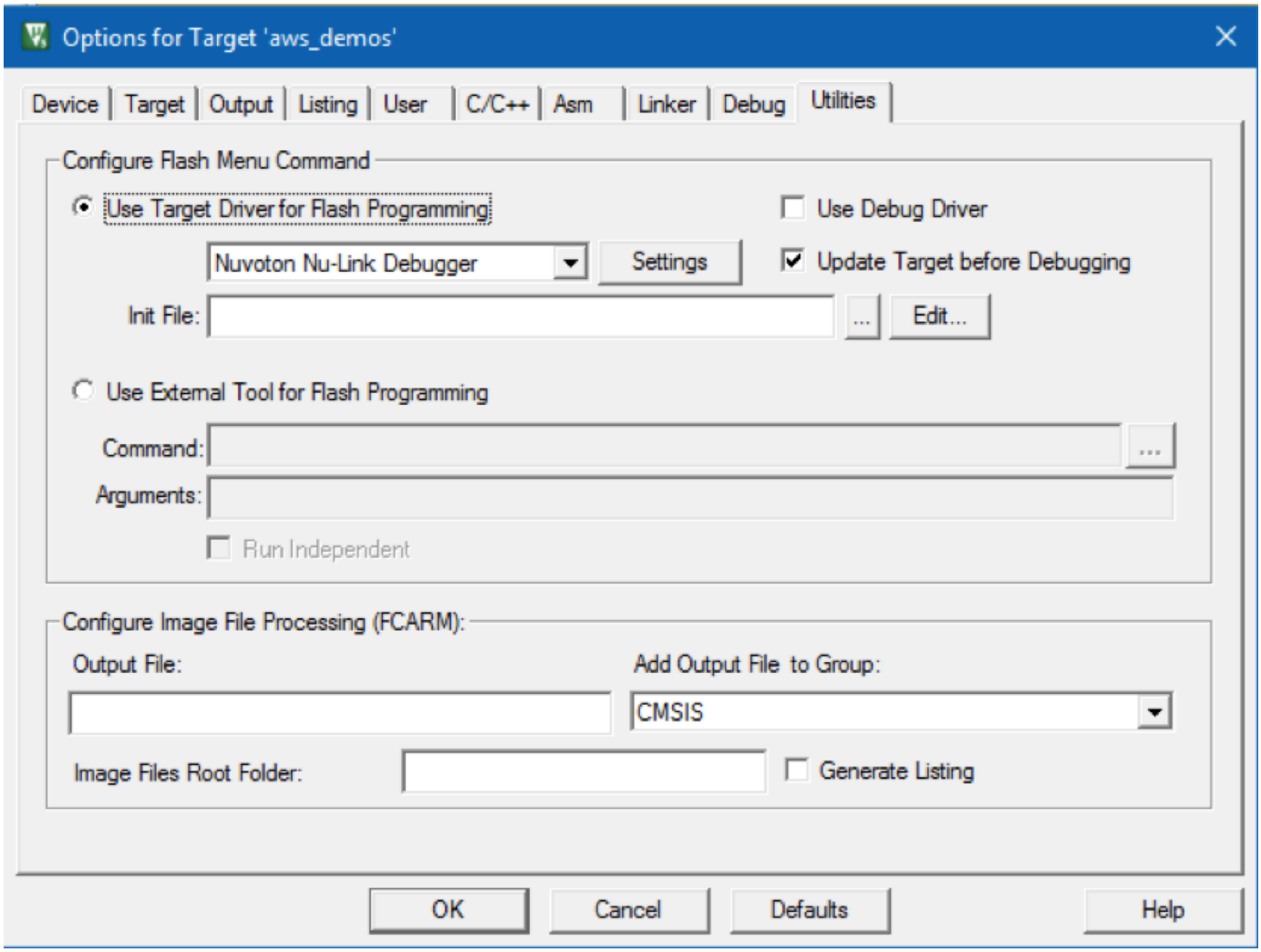


µVision 디버그 설정 문제 해결

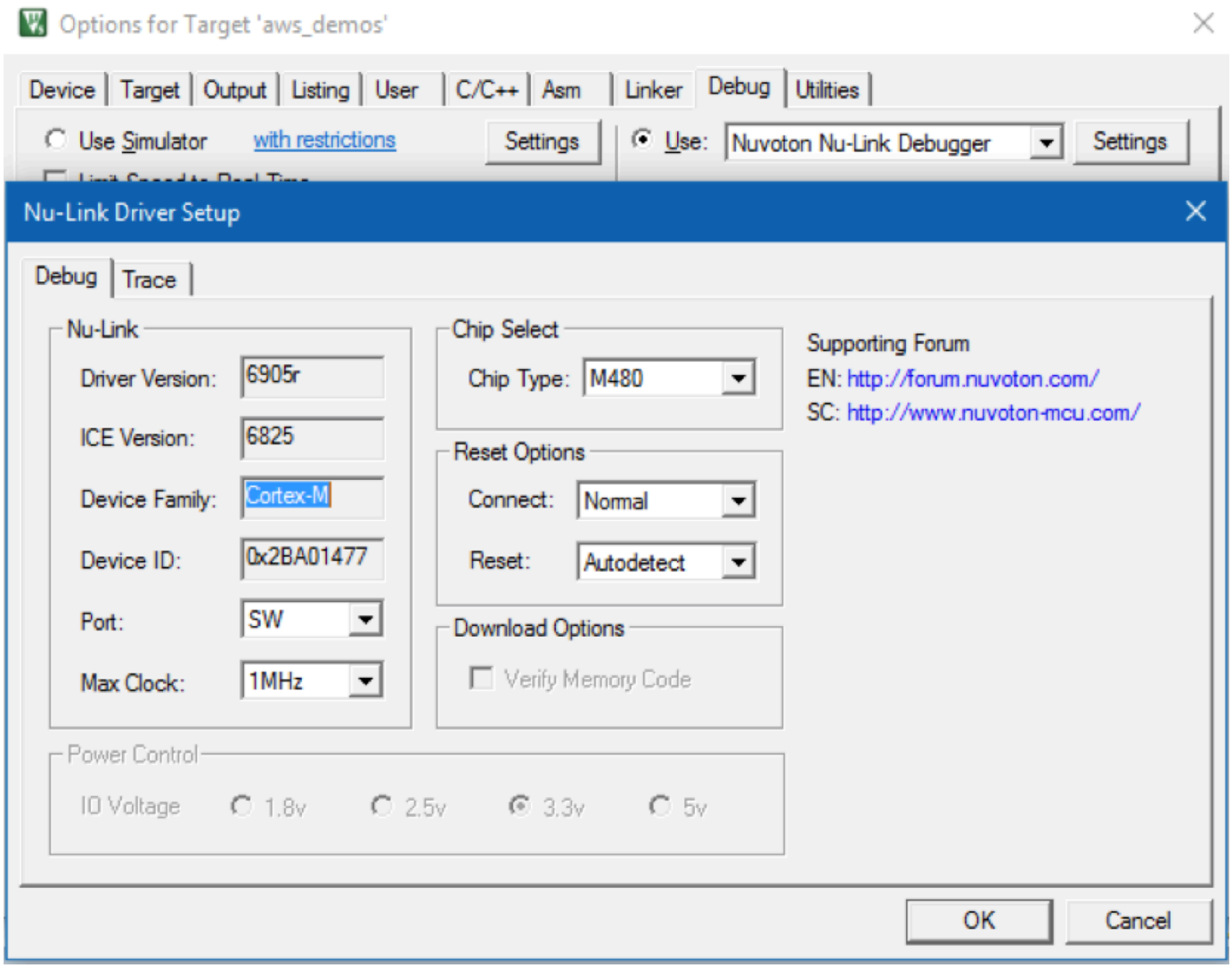
애플리케이션을 디버깅하는 동안 문제가 발생하는 경우 Keil µVision에서 디버그 설정이 올바르게 설정되었는지 확인합니다.

µVision 디버그 설정이 올바른지 확인하려면

1. Keil µVision을 엽니다.
2. IDE에서 aws_demo 프로젝트를 마우스 오른쪽 버튼으로 클릭한 다음 Options(옵션)를 선택합니다.
3. Utilities(유틸리티) 탭에서 Use Target Driver for Flash Programming(플래시 프로그래밍에 대상 드라이버 사용)이 선택되어 있는지 확인하고 Nuvoton Nu-Link Debugger(Nuvoton Nu-Link 디버거)가 대상 드라이버로 설정되어 있는지 확인합니다.



4. Debug(디버깅) 탭에서 Nuvoton Nu-Link Debugger(Nuvoton Nu-Link 디버거) 옆의 Settings(설정)를 선택합니다.



5. Chip Type(칩 유형)이 M480으로 설정되어 있는지 확인합니다.

NXP LPC54018 IoT 모듈 시작하기

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

이 자습서에서는 NXP LPC54018 IoT 모듈을 시작하기 위한 지침을 제공합니다. NXP LPC54018 IoT 모듈이 없는 경우 AWS 파트너 디바이스 카탈로그를 방문하여 파트너로부터 구매하세요. USB 케이블을 사용하여 NXP LPC54018 IoT Module을 컴퓨터에 연결합니다.

시작하기 전에 AWS IoT 및 FreeRTOS 다운로드를 구성하여 디바이스를 클라우드에 AWS 연결해야 합니다. 자세한 내용은 첫 번째 단계 섹션을 참조하세요. 이 자습서에서는 FreeRTOS 다운로드 디렉터리의 경로를 *freertos*라고 합니다.

개요

이 자습서에는 다음의 시작하기 단계에 대한 지침이 포함되어 있습니다.

1. 보드를 호스트 시스템에 연결합니다.
2. 마이크로 컨트롤러 보드용 내장형 애플리케이션을 개발 및 디버깅하기 위한 소프트웨어를 호스트 시스템에 설치합니다.
3. FreeRTOS 데모 애플리케이션을 바이너리 이미지로 크로스 컴파일합니다.
4. 애플리케이션 바이너리 이미지를 보드에 로드한 후 애플리케이션을 실행합니다.

NXP 하드웨어 설정

NXP LPC54018을 설정하려면

- 컴퓨터를 NXP LPC54018의 USB 포트에 연결합니다.

JTAG 디버거를 설정하려면

NXP LPC54018 보드에서 실행되는 코드를 시작하고 디버깅하려면 JTAG 디버거가 필요합니다. FreeRTOS는 OM40006 IoT 모듈을 사용하여 테스트되었습니다. 지원되는 디버거에 대한 자세한 내용은 OM40007 LPC54018 IoT Module 제품 페이지에서 제공되는 User Manual for NXP LPC54018 IoT Module을 참조하세요.

1. OM40006 IoT 모듈 디버거를 사용하는 경우 디버거의 20핀 커넥터를 NXP IoT 모듈의 10핀 커넥터에 연결하려면 변환기 케이블을 사용합니다.
2. 미니 USB-USB 케이블을 사용하여 NXP LPC54018과 OM40006 IoT 모듈 디버거를 컴퓨터의 USB 포트에 연결합니다.

개발 환경 설정

FreeRTOS는 NXP LPC54018 IoT 모듈에 대해 IAR Embedded Workbench 및 MCUXpresso라는 두 개의 IDE를 지원합니다.

시작하기 전에 이러한 IDE 중 하나를 설치합니다.

IAR Embedded Workbench for ARM을 설치하려면

1. IAR Embedded Workbench for ARM로 이동하여 소프트웨어를 다운로드합니다.

   > **Note**
   >
   > IAR Embedded Workbench for ARM에는 Microsoft Windows가 필요합니다.

2. 설치 관리자를 실행하고 메시지에 따라 완료합니다.
3. License Wizard(라이선스 마법사)에서 Register with IAR Systems to get an evaluation license(IAR Systems에 등록하여 평가 라이선스 받기)를 선택합니다.
4. 데모를 실행하기 전에 디바이스에 부트로더를 배치합니다.

NXP의 MCUXpresso를 설치하려면

1. NXP에서 MCUXpresso 설치 관리자를 다운로드하여 설치합니다.

   > **Note**
   >
   > 버전 10.3.x 이상이 지원됩니다.

2. MCUXpresso SDK로 이동하여 Build your SDK(SDK 빌드)를 선택합니다.

   > **Note**
   >
   > 버전 2.5 이상이 지원됩니다.

3. Select Development Board(개발 보드 선택)를 선택합니다.
4. Select Development Board(개발 보드 선택)의 Search by Name(이름으로 검색)에 **`LPC54018-IoT-Module`**를 입력합니다.
5. Boards(보드)에서 LPC54018-IoT-Module을 선택합니다.
6. 하드웨어 세부 정보를 확인한 다음 Build MCUXepresso SDK(MCUXepresso SDK 빌드)를 선택합니다.
7. MCUXpresso IDE를 사용하는 Windows용 SDK가 이미 빌드되어 있습니다. Download SDK를 선택합니다. 다른 운영 체제를 사용하는 경우 Host OS(호스트 OS)에서 해당 운영 체제를 선택한 다음 Download SDK(SDK 다운로드)를 선택합니다.
8. MCUXpresso IDE를 시작하고 Installed SDKs(설치된 SDK) 탭을 선택합니다.

9. 다운로드한 SDK 아카이브 파일을 Installed SDKs(설치한 SDK) 창으로 끌어서 놓습니다.

설치 중에 문제가 발생할 경우 NXP Support(NXP 지원) 또는 NXP Developer Resources(NXP 개발자 리소스)를 참조하십시오.

FreeRTOS 데모 프로젝트 빌드 및 실행

IDE로 FreeRTOS 데모 가져오기

FreeRTOS 샘플 코드를 IAR Embedded Workbench IDE로 가져오려면

1. IAR Embedded Workbench를 열고 File(파일) 메뉴에서 Open Workspace(작업 공간 열기)를 선택합니다.
2. search-directory(검색 디렉터리) 텍스트 상자에 `projects/nxp/lpc54018iotmodule/iar/aws_demos`를 입력하고 aws_demos.eww를 선택합니다.
3. Project(프로젝트) 메뉴에서 Rebuild All(모두 다시 빌드)을 선택합니다.

FreeRTOS 샘플 코드를 MCUXpresso IDE로 가져오려면

1. MCUXpresso를 열고 File(파일) 메뉴에서 Open Projects From File System(파일 시스템에서 프로젝트 열기)을 선택합니다.
2. 디렉터리 텍스트 상자에 `projects/nxp/lpc54018iotmodule/mcuxpresso/aws_demos`를 입력하고 마침을 선택합니다.
3. Project(프로젝트) 메뉴에서 Build All(모두 빌드)을 선택합니다.

FreeRTOS 데모 프로젝트 실행

IAR Embedded Workbench IDE로 FreeRTOS 데모 프로젝트를 실행하려면

1. IDE의 프로젝트 메뉴에서 Make를 선택합니다.
2. Project(프로젝트) 메뉴에서 Download and Debug(다운로드 및 디버그)를 선택합니다.
3. Debug(디버그) 메뉴에서 Start Debugging(디버깅 시작)을 선택합니다.
4. 디버거가 `main`의 중단점에서 중지되면 Debug(디버그) 메뉴에서 Go(이동)를 선택합니다.

> **Note**
>
> J-Link Device Selection(J-Link 디바이스 선택) 대화 상자가 열리면 OK(확인)를 선택하여 계속합니다. Target Device Settings(대상 디바이스 설정) 대화 상자에서 Unspecified(지정 안 함)를 선택하고 Cortex-M4를 선택한 다음 OK(확인)를 선택합니다. 이 단계는 한 번만 수행하면 됩니다.

MCUexpresso IDE로 FreeRTOS 데모 프로젝트를 실행하려면

1. IDE의 프로젝트 메뉴에서 빌드를 선택합니다.
2. 처음 디버깅하는 경우 `aws_demos` 프로젝트를 선택하고 디버깅 도구 모음에서 파란색 디버그 버튼을 선택합니다.
3. 발견된 디버그 프로브가 표시됩니다. 사용할 프로브를 선택한 다음 OK(확인)를 선택하여 디버깅을 시작합니다.

> **Note**
>
> 디버거가 `main()`의 중단점에서 중지되면 디버그 다시 시작 버튼
>
> 
>
> 을 한 번 눌러 디버깅 세션을 재설정합니다. (이 단계는 NXP54018-IoT-Module용 MCUXpresso 디버거의 버그로 인해 필요합니다).

4. 디버거가 `main()`의 중단점에서 중지되면 Debug(디버그) 메뉴에서 Go(이동)를 선택합니다.

클라우드에서 MQTT 메시지 모니터링

AWS IoT 콘솔에서 MQTT 클라이언트를 사용하여 디바이스가 AWS 클라우드로 보내는 메시지를 모니터링할 수 있습니다.

MQTT 클라이언트를 사용하여 AWS IoT MQTT 주제를 구독하려면

1. AWS IoT 콘솔에 로그인합니다.
2. 탐색 창에서 Test(테스트)를 선택하여 MQTT 클라이언트를 엽니다.
3. 구독 주제에 ***your-thing-name*/example/topic**을 입력한 다음 주제 구독을 선택합니다.

문제 해결

FreeRTOS 시작하기에 대한 일반 문제 해결 정보는 시작하기 문제 해결 섹션을 참조하세요.

Renesas Starter Kit+ for RX65N-2MB 시작하기

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

이 자습서에서는 Renesas Starter Kit+ for RX65N-2MB를 시작하기 위한 지침을 제공합니다. RX65N-2MB용 Renesas RSK+가 없는 경우 AWS 파트너 디바이스 카탈로그를 방문하여 파트너로부터 구매하세요.

시작하기 전에 AWS IoT 및 FreeRTOS 다운로드를 구성하여 디바이스를 클라우드에 AWS 연결해야 합니다. 자세한 내용은 첫 번째 단계 섹션을 참조하세요. 이 자습서에서는 FreeRTOS 다운로드 디렉터리의 경로를 *freertos*라고 합니다.

개요

이 자습서에는 다음의 시작하기 단계에 대한 지침이 포함되어 있습니다.

1. 보드를 호스트 시스템에 연결합니다.
2. 마이크로 컨트롤러 보드용 내장형 애플리케이션을 개발 및 디버깅하기 위한 소프트웨어를 호스트 시스템에 설치합니다.
3. FreeRTOS 데모 애플리케이션을 바이너리 이미지로 크로스 컴파일합니다.
4. 애플리케이션 바이너리 이미지를 보드에 로드한 후 애플리케이션을 실행합니다.

Renesas 하드웨어 설정

RSK+ for RX65N-2MB를 설정하려면

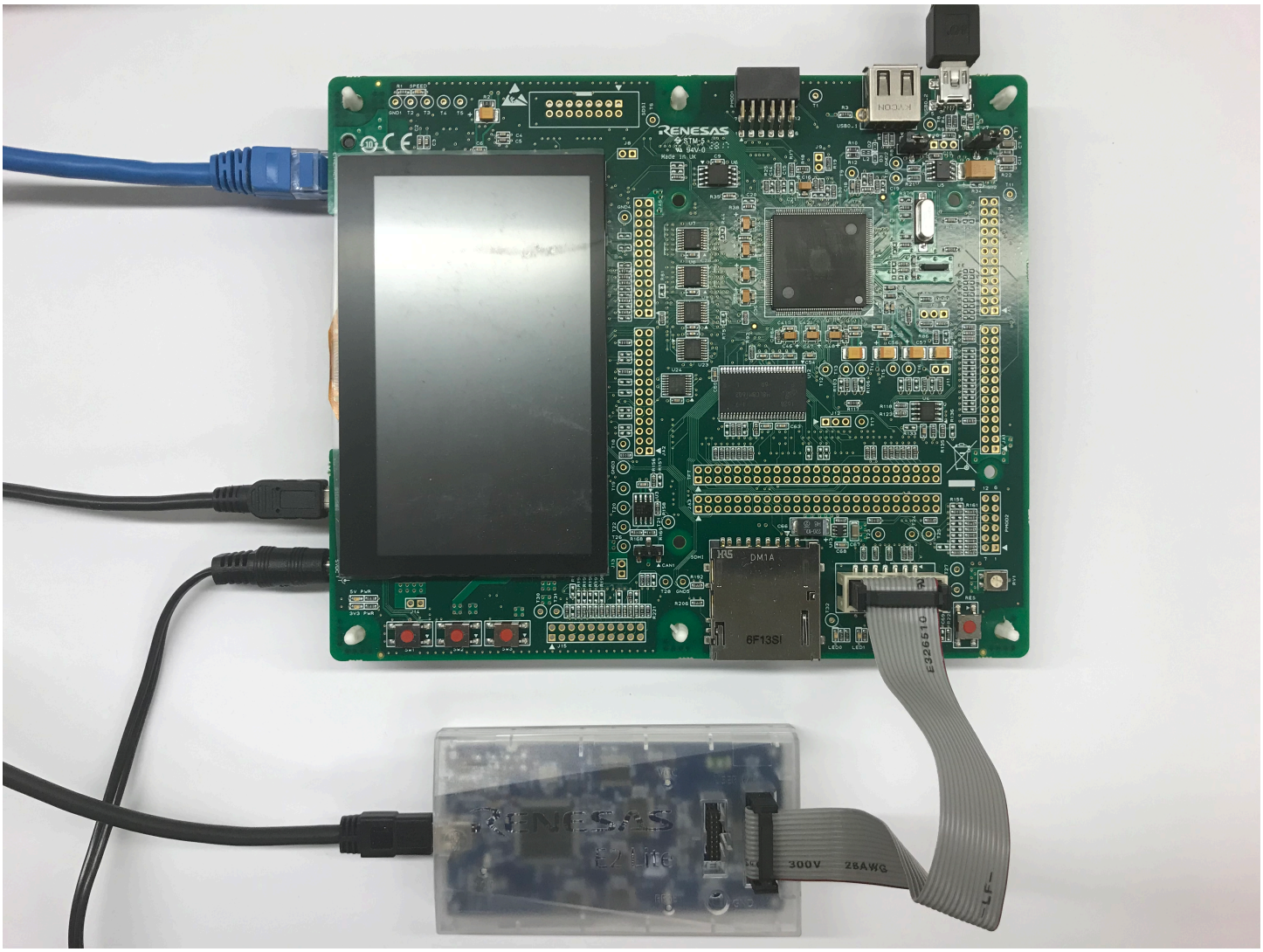
1. 양의 +5V 전원 어댑터를 RSK+ for RX65N-2MB의 PWR 커넥터에 연결합니다.
2. 컴퓨터를 RSK+ for RX65N-2MB의 USB2.0 FS 포트에 연결합니다.
3. 컴퓨터를 RSK+ for RX65N-2MB의 USB-to-serial 포트에 연결합니다.

4. 라우터 또는 인터넷에 연결된 이더넷 포트를 RSK+ for RX65N-2MB의 이더넷 포트에 연결합니다.

E2 Lite 디버거 모듈을 설정하려면

1. 14핀 리본 케이블을 사용하여 E2 Lite 디버거 모듈을 RSK+ for RX65N-2MB의 ‘E1/E2 Lite’ 포트에 연결합니다.
2. USB 케이블을 사용하여 E2 Lite 디버거 모듈을 호스트 머신에 연결합니다. E2 Lite 디버거가 보드와 컴퓨터 둘 다에 연결된 경우 디버거에서 녹색 ‘ACT’ LED가 깜박입니다.
3. 디버거가 호스트 컴퓨터와 RSK+ for RX65N-2MB에 연결되고 나면 E2 Lite 디버거 드라이버가 설치되기 시작합니다.

   드라이버를 설치하려면 관리자 권한이 필요합니다.



개발 환경 설정

RSK+ for RX65N-2MB에 대한 FreeRTOS 구성을 설정하려면 Renesas e^{2}studio IDE 및 CC-RX 컴파일러를 사용합니다.

> **Note**
>
> Renesas e^{2}studio IDE 및 CC-RX 컴파일러는 Windows 7, 8, 10 운영 체제에서만 지원됩니다.

e^{2}studio 다운로드 및 설치

1. Renesas e^{2}studio installer 다운로드 페이지로 이동하여 오프라인 설치 관리자를 다운로드합니다.
2. 그러면 Renesas 로그인 페이지로 이동합니다.

   Renesas 계정이 있는 경우 로그인 보안 인증 정보를 입력한 다음 로그인을 선택합니다.

   계정이 없는 경우 Register now(지금 등록)을 선택하고 최초의 등록 단계를 따릅니다. 그러면 Renesas 계정을 활성화하기 위한 링크가 포함된 이메일을 받게 됩니다. 이 링크를 따라서 Renesas 등록을 완료한 다음 Renesas에 로그인합니다.
3. 로그인한 후 e^{2}studio 설치 관리자를 컴퓨터에 다운로드합니다.
4. 설치 관리자를 열고 단계에 따라 완료합니다.

자세한 내용은 Renesas 웹사이트에서 e^{2}studio를 참조하세요.

RX Family C/C++ 컴파일러 패키지 다운로드 및 설치

1. RX Family C/C++ Compiler Package 다운로드 페이지로 이동하여 V3.00.00 패키지를 다운로드합니다.
2. 실행 파일을 열고 컴파일러를 설치합니다.

자세한 내용은 Renesas 웹사이트에서 RX Family용 C/C++ 컴파일러 패키지를 참조하십시오.

> **Note**
>
> 컴파일러는 평가판에 대해서만 무료로 제공되며 60일 동안 유효합니다. 61일째는 라이선스 키를 받아야 합니다. 자세한 내용은 평가 소프트웨어 도구를 참조하십시오.

FreeRTOS 샘플 빌드 및 실행

이제 데모를 구성했으므로 보드에서 데모 프로젝트를 빌드 및 실행할 준비가 된 것입니다.

e²studio에서 FreeRTOS 데모 빌드

e²studio에서 데모를 가져와서 빌드하기

1. 시작 메뉴에서 e²studio를 시작합니다.
2. Select a directory as a workspace(작업 영역으로 디렉터리 선택) 창에서 작업하려는 폴더를 찾아 시작을 선택합니다.
3. 처음 e²studio를 열면 Toolchain Registry(도구 체인 등록) 창이 열립니다. Renesas Toolchains(Renesas 도구 체인)을 선택하고 **`CC-RX v3.00.00`**이 선택되어 있는지 확인합니다. 등록을 선택한 다음 확인을 선택합니다.
4. 처음으로 e²studio를 열 경우 Code Generator Registration(코드 생성기 등록) 창이 나타납니다. 확인을 선택합니다.
5. Code Generator COM component register(코드 생성기 COM 구성요소 등록)창이 나타납니다. Please restart e²studio to use Code Generator에서 확인을 선택합니다.
6. e²studio 다시 시작 창이 나타납니다. 확인을 선택합니다.
7. e²studio가 다시 시작됩니다. Select a directory as a workspace(작업 영역으로 디렉터리 선택) 창에서 시작을 선택합니다.
8. e²studio 시작 화면에서 e²studio 워크벤치로 이동 화살표 아이콘을 선택합니다.
9. Project Explorer 창을 마우스 오른쪽 버튼으로 클릭하고 가져오기를 선택합니다.
10. 가져오기 마법사에서 General(일반), Existing Projects into Workspace(기존 프로젝트를 작업 공간으로)를 선택한 후 다음을 선택합니다.
11. 찾아보기를 선택하고 `projects/renesas/rx65n-rsk/e2studio/aws_demos` 디렉터리를 찾은 후 마침을 선택합니다.
12. 프로젝트 메뉴에서 프로젝트, Build All(모두 빌드)를 선택합니다.

    빌드 콘솔이 라이선스 관리자가 설치되어 있지 않다는 경고 메시지를 표시합니다. CC-RX 컴파일러용 라이선스 키가 없다면 이 메시지를 무시할 수 있습니다. 라이선스 관리자를 설치하려면 License Manager(라이선스 관리자) 다운로드 페이지를 참조하십시오.

FreeRTOS 프로젝트 실행

e²studio에서 프로젝트 실행하기

1. E2 Lite 디버거 모듈을 RSK+ for RX65N-2MB에 연결했는지 확인합니다.
2. 상단 메뉴에서 Run(실행), Debug Configuration(디버그 구성)을 선택합니다.
3. Renesas GDB Hardware Debugging(Renesas GDB 하드웨어 디버깅)을 확장하고 aws_demos HardwareDebug를 선택합니다.
4. Debugger(디버거) 탭을 선택한 다음 Connection Settings(연결 설정) 탭을 선택합니다. 연결 설정이 올바른지 확인합니다.
5. Debug(디버그)를 선택하여 코드를 보드에 다운로드하고 디버깅을 시작합니다.

   `e2-server-gdb.exe`에 대한 방화벽 경고 메시지가 표시될 수 있습니다. Private networks, such as my home or work network(프라이빗 네트워크(예: 홈 또는 직장 네트워크))를 선택한 다음 Allow access(액세스 허용)를 선택합니다.
6. e²studio가 Renesas Debug Perspective(Renesas 디버그 관점)로 변경하도록 요청할 수 있습니다. 예(Yes)를 선택합니다.

   E2 Lite 디버거에서 녹색 'ACT' LED가 켜집니다.
7. 코드가 보드에 다운로드되면 다시 시작을 선택하여 첫 번째 줄의 main 함수에 대해 코드를 실행합니다. 다시 시작을 다시 선택하여 나머지 코드를 실행합니다.

클라우드에서 MQTT 메시지 모니터링

AWS IoT 콘솔에서 MQTT 클라이언트를 사용하여 디바이스가 AWS 클라우드로 보내는 메시지를 모니터링할 수 있습니다.

MQTT 클라이언트를 사용하여 AWS IoT MQTT 주제를 구독하려면

1. AWS IoT 콘솔에 로그인합니다.
2. 탐색 창에서 Test(테스트)를 선택하여 MQTT 클라이언트를 엽니다.
3. 구독 주제에 ***your-thing-name*/example/topic**을 입력한 다음 주제 구독을 선택합니다.

Renesas에서 릴리스한 최신 프로젝트는 GitHub에서 `amazon-freertos` 리포지토리의 `renesas-rx` 포크를 참조하세요.

문제 해결

FreeRTOS 시작하기에 대한 일반 문제 해결 정보는 시작하기 문제 해결 섹션을 참조하세요.

STMicroelectronics STM32L4 Discovery Kit IoT Node 시작하기

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

이 자습서에서는 STMicroelectronics STM32L4 Discovery Kit IoT 노드를 시작하기 위한 지침을 제공합니다. STMicroelectronics STM32L4 Discovery Kit IoT 노드가 아직 없는 경우 AWS 파트너 디바이스 카탈로그를 방문하여 파트너로부터 구매하세요.

최신 Wi-Fi 펌웨어를 설치했는지 확인합니다. 최신 Wi-Fi 펌웨어를 다운로드하려면 STM32L4 Discovery 키트 IoT 노드, 저전력 무선, Bluetooth Low Energy, NFC, SubGHz, Wi-Fi를 참조하십시오. Binary Resources(바이너리 리소스)에서 Inventek ISM 43362 Wi-Fi module firmware update (read the readme file for instructions)(Inventek ISM 43362 Wi-Fi 모듈 펌웨어 업데이트(지침은 readme 파일 참조))를 선택합니다.

시작하기 전에 AWS IoT 및 FreeRTOS 다운로드를 구성하여 디바이스를 AWS 클라우드에 연결해야 합니다. 자세한 내용은 첫 번째 단계 섹션을 참조하세요. 이 자습서에서는 FreeRTOS 다운로드 디렉터리의 경로를 *freertos*라고 합니다.

개요

이 자습서에는 다음의 시작하기 단계에 대한 지침이 포함되어 있습니다.

1. 마이크로 컨트롤러 보드용 내장형 애플리케이션을 개발 및 디버깅하기 위한 소프트웨어를 호스트 시스템에 설치합니다.
2. FreeRTOS 데모 애플리케이션을 바이너리 이미지로 크로스 컴파일합니다.
3. 애플리케이션 바이너리 이미지를 보드에 로드한 후 애플리케이션을 실행합니다.

개발 환경 설정

System Workbench for STM32 설치

1. OpenSTM32.org로 이동합니다.
2. OpenSTM32 웹 페이지에서 등록합니다. System Workbench를 다운로드하려면 로그인해야 합니다.
3. System Workbench for STM32 설치 관리자로 이동하여 System Workbench를 다운로드하고 설치합니다.

설치 중에 문제가 발생하면 System Workbench 웹 사이트의 FAQ를 참조하십시오.

FreeRTOS 데모 프로젝트 빌드 및 실행

FreeRTOS 데모를 STM32 System Workbench로 가져오기

1. STM32 System Workbench를 열고 새 작업 공간의 이름을 입력합니다.
2. 파일 메뉴에서 가져오기를 선택합니다. General(일반)을 확장하고 Existing Projects into Workspace(기존 프로젝트를 작업 공간으로)를 선택한 다음 Next(다음)를 선택합니다.
3. Select Root Directory(루트 디렉터리 선택)에 `projects/st/stm32l475_discovery/ac6/aws_demos`를 입력합니다.
4. `aws_demos` 프로젝트가 기본적으로 선택되어야 합니다.
5. Finish(완료)를 선택하여 프로젝트를 STM32 System Workbench로 가져옵니다.
6. Project(프로젝트) 메뉴에서 Build All(모두 빌드)을 선택합니다. 오류 없이 프로젝트가 컴파일되는지 확인합니다.

FreeRTOS 데모 프로젝트 실행

1. USB 케이블을 사용하여 STMicroelectronics STM32L4 Discovery Kit IoT Node를 컴퓨터에 연결합니다.
2. Project Explorer에서 `aws_demos`를 마우스 오른쪽 버튼으로 클릭하고 Debug As(다른 형식으로 디버그)를 선택한 다음 Ac6 STM32 C/C++ Application(Ac6 STM32 C/C++ 애플리케이션)을 선택합니다.

   디버그 세션을 처음 시작할 때 디버그 오류가 발생하는 경우 다음 단계를 따릅니다.

1. STM32 System Workbench의 Run(실행) 메뉴에서 Debug Configurations(디버그 구성)를 선택합니다.
2. aws_demos Debug(aws_demos 디버그)를 선택합니다. (Ac6 STM32 Debugging(Ac6 STM32 디버깅)을 확장해야 할 수 있습니다.)
3. Debugger(디버거) 탭을 선택합니다.
4. Configuration Script(구성 스크립트)에서 Show Generator Options(생성기 옵션 표시)를 선택합니다.
5. Mode Setup(모드 설정)에서 Reset Mode(모드 재설정)를 Software System Reset(소프트웨어 시스템 재설정)으로 설정합니다. [Apply]를 선택한 다음 [Debug]를 선택합니다.

3. 디버거가 `main()`의 중단점에서 중지되면 실행 메뉴에서 다시 시작을 선택합니다.

FreeRTOS에서 CMake 사용

FreeRTOS 개발용 IDE를 사용하지 않으려는 경우 CMake를 사용하여 데모 애플리케이션이나 타사 코드 편집기 및 디버깅 도구를 사용하여 개발한 애플리케이션을 빌드하고 실행할 수 있습니다.

먼저 생성된 빌드 파일을 포함할 폴더(*`build-folder`*)를 생성합니다.

빌드 파일을 생성할 때는 다음 명령을 사용합니다.

```
cmake -DVENDOR=st -DBOARD=stm32l475_discovery -DCOMPILER=arm-gcc -S freertos -B build-folder
```

`arm-none-eabi-gcc`가 셸 경로에 없으면 `AFR_TOOLCHAIN_PATH` CMake 변수도 설정해야 합니다. 예시:

```
-D AFR_TOOLCHAIN_PATH=/home/user/opt/gcc-arm-none-eabi/bin
```

FreeRTOS에서 CMake를 사용하는 방법에 대한 자세한 내용은 FreeRTOS에서 CMake 사용 섹션을 참조하세요.

클라우드에서 MQTT 메시지 모니터링

AWS IoT 콘솔에서 MQTT 클라이언트를 사용하여 디바이스가 AWS 클라우드로 보내는 메시지를 모니터링할 수 있습니다.

MQTT 클라이언트를 사용하여 AWS IoT MQTT 주제를 구독하려면

1. AWS IoT 콘솔에 로그인합니다.
2. 탐색 창에서 Test(테스트)를 선택하여 MQTT 클라이언트를 엽니다.
3. 구독 주제에 ***your-thing-name*/example/topic**을 입력한 다음 주제 구독을 선택합니다.

문제 해결

데모 애플리케이션의 UART 출력에 다음이 나타나면 Wi-Fi 모듈의 펌웨어를 업데이트해야 할 수 있습니다.

```
[Tmr Svc] WiFi firmware version is: xxxxxxxxxxxx
[Tmr Svc] [WARN] WiFi firmware needs to be updated.
```

최신 Wi-Fi 펌웨어를 다운로드하려면 STM32L4 Discovery 키트 IoT 노드, 저전력 무선, Bluetooth Low Energy, NFC, SubGHz, Wi-Fi를 참조하십시오. Binary Resources(바이너리 리소스)에서 Inventek ISM 43362 Wi-Fi module firmware update(Inventek ISM 43362 Wi-Fi 모듈 펌웨어 업데이트)의 다운로드 링크를 선택합니다.

FreeRTOS 시작하기에 대한 일반 문제 해결 정보는 시작하기 문제 해결 섹션을 참조하세요.

Texas Instruments CC3220SF-LAUNCHXL 시작하기

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

이 자습서에서는 Texas Instruments CC3220SF-LAUNCHXL을 시작하기 위한 지침을 제공합니다. Texas Instruments(TI) CC3220SF-LAUNCHXL 개발 키트가 없는 경우 AWS 파트너 디바이스 카탈로그를 방문하여 파트너로부터 구매하세요.

시작하기 전에 AWS IoT 및 FreeRTOS 다운로드를 구성하여 디바이스를 클라우드에 AWS 연결해야 합니다. 자세한 내용은 첫 번째 단계 섹션을 참조하세요. 이 자습서에서는 FreeRTOS 다운로드 디렉터리의 경로를 *freertos*라고 합니다.

개요

이 자습서에는 다음의 시작하기 단계에 대한 지침이 포함되어 있습니다.

1. 마이크로 컨트롤러 보드용 내장형 애플리케이션을 개발 및 디버깅하기 위한 소프트웨어를 호스트 시스템에 설치합니다.
2. FreeRTOS 데모 애플리케이션을 바이너리 이미지로 크로스 컴파일합니다.
3. 애플리케이션 바이너리 이미지를 보드에 로드한 후 애플리케이션을 실행합니다.

개발 환경 설정

아래 단계에 따라 개발 환경을 설정하고 FreeRTOS를 시작합니다.

FreeRTOS는 TI CC3220SF-LAUNCHXL 개발 키트 Code Composer Studio와 IAR Embedded Workbench 버전 8.32라는 두 개의 IDE를 지원합니다. 이 중 하나의 IDE를 사용하여 시작할 수 있습니다.

Install Code Composer 설치

1. TI Code Composer Studio로 이동합니다.
2. 각 호스트 시스템(Windows, macOS 또는 Linux 64비트)의 플랫폼에 맞는 오프라인 설치 관리자를 다운로드합니다.
3. 오프라인 설치 관리자의 압축을 풀고 실행합니다. 다음에 나타나는 메시지를 따릅니다.
4. Product Families to Install(설치할 제품군)에서 SimpleLink Wi-Fi CC32xx Wireless MCUs(SimpleLink Wi-Fi CC32xx 무선 MCU)를 선택합니다.
5. 다음 페이지에서 프로브 디버깅에 대한 기본 설정을 수락하고 Finish(완료)를 선택합니다.

Code Composer Studio를 설치하는 동안 문제가 발생할 경우 TI 개발 도구 지원, Code Composer Studio FAQ 및 CCS 문제 해결을 참조하십시오.

IAR Embedded Workbench 설치

1. ARM용 IAR Embedded Workbench의 버전 8.32용 Windows 설치 관리자를 다운로드하여 실행합니다. Debug probe drivers(프로브 드라이버 디버깅)에서 TI XDS가 선택되어 있는지 확인합니다.
2. 설치를 완료하고 프로그램을 시작합니다. License Wizard(라이선스 마법사) 페이지에서 Register with IAR Systems to get an evaluation license(IAR 시스템에 등록하여 평가 라이선스 받기)를 선택하거나 고유의 IAR 라이선스를 사용합니다.

SimpleLink CC3220 SDK 설치

SimpleLink CC3220 SDK를 설치합니다. SimpleLink Wi-Fi CC3220 SDK에는 CC3220SF 프로그래밍 가능한 MCU용 드라이버, 40개 이상의 샘플 애플리케이션 및 샘플을 사용하는 데 필요한 설명서가 포함되어 있습니다.

Uniflash 설치

Uniflash를 설치합니다. CCS Uniflash는 TI MCU에서 온칩 플래시 메모리를 프로그래밍하는 데 사용되는 독립형 도구입니다. Uniflash에는 GUI, 명령줄 및 스크립팅 인터페이스가 있습니다.

최신 서비스 팩 설치

1. TI CC3220SF-LAUNCHXL에서 SOP 점퍼를 중간 핀 집합(위치 = 1)에 놓고 보드를 재설정합니다.
2. Uniflash를 시작합니다. CC3220SF LaunchPad 보드가 Detected Devices(검색된 디바이스)에 나타나면 Start(시작)를 선택합니다. 보드가 검색되지 않는 경우 새 구성 아래의 보드 목록에서 CC3220SF-LAUNCHXL을 선택한 다음 이미지 생성자 시작을 선택합니다.
3. 새 프로젝트를 선택합니다.
4. Start new project(새 프로젝트 시작) 페이지에서 프로젝트의 이름을 입력합니다. Device Type(디바이스 유형)에서 CC3220SF를 선택합니다. Device Mode(디바이스 모드)에서 Develop(개발)를 선택한 다음 Create Project(프로젝트 생성)를 선택합니다.
5. Uniflash 애플리케이션 창의 오른쪽에서 연결을 선택합니다.
6. 왼쪽 열에서 고급, 파일 및 서비스 팩을 차례로 선택합니다.
7. Browse(찾아보기)를 선택한 다음 CC3220SF SimpleLink SDK를 설치한 위치로 이동합니다. 서비스 팩은 `ti/simplelink_cc32xx_sdk_`*`VERSION`*`/tools/cc32xx_tools/servicepack-cc3x20/sp_`*`VERSION`*`.bin`에 있습니다.
8. Burn()
   버튼을 선택한 다음 프로그램 이미지(생성 및 프로그램)을 선택하여 서비스 팩을 설치합니다. SOP 점퍼를 다시 0 위치로 전환하고 보드를 재설정해야 합니다.

Wi-Fi 프로비저닝 구성

보드에 대한 Wi-Fi 설정을 구성하려면 다음 중 하나를 수행합니다.

- FreeRTOS 데모 구성에 설명된 FreeRTOS 데모 애플리케이션을 구성합니다.
- Texas Instruments의 SmartConfig를 사용합니다.

FreeRTOS 데모 프로젝트 빌드 및 실행

TI Code Composer에서 FreeRTOS 데모 프로젝트 빌드 및 실행

FreeRTOS 데모를 TI Code Composer로 가져오려면

1. TI Code Composer를 열고 OK(확인)를 선택하여 기본 작업 영역 이름을 수락합니다.
2. Getting Started(시작하기) 페이지에서 Import Project(프로젝트 가져오기)를 선택합니다.
3. Select search-directory(검색 디렉터리 선택)에 `projects/ti/cc3220_launchpad/ccs/aws_demos`를 입력합니다. `aws_demos` 프로젝트가 기본적으로 선택되어야 합니다. 프로젝트를 TI Code Composer로 가져오려면 Finish(완료)를 선택합니다.
4. Project Explorer(프로젝트 탐색기)에서 aws_demos를 두 번 클릭하여 프로젝트를 활성화합니다.
5. Project(프로젝트)에서 Build Project(프로젝트 빌드)를 선택하여 오류나 경고 없이 프로제트가 컴파일되는지 확인합니다.

TI Code Composer에서 FreeRTOS 데모를 실행하려면

1. Texas Instruments CC3220SF-LAUNCHXL의 Sense On Power(SOP) 점퍼가 0 위치에 있는지 확인합니다. 자세한 내용은 SimpleLink Wi-Fi CC3x20, CC3x3x Network Processor User's Guide를 참조하세요.
2. USB 케이블을 사용하여 Texas Instruments CC3220SF-LAUNCHXL을 컴퓨터에 연결합니다.
3. 프로젝트 탐색기에서 `CC3220SF.ccxml`을 활성 대상 구성으로 선택했는지 확인합니다. 활성화하려면 파일을 마우스 오른쪽 버튼으로 클릭하고 Set as active target configuration(활성 대상 구성으로 설정)을 선택합니다.
4. TI Code Composer의 Run(실행)에서 Debug(디버그)를 선택합니다.
5. 디버거가 `main()`의 중단점에서 중지되면 실행 메뉴로 이동하여 다시 시작을 선택합니다.

IAR Embedded Workbench에서 FreeRTOS 데모 프로젝트 빌드 및 실행

FreeRTOS 데모를 IAR Embedded Workbench로 가져오려면

1. IAR Embedded Workbench를 열고 File(파일)을 선택한 다음 Open Workspace(작업 공간 열기)를 선택합니다.
2. `projects/ti/cc3220_launchpad/iar/aws_demos`로 이동하고 aws_demos.eww를 선택한 다음 확인을 선택합니다.
3. 프로젝트 이름(`aws_demos`)을 마우스 오른쪽 버튼으로 클릭한 다음 Make(만들기)를 선택합니다.

IAR Embedded Workbench에서 FreeRTOS 데모를 실행하려면

1. Texas Instruments CC3220SF-LAUNCHXL의 Sense On Power(SOP) 점퍼가 0 위치에 있는지 확인합니다. 자세한 내용은 SimpleLink Wi-Fi CC3x20, CC3x3x Network Processor User's Guide를 참조하세요.
2. USB 케이블을 사용하여 Texas Instruments CC3220SF-LAUNCHXL을 컴퓨터에 연결합니다.
3. 프로젝트를 다시 빌드합니다.

   프로젝트를 다시 빌드하려면 Project(프로젝트) 메뉴에서 Make(만들기)를 선택합니다.
4. Project(프로젝트) 메뉴에서 Download and Debug(다운로드 및 디버그)를 선택합니다. "Warning: Failed to initialize EnergyTrace(경고: EnergyTrace를 초기화하지 못함)"가 표시되는 경우 이 메시지를 무시할 수 있습니다. EnergyTrace에 대한 자세한 내용은 MSP EnergyTrace Technology를 참조하십시오.
5. 디버거가 `main()`의 중단점에서 중지되면 디버깅 메뉴로 이동하여 이동을 선택합니다.

FreeRTOS에서 CMake 사용

FreeRTOS 개발용 IDE를 사용하지 않으려는 경우 CMake를 사용하여 데모 애플리케이션이나 타사 코드 편집기 및 디버깅 도구를 사용하여 개발한 애플리케이션을 빌드하고 실행할 수 있습니다.

CMake로 FreeRTOS 데모를 빌드하려면

1. 생성된 빌드 파일을 포함할 폴더(*`build-folder`*)를 생성합니다.
2. 검색 경로($PATH 환경 변수)에 TI CGT 컴파일러 바이너리가 있는 폴더(예: `C:\ti\ccs910\ccs\tools\compiler\ti-cgt-arm_18.12.2.LTS\bin`)가 포함되어 있는지 확인하십시오.

   TI 보드에서 TI ARM 컴파일러를 사용하는 경우 다음 명령을 사용하여 소스 코드에서 빌드 파일을 생성합니다.

   ```
   cmake -DVENDOR=ti -DBOARD=cc3220_launchpad -DCOMPILER=arm-ti -S freertos -B build-folder
   ```

자세한 내용은 FreeRTOS에서 CMake 사용 단원을 참조하십시오.

클라우드에서 MQTT 메시지 모니터링

AWS IoT 콘솔에서 MQTT 클라이언트를 사용하여 디바이스가 AWS 클라우드로 보내는 메시지를 모니터링할 수 있습니다.

MQTT 클라이언트를 사용하여 AWS IoT MQTT 주제를 구독하려면

1. AWS IoT 콘솔에 로그인합니다.
2. 탐색 창에서 Test(테스트)를 선택하여 MQTT 클라이언트를 엽니다.
3. 구독 주제에 ***your-thing-name*/example/topic**을 입력한 다음 주제 구독을 선택합니다.

문제 해결

AWS IoT 콘솔의 MQTT 클라이언트에 메시지가 표시되지 않으면 보드에 대한 디버그 설정을 구성해야 할 수 있습니다.

TI 보드에 대한 디버그 설정을 구성하려면

1. Code Composer의 Project Explorer(프로젝트 탐색기)에서 aws_demos를 선택합니다.
2. Run(실행) 메뉴에서 Debug Configurations(디버그 구성)를 선택합니다.
3. 탐색 창에서 aws_demos를 선택합니다.
4. Target(대상) 탭의 Connection Options(연결 옵션)에서 Reset the target on a connect(연결 시 대상 재설정)를 선택합니다.
5. [Apply]를 선택한 다음 [Close]를 선택합니다.

이러한 단계를 수행해도 효과가 없으면 직렬 터미널에서 프로그램의 출력을 살펴봅니다. 문제의 원인을 표시하는 텍스트가 나타나야 합니다.

FreeRTOS 시작하기에 대한 일반 문제 해결 정보는 시작하기 문제 해결 섹션을 참조하세요.

Windows Device Simulator 시작하기

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

이 자습서에서는 FreeRTOS Windows Device Simulator를 시작하기 위한 지침을 제공합니다.

시작하기 전에 AWS IoT 및 FreeRTOS 다운로드를 구성하여 디바이스를 AWS 클라우드에 연결해야 합니다. 자세한 내용은 첫 번째 단계 섹션을 참조하세요. 이 자습서에서는 FreeRTOS 다운로드 디렉터리의 경로를 *freertos*라고 합니다.

FreeRTOS는 지정한 플랫폼용 FreeRTOS 라이브러리 및 샘플 애플리케이션이 포함된 zip 파일로 릴리스됩니다. Windows 머신에서 샘플을 실행하려면 Windows에서 실행하도록 이식된 라이브러리와 샘플을 다운로드합니다. 이 파일 세트를 Windows용 FreeRTOS 시뮬레이터라고 합니다.

> **Note**
>
> Amazon EC2 Windows 인스턴스에서는 이 자습서를 성공적으로 실행할 수 없습니다.

개발 환경 설정

1. 최신 버전의 WinPCap을 설치합니다.
2. Microsoft Visual Studio를 설치합니다.

   Visual Studio 2017 및 2019 버전이 지원됩니다. 모든 Visual Studio 버전이 지원됩니다 (Community, Professional 또는 Enterprise).

   IDE 외에도 Desktop development with C++ 구성 요소를 설치합니다.

   최신 Windows 10 SDK를 설치합니다. C++ 구성 요소를 사용한 데스크톱 개발의 선택 사항 섹션에서 이 옵션을 선택할 수 있습니다.
3. 활성 유선 이더넷 연결이 있는지 확인합니다.
4. (선택 사항) FreeRTOS 프로젝트를 빌드하기 위해 CMake 기반 빌드 시스템을 사용하려면 최신 버전의 CMake를 설치하세요. FreeRTOS를 사용하려면 CMake 버전 3.13 이상이 필요합니다.

FreeRTOS 데모 프로젝트 빌드 및 실행

Visual Studio 또는 CMake를 사용하여 FreeRTOS 프로젝트를 빌드할 수 있습니다.

Visual Studio IDE를 사용하여 FreeRTOS 데모 프로젝트 빌드 및 실행

1. Visual Studio에서 프로젝트를 로드합니다.

   Visual Studio의 파일 메뉴에서 열기를 선택합니다. File/Solution(파일/솔루션)을 선택하고 `projects/pc/windows/visual_studio/aws_demos/aws_demos.sln` 파일로 이동한 다음 열기를 선택합니다.
2. 데모 프로젝트의 목표를 재설정합니다.

제공된 데모 프로젝트는 Windows SDK에 따라 달라지지만 Windows SDK 버전이 지정되어 있지 않습니다. 기본적으로 IDE에서 컴퓨터에 없는 SDK 버전으로 데모를 빌드하려고 시도할 수 있습니다. Windows SDK 버전을 설정하려면 `aws_demos`를 마우스 오른쪽 버튼으로 클릭하고 Retarget Projects(프로젝트 대상 재지정)를 선택합니다. Review Solution Actions(솔루션 작업 검토) 창이 열립니다. 컴퓨터에 있는 Windows SDK 버전(드롭다운의 초기 값)을 선택하고 확인을 선택합니다.

3. 프로젝트를 빌드 및 실행합니다.

   구축 메뉴에서 솔루션 구축을 선택하고 솔루션이 오류나 경고 없이 빌드되는지 확인합니다. 디버깅, Start Debugging(디버깅 시작)을 선택하여 프로젝트를 실행합니다. 처음 실행하는 경우 네트워크 인터페이스를 선택해야 합니다.

CMake로 FreeRTOS 데모 프로젝트 빌드 및 실행

CMake 명령줄 도구 대신 CMake GUI를 사용하여 Windows Simulator용 데모 프로젝트를 빌드하는 것이 좋습니다.

CMake를 설치한 후 CMake GUI를 엽니다. Windows에서는 시작 메뉴의 CMake, CMake (cmake-gui) 아래에서 찾을 수 있습니다.

1. FreeRTOS 소스 코드 디렉터리를 설정합니다.

   GUI의 소스 코드 위치에서 FreeRTOS 소스 코드 디렉터리(*`freertos`*)를 설정합니다.

   바이너리를 빌드할 위치에 *`freertos`*`/build`를 설정합니다.

2. CMake 프로젝트를 구성합니다.

   CMake GUI에서 항목 추가를 선택하고 Add Cache Entry(캐시 항목 추가) 창에서 다음 값을 설정합니다.

   명칭

   AFR_BOARD

   유형

   STRING

   값

   pc.windows

설명

(선택 사항)

3. 구성을 선택합니다. CMake에서 빌드 디렉터리를 만들 것인지 묻는 메시지가 나타나면 예를 선택한 다음 Specify the generator for this project(이 프로젝트의 생성기 지정)에서 생성기를 선택합니다. Visual Studio를 생성기로 사용하는 것이 좋지만 Ninja도 지원됩니다. (Visual Studio 2019 사용 시에는 플랫폼을 기본 설정 대신 Win32로 설정해야 하는 것에 유의하십시오.) 다른 생성기 옵션은 그대로 두고 마침을 선택합니다.
4. CMake 프로젝트를 생성하여 엽니다.

   프로젝트를 구성한 후, CMake GUI는 생성된 프로젝트에 사용할 수 있는 모든 옵션을 표시합니다. 이 자습서에서는 옵션을 기본값으로 두어도 됩니다.

   생성을 선택하여 Visual Studio 솔루션을 만든 다음 프로젝트 열기를 선택하여 Visual Studio에서 프로젝트를 엽니다.

   Visual Studio에서 `aws_demos` 프로젝트를 마우스 오른쪽 버튼으로 클릭하고 Set as StartUp Project(시작 프로젝트로 설정)를 선택합니다. 이를 통해 프로젝트를 빌드하고 실행할 수 있습니다. 처음 실행하는 경우 네트워크 인터페이스를 선택해야 합니다.

FreeRTOS에서 CMake를 사용하는 방법에 대한 자세한 내용은 FreeRTOS에서 CMake 사용 섹션을 참조하세요.

네트워크 인터페이스 구성

데모 프로젝트의 첫 번째 실행 시 사용할 네트워크 인터페이스를 선택해야 합니다. 프로그램에서 네트워크 인터페이스가 열거됩니다. 유선 이더넷 인터페이스의 번호를 찾습니다. 출력은 다음과 같아야 합니다.

```
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.
```

유선 이더넷 인터페이스의 번호를 식별한 후 애플리케이션 창을 닫습니다. 위의 예에서 사용할 번호는 1입니다.

`FreeRTOSConfig.h`를 열고 `configNETWORK_INTERFACE_TO_USE`를 유선 네트워크 인터페이스에 해당하는 번호로 설정합니다.

> **Important**
>
> 이더넷 인터페이스만 지원됩니다. Wi-Fi는 지원되지 않습니다. 자세한 내용은 WinPcap FAQ의 Q-16: WinPCAP에서 지원하는 네트워크 어댑터는 무엇입니까? 항목을 참조하십시오.

클라우드에서 MQTT 메시지 모니터링

AWS IoT 콘솔에서 MQTT 클라이언트를 사용하여 디바이스가 AWS 클라우드로 보내는 메시지를 모니터링할 수 있습니다.

MQTT 클라이언트를 사용하여 AWS IoT MQTT 주제를 구독하려면

1. AWS IoT 콘솔에 로그인합니다.
2. 탐색 창에서 Test(테스트)를 선택하여 MQTT 클라이언트를 엽니다.
3. 구독 주제에 ***your-thing-name*/example/topic**을 입력한 다음 주제 구독을 선택합니다.

문제 해결

Windows의 일반적인 문제 해결

Visual Studio로 데모 프로젝트를 빌드할 때 다음 오류가 발생할 수 있습니다.

```
Error "The Windows SDK version X.Y was not found" when building the provided Visual
 Studio solution.
```

프로젝트는 컴퓨터에 있는 Windows SDK 버전을 대상으로 해야 합니다.

FreeRTOS 시작하기에 대한 일반 문제 해결 정보는 시작하기 문제 해결 섹션을 참조하세요.

Xilinx Avnet MicroZed Industrial IoT Kit 시작하기

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

이 자습서에서는 Xilinx Avnet MicroZed Industrial IoT 키트를 시작하기 위한 지침을 제공합니다. Xilinx Avnet MicroZed Industrial IoT 키트가 없는 경우 AWS 파트너 디바이스 카탈로그를 방문하여 파트너로부터 구매하세요.

시작하기 전에 AWS IoT 및 FreeRTOS 다운로드를 구성하여 디바이스를 AWS 클라우드에 연결해야 합니다. 자세한 내용은 첫 번째 단계 섹션을 참조하세요. 이 자습서에서는 FreeRTOS 다운로드 디렉터리의 경로를 *freertos*라고 합니다.

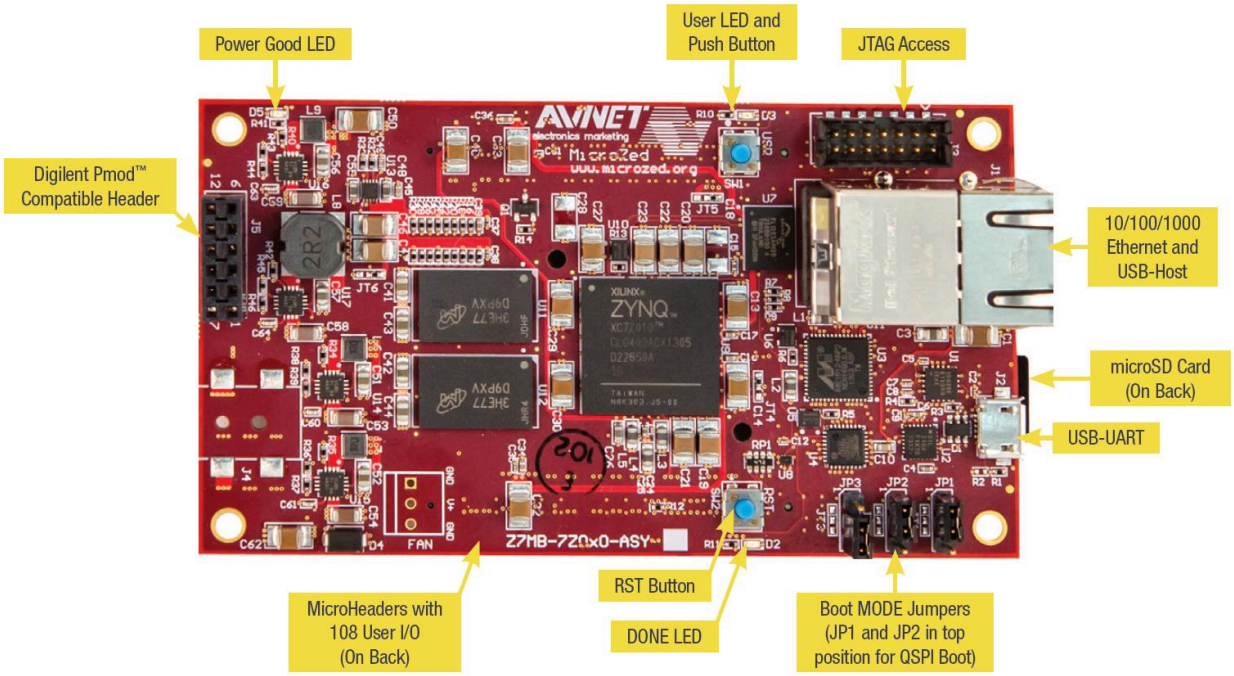
개요

이 자습서에는 다음의 시작하기 단계에 대한 지침이 포함되어 있습니다.

1. 보드를 호스트 시스템에 연결합니다.
2. 마이크로 컨트롤러 보드용 내장형 애플리케이션을 개발 및 디버깅하기 위한 소프트웨어를 호스트 시스템에 설치합니다.
3. FreeRTOS 데모 애플리케이션을 바이너리 이미지로 크로스 컴파일합니다.
4. 애플리케이션 바이너리 이미지를 보드에 로드한 후 애플리케이션을 실행합니다.

MicroZed 하드웨어 설정

다음 다이어그램은 MicroZed 하드웨어를 설정할 때 유용할 수 있습니다.



MicroZed 보드를 설정하려면

1. 컴퓨터를 MicroZed 보드의 USB-UART 포트에 연결합니다.
2. 컴퓨터를 MicroZed 보드의 JTAG Access 포트에 연결합니다.
3. 라우터 또는 인터넷에 연결된 이더넷 포트를 MicroZed 보드의 이더넷 및 USB-Host 포트에 연결합니다.

개발 환경 설정

MicroZed 키트에 대한 FreeRTOS 구성을 설정하려면 Xilinx Software Development Kit(XSDK)를 사용해야 합니다. XSDK는 Windows 및 Linux에서 지원됩니다.

XSDK 다운로드 및 설치

Xilinx 소프트웨어를 설치하려면 무료 Xilinx 계정이 필요합니다.

XSDK를 다운로드하려면

1. Software Development Kit Standalone WebInstall Client 다운로드 페이지로 이동합니다.
2. 운영 체제에 적합한 옵션을 선택합니다.
3. Xilinx 로그인 페이지로 이동합니다.

   Xilinx 계정이 있는 경우 로그인 보안 인증 정보를 입력한 다음 로그인을 선택합니다.

계정이 없는 경우 Create your account를 선택합니다. 등록하면 Xilinx 계정을 활성화하기 위한 링크가 포함된 이메일을 받게 됩니다.

4. Name and Address Verification 페이지에 정보를 입력한 후 Next를 선택합니다. 다운로드를 시작할 준비가 됩니다.
5. `Xilinx_SDK_`*`version`*`_os` 파일을 저장합니다.

XSDK를 설치하려면

1. `Xilinx_SDK_`*`version`*`_os` 파일을 엽니다.
2. Select Edition to Install에서 Xilinx Software Development Kit (XSDK)를 선택한 후 Next를 선택합니다.
3. 설치 마법사의 다음 페이지의 Installation Options에서 Install Cable Drivers를 선택한 후 Next를 선택합니다.

컴퓨터가 MicroZed's USB-UART 연결을 감지하지 못하는 경우 CP210x USB-to-UART Bridge VCP 드라이버를 수동으로 설치합니다. 지침은 Silicon Labs CP210x USB-to-UART Installation Guide를 참조하십시오.

XSDK에 대한 자세한 내용은 Xilink 웹 사이트에서 Getting Started with Xilinx SDK를 참조하십시오.

FreeRTOS 데모 프로젝트 빌드 및 실행

XSDK IDE에서 FreeRTOS 데모 열기

1. 작업 영역 디렉터리가 *`freertos`*`/projects/xilinx/microzed/xsdk`로 설정된 상태로 XSDK IDE를 시작합니다.
2. 시작 페이지를 닫습니다. 메뉴에서 Project를 선택한 후 Build Automatically를 지웁니다.
3. 메뉴에서 File을 선택한 후 Import를 선택합니다.
4. Select 페이지에서 General을 확장하고 Existing Projects into Workspace를 선택한 후 Next를 선택합니다.
5. Import Projects(프로젝트 가져오기) 페이지에서 Select root directory(루트 디렉터리 선택)를 선택한 후 데모 프로젝트 *`freertos`*`/projects/xilinx/microzed/xsdk/aws_demos`의 루트 디렉터리를 입력합니다. 디렉터리를 찾아보려면 Browse를 선택합니다.

   루트 디렉터리를 지정하면 Import Projects 페이지에 해당 디렉터리의 프로젝트가 표시됩니다. 사용 가능한 모든 프로젝트가 기본적으로 선택되어 있습니다.

> **Note**
>
> Import Projects 페이지의 상단에 경고가 표시되는 경우("Some projects cannot be imported because they already exist in the workspace") 이를 무시할 수 있습니다.

6. 프로젝트가 모두 선택된 상태에서 Finish를 선택합니다.
7. 프로젝트 창에 `aws_bsp`, `fsbl` 및 `MicroZed_hw_platform_0` 프로젝트가 표시되지 않는 경우 이전 단계의 #3부터 반복하되, 루트 디렉터리를 *`freertos`*`/vendors/xilinx`로 설정하고 `aws_bsp`, `fsbl` 및 `MicroZed_hw_platform_0`을 가져옵니다.
8. 메뉴에서 Window를 선택한 후 Preferences를 선택합니다.
9. 탐색 창에서 Run/Debug를 확장하고 String Substitution을 선택한 후 New를 선택합니다.
10. New String Substitution Variable의 Name에 **`AFR_ROOT`**를 입력합니다. Value(값)에 *`freertos`*`/projects/xilinx/microzed/xsdk/aws_demos`의 루트 경로를 입력합니다. OK를 선택한 후 OK를 선택하여 변수를 저장하고 Preferences를 닫습니다.

FreeRTOS 데모 프로젝트 빌드

1. XSDK IDE의 메뉴에서 Project를 선택한 후 Clean을 선택합니다.
2. Clean에서 옵션을 기본 값으로 둔 후 OK를 선택합니다. XSDK가 프로젝트를 모두 정리하고 빌드한 후 `.elf` 파일을 생성합니다.

> **Note**
>
> 모든 프로젝트를 정리하지 않고 빌드하려면 프로젝트를 선택한 후 Build All(모두 빌드)을 선택합니다.
> 개별 프로젝트를 빌드하려면 빌드하려는 프로젝트를 선택하고 프로젝트를 선택한 후 빌드 프로젝트를 선택합니다.

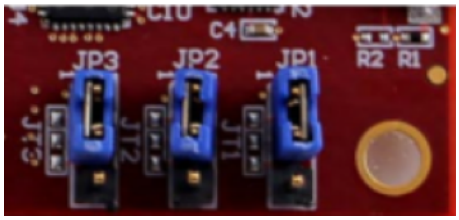
FreeRTOS 데모 프로젝트의 부팅 이미지 생성

1. XSDK IDE에서 aws_demos를 마우스 오른쪽 버튼으로 클릭한 후 Create Boot Image를 선택합니다.
2. Create Boot Image에서 Create new BIF file을 선택합니다.

3. 출력 BIF 파일 경로 옆의 찾아보기를 선택한 후 *<freertos>*/vendors/xilinx/microzed/aws_demos/aws_demos.bif에 있는 aws_demos.bif를 선택합니다.
4. 추가를 선택합니다.
5. Add new boot image partition(새 부팅 이미지 파티션 추가)에서 File path(파일 경로) 옆의 찾아보기를 선택한 후 vendors/xilinx/fsbl/Debug/fsbl.elf에 있는 fsbl.elf를 선택합니다.
6. Partition type에서 bootloader를 선택한 후 OK를 선택합니다.
7. Create Boot Image에서 Create Image를 선택합니다. Override Files(파일 재정의)에서 확인을 선택하여 기존 aws_demos.bif를 덮어쓰고 projects/xilinx/microzed/xsdk/aws_demos/BOOT.bin에서 BOOT.bin 파일을 생성합니다.

JTAG 디버깅

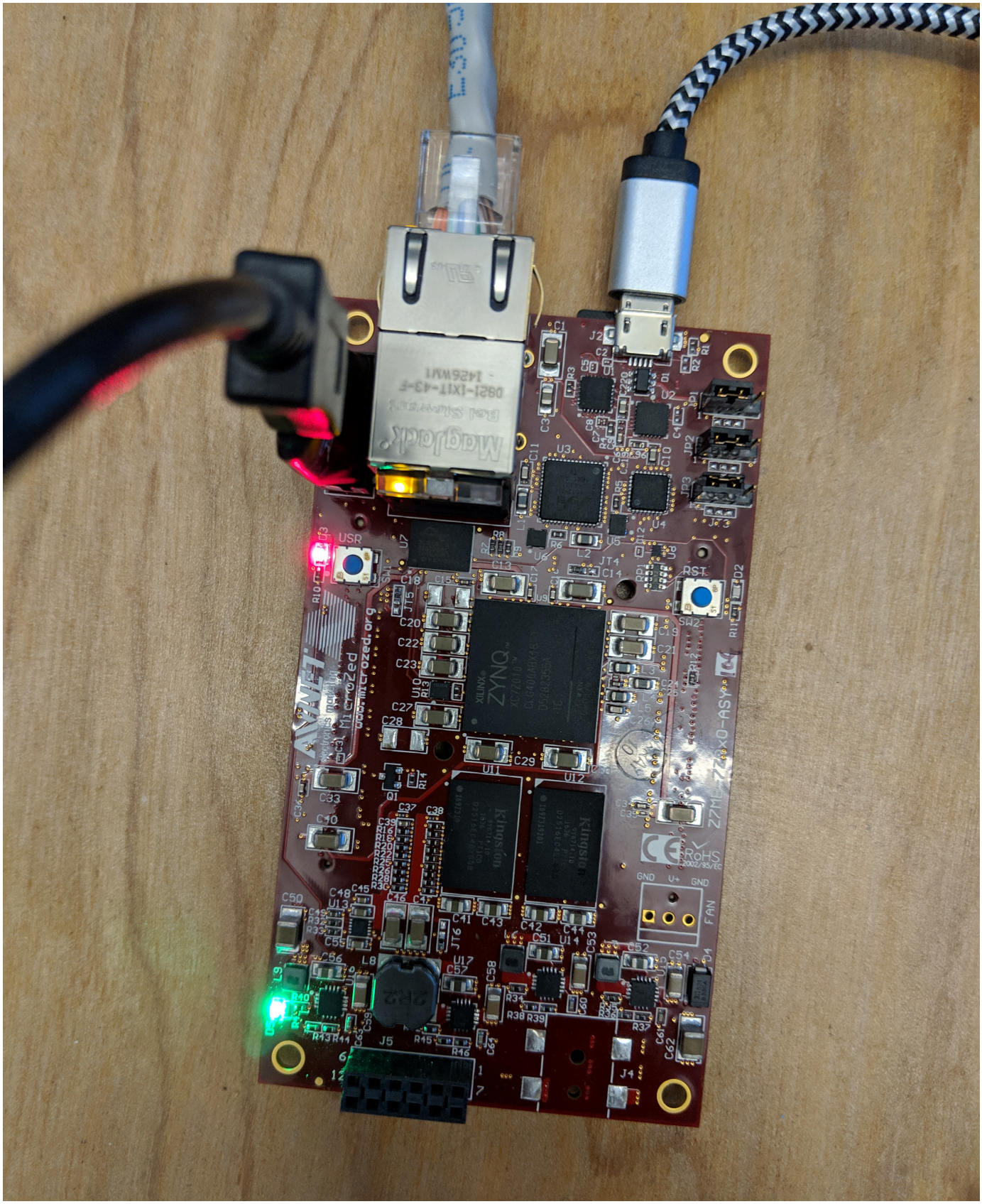
1. MicroZed 보드의 부팅 모드 점퍼를 JTAG 부팅 모드로 설정합니다.



2. USB-UART 포트 바로 아래에 있는 MicroSD 카드 슬롯에 MicroSD 카드를 삽입합니다.

> **Note**
>
> 디버깅하기 전에 MicroSD 카드에 있는 모든 콘텐츠를 백업하십시오.

보드는 다음과 같은 모습이어야 합니다.


USR
RST
ZYNQ
XILINX
Kingston
GND
V+
GND
FAN
RoHS

3. XSDK IDE에서 aws_demos를 마우스 오른쪽 버튼으로 클릭하고 Debug As를 선택한 후 1 Launch on System Hardware (System Debugger)를 선택합니다.
4. 디버거가 `main()`의 중단점에서 중지되면 메뉴에서 Run을 선택한 후 Resume을 선택합니다.

> **Note**
>
> 애플리케이션을 처음 실행할 때 새 인증서-키 페어가 비휘발성 메모리로 가져오기 됩니다. 후속 실행의 경우 이미지와 `BOOT.bin` 파일을 다시 빌드하기 전에 `main.c` 파일에서 `vDevModeKeyProvisioning()`을 주석 처리할 수 있습니다. 그러면 실행할 때마다 인증서와 키를 스토리지에 복사하지 않아도 됩니다.

MicroSD 카드 또는 QSPI 플래시에서 MicroZed 보드를 부팅하여 FreeRTOS 데모 프로젝트를 실행하도록 선택할 수 있습니다. 지침은 FreeRTOS 데모 프로젝트의 부팅 이미지 생성 및 FreeRTOS 데모 프로젝트 실행 섹션을 참조하세요.

FreeRTOS 데모 프로젝트 실행

FreeRTOS 데모 프로젝트를 실행하려면 MicroSD 카드 또는 QSPI 플래시에서 MicroZed 보드를 부팅하면 됩니다.

FreeRTOS 데모 프로젝트를 실행하기 위해 MicroZed 보드를 설정할 때 MicroZed 하드웨어 설정의 다이어그램을 참조하세요. MicroZed 보드를 컴퓨터에 연결했는지 확인합니다.

MicroSD 카드에서 FreeRTOS 프로젝트 부팅

Xilinx MicroZed Industrial IoT Kit와 함께 제공되는 MicroSD 카드를 포맷합니다.

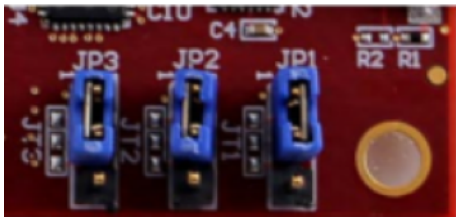
1. `BOOT.bin` 파일을 MicroSD 카드에 복사합니다.
2. USB-UART 포트 바로 아래의 MicroSD 카드 슬롯에 카드를 삽입합니다.
3. MicroZed 부팅 모드 점퍼를 SD 부팅 모드로 설정합니다.



4. RST 버튼을 눌러 디바이스를 재설정하고 애플리케이션 부팅을 시작합니다. USB-UART 포트에서 USB-UART 케이블을 뽑은 후 케이블을 다시 삽입할 수도 있습니다.

QSPI 플래시에서 FreeRTOS 데모 프로젝트 부팅

1. MicroZed 보드의 부팅 모드 점퍼를 JTAG 부팅 모드로 설정합니다.



2. 컴퓨터가 USB-UART 및 JTAG Access 포트에 연결되어 있는지 확인합니다. 녹색 Power Good LED 조명이 켜져야 합니다.
3. XSDK IDE의 메뉴에서 Xilinx를 선택한 후 Program Flash를 선택합니다.
4. Program Flash Memory에서 하드웨어 플랫폼이 자동으로 채워져야 합니다. Connection에서 보드를 호스트 컴퓨터와 연결할 MicroZed 하드웨어 서버를 선택합니다.

> **Note**
>
> Xilinx Smart Lync JTAG 케이블을 사용하는 경우 XSDK IDE에 하드웨어 서버를 만들어야 합니다. New를 선택한 후 서버를 정의합니다.

5. Image File에 `BOOT.bin` 이미지 파일의 디렉터리 경로를 입력합니다. 파일을 찾아보려면 Browse를 선택합니다.
6. Offset에 **`0x0`**을 입력합니다.
7. FSBL File에 `fsbl.elf` 파일의 디렉터리 경로를 입력합니다. 파일을 찾아보려면 Browse를 선택합니다.
8. Program을 선택하여 보드를 프로그래밍합니다.
9. QSPI 프로그래밍을 완료한 후 USB-UART 케이블을 제거하여 보드의 전원을 끕니다.
10. MicroZed 보드의 부팅 모드 점퍼를 QSPI 부팅 모드로 설정합니다.
11. USB-UART 포트 바로 아래에 있는 MicroSD 카드 슬롯에 카드를 삽입합니다.

> **Note**
>
> MicroSD 카드에 있는 모든 콘텐츠를 백업하십시오.

12. RST 버튼을 눌러 디바이스를 재설정하고 애플리케이션 부팅을 시작합니다. USB-UART 포트에서 USB-UART 케이블을 뽑은 후 케이블을 다시 삽입할 수도 있습니다.

클라우드에서 MQTT 메시지 모니터링

AWS IoT 콘솔에서 MQTT 클라이언트를 사용하여 디바이스가 AWS 클라우드로 보내는 메시지를 모니터링할 수 있습니다.

MQTT 클라이언트를 사용하여 AWS IoT MQTT 주제를 구독하려면

1. AWS IoT 콘솔에 로그인합니다.
2. 탐색 창에서 Test(테스트)를 선택하여 MQTT 클라이언트를 엽니다.
3. 구독 주제에 ***your-thing-name*/example/topic**을 입력한 다음 주제 구독을 선택합니다.

문제 해결

잘못된 경로와 관련된 빌드 오류가 발생한 경우 FreeRTOS 데모 프로젝트 빌드에 설명된 대로 프로젝트를 정리하고 다시 빌드해 보십시오.

Windows를 사용하는 경우 Windows XSDK IDE에서 문자열 대체 변수를 설정할 때 슬래시를 사용해야 합니다.

FreeRTOS 시작하기에 대한 일반 문제 해결 정보는 시작하기 문제 해결 섹션을 참조하세요.

FreeRTOS 무선 업데이트(OTA)

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

무선 업데이트(OTA)를 사용하면 플릿에 있는 하나 이상의 디바이스에 펌웨어 업데이트를 배포할 수 있습니다. OTA 업데이트는 디바이스 펌웨어를 업데이트하도록 설계되었지만, OTA 업데이트를 사용하여 AWS IoT에 등록된 하나 이상의 디바이스에 파일을 전송할 수 있습니다. 무선으로 업데이트를 전송할 경우 파일을 수신하는 디바이스에서 파일이 중간에 변조되지 않았음을 확인할 수 있도록 파일에 디지털 방식으로 서명하는 것이 좋습니다.

AWS IoT용 코드 서명을 사용하여 파일에 서명하거나 자체 코드 서명 도구를 사용하여 파일에 서명할 수 있습니다.

OTA 업데이트를 생성할 경우 OTA 업데이트 관리자 서비스에서는 디바이스에 업데이트가 사용 가능함을 알려주는 AWS IoT 작업을 생성합니다. OTA 데모 애플리케이션은 디바이스에서 실행되며 AWS IoT 작업에 대한 알림 주제를 구독하고 업데이트 메시지를 수신하는 FreeRTOS 작업을 생성합니다. 업데이트를 사용할 수 있는 경우 OTA 에이전트는 선택한 설정에 따라 HTTP 또는 MQTT 프로토콜을 사용하여 AWS IoT 에 요청을 게시하고 업데이트를 수신합니다. OTA 에이전트는 다운로드한 파일의 디지털 서명을 확인하고 유효할 경우 펌웨어 업데이트를 설치합니다. FreeRTOS OTA 업데이트 데모 애플리케이션을 사용하지 않을 경우 OTA 에이전트 라이브러리를 자체 애플리케이션에 통합하여 펌웨어 업데이트 기능을 가져와야 합니다.

FreeRTOS 무선(OTA) 업데이트를 사용하여 다음을 수행할 수 있습니다.

- 배포 전에 펌웨어에 디지털 방식으로 서명합니다.
- 새 펌웨어 이미지를 단일 디바이스, 디바이스 그룹 또는 전체 플릿에 배포합니다.
- 그룹에 추가되거나, 재설정되거나, 다시 프로비저닝되는 디바이스에 펌웨어를 배포합니다.
- 디바이스에 배포된 이후에 새 펌웨어의 신뢰성과 무결성을 확인합니다.
- 배포 진행 상황을 모니터링합니다.
- 실패한 배포를 디버깅합니다.

OTA 리소스에 태그 지정

필요할 경우 자체 메타데이터를 태그의 형태로 업데이트 및 스트림에 배정하면 OTA 리소스를 쉽게 관리할 수 있습니다. 태그를 사용하면 AWS IoT 리소스를 다양한 방법(예: 용도, 소유자 또는 환경)으로 분류할 수 있습니다. 이는 동일한 유형의 리소스가 많을 때 유용합니다. 지정한 태그를 기반으로 리소스를 신속하게 식별할 수 있습니다.

자세한 내용을 알아보려면 AWS IoT 리소스에 태그 지정을 참조하세요.

OTA 업데이트 사전 조건

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

무선 업데이트(OTA)를 사용하려면 다음을 수행합니다.

- HTTP를 사용한 OTA 업데이트 사전 조건 또는 MQTT를 사용한 OTA 업데이트 사전 조건를 확인합니다.
- 업데이트를 저장할 Amazon S3 버킷 생성.
- OTA 업데이트 서비스 역할 생성.
- OTA 사용자 정책 생성.
- 코드 서명 인증서 생성.
- 에 코드 서명을 사용하는 경우 AWS IoT에 대한 코드 서명에 대한 액세스 권한 부여 AWS IoT.
- OTA 라이브러리와 함께 FreeRTOS 다운로드.

업데이트를 저장할 Amazon S3 버킷 생성

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

OTA 업데이트 파일은 Amazon S3 버킷에 저장됩니다.

Code Signing for AWS IoT를 사용하는 경우 코드 서명 작업을 생성하는 데 사용하는 명령은 소스 버킷(서명되지 않은 펌웨어 이미지가 있는 위치)과 대상 버킷(서명된 펌웨어 이미지가 작성된 위치)을 사용합니다. 소스와 대상 모두에 동일한 버킷을 지정할 수 있습니다. 원래 파일을 덮어쓰지 않도록 파일 이름이 GUID로 변경됩니다.

Amazon S3 버킷을 생성하려면

1. https://console.aws.amazon.com/s3/에서 Amazon S3 콘솔에 로그인합니다.
2. 버킷 생성을 선택합니다.
3. 버킷 이름을 입력합니다.
4. 퍼블릭 액세스 차단을 위한 버킷 설정에서 모든 퍼블릭 액세스 차단을 선택한 상태로 유지하여 기본 권한을 수락합니다.
5. 버킷 버전 관리에서 활성화를 선택하여 모든 버전을 동일 버킷 내에 유지합니다.
6. 버킷 생성을 선택합니다.

Amazon S3에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서를 참조하세요.

OTA 업데이트 서비스 역할 생성

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

OTA 업데이트 서비스에서는 이 역할을 사용하여 고객 대신 OTA 업데이트 작업을 생성하고 관리한다고 가정합니다.

OTA 서비스 역할을 생성하려면

1. https://console.aws.amazon.com/iam/ 로그인합니다.
2. 탐색 창에서 역할을 선택합니다.
3. 역할 생성을 선택합니다.
4. [신뢰할 수 있는 유형의 엔터티 선택(Select type of trusted entity)] 아래에서 AWS 서비스를 선택합니다.

5. AWS 서비스 목록에서 IoT를 선택합니다.
6. 사용 사례 선택(Select your use case) 아래에서 IoT를 선택합니다.
7. 다음: 권한을 선택합니다.
8. 다음: 태그를 선택합니다.
9. 다음: 검토를 선택합니다.
10. 역할 이름과 설명을 입력한 후 역할 생성을 선택합니다.

IAM 역할에 대한 자세한 내용은 IAM 역할을 참조하세요.

OTA 서비스 역할에 OTA 업데이트 권한을 추가하려면

1. IAM 콘솔 페이지의 검색 상자에 역할 이름을 입력한 후 목록에서 해당 역할을 선택합니다.
2. 정책 연결을 선택합니다.
3. 검색 상자에 'AmazonFreeRTOSOTAUpdate'를 입력하고 필터링된 정책 목록에서 `AmazonFreeRTOSOTAUpdate`를 선택한 후 정책 연결을 선택하여 정책을 서비스 역할에 연결합니다.

OTA 서비스 역할에 필요한 IAM 권한을 추가하려면

1. IAM 콘솔 페이지의 검색 상자에 역할 이름을 입력한 후 목록에서 해당 역할을 선택합니다.
2. 인라인 정책 추가(Add inline policy)를 선택합니다.
3. JSON 탭을 선택합니다.
4. 다음 정책 문서를 복사해 텍스트 상자에 붙여 넣습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "iam:PassRole"
            ],
            "Resource": "arn:aws:iam::your_account_id:role/your_role_name"
      }
    ]
```

```
}
```

*your_account_id*를 AWS 계정 ID로 바꾸고 *your_role_name*을 OTA 서비스 역할의 이름으로 바꿔야 합니다.

5. 정책 검토를 선택합니다.
6. 정책 이름을 입력한 후 정책 생성을 선택합니다.

> **Note**
>
> Amazon S3 버킷 이름이 'afr-ota'로 시작하는 경우에는 다음 절차가 필요하지 않습니다. 이 경우 AWS 관리형 정책에는 필요한 권한이 `AmazonFreeRTOSOTAUpdate` 이미 포함되어 있습니다.

OTA 서비스 역할에 필요한 Amazon S3 권한을 추가하려면

1. IAM 콘솔 페이지의 검색 상자에 역할 이름을 입력한 후 목록에서 해당 역할을 선택합니다.
2. 인라인 정책 추가(Add inline policy)를 선택합니다.
3. JSON 탭을 선택합니다.
4. 다음 정책 문서를 복사하여 상자에 붙여넣습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObjectVersion",
                "s3:GetObject",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::example-bucket/*"
            ]
        }
    ]
}
```

이 정책은 Amazon S3 객체를 읽을 수 있는 OTA 서비스 역할 권한을 부여합니다. *example-bucket*을 해당 버킷 이름으로 바꿉니다.

5. 정책 검토를 선택합니다.
6. 정책 이름을 입력한 후 정책 생성을 선택합니다.

OTA 사용자 정책 생성

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

IAM 사용자에게 무선 업데이트(OTA) 수행 권한을 부여해야 합니다. 사용자에게 다음에 대한 권한이 있어야 합니다.

- 펌웨어 업데이트를 저장할 S3 액세스
- 에 저장된 인증서에 액세스합니다 AWS Certificate Manager.
- AWS IoT MQTT 기반 파일 전송 기능에 액세스합니다.
- FreeRTOS OTA 업데이트에 액세스
- AWS IoT 작업에 액세스합니다.
- IAM에 액세스
- 에 대한 액세스 코드 서명입니다 AWS IoT. 에 대한 코드 서명에 대한 액세스 권한 부여 AWS IoT을(를) 참조하세요.
- FreeRTOS 하드웨어 플랫폼 나열
- AWS IoT 리소스에 태그를 지정하고 태그를 해제합니다.

액세스 권한을 제공하려면 사용자, 그룹 또는 역할에 권한을 추가하세요:

- 의 사용자 및 그룹 AWS IAM Identity Center:

  권한 세트를 생성합니다. AWS IAM Identity Center 사용 설명서의 권한 세트 생성의 지침을 따릅니다.
- 보안 인증 공급자를 통해 IAM에서 관리되는 사용자:

ID 페더레이션을 위한 역할을 생성합니다. IAM 사용 설명서의 Create a role for a third-party identity provider (federation)의 지침을 따릅니다.

- IAM 사용자:
  - 사용자가 맡을 수 있는 역할을 생성합니다. IAM 사용 설명서에서 Create a role for an IAM user의 지침을 따릅니다.
  - (권장되지 않음)정책을 사용자에게 직접 연결하거나 사용자를 사용자 그룹에 추가합니다. IAM 사용 설명서에서 사용자(콘솔)에 권한 추가의 지침을 따르세요.

코드 서명 인증서 생성

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

펌웨어 이미지에 디지털 방식으로 서명하려면 코드 서명 인증서와 프라이빗 키가 필요합니다. 테스트 목적으로 자체 서명된 인증서와 프라이빗 키를 생성할 수 있습니다. 프로덕션 환경에서는 잘 알려진 인증 기관(CA)을 통해 인증서를 구입하십시오.

플랫폼에 따라 필요한 코드 서명 인증서 유형이 다릅니다. 다음 단원에서는 서로 다른 FreeRTOS 적격 플랫폼에 대한 코드 서명 인증서를 생성하는 방법을 설명합니다.

주제

- Texas Instruments CC3220SF-LAUNCHXL을 위한 코드 서명 인증서 생성
- Microchip Curiosity PIC32MZEF에 대한 코드 서명 인증서 생성
- Espressif ESP32에 대한 코드 서명 인증서 생성
- Nordic nrf52840-dk에 대한 코드 서명 인증서 생성
- FreeRTOS Windows 시뮬레이터용 코드 서명 인증서 생성
- 사용자 지정 하드웨어에 대한 코드 서명 인증서 생성

Texas Instruments CC3220SF-LAUNCHXL을 위한 코드 서명 인증서 생성

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

SimpleLink Wi-Fi CC3220SF Wireless Microcontroller Launchpad Development Kit는 펌웨어 코드 서명을 위한 두 가지 인증서 체인을 지원합니다.

- 프로덕션(인증서 카탈로그)

  프로덕션 인증서 체인을 사용하려면 상용 코드 서명 인증서를 구매하고 TI Uniflash 도구를 사용하여 보드를 프로덕션 모드로 설정해야 합니다.

- 테스트 및 개발(인증서 실습)

  플레이그라운드 인증서 체인을 사용하면 자체 서명된 코드 서명 인증서를 사용하여 OTA 업데이트를 시험해 볼 수 있습니다.

AWS Command Line Interface 를 사용하여 코드 서명 인증서, 프라이빗 키 및 인증서 체인을 로 가져옵니다 AWS Certificate Manager. 자세한 내용은 AWS Command Line Interface 사용 설명서에서 AWS CLI설치를 참조하세요.

최신 버전의 SimpleLink CC3220 SDK를 다운로드하여 설치합니다. 기본적으로 필요한 파일은 다음 위치에 있습니다.

`C:\ti\simplelink_cc32xx_sdk_version\tools\cc32xx_tools\certificate-playground`(Windows)

`/Applications/Ti/simplelink_cc32xx_version/tools/cc32xx_tools/certificate-playground`(macOS)

SimpleLink CC3220 SDK의 인증서는 DER 형식입니다. 자체 서명된 코드 서명 인증서를 생성하려면 이를 PEM 형식으로 변환해야 합니다.

다음 단계에 따라 Texas Instruments 플레이그라운드 인증서 계층 구조에 연결되고 AWS Certificate Manager 및 Code Signing for AWS IoT criteria를 충족하는 코드 서명 인증서를 생성합니다.

> **Note**
>
> 코드 서명 인증서를 생성하려면 컴퓨터에 OpenSSL을 설치합니다. OpenSSL을 설치한 후 명령 프롬프트나 터미널 환경에서 `openssl`이 OpenSSL 실행 파일에 할당되었는지 확인합니다.

자체 서명된 코드 서명 인증서를 생성하려면

1. 명령 프롬프트 창이나 터미널을 관리자 권한으로 엽니다.
2. 작업 디렉터리에서 다음 텍스트를 사용하여 `cert_config.txt` 파일을 생성합니다. *test_signer@amazon.com*을 사용자의 이메일 주소로 바꿉니다.

```
[ req ]
prompt             = no
distinguished_name = my dn

[ my dn ]
commonName = test_signer@amazon.com

[ my_exts ]
keyUsage         = digitalSignature
extendedKeyUsage = codeSigning
```

3. 프라이빗 키 및 인증서 서명 요청(CSR)을 생성합니다.

```
openssl req -config cert_config.txt -extensions my_exts -nodes -days 365 -newkey
 rsa:2048 -keyout tisigner.key -out tisigner.csr
```

4. Texas Instruments 실습 루트 CA 프라이빗 키를 DER 형식에서 PEM 형식으로 변환합니다.

   TI 실습 루트 CA 프라이빗 키는 다음 위치에 있습니다.

   `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. Texas Instruments 실습 루트 CA 인증서를 DER 형식에서 PEM 형식으로 변환합니다.

TI 실습 루트 인증서는 다음 위치에 있습니다.

`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. Texas Instruments 루트 CA로 CSR에 서명합니다.

```
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. 코드 서명 인증서(`tisigner.crt.pem`)를 DER 형식으로 변환합니다.

```
openssl x509 -in tisigner.crt.pem -out tisigner.crt.der -outform DER
```

> **Note**
>
> 나중에 TI 개발 보드에 `tisigner.crt.der` 인증서를 작성합니다.

8. 코드 서명 인증서, 프라이빗 키 및 인증서 체인을 AWS Certificate Manager다음으로 가져옵니다.

```
aws acm import-certificate --certificate fileb://tisigner.crt.pem --private-key
 fileb://tisigner.key --certificate-chain fileb://dummy-root-ca-cert.pem
```

이 명령은 인증서에 대한 ARN을 표시합니다. OTA 업데이트 작업을 생성할 때 이 ARN이 필요합니다.

> **Note**
>
> 이 단계는 용 코드 서명을 사용하여 펌웨어 이미지에 서명한다는 가정하 AWS IoT 에 작성되었습니다. 용 코드 서명을 사용하는 AWS IoT 것이 좋지만 펌웨어 이미지에 수동으로 서명할 수 있습니다.

Microchip Curiosity PIC32MZEF에 대한 코드 서명 인증서 생성

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

Microchip Curiosity PIC32MZEF에서는 ECDSA 코드 서명 인증서로 자체 서명된 SHA256을 지원합니다.

> **Note**
>
> 코드 서명 인증서를 생성하려면 컴퓨터에 OpenSSL을 설치합니다. OpenSSL을 설치한 후 명령 프롬프트나 터미널 환경에서 `openssl`이 OpenSSL 실행 파일에 할당되었는지 확인합니다. AWS Command Line Interface 를 사용하여 코드 서명 인증서, 프라이빗 키 및 인증서 체인을 로 가져옵니다 AWS Certificate Manager. 설치에 대한 자세한 내용은 설치를 AWS CLI참조하세요. AWS CLI

1. 작업 디렉터리에서 다음 텍스트를 사용하여 `cert_config.txt` 파일을 생성합니다. *`test_signer@amazon.com`*을 사용자의 이메일 주소로 바꿉니다.

```
[ req ]
prompt             = no
distinguished_name = my_dn

[ my_dn ]
commonName = test_signer@amazon.com

[ my_exts ]
keyUsage         = digitalSignature
extendedKeyUsage = codeSigning
```

2. ECDSA 코드 서명 프라이빗 키를 생성합니다.

```
openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -pkeyopt
 ec_param_enc:named_curve -outform PEM -out ecdsasigner.key
```

3. ECDSA 코드 서명 인증서를 생성합니다.

```
openssl req -new -x509 -config cert_config.txt -extensions my_exts -nodes -days 365
 -key ecdsasigner.key -out ecdsasigner.crt
```

4. 코드 서명 인증서, 프라이빗 키 및 인증서 체인을 AWS Certificate Manager다음으로 가져옵니다.

```
aws acm import-certificate --certificate fileb://ecdsasigner.crt --private-key
 fileb://ecdsasigner.key
```

이 명령은 인증서에 대한 ARN을 표시합니다. OTA 업데이트 작업을 생성할 때 이 ARN이 필요합니다.

> **Note**
>
> 이 단계는 용 코드 서명을 사용하여 펌웨어 이미지에 서명한다는 가정하 AWS IoT 에 작성되었습니다. 용 코드 서명을 사용하는 AWS IoT 것이 좋지만 펌웨어 이미지에 수동으로 서명할 수 있습니다.

Espressif ESP32에 대한 코드 서명 인증서 생성

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

Espressif ESP32 보드는 ECDSA 코드 서명 인증서로 자체 서명된 SHA-256을 지원합니다.

> **Note**
>
> 코드 서명 인증서를 생성하려면 컴퓨터에 OpenSSL을 설치합니다. OpenSSL을 설치한 후 명령 프롬프트나 터미널 환경에서 `openssl`이 OpenSSL 실행 파일에 할당되었는지 확인합니다. AWS Command Line Interface 를 사용하여 코드 서명 인증서, 프라이빗 키 및 인증서 체인을 로 가져옵니다 AWS Certificate Manager. 설치에 대한 자세한 내용은 설치를 AWS CLI참조하세요. AWS CLI

1. 작업 디렉터리에서 다음 텍스트를 사용하여 `cert_config.txt` 파일을 생성합니다. *test_signer@amazon.com*을 사용자의 이메일 주소로 바꿉니다.

```
[ req ]
prompt             = no
distinguished_name = my_dn

[ my_dn ]
commonName = test_signer@amazon.com

[ my_exts ]
keyUsage         = digitalSignature
extendedKeyUsage = codeSigning
```

2. ECDSA 코드 서명 프라이빗 키를 생성합니다.

```
openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -pkeyopt
 ec_param_enc:named_curve -outform PEM -out ecdsasigner.key
```

3. ECDSA 코드 서명 인증서를 생성합니다.

```
openssl req -new -x509 -config cert_config.txt -extensions my_exts -nodes -days 365
 -key ecdsasigner.key -out ecdsasigner.crt
```

4. 코드 서명 인증서, 프라이빗 키 및 인증서 체인을 AWS Certificate Manager다음으로 가져옵니다.

```
aws acm import-certificate --certificate fileb://ecdsasigner.crt --private-key
 fileb://ecdsasigner.key
```

이 명령은 인증서에 대한 ARN을 표시합니다. OTA 업데이트 작업을 생성할 때 이 ARN이 필요합니다.

> **Note**
>
> 이 단계는 용 코드 서명을 사용하여 펌웨어 이미지에 서명한다는 가정하 AWS IoT 에 작성되었습니다. 용 코드 서명을 사용하는 AWS IoT 것이 좋지만 펌웨어 이미지에 수동으로 서명할 수 있습니다.

Nordic nrf52840-dk에 대한 코드 서명 인증서 생성

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

Nordic nrf52840-dk는 ECDSA 코드 서명 인증서로 자체 서명된 SHA256을 지원합니다.

> **Note**
>
> 코드 서명 인증서를 생성하려면 컴퓨터에 OpenSSL을 설치합니다. OpenSSL을 설치한 후 명령 프롬프트나 터미널 환경에서 `openssl`이 OpenSSL 실행 파일에 할당되었는지 확인합니다. AWS Command Line Interface 를 사용하여 코드 서명 인증서, 프라이빗 키 및 인증서 체인을로 가져옵니다 AWS Certificate Manager. 설치에 대한 자세한 내용은 설치를 AWS CLI참조하세요. AWS CLI

1. 작업 디렉터리에서 다음 텍스트를 사용하여 `cert_config.txt` 파일을 생성합니다. *test_signer@amazon.com*을 사용자의 이메일 주소로 바꿉니다.

```
[ req ]
prompt             = no
distinguished_name = my_dn

[ my_dn ]
commonName = test_signer@amazon.com

[ my_exts ]
keyUsage         = digitalSignature
extendedKeyUsage = codeSigning
```

2. ECDSA 코드 서명 프라이빗 키를 생성합니다.

```
openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -pkeyopt
 ec_param_enc:named_curve -outform PEM -out ecdsasigner.key
```

3. ECDSA 코드 서명 인증서를 생성합니다.

```
openssl req -new -x509 -config cert_config.txt -extensions my_exts -nodes -days 365
 -key ecdsasigner.key -out ecdsasigner.crt
```

4. 코드 서명 인증서, 프라이빗 키 및 인증서 체인을 AWS Certificate Manager다음으로 가져옵니다.

```
aws acm import-certificate --certificate fileb://ecdsasigner.crt --private-key
 fileb://ecdsasigner.key
```

이 명령은 인증서에 대한 ARN을 표시합니다. OTA 업데이트 작업을 생성할 때 이 ARN이 필요합니다.

> **Note**
>
> 이 단계는 용 코드 서명을 사용하여 펌웨어 이미지에 서명한다는 가정하 AWS IoT 에 작성되었습니다. 용 코드 서명을 사용하는 AWS IoT 것이 좋지만 펌웨어 이미지에 수동으로 서명할 수 있습니다.

FreeRTOS Windows 시뮬레이터용 코드 서명 인증서 생성

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

FreeRTOS Windows 시뮬레이터에서 OTA 업데이트를 수행하려면 ECDSA P-256 키 및 SHA-256 해시와 함께 코드 서명 인증서가 필요합니다. 코드 서명 인증서가 없는 경우 다음 단계를 따라 인증서를 생성합니다.

> **Note**
>
> 코드 서명 인증서를 생성하려면 컴퓨터에 OpenSSL을 설치합니다. OpenSSL을 설치한 후 명령 프롬프트나 터미널 환경에서 `openssl`이 OpenSSL 실행 파일에 할당되었는지 확인합니다. AWS Command Line Interface 를 사용하여 코드 서명 인증서, 프라이빗 키 및 인증서 체인을 로 가져옵니다 AWS Certificate Manager. 설치에 대한 자세한 내용은 설치를 AWS CLI참조하세요. AWS CLI

1. 작업 디렉터리에서 다음 텍스트를 사용하여 `cert_config.txt` 파일을 생성합니다. *test_signer@amazon.com*을 사용자의 이메일 주소로 바꿉니다.

```
[ req ]
prompt             = no
distinguished_name = my_dn

[ my_dn ]
commonName = test_signer@amazon.com

[ my_exts ]
keyUsage         = digitalSignature
extendedKeyUsage = codeSigning
```

2. ECDSA 코드 서명 프라이빗 키를 생성합니다.

```
openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -pkeyopt
 ec_param_enc:named_curve -outform PEM -out ecdsasigner.key
```

3. ECDSA 코드 서명 인증서를 생성합니다.

```
openssl req -new -x509 -config cert_config.txt -extensions my_exts -nodes -days 365
 -key ecdsasigner.key -out ecdsasigner.crt
```

4. 코드 서명 인증서, 프라이빗 키 및 인증서 체인을 AWS Certificate Manager다음으로 가져옵니다.

```
aws acm import-certificate --certificate fileb://ecdsasigner.crt --private-key
 fileb://ecdsasigner.key
```

이 명령은 인증서에 대한 ARN을 표시합니다. OTA 업데이트 작업을 생성할 때 이 ARN이 필요합니다.

> **Note**
>
> 이 단계는 용 코드 서명을 사용하여 펌웨어 이미지에 서명한다는 가정하 AWS IoT 에 작성되었습니다. 용 코드 서명을 사용하는 AWS IoT 것이 좋지만 펌웨어 이미지에 수동으로 서명할 수 있습니다.

사용자 지정 하드웨어에 대한 코드 서명 인증서 생성

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

적절한 도구 세트를 사용하여 하드웨어에 대해 자체 서명된 인증서와 프라이빗 키를 생성합니다.

AWS Command Line Interface 를 사용하여 코드 서명 인증서, 프라이빗 키 및 인증서 체인을 로 가져옵니다 AWS Certificate Manager. 설치에 대한 자세한 내용은 설치를 AWS CLI참조하세요. AWS CLI

코드 서명 인증서를 생성한 후 AWS CLI 를 사용하여 ACM으로 가져올 수 있습니다.

```
aws acm import-certificate --certificate fileb://code-sign.crt --private-key fileb://
code-sign.key
```

이 명령의 출력에는 인증서에 대한 ARN이 표시됩니다. OTA 업데이트 작업을 생성할 때 이 ARN이 필요합니다.

ACM에서 특정 알고리즘 및 키 크기를 사용하려면 인증서가 필요합니다. 자세한 내용은 인증서를 가져오기 위한 사전 조건을 참조하십시오. ACM에 대한 자세한 내용은 AWS Certificate Manager으로 인증서 가져오기를 참조하세요.

코드 서명 인증서의 내용을 복사하여 나중에 다운로드할 FreeRTOS 코드의 일부인 `aws_ota_codesigner_certificate.h` 파일에 붙여넣고 서식을 지정해야 합니다.

에 대한 코드 서명에 대한 액세스 권한 부여 AWS IoT

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

프로덕션 환경에서 펌웨어 업데이트에 디지털 방식으로 서명하여 업데이트의 신뢰성과 무결성을 확인해야 합니다. 업데이트에 수동으로 서명하거나 용 코드 서명을 사용하여 코드에 서명 AWS IoT 할 수 있습니다. FreeRTOS용 코드 서명을 사용하려면 IAM 사용자 계정에 FreeRTOS용 코드 서명에 대한 액세스 권한을 부여해야 합니다.

액세스 권한을 제공하려면 사용자, 그룹 또는 역할에 권한을 추가하세요:

- 의 사용자 및 그룹 AWS IAM Identity Center:

  권한 세트를 생성합니다. AWS IAM Identity Center 사용 설명서의 권한 세트 생성의 지침을 따릅니다.

- 보안 인증 공급자를 통해 IAM에서 관리되는 사용자:

  ID 페더레이션을 위한 역할을 생성합니다. IAM 사용 설명서의 Create a role for a third-party identity provider (federation)의 지침을 따릅니다.

- IAM 사용자:
  - 사용자가 맡을 수 있는 역할을 생성합니다. IAM 사용 설명서에서 Create a role for an IAM user의 지침을 따릅니다.
  - (권장되지 않음)정책을 사용자에게 직접 연결하거나 사용자를 사용자 그룹에 추가합니다. IAM 사용 설명서에서 사용자(콘솔)에 권한 추가의 지침을 따르세요.

OTA 라이브러리와 함께 FreeRTOS 다운로드

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

GitHub에서 FreeRTOS를 복제하거나 다운로드할 수 있습니다. 자세한 내용은 README.md 파일을 참조하십시오.

OTA 데모 애플리케이션 설정 및 실행에 대한 자세한 내용은 OTA(Over-the-Air) 업데이트 데모 애플리케이션 단원을 참조하십시오.

MQTT를 사용한 OTA 업데이트 사전 조건

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

이 단원에서는 MQTT를 사용하여 무선 업데이트(OTA)를 수행하기 위한 일반적인 요구 사항에 대해 설명합니다.

액세스 권한을 제공하려면 사용자, 그룹 또는 역할에 권한을 추가하세요:

- 의 사용자 및 그룹 AWS IAM Identity Center:

  권한 세트를 생성합니다. AWS IAM Identity Center 사용 설명서의 권한 세트 생성의 지침을 따릅니다.
- 보안 인증 공급자를 통해 IAM에서 관리되는 사용자:

  ID 페더레이션을 위한 역할을 생성합니다. IAM 사용 설명서의 Create a role for a third-party identity provider (federation)의 지침을 따릅니다.
- IAM 사용자:
  - 사용자가 맡을 수 있는 역할을 생성합니다. IAM 사용 설명서에서 Create a role for an IAM user의 지침을 따릅니다.
  - (권장되지 않음)정책을 사용자에게 직접 연결하거나 사용자를 사용자 그룹에 추가합니다. IAM 사용 설명서에서 사용자(콘솔)에 권한 추가의 지침을 따르세요.

최소 요구 사항

- 디바이스 펌웨어에는 필요한 FreeRTOS 라이브러리(coreMQTT, OTA 에이전트 및 해당 종속성)가 포함되어 있어야 합니다.
- FreeRTOS 버전 1.4.0 이상이 필요합니다. 하지만 가능한 경우 최신 버전을 사용하는 것이 좋습니다.

구성

버전 201912.00부터 FreeRTOS OTA는 HTTP 또는 MQTT 프로토콜을 사용하여 펌웨어 업데이트 이미지를에서 디바이스 AWS IoT 로 전송할 수 있습니다. FreeRTOS에서 OTA 업데이트를 생성할 때 두 프로토콜을 모두 지정하면 각 디바이스에서 이미지를 전송하는 데 사용되는 프로토콜을 결정합니다. 자세한 정보는 HTTP를 사용한 OTA 업데이트 사전 조건을 참조하세요.

기본적으로 `aws_ota_agent_config.h`의 OTA 프로토콜은 MQTT 프로토콜을 사용하도록 구성되어 있습니다.

```
/**
 * @brief The protocol selected for OTA control operations.
 * This configuration parameter sets the default protocol for all the OTA control
```

```
 * operations like requesting OTA job, updating the job status etc.
 *
 * Note - Only MQTT is supported at this time for control operations.
 */
#define configENABLED_CONTROL_PROTOCOL       ( OTA_CONTROL_OVER_MQTT )
/**
 * @brief The protocol selected for OTA data operations.
 * This configuration parameter sets the protocols selected for the data operations
 * like requesting file blocks from the service.
 *
 * Note - Both MQTT and HTTP are supported for data transfer. This configuration
 parameter
 * can be set to the following -
 * Enable data over MQTT - ( OTA_DATA_OVER_MQTT )
 * Enable data over HTTP - ( OTA_DATA_OVER_HTTP)
 * Enable data over both MQTT & HTTP ( OTA_DATA_OVER_MQTT | OTA_DATA_OVER_HTTP )
 */
#define configENABLED_DATA_PROTOCOLS         ( OTA_DATA_OVER_MQTT )
 /**
  * @brief The preferred protocol selected for OTA data operations.
  *
  * Primary data protocol will be the protocol used for downloading files if more than
  * one protocol is selected while creating OTA job. Default primary data protocol is
 MQTT
  * and the following update here switches to HTTP as primary.
  *
  * Note - use OTA_DATA_OVER_HTTP for HTTP as primary data protocol.
  */
#define configOTA_PRIMARY_DATA_PROTOCOL     ( OTA_DATA_OVER_MQTT )
```

디바이스별 구성

없음.

메모리 사용량

MQTT를 데이터 전송에 사용하는 경우, 제어 작업과 데이터 작업 간에 공유되므로 MQTT 연결을 위한 추가 힙 메모리가 필요하지 않습니다.

디바이스 정책

MQTT를 사용하여 OTA 업데이트를 수신하는 각 디바이스는에 사물로 등록되어야 하며 AWS IoT 사물에는 여기에 나열된 것과 같은 연결된 정책이 있어야 합니다. `"Action"` 및 `"Resource"` 객체의 항목

에 대한 자세한 내용은 AWS IoT 핵심 정책 작업 및 AWS IoT 핵심 작업 리소스에서 확인할 수 있습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:Connect",
            "Resource": "arn:partition:iot:region:account:client/
${iot:Connection.Thing.ThingName}"
        },
        {
            "Effect": "Allow",
            "Action": "iot:Subscribe",
            "Resource": [
                "arn:partition:iot:region:account:topicfilter/$aws/things/
${iot:Connection.Thing.ThingName}/streams/*",
                "arn:partition:iot:region:account:topicfilter/$aws/things/
${iot:Connection.Thing.ThingName}/jobs/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish",
                "iot:Receive"
            ],
            "Resource": [
                "arn:partition:iot:region:account:topic/$aws/things/
${iot:Connection.Thing.ThingName}/streams/*",
                "arn:partition:iot:region:account:topic/$aws/things/
${iot:Connection.Thing.ThingName}/jobs/*"
            ]
        }
    ]
}
```

Notes

- `iot:Connect` 권한을 통해 디바이스는 MQTT를 AWS IoT 통해에 연결할 수 있습니다.

- AWS IoT 작업 주제에 대한 `iot:Subscribe` 및 `iot:Publish` 권한(`.../jobs/*`)을 사용하면 연결된 디바이스가 작업 알림 및 작업 문서를 수신하고 작업 실행의 완료 상태를 게시할 수 있습니다.
- AWS IoT OTA 스트림(`.../streams/*`) 주제에 대한 `iot:Subscribe` 및 `iot:Publish` 권한을 통해 연결된 디바이스는 OTA 업데이트 데이터를 가져올 수 있습니다 AWS IoT. MQTT를 통해 펌웨어 업데이트를 수행하려면 이러한 권한이 필요합니다.
- `iot:Receive` 권한을 통해 AWS IoT Core 는 연결된 디바이스에 해당 주제에 대한 메시지를 게시할 수 있습니다. MQTT 메시지를 전송할 때마다 이 권한을 확인합니다. 이 권한을 사용하면 주제를 현재 구독 중인 클라이언트에 대한 액세스 권한을 취소할 수 있습니다.

HTTP를 사용한 OTA 업데이트 사전 조건

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

이 단원에서는 HTTP를 사용하여 무선 업데이트(OTA)를 수행하기 위한 일반적인 요구 사항에 대해 설명합니다. 버전 201912.00부터 FreeRTOS OTA는 HTTP 또는 MQTT 프로토콜을 사용하여 펌웨어 업데이트 이미지를에서 디바이스 AWS IoT 로 전송할 수 있습니다.

> **Note**
>
> - 펌웨어 이미지를 전송하는 데 HTTP 프로토콜을 사용할 수도 있지만 작업 실행 알림, 작업 문서 및 실행 상태 업데이트 송수신 등 AWS IoT Core 와의 다른 상호 작용에서 coreMQTT 라이브러리를 사용하기 때문에 coreMQTT 라이브러리가 여전히 필요합니다.
> - OTA 업데이트 작업에 대해 MQTT 및 HTTP 프로토콜을 모두 지정하는 경우 각 디바이스의 OTA 에이전트 소프트웨어 설정에 따라 펌웨어 이미지를 전송하는 데 사용되는 프로토콜이 결정됩니다. OTA 에이전트를 기본 MQTT 프로토콜 메서드에서 HTTP 프로토콜로 변경하려면 장치의 FreeRTOS 소스 코드를 컴파일하는 데 사용되는 헤더 파일을 수정할 수 있습니다.

최소 요구 사항

- 디바이스 펌웨어에는 필요한 FreeRTOS 라이브러리(coreMQTT, HTTP, OTA 에이전트 및 해당 종속성)가 포함되어 있어야 합니다.

- OTA 프로토콜의 구성을 변경하여 HTTP를 통한 OTA 데이터 전송을 활성화하려면 FreeRTOS 버전 201912.00 이상이 필요합니다.

구성

\vendors\boards*board*\aws_demos\config_files\aws_ota_agent_config.h 파일에서 다음과 같은 OTA 프로토콜 구성을 참조하십시오.

```
/**
 * @brief The protocol selected for OTA control operations.
 * This configuration parameter sets the default protocol for all the OTA control
 * operations like requesting an OTA job, updating the job status, and so on.
 *
 * Note - Only MQTT is supported at this time for control operations.
 */
#define configENABLED_CONTROL_PROTOCOL       ( OTA_CONTROL_OVER_MQTT )
/**
 * @brief The protocol selected for OTA data operations.
 * This configuration parameter sets the protocols selected for the data operations
 * like requesting file blocks from the service.
 *
 * Note - Both MQTT and HTTP are supported for data transfer. This configuration
 parameter
 * can be set to the following -
 * Enable data over MQTT - ( OTA_DATA_OVER_MQTT )
 * Enable data over HTTP - ( OTA_DATA_OVER_HTTP)
 * Enable data over both MQTT & HTTP ( OTA_DATA_OVER_MQTT | OTA_DATA_OVER_HTTP )
 */
#define configENABLED_DATA_PROTOCOLS         ( OTA_DATA_OVER_MQTT )
 /**
  * @brief The preferred protocol selected for OTA data operations.
  *
  * Primary data protocol will be the protocol used for downloading files if more than
  * one protocol is selected while creating OTA job. Default primary data protocol is
 MQTT
  * and the following update here switches to HTTP as primary.
  *
  * Note - use OTA_DATA_OVER_HTTP for HTTP as primary data protocol.
  */
#define configOTA_PRIMARY_DATA_PROTOCOL     ( OTA_DATA_OVER_MQTT )
```

HTTP를 통한 OTA 데이터 전송을 활성화하려면

1. `configENABLED_DATA_PROTOCOLS`를 `OTA_DATA_OVER_HTTP`로 변경합니다.
2. OTA가 업데이트되면 MQTT 또는 HTTP 프로토콜을 사용할 수 있도록 두 프로토콜을 모두 지정할 수 있습니다. `configOTA_PRIMARY_DATA_PROTOCOL`을 `OTA_DATA_OVER_HTTP`로 변경하여 디바이스에서 사용하는 기본 프로토콜을 HTTP로 설정할 수 있습니다.

> **Note**
>
> HTTP는 OTA 데이터 작업에만 지원됩니다. 제어 작업의 경우 MQTT를 사용해야 합니다.

디바이스별 구성

ESP32

RAM 양의 제한으로 인해 HTTP를 OTA 데이터 프로토콜로 활성화할 때는 BLE를 꺼야 합니다. `vendors/espressif/boards/esp32/aws_demos/config_files/aws_iot_network_config.h` 파일에서 `AWSIOT_NETWORK_TYPE_WIFI`를 `configENABLED_NETWORKS` 전용으로 변경합니다.

```
/**
     * @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 )
```

메모리 사용량

MQTT를 데이터 전송에 사용하는 경우 제어 및 데이터 작업 간에 공유되므로 MQTT 연결에 추가 힙 메모리가 필요하지 않습니다. 그러나 HTTP를 통해 데이터를 전송하려면 추가 힙 메모리가 필요합니다. 다음은 FreeRTOS `xPortGetFreeHeapSize` API를 사용하여 계산된 모든 지원되는 플랫폼의 힙 메모리 사용량 데이터입니다. OTA 라이브러리를 사용하기에 충분한 RAM이 있는지 확인해야 합니다.

Texas Instruments CC3220SF-LAUNCHXL

제어 작업(MQTT): 12KB

데이터 작업(HTTP): 10KB

> **Note**
>
> TI는 하드웨어에서 SSL을 수행하기 때문에 RAM을 훨씬 적게 사용하므로 mbedtls 라이브러리를 사용하지 않습니다.

Microchip Curiosity PIC32MZEF

제어 작업(MQTT): 65KB

데이터 작업(HTTP): 43KB

Espressif ESP32

제어 작업(MQTT): 65KB

데이터 작업(HTTP): 45KB

> **Note**
>
> ESP32의 BLE는 약 87KB의 RAM을 사용합니다. 위의 디바이스별 구성에서 언급한 모든 내용을 활성화할 수 있는 RAM이 충분하지 않습니다.

Windows 시뮬레이터

제어 작업(MQTT): 82KB

데이터 작업(HTTP): 63KB

Nordic nrf52840-dk

HTTP는 지원되지 않습니다.

디바이스 정책

이 정책을 사용하면 OTA 업데이트에 MQTT 또는 HTTP를 사용할 수 있습니다.

HTTP를 사용하여 OTA 업데이트를 수신하는 각 장치는 AWS IoT 에 항목으로 등록되어야 하며, 여기에 나열된 것과 같은 연결 정책이 있어야 합니다. `"Action"` 및 `"Resource"` 객체의 항목에 대한 자세한 내용은 AWS IoT 핵심 정책 작업 및 AWS IoT 핵심 작업 리소스에서 확인할 수 있습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:Connect",
            "Resource": "arn:partition:iot:region:account:client/
${iot:Connection.Thing.ThingName}"
        },
        {
            "Effect": "Allow",
            "Action": "iot:Subscribe",
            "Resource": [
                "arn:partition:iot:region:account:topicfilter/$aws/things/
${iot:Connection.Thing.ThingName}/jobs/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish",
                "iot:Receive"
            ],
            "Resource": [
                "arn:partition:iot:region:account:topic/$aws/things/
${iot:Connection.Thing.ThingName}/jobs/*"
            ]
        }
    ]
```

```
}
```

Notes

- `iot:Connect` 권한을 사용하면 MQTT를 통해 디바이스를 AWS IoT 에 연결할 수 있습니다.
- AWS IoT 작업 주제에 대한 `iot:Subscribe` 및 `iot:Publish` 권한(`.../jobs/*`)을 사용하면 연결된 디바이스가 작업 알림 및 작업 문서를 수신하고 작업 실행의 완료 상태를 게시할 수 있습니다.
- `iot:Receive` 권한을 통해 AWS IoT Core 는 해당 주제에 대한 메시지를 현재 연결된 디바이스에 게시할 수 있습니다. MQTT 메시지를 전송할 때마다 이 권한을 확인합니다. 이 권한을 사용하면 주제를 현재 구독 중인 클라이언트에 대한 액세스 권한을 취소할 수 있습니다.

OTA 자습서

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

이 단원에는 OTA 업데이트를 사용하여 FreeRTOS를 실행하는 디바이스에서 펌웨어를 업데이트하는 방법을 설명하는 자습서가 포함되어 있습니다. 펌웨어 이미지 외에도 OTA 업데이트를 사용하여 모든 유형의 파일을 AWS IoT에 연결된 디바이스로 보낼 수 있습니다.

AWS IoT 콘솔 또는를 사용하여 OTA 업데이트를 AWS CLI 생성할 수 있습니다. 콘솔을 사용하면 많은 작업이 자동으로 수행되므로 OTA를 가장 쉽게 시작할 수 있습니다. AWS CLI 는 OTA 업데이트 작업을 자동화하거나, 많은 수의 디바이스로 작업하거나, FreeRTOS에 적합하지 않은 디바이스를 사용할 때 유용합니다. FreeRTOS용 디바이스 검증에 대한 자세한 내용은 FreeRTOS 파트너 웹 사이트를 참조하세요.

OTA 업데이트를 생성하려면

1. 초기 펌웨어 버전을 하나 이상의 디바이스에 배포합니다.
2. 펌웨어가 올바르게 실행되는지 확인합니다.
3. 펌웨어를 업데이트해야 하는 경우 코드를 변경하고 새 이미지를 빌드합니다.
4. 펌웨어에 수동으로 서명할 경우 서명 후 서명된 펌웨어 이미지를 Amazon S3 버킷에 업로드합니다. 에 코드 서명을 사용하는 경우 서명되지 않은 펌웨어 이미지를 Amazon S3 버킷에 AWS IoT업로드합니다.

5. OTA 업데이트를 생성합니다.

OTA 업데이트를 생성할 때 이미지 전송 프로토콜(MQTT 또는 HTTP)을 지정하거나 둘 다 지정하여 디바이스가 선택할 수 있도록 합니다. 디바이스의 FreeRTOS OTA 에이전트가 업데이트된 펌웨어 이미지를 수신하고 새 이미지의 디지털 서명, 체크섬 및 버전 번호를 확인합니다. 펌웨어 업데이트가 확인된 후 디바이스가 재설정되고 나면 애플리케이션 정의 로직에 따라 업데이트를 커밋합니다. 디바이스에서 FreeRTOS를 실행하고 있지 않은 경우 디바이스에서 실행되는 OTA 에이전트를 구현해야 합니다.

초기 펌웨어 설치

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

펌웨어를 업데이트하려면 OTA 에이전트 라이브러리를 사용하여 OTA 업데이트 작업을 수신하는 초기 펌웨어 버전을 설치해야 합니다. FreeRTOS를 실행하지 않는 경우 이 단계를 건너뜁니다. 대신 OTA 에이전트 구현을 디바이스에 복사해야 합니다.

주제

- Texas Instruments CC3220SF-LAUNCHXL에 초기 펌웨어 버전 설치
- Microchip Curiosity PIC32MZEF에 초기 펌웨어 버전 설치
- Espressif ESP32에 초기 펌웨어 버전 설치
- Nordic nRF52840 DK에 초기 펌웨어 버전 설치
- Windows 시뮬레이터의 초기 펌웨어
- 사용자 지정 보드에 초기 펌웨어 버전 설치

Texas Instruments CC3220SF-LAUNCHXL에 초기 펌웨어 버전 설치

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

이 단계는 Texas Instruments CC3220SF-LAUNCHXL에 대한 FreeRTOS OTA 데모 다운로드, 빌드, 플래시 및 실행에 설명된 대로 aws_demos 프로젝트를 이미 빌드했다는 가정하에 작성되었습니다.

1. Texas Instruments CC3220SF-LAUNCHXL에서 SOP 점퍼를 중간 핀 집합(위치 = 1)에 두고 보드를 재설정합니다.
2. TI Uniflash 도구를 다운로드한 후 설치합니다.
3. Uniflash를 시작합니다. 구성 목록에서 CC3220SF-LAUNCHXL을 선택한 후 Start Image Creator(이미지 생성자 시작)를 선택합니다.
4. 새 프로젝트를 선택합니다.
5. Start new project(새 프로젝트 시작) 페이지에서 프로젝트의 이름을 입력합니다. Device Type(디바이스 유형)에서 CC3220SF를 선택합니다. Device Mode(디바이스 모드)에서 Develop(개발)을 선택합니다. 프로젝트 생성을 선택합니다.
6. 터미널 에뮬레이터를 연결 해제합니다.
7. Uniflash 애플리케이션 창의 오른쪽에서 연결을 선택합니다.
8. 고급, 파일에서 사용자 파일을 선택합니다.
9. 파일 선택기 창에서 파일 추가 아이콁



을 선택합니다.
10. /Applications/Ti/simplelink_cc32xx_sdk_*version*/tools/cc32xx_tools/certificate-playground 디렉터리로 이동한 후 dummy-root-ca-cert, 열기, 쓰기를 차례로 선택합니다.
11. 파일 선택기 창에서 파일 추가 아이콁



을 선택합니다.
12. 코드 서명 인증서 및 프라이빗 키를 생성한 작업 디렉터리로 이동하여 tisigner.crt.der, 열기, 쓰기를 차례로 선택합니다.
13. 작업 드롭다운 목록에서 Select MCU Image(MCU 이미지 선택)을 선택한 후 찾아보기를 선택하여 디바이스에 쓰기에 사용할 펌웨어 이미지(aws_demos.bin)를 선택합니다. 이 파일은 *freertos*/vendors/ti/boards/cc3220_launchpad/aws_demos/ccs/Debug 디렉터리에 위치합니다. Open을 선택합니다.
    a. 파일 대화 상자에서 파일 이름이 mcuflashimg.bin으로 설정되어 있는지 확인합니다.
    b. 공급업체 확인란을 선택합니다.
    c. File Token(파일 토큰)에 **1952007250**을 입력합니다.

d. Private Key File Name(프라이빗 키 파일 이름)에서 찾아보기를 선택한 후 코드 서명 인증서와 프라이빗 키를 생성한 작업 디렉터리에서 `tisigner.key`를 선택합니다.

e. Certification File Name(인증서 파일 이름)에서 `tisigner.crt.der`을 선택합니다.

f. 쓰기를 선택합니다.

14. 왼쪽 창의 파일에서 서비스 팩을 선택합니다.

15. Service Pack File Name(서비스 팩 파일 이름)에서 찾아보기를 선택하고 `simplelink_cc32x_sdk_version/tools/cc32xx_tools/servicepack-cc3x20`으로 이동하여 `sp_3.7.0.1_2.0.0.0_2.2.0.6.bin`을 선택한 후 열기를 선택합니다.

16. 왼쪽 창의 파일에서 Trusted Root-Certificate Catalog(신뢰할 수 있는 루트 인증서 카탈로그)를 선택합니다.

17. Use default Trusted Root-Certificate Catalog(기본 신뢰할 수 있는 루트 인증서 카탈로그 사용) 확인란의 선택을 취소합니다.

18. Source File(소스 파일)에서 찾아보기를 선택하고 simplelink_cc32xx_sdk_***version***/tools/cc32xx_tools/certificate-playground/certcatalogPlayGround20160911.lst를 선택한 후 열기를 선택합니다.

19. Signature Source File(서명 소스 파일)에서 찾아보기를 선택하고 simplelink_cc32xx_sdk_***version***/tools/cc32xx_tools/certificate-playground/certcatalogPlayGround20160911.lst.signed_3220.bin을 선택한 후 열기를 선택합니다.

20. 

    버튼을 선택하여 프로젝트를 저장합니다.

21. 

    버튼을 선택합니다.

22. Program Image(Create and Program)(프로그램 이미지[생성 및 프로그램])을 선택합니다.

23. 프로그래밍 프로세스가 완료되면 SOP 점퍼를 첫 번째 핀 집합(위치 = 0)에 두고 보드를 재설정한 후 터미널 에뮬레이터를 다시 연결하여 Code Composer Studio에서 출력을 디버깅할 때와 출력이 동일한지 확인합니다. 터미널 출력의 애플리케이션 버전 번호를 메모해 두십시오. 나중에 이 버전 번호를 사용하여 OTA 업데이트에 의해 펌웨어가 업데이트되었는지 확인합니다.

    터미널에 다음과 같은 출력이 표시됩니다.

```
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
```

Microchip Curiosity PIC32MZEF에 초기 펌웨어 버전 설치

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

이 단계는 Microchip Curiosity PIC32MZEF에 대한 FreeRTOS OTA 데모 다운로드, 빌드, 플래시 및 실행에 설명된 대로 aws_demos 프로젝트를 이미 빌드했다는 가정하에 작성되었습니다.

데모 애플리케이션을 보드에 번인하려면

1. aws_demos 프로젝트를 다시 빌드하고 오류 없이 컴파일되는지 확인합니다.

2. 도구 모음에서

   아이콘을 선택합니다.

3. 프로그래밍 프로세스가 완료된 후 ICD 4 디버거를 연결 해제하고 보드를 재설정합니다. 터미널 에뮬레이터를 다시 연결하여 MPLAB X IDE에서 데모를 디버깅할 때와 출력이 동일한지 확인합니다.

   터미널에 다음과 비슷한 출력이 표시됩니다.

```
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
```

다음 절차에서는 참조 부트로더와 암호화 서명이 있는 애플리케이션으로 구성된 통합 16진수 파일 또는 출고 시 이미지를 생성합니다. 부트로더는 부팅 시 애플리케이션의 암호화 서명을 확인하고 OTA 업데이트를 지원합니다.

출고 시 이미지를 빌드 후 플래시하려면

1. Source Forge(소스 Forge)에서 설치된 SRecord 도구가 있는지 확인합니다. `srec_cat` 및 `srec_info` 프로그램이 포함된 디렉터리가 시스템 경로에 있는지 확인합니다.
2. 출고 시 이미지에 대한 OTA 시퀀스 번호와 애플리케이션 버전을 업데이트합니다.
3. `aws_demos` 프로젝트를 빌드합니다.
4. `factory_image_generator.py` 스크립트를 실행하여 출고 시 이미지를 생성합니다.

```
factory_image_generator.py -b mplab.production.bin -p MCHP-Curiosity-PIC32MZEF -k
 private_key.pem  -x aws_bootloader.X.production.hex
```

이 명령은 다음 파라미터를 사용합니다.

- `mplab.production.bin`: 애플리케이션 이진 파일
- `MCHP-Curiosity-PIC32MZEF`: 플랫폼 이름
- `private_key.pem`: 코드 서명 프라이빗 키
- `aws_bootloader.X.production.hex`: 부트로더 16진수 파일

`aws_demos` 프로젝트를 빌드할 때 애플리케이션 이진 이미지와 부트로더 16진수 파일이 프로세스의 일부로 빌드됩니다. `vendors/microchip/boards/curiosity_pic32mzef/aws_demos/` 디렉터리 아래의 각 프로젝트에는 이러한 파일이 포함된 `dist/pic32mz_ef_curiosity/production/` 디렉터리가 있습니다. 생성된 통합 16진수 파일의 이름은 `mplab.production.factory.unified.hex`입니다.

5. MPLab IPE 도구를 사용하여 생성된 16진수 파일을 디바이스에 프로그래밍합니다.
6. 이미지가 업로드될 때 보드의 UART 출력을 보고 출고 시 이미지가 작동하는지 확인할 수 있습니다. 모든 것이 올바르게 설정되었으면 이미지 부트가 성공적으로 표시됩니다.

```
[prvValidateImage] Validating image at Bank : 0
[prvValidateImage] Valid magic code at: 0xbd000000
[prvValidateImage] Valid image flags: 0xfc at: 0xbd000000
[prvValidateImage] Addresses are valid.
[prvValidateImage] Crypto signature is valid.
[...]
[prvBOOT_ValidateImages] Booting image with sequence number 1 at 0xbd000000
```

7. 인증서가 잘못 구성되었거나 OTA 이미지가 제대로 서명되지 않은 경우 다음과 같은 메시지가 표시된 후 칩의 부트로더가 잘못된 업데이트를 지웁니다. 코드 서명 인증서가 일치하는지 확인하고 이전 단계를 주의해서 검토합니다.

```
[prvValidateImage] Validating image at Bank : 0
[prvValidateImage] Valid magic code at: 0xbd000000
[prvValidateImage] Valid image flags: 0xfc at: 0xbd000000
```

```
[prvValidateImage] Addresses are valid.
[prvValidateImage] Crypto signature is not valid.
[prvBOOT_ValidateImages] Validation failed for image at 0xbd000000
[BOOT_FLASH_EraseBank] Bank erased at : 0xbd000000
```

Espressif ESP32에 초기 펌웨어 버전 설치

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

이 안내서는 Espressif ESP32-DevKitC 및 ESP-WROVER-KIT 시작하기 및 무선 업데이트(OTA) 사전 조건의 단계를 이미 수행했다는 가정하에 작성되었습니다. OTA 업데이트를 시도하기 전에 FreeRTOS 시작하기에서 설명한 MQTT 데모 프로젝트를 실행하여 보드와 도구 체인이 올바르게 설정되어 있는지 확인할 수 있습니다.

초기 출고 시 이미지를 보드에 플래시하려면

1. *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h를 열고 `#define CONFIG_MQTT_DEMO_ENABLED`를 주석으로 처리한 다음 `CONFIG_OTA_UPDATE_DEMO_ENABLED`를 정의합니다.
2. OTA 업데이트 사전 조건에서 생성한 SHA-256/ECDSA PEM 형식 코드 서명 인증서를 demos/include/aws_ota_codesigner_certificate.h에 복사합니다. 다음 방식으로 형식을 지정해야 합니다.

   ```
   static const char signingcredentialSIGNING_CERTIFICATE_PEM[] = "-----BEGIN
    CERTIFICATE-----\n"
   "...base64 data...\n"
   "-----END CERTIFICATE-----\n";
   ```

3. OTA 업데이트 데모를 선택한 상태에서 ESP32 시작하기에 요약된 것과 동일한 단계에 따라 이미지를 빌드 후 플래시합니다. 프로젝트를 이미 빌드하여 플래시한 경우 먼저 `make clean`을 실행해야 할 수 있습니다. `make flash monitor`를 실행하면 다음과 유사한 내용이 표시됩니다. 데모 애플리케이션은 한 번에 여러 작업을 실행하므로 일부 메시지의 순서가 다를 수 있습니다.

```
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. 이제 ESP32 보드에서 OTA 업데이트를 수신합니다. ESP-IDF 모니터는 `make flash monitor` 명령에 의해 시작됩니다. Ctrl+]를 눌러 종료할 수 있습니다. 선호하는 TTY 터미널 프로그램(예: PuTTY, Tera Term, GNU Screen)을 사용하여 보드의 직렬 출력을 수신할 수도 있습니다. 보드의 직렬 포트에 연결할 때 재부팅될 수 있습니다.

Nordic nRF52840 DK에 초기 펌웨어 버전 설치

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

이 안내서는 Nordic nRF52840-DK 시작하기 및 무선 업데이트(OTA) 사전 조건의 단계를 이미 수행했다는 가정하에 작성되었습니다. OTA 업데이트를 시도하기 전에 FreeRTOS 시작하기에서 설명한 MQTT 데모 프로젝트를 실행하여 보드와 도구 체인이 올바르게 설정되어 있는지 확인할 수 있습니다.

초기 출고 시 이미지를 보드에 플래시하려면

1. *freertos*/vendors/nordic/boards/nrf52840-dk/aws_demos/config_files/aws_demo_config.h를 엽니다.
2. #define CONFIG_MQTT_DEMO_ENABLED를 #define democonfigOTA_UPDATE_DEMO_ENABLED로 바꿉니다.
3. OTA 업데이트 데모를 선택한 상태에서 Nordic nRF52840-DK 시작하기에 요약된 것과 동일한 단계에 따라 이미지를 빌드 후 플래시합니다.

   다음과 유사한 출력 화면이 표시되어야 합니다.

```
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
```

   이제 보드에서 OTA 업데이트를 수신합니다.

Windows 시뮬레이터의 초기 펌웨어

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

Windows 시뮬레이터를 사용하는 경우 초기 펌웨어 버전을 플래시할 필요가 없습니다. Windows 시뮬레이터는 `aws_demos` 애플리케이션의 일부이며, 펌웨어를 포함합니다.

사용자 지정 보드에 초기 펌웨어 버전 설치

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

IDE를 사용하여 `aws_demos` 프로젝트를 빌드합니다. 이때 OTA 라이브러리를 포함해야 합니다. FreeRTOS 소스 코드의 구조에 대한 자세한 내용은 FreeRTOS 데모 단원을 참조하세요.

FreeRTOS 프로젝트 또는 디바이스에 코드 서명 인증서, 프라이빗 키 및 인증서 신뢰 체인을 포함해야 합니다.

적절한 도구를 사용하여 애플리케이션을 보드에 번인하고 올바르게 실행되는지 확인합니다.

펌웨어 버전 업데이트

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

FreeRTOS에 포함된 OTA 에이전트에서 업데이트 버전을 확인한 후 기존 펌웨어 버전보다 최신 버전이 있는 경우에만 해당 버전을 설치합니다. 다음 단계에서는 OTA 데모 애플리케이션의 펌웨어 버전을 높이는 방법을 보여 줍니다.

1. IDE에서 aws_demos 프로젝트를 엽니다.
2. demos/include/aws_application_version.h를 열고 APP_VERSION_BUILD 토큰 값을 늘립니다.
3. Microchip Curiosity PIC32MZEF를 사용 중인 경우 vendors/microchip/boards/curiosity_pic32mzef/bootloader/bootloader/utility/user-config/ota-descriptor.config에서 OTA 시퀀스 번호를 늘립니다. 새 OTA 이미지가 생성될 때마다 OTA 시퀀스 번호가 증가해야 합니다.
4. 프로젝트를 다시 빌드합니다.

업데이트를 저장할 Amazon S3 버킷 생성에 설명된 대로 펌웨어 업데이트를 생성된 Amazon S3 버킷에 복사해야 합니다. Amazon S3에 복사할 파일 이름은 사용 중인 하드웨어 플랫폼에 따라 다릅니다.

- Texas Instruments CC3220SF-LAUNCHXL: vendors/ti/boards/cc3220_launchpad/aws_demos/ccs/debug/aws_demos.bin
- Microchip Curiosity PIC32MZEF: vendors/microchip/boards/curiosity_pic32mzef/aws_demos/mplab/dist/pic32mz_ef_curiosity/production/mplab.production.ota.bin
- Espressif ESP32: vendors/espressif/boards/esp32/aws_demos/make/build/aws_demos.bin

OTA 업데이트 생성(AWS IoT 콘솔)

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

1. AWS IoT 콘솔의 탐색 창에서 관리를 선택한 다음 작업을 선택합니다.
2. 생성(Create)을 선택합니다.
3. FreeRTOS 무선 업데이트(OTA) 작업 생성에서 OTA 업데이트 작업 생성을 선택합니다.
4. OTA 업데이트를 단일 디바이스 또는 디바이스 그룹에 배포할 수 있습니다. 업데이트할 디바이스 선택에서 선택을 선택합니다. 단일 디바이스를 업데이트하려면 사물 탭을 선택합니다. 디바이스 그룹을 업데이트하려면 사물 그룹 탭을 선택합니다.

5. 단일 디바이스를 업데이트할 경우 디바이스와 연결된 IoT 사물 옆에 있는 확인란을 선택합니다. 디바이스 그룹을 업데이트할 경우 디바이스와 연결된 사물 그룹 옆에 있는 확인란을 선택합니다. 다음을 선택합니다.
6. Select the protocol for firmware image transfer(펌웨어 이미지 전송을 위한 프로토콜 선택)에서 HTTP, MQTT를 선택하거나 둘 다 선택하여 각 디바이스가 사용할 프로토콜을 결정할 수 있도록 합니다.
7. 펌웨어 이미지 선택 및 서명에서 나를 위해 새 펌웨어 이미지에 서명을 선택합니다.
8. Code signing profile(코드 서명 프로필)에서 생성을 선택합니다.
9. Create a code signing profile(코드 서명 프로필 생성)에 코드 서명 프로필의 이름을 입력합니다.

    a. 디바이스 하드웨어 플랫폼에서 하드웨어 플랫폼을 선택합니다.

    > **Note**
    >
    > FreeRTOS에 적격한 하드웨어 플랫폼만 이 목록에 표시됩니다. 비적격 플랫폼을 테스트하는 경우 서명에 ECDSA P-256 SHA-256 ciphersuite를 사용하면 Windows Simulator 코드 서명 프로파일을 선택하여 호환되는 서명을 생성할 수 있습니다. 정규화되지 않은 플랫폼을 사용 중이고 ECDSA P-256 SHA-256 이외의 ciphersuite를 사용하여 서명하는 경우 용 코드 서명을 사용하거나 펌웨어 업데이트에 직접 서명 AWS IoT할 수 있습니다. 자세한 내용은 펌웨어 업데이트에 디지털 방식으로 서명 단원을 참조하십시오.

    b. 코드 서명 인증서에서 선택을 선택하여 이전에 가져온 인증서를 선택하거나 가져오기를 선택하여 새 인증서를 가져옵니다.

    c. 디바이스에 있는 코드 서명 인증서의 경로 이름에 디바이스에 있는 코드 서명 인증서에 대한 정규화된 경로 이름을 입력합니다. 인증서 위치는 플랫폼마다 다릅니다. 초기 펌웨어 설치의 지침을 수행할 때 코드 서명 인증서를 배치한 위치입니다.

    > **Important**
    >
    > Texas Instruments CC3220SF-LAUNCHXL에서 코드 서명 인증서가 파일 시스템의 루트에 있는 경우 파일 이름 앞에 슬래시(/)를 포함하지 마십시오. 그렇지 않으면 인증 중에 `file not found` 오류가 발생하고 OTA 업데이트가 실패합니다.

10. S3에서 펌웨어 이미지 선택 또는 업로드에서 선택을 선택합니다. Amazon S3 버킷 목록이 표시됩니다. 펌웨어 업데이트가 포함된 버킷을 선택한 후 버킷에서 펌웨어 업데이트를 선택합니다.

> **Note**
>
> Microchip Curiosity PIC32MZEF 데모 프로젝트에서는 기본 이름 `mplab.production.bin` 및 `mplab.production.ota.bin`으로 두 이진 이미지를 생성합니다. OTA 업데이트를 위해 이미지를 업로드할 때 두 번째 파일을 사용합니다.

11. 디바이스에 있는 펌웨어 이미지의 경로 이름에 OTA 작업이 펌웨어 이미지를 복사할 디바이스의 위치에 대한 정규화된 경로 이름을 입력합니다. 이 위치는 플랫폼마다 다릅니다.

> **Important**
>
> Texas Instruments CC3220SF-LAUNCHXL에서는 보안 제한으로 인해 펌웨어 이미지 경로 이름이 `/sys/mcuflashimg.bin`이어야 합니다.

12. 파일 유형에서 0~255 범위의 정수 값을 입력합니다. 입력한 파일 유형은 MCU로 전달되는 작업 문서에 추가됩니다. MCU 펌웨어/소프트웨어 개발자는 이 값으로 수행할 작업에 대한 완전한 소유권을 갖습니다. 가능한 시나리오로는 기본 프로세서와 독립적으로 펌웨어를 업데이트할 수 있는 보조 프로세서가 있는 MCU가 있습니다. 디바이스가 OTA 업데이트 작업을 받으면 파일 유형을 사용하여 업데이트가 필요한 프로세서를 식별할 수 있습니다.
13. OTA 업데이트 작업의 IAM 역할에서 OTA 업데이트 서비스 역할 생성의 지침에 따라 역할을 선택합니다.
14. 다음을 선택합니다.
15. OTA 업데이트 작업에 대한 ID와 설명을 입력합니다.
16. 작업 유형에서 작업이 선택한 디바이스/그룹에 배포된 후 완료됩니다(스냅샷)를 선택합니다.
17. 작업에 적합한 선택적 구성(작업 실행 롤아웃, 작업 중단, 작업 실행 제한 시간 및 태그)을 선택합니다.
18. 생성(Create)을 선택합니다.

이전에 서명된 펌웨어 이미지를 사용하려면

1. 펌웨어 이미지 선택 및 서명에서 이전에 서명된 펌웨어 이미지 선택을 선택합니다.
2. 디바이스에 있는 펌웨어 이미지의 경로 이름에 OTA 작업이 펌웨어 이미지를 복사할 디바이스의 위치에 대한 정규화된 경로 이름을 입력합니다. 이 위치는 플랫폼마다 다릅니다.
3. 이전 코드 서명 작업에서 선택을 선택한 후 OTA 업데이트에 사용 중인 펌웨어 이미지에 서명하는 데 사용된 이전 코드 서명 작업을 선택합니다.

서명된 사용자 지정 펌웨어 이미지 사용

1. 펌웨어 이미지 선택 및 서명에서 서명된 내 사용자 지정 펌웨어 이미지 사용을 선택합니다.
2. 디바이스에 있는 코드 서명 인증서의 경로 이름에 디바이스에 있는 코드 서명 인증서에 대한 정규화된 경로 이름을 입력합니다. 이 경로 이름은 플랫폼에 따라 다릅니다.
3. 디바이스에 있는 펌웨어 이미지의 경로 이름에 OTA 작업이 펌웨어 이미지를 복사할 디바이스의 위치에 대한 정규화된 경로 이름을 입력합니다. 이 위치는 플랫폼마다 다릅니다.
4. 서명 아래에 PEM 형식 서명을 붙여 넣습니다.
5. 원래 해시 알고리즘에서 파일 서명을 생성할 때 사용한 해시 알고리즘을 선택합니다.
6. 원래 암호화 알고리즘에서 파일 서명을 생성할 때 사용한 알고리즘을 선택합니다.
7. Amazon S3에서 펌웨어 이미지 선택에서 Amazon S3 버킷과 Amazon S3 버킷에 있는 서명된 펌웨어 이미지를 선택합니다.

코드 서명 정보를 지정한 후 OTA 업데이트 작업 유형, 서비스 역할 및 업데이트 ID를 지정합니다.

> **Note**
>
> OTA 업데이트에 대한 작업 ID에 개인 식별 정보(OTA)를 사용하지 마십시오. 개인 식별 정보의 예는 다음과 같습니다.
>
> - 이름
> - IP 주소
> - 이메일 주소
> - 위치
> - 은행 세부 정보
> - 의료 정보

1. 작업 유형에서 작업이 선택한 디바이스/그룹에 배포된 후 완료됩니다(스냅샷)를 선택합니다.
2. OTA 업데이트 작업의 IAM 역할에서 OTA 서비스 역할을 선택합니다.
3. 작업에 대한 영숫자 ID를 입력한 후 생성을 선택합니다.

작업이 AWS IoT 콘솔에 진행 중 상태로 표시됩니다.

> **Note**
>
> - AWS IoT 콘솔은 작업 상태를 자동으로 업데이트하지 않습니다. 브라우저를 새로 고쳐 업데이트를 확인합니다.

직렬 UART 터미널을 디바이스에 연결합니다. 디바이스에서 업데이트된 펌웨어를 다운로드하고 있다고 나타내는 출력이 표시됩니다.

디바이스가 업데이트된 펌웨어를 다운로드하고 다시 시작된 이후에 펌웨어를 설치합니다. UART 터미널에서 무슨 일이 일어나고 있는지 알 수 있습니다.

콘솔을 사용하여 OTA 업데이트를 생성하는 방법을 보여 주는 자습서는 OTA(Over-the-Air) 업데이트 데모 애플리케이션 단원을 참조하십시오.

를 사용하여 OTA 업데이트 생성 AWS CLI

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

AWS CLI 를 사용하여 OTA 업데이트를 생성할 때 다음을 수행합니다.

1. 펌웨어 이미지에 디지털 방식으로 서명합니다.
2. 디지털 방식으로 서명된 펌웨어 이미지의 스트림을 생성합니다.
3. OTA 업데이트 작업을 시작합니다.

펌웨어 업데이트에 디지털 방식으로 서명

AWS CLI 를 사용하여 OTA 업데이트를 수행할 때에 대한 코드 서명을 사용하거나 펌웨어 업데이트에 직접 서명 AWS IoT할 수 있습니다. Code Signing for에서 지원하는 암호화 서명 및 해싱 알고리즘 목록은 SigningConfigurationOverrides를 AWS IoT참조하세요. SigningConfigurationOverrides Code Signing for에서 지원하지 않는 암호화 알고리즘을 사용하려면 펌웨어 바이너리에 서명한 후 Amazon S3에 업로드해야 AWS IoT합니다.

용 코드 서명을 사용하여 펌웨어 이미지에 서명 AWS IoT

Code Signing for를 사용하여 펌웨어 이미지에 서명하려면 AWS SDKs 또는 명령줄 도구 중 하나를 사용할 AWS IoT수 있습니다. 에 대한 코드 서명에 대한 자세한 내용은에 대한 코드 서명을 AWS IoT참조하세요. AWS IoT

코드 서명 도구를 설치하고 구성한 후 서명되지 않은 펌웨어 이미지를 Amazon S3 버킷에 복사하고 다음 AWS CLI 명령을 사용하여 코드 서명 작업을 시작합니다. put-signing-profile 명령은 재사용 가능한 코드 서명 프로필을 생성합니다. start-signing-job 명령은 서명 작업을 시작합니다.

```
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* 및 *your-destination-bucket-name*은 동일한 Amazon S3 버킷일 수 있습니다.

다음은 put-signing-profile 및 start-signing-job 명령의 파라미터입니다.

`source`

S3 버킷에서 서명되지 않은 펌웨어 위치를 지정합니다.

- `bucketName`: S3 버킷의 이름
- `key`: S3 버킷의 펌웨어 키(파일 이름)
- `version`: S3 버킷에 있는 펌웨어의 S3 버전 펌웨어 버전과 다릅니다. Amazon S3 콘솔로 이동한 후 버킷을 선택하고 페이지 상단의 버전 옆에 있는 표시를 선택하여 찾을 수 있습니다.

`destination`

S3 버킷의 서명된 펌웨어가 복사될 디바이스의 대상. 이 파라미터의 형식은 `source` 파라미터의 형식과 동일합니다.

`signing-material`

코드 서명 인증서의 ARN입니다. 이 ARN은 인증서를 ACM으로 가져올 때 생성됩니다.

`signing-parameters`

서명을 위한 키-값 페어의 맵입니다. 여기에는 서명 중에 사용할 정보가 포함될 수 있습니다.

> **Note**
>
> AWS IoT용 코드 서명을 사용하여 OTA 업데이트에 서명하기 위한 코드 서명 프로필을 생성할 때 이 파라미터가 필요합니다.

`platform`

OTA 업데이트를 배포할 하드웨어 플랫폼의 `platformId`입니다.

사용 가능한 플랫폼 및 `platformId` 값 목록을 반환하려면 aws signer list-signing-platforms 명령을 사용합니다.

서명 작업이 시작되고 서명된 펌웨어 이미지를 대상 Amazon S3 버킷에 씁니다. 서명된 펌웨어 이미지의 파일 이름은 GUID입니다. 스트림을 생성할 때 이 파일 이름이 필요합니다. Amazon S3 콘솔로 이동한 후 버킷을 선택하여 파일 이름을 찾을 수 있습니다. GUID 파일 이름을 가진 파일이 표시되지 않는 경우 브라우저를 새로 고칩니다.

이 명령은 작업 ARN 및 작업 ID를 표시합니다. 이러한 값은 나중에 필요합니다. 에 대한 코드 서명에 대한 자세한 내용은에 대한 코드 서명을 AWS IoT참조하세요. AWS IoT

펌웨어 이미지에 수동으로 서명

펌웨어 이미지에 디지털 방식으로 서명하고 서명된 펌웨어 이미지를 Amazon S3 버킷에 업로드합니다.

펌웨어 업데이트 스트림 생성

스트림은 디바이스에서 사용할 수 있는 데이터에 대한 추상적 인터페이스입니다. 스트림은 서로 다른 위치 또는 다른 클라우드 기반 서비스에 저장된 데이터에 액세스하는 복잡성을 숨길 수 있습니다. OTA

업데이트 관리자 서비스를 사용하면 Amazon S3의 여러 위치에 저장된 여러 데이터를 사용하여 OTA 업데이트를 수행할 수 있습니다.

AWS IoT OTA 업데이트를 생성할 때 서명된 펌웨어 업데이트가 포함된 스트림을 생성할 수도 있습니다. 서명된 펌웨어 이미지를 식별하는 JSON 파일(`stream.json`)을 생성합니다. JSON 파일에는 다음이 포함되어야 합니다.

```
[
  {
    "fileId":"your_file_id",
    "s3Location":{
      "bucket":"your_bucket_name",
      "key":"your_s3_object_key"
    }
  }
]
```

다음은 JSON 파일의 속성입니다.

`fileId`

펌웨어 이미지를 식별하는 0~255 사이의 임의 정수입니다.

`s3Location`

스트리밍할 펌웨어에 대한 버킷 및 키입니다.

`bucket`

서명되지 않은 펌웨어 이미지가 저장되는 Amazon S3 버킷입니다.

`key`

Amazon S3 버킷에 있는 서명된 펌웨어 이미지의 파일 이름입니다. Amazon S3 콘솔에서 버킷 내용을 조사하여 이 값을 찾을 수 있습니다.

에 코드 서명을 사용하는 경우 AWS IoT파일 이름은 Code Signing for에서 생성한 GUID입니다 AWS IoT.

create-stream AWS CLI 명령을 사용하여 스트림을 만듭니다.

```
aws iot create-stream \
```

```
    --stream-id your_stream_id \
    --description your_description \
    --files file://stream.json \
    --role-arn your_role_arn
```

create-stream AWS CLI 명령의 인수는 다음과 같습니다.

`stream-id`

스트림을 식별하는 임의 문자열입니다.

`description`

스트림에 대한 설명(선택 사항)입니다.

`files`

스트리밍할 펌웨어 이미지에 대한 데이터를 포함하는 JSON 파일에 대한 하나 이상의 참조입니다. JSON 파일은 다음 속성을 포함해야 합니다.

`fileId`

임의 파일 ID입니다.

`s3Location`

서명된 펌웨어 이미지가 저장되는 버킷 이름 및 서명된 펌웨어 이미지의 키(파일 이름)입니다.

`bucket`

서명된 펌웨어 이미지가 저장되는 Amazon S3 버킷입니다.

`key`

서명된 펌웨어 이미지의 키(파일 이름)입니다.

에 코드 서명을 사용하는 경우 AWS IoT이 키는 GUID입니다.

다음은 예 `stream.json` 파일입니다.

```
[
    {
        "fileId":123,
        "s3Location": {
            "bucket":"codesign-ota-bucket",
```

```
            "key":"48c67f3c-63bb-4f92-a98a-4ee0fbc2bef6"
        }
    }
]
```

`role-arn`

펌웨어 이미지가 저장된 Amazon S3 버킷에 대한 액세스 권한도 부여하는 OTA 서비스 역할입니다.

서명된 펌웨어 이미지의 Amazon S3 객체 키를 찾으려면 aws signer describe-signing-job --job-id ***my-job-id*** 명령을 사용합니다. 여기서 `my-job-id`는 create-signing-job AWS CLI 명령으로 표시되는 작업 ID입니다. describe-signing-job 명령의 출력에는 서명된 펌웨어 이미지의 키가 포함됩니다.

```
... text deleted for brevity ...
  "signedObject": {
    "s3": {
      "bucketName": "ota-bucket",
      "key": "7309da2c-9111-48ac-8ee4-5a4262af4429"
    }
  }
... text deleted for brevity ...
```

OTA 업데이트 생성

create-ota-update AWS CLI 명령을 사용하여 OTA 업데이트 작업을 생성합니다.

> **Note**
>
> OTA 업데이트 작업 ID에 개인 식별 정보(PII)를 사용하지 마십시오. 개인 식별 정보의 예는 다음과 같습니다.
>
> - 이름
> - IP 주소
> - 이메일 주소
> - 위치
> - 은행 세부 정보
> - 의료 정보

```
aws iot  create-ota-update \
    --ota-update-id value \
    [--description value] \
    --targets value \
    [--protocols value] \
    [--target-selection value] \
    [--aws-job-executions-rollout-config value] \
    [--aws-job-presigned-url-config value] \
    [--aws-job-abort-config value] \
    [--aws-job-timeout-config value] \
    --files value \
    --role-arn value \
    [--additional-parameters value] \
    [--tags value]  \
    [--cli-input-json value] \
    [--generate-cli-skeleton]
```

cli-input-json 형식

```
{
  "otaUpdateId": "string",
  "description": "string",
  "targets": [
    "string"
  ],
  "protocols": [
    "string"
  ],
  "targetSelection": "string",
  "awsJobExecutionsRolloutConfig": {
    "maximumPerMinute": "integer",
    "exponentialRate": {
      "baseRatePerMinute": "integer",
      "incrementFactor": "double",
      "rateIncreaseCriteria": {
        "numberOfNotifiedThings": "integer",
        "numberOfSucceededThings": "integer"
      }
    }
  },
  "awsJobPresignedUrlConfig": {
    "expiresInSec": "long"
  },
```

```
  "awsJobAbortConfig": {
    "abortCriteriaList": [
      {
        "failureType": "string",
        "action": "string",
        "thresholdPercentage": "double",
        "minNumberOfExecutedThings": "integer"
      }
    ]
  },
  "awsJobTimeoutConfig": {
    "inProgressTimeoutInMinutes": "long"
  },
  "files": [
    {
      "fileName": "string",
      "fileType": "integer",
      "fileVersion": "string",
      "fileLocation": {
        "stream": {
          "streamId": "string",
          "fileId": "integer"
        },
        "s3Location": {
          "bucket": "string",
          "key": "string",
          "version": "string"
        }
      },
      "codeSigning": {
        "awsSignerJobId": "string",
        "startSigningJobParameter": {
          "signingProfileParameter": {
            "certificateArn": "string",
            "platform": "string",
            "certificatePathOnDevice": "string"
          },
          "signingProfileName": "string",
          "destination": {
            "s3Destination": {
              "bucket": "string",
              "prefix": "string"
            }
          }
```

```
        },
        "customCodeSigning": {
          "signature": {
            "inlineDocument": "blob"
          },
          "certificateChain": {
            "certificateName": "string",
            "inlineDocument": "string"
          },
          "hashAlgorithm": "string",
          "signatureAlgorithm": "string"
        }
      },
      "attributes": {
        "string": "string"
      }
    }
  ],
  "roleArn": "string",
  "additionalParameters": {
    "string": "string"
  },
  "tags": [
    {
      "Key": "string",
      "Value": "string"
    }
  ]
}
```

cli-input-json 필드

명칭	유형	설명
otaUpdateId	문자열 (최대:128분:1)	생성할 OTA 업데이트의 ID입니다.
description	문자열 (최대:2028)	OTA 업데이트에 대한 설명입니다.

명칭	유형	설명
`targets`	list	OTA 업데이트를 수신할 대상 디바이스입니다.
`protocols`	list	OTA 업데이트 이미지를 전송하는 데 사용되는 프로토콜입니다. 유효한 값은 [HTTP], [MQTT], [HTTP, MQTT]입니다. HTTP와 MQTT가 모두 지정되면 대상 디바이스가 프로토콜을 선택할 수 있습니다.
`targetSelection`	문자열	대상으로 지정된 모든 사물이 업데이트를 마친 후 업데이트를 계속해서 실행할지(CONTINUOUS), 혹은 완료할지(SNAPSHOT) 지정합니다. CONTINUOUS인 경우에는 대상에서 변경이 감지되면 사물에서 업데이트를 계속 실행할 수 있습니다. 예를 들어 사물이 대상 그룹에 추가되면 처음에 그룹에 포함되었던 모든 사물에서 업데이트가 완료되었다고 해도 사물에서 업데이트가 계속 실행됩니다. 유효 값: CONTINUOUS | SNAPSHOT 열거형: CONTINUOUS | SNAPSHOT
`awsJobExecutionsRolloutConfig`		OTA 업데이트의 롤아웃에 대한 구성.
`maximumPerMinute`	정수 (최대:1000분:1)	분당 시작한 OTA 업데이트 작업 실행 최대 수.

명칭	유형	설명
exponentialRate		작업 롤아웃 증가 속도입니다. 이 파라미터를 사용하면 작업 롤아웃에 대한 기하급수적 속도 증가를 정의할 수 있습니다.
baseRatePerMinute	정수 (최대:1000분:1)	작업 롤아웃 시작 시 대기 중인 작업에 대한 알림을 받을 분당 최소 사물 수입니다. 이는 롤아웃의 초기 속도입니다.
rateIncreaseCriteria		작업에 대한 롤아웃 속도 증가를 시작하는 기준입니다. AWS IoT 는 소수점 뒤에 최대 1자리까지 지원합니다(예: 1.5는 지원하지만 1.55는 지원하지 않음).
numberOfNotifiedThings	정수 (최소:1)	이 사물 수에 대한 알림을 받으면 롤아웃 속도가 증가하기 시작합니다.
numberOfSucceededThings	정수 (최소:1)	이 사물 수가 작업 실행에서 성공하면 롤아웃 속도가 증가하기 시작합니다.
awsJobPresignedUrlConfig		미리 서명된 URL의 구성 정보입니다.
expiresInSec	long	미리 서명된 URL의 유효 시간(초)입니다. 유효한 값은 60~3,600이며, 기본값은 1,800초입니다. 사전 서명된 URL은 작업 문서에 대한 요청이 수신될 때 생성됩니다.

명칭	유형	설명
awsJobAbortConfig		작업 중단이 발생하는 시기와 방법을 결정하는 기준입니다.
abortCriteriaList	list	작업을 중단할 시기와 방법을 결정하는 기준 목록입니다.
failureType	문자열	작업 중단을 시작할 수 있는 작업 실행 실패 유형입니다. enum: FAILED | REJECTED | TIMED_OUT | ALL
action	문자열	작업 중단을 시작하기 위해 수행할 작업 작업 유형입니다. enum: CANCEL
minNumberOfExecutedThings	정수 (최소:1)	작업을 중단하기 전에 작업 실행 알림을 받아야 하는 최소 사물 수입니다.
awsJobTimeoutConfig		각 디바이스가 작업 실행을 마쳐야 하는 시간을 지정합니다. 타이머는 작업 실행 상태가 IN_PROGRESS 에 설정되면 시작합니다. 타이머가 만료하기 전에 작업 실행 상태가 다른 터미널 상태로 설정되어 있지 않으면 TIMED_OUT 로 자동으로 설정됩니다.

명칭	유형	설명
inProgressTimeoutInMinutes	long	이 디바이스가 이 작업 실행을 마쳐야 하는 시간(분)을 지정합니다. 제한 시간 간격은 1분~7일(1~10080분) 사이의 어떤 값이든 가능합니다. 진행 중 타이머는 업데이트될 수 없으며 이 작업에 대한 모든 작업 실행에 적용됩니다. 작업 실행이 이 간격보다 오랫동안 IN_PROGRESS 상태를 유지할 때마다 작업 실행은 실패하며 터미널 TIMED_OUT 상태로 전환합니다.
files	list	OTA 업데이트에서 스트리밍되는 파일입니다.
fileName	문자열	파일 이름입니다
fileType	정수 최대 범위: 255 최소 범위: 0	디바이스가 클라우드에서 수신한 파일 유형을 식별할 수 있도록 작업 문서에 포함할 수 있는 정수 값입니다.
fileVersion	문자열	파일 버전입니다.
fileLocation		업데이트된 펌웨어 위치
stream		OTA 업데이트를 포함하는 스트림.
streamId	문자열 (최대:128분:1)	스트림 ID입니다.

명칭	유형	설명
fileId	정수 (최대:255분:0)	스트림과 연결된 파일의 ID입니다.
s3Location		S3에 업데이트된 펌웨어 위치.
bucket	문자열 (최소:1)	S3 버킷
key	문자열 (최소:1)	S3 키입니다.
version	문자열	S3 버킷 버전
codeSigning		파일의 코드 서명 메서드입니다.
awsSignerJobId	문자열	파일 서명을 위해 생성된 AWSSignerJob의 ID입니다.
startSigningJobParameter		코드 서명 작업을 설명합니다.
signingProfileParameter		코드 서명 프로파일을 설명합니다.
certificateArn	문자열	인증서 ARN.
platform	문자열	디바이스 하드웨어 플랫폼.
certificatePathOnDevice	문자열	디바이스의 코드 서명 인증서의 위치.
signingProfileName	문자열	코드 서명 프로파일 이름.
destination		코드 서명한 파일을 쓸 위치.

명칭	유형	설명
s3Destination		업데이트된 펌웨어의 S3의 위치를 설명합니다.
bucket	문자열 (최소:1)	업데이트된 펌웨어를 포함하는 S3 버킷.
prefix	문자열	S3 접두사.
customCodeSigning		파일 코드 서명을 위한 사용자 지정 메서드입니다.
signature		파일 서명입니다.
inlineDocument	blob	코드 서명에 대한 base64 인코딩 바이너리 표현입니다.
certificateChain		인증서 체인입니다.
certificateName	문자열	인증서 이름입니다
inlineDocument	문자열	코드 서명 인증서 체인에 대한 base64 인코딩 바이너리 표현입니다.
hashAlgorithm	문자열	파일의 코드 서명에 사용되는 해시 알고리즘입니다.
signatureAlgorithm	문자열	파일의 코드 서명에 사용되는 서명 알고리즘입니다.
attributes	map	이름/속성 페이의 목록입니다.
roleArn	문자열 (최대:2048분:20)	OTA 업데이트 작업을 생성하기 위해 Amazon S3, AWS IoT 작업 및 AWS 코드 서명 리소스에 대한 AWS IoT 액세스 권한을 부여하는 IAM 역할입니다.

명칭	유형	설명
additionalParameters	map	이름-값 페어로 추가되는 OTA 업데이트 파라미터의 목록입니다.
tags	list	업데이트 관리에 사용할 수 있는 메타데이터입니다.
Key	문자열 (최대:128분:1)	태그 키.
Value	문자열 (최대: 256분:1)	태그 값입니다.

출력

```
{
  "otaUpdateId": "string",
  "awsIotJobId": "string",
  "otaUpdateArn": "string",
  "awsIotJobArn": "string",
  "otaUpdateStatus": "string"
}
```

AWS CLI 출력 필드

명칭	유형	설명
otaUpdateId	문자열 (최대:128분:1)	OTA 업데이트 ID입니다.
awsIotJobId	문자열	OTA 업데이트와 연결된 AWS IoT 작업 ID입니다.
otaUpdateArn	문자열	OTA 업데이트 ARN입니다.

명칭	유형	설명
awsIotJobArn	문자열	OTA 업데이트와 연결된 AWS IoT 작업 ARN입니다.
otaUpdateStatus	문자열	OTA 업데이트 상태입니다. 열거형: CREATE_PENDING | CREATE_IN_PROGRESS | CREATE_COMPLETE | CREATE_FAILED

다음은 코드 서명을 사용하는 create-ota-update 명령에 전달되는 JSON 파일의 예입니다 AWS IoT.

```
[
  {
    "fileName": "firmware.bin",
    "fileType": 1,
    "fileLocation": {
      "stream": {
        "streamId": "004",
        "fileId":123
      }
    },
    "codeSigning": {
      "awsSignerJobId": "48c67f3c-63bb-4f92-a98a-4ee0fbc2bef6"
    }
  }
]
```

다음은 인라인 파일을 사용하여 사용자 지정 코드 서명 구성 요소를 제공하는 create-ota-update AWS CLI 명령에 전달되는 JSON 파일의 예입니다.

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

다음은 FreeRTOS OTA가 코드 서명 작업을 시작하고 코드 서명 프로파일 및 스트림을 생성할 수 있도록 허용하는 create-ota-update AWS CLI 명령에 전달되는 JSON 파일의 예입니다.

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

다음은 기존 프로파일로 코드 서명 작업을 시작하고 지정된 스트림을 사용하는 OTA 업데이트를 생성하는 create-ota-update AWS CLI 명령에 전달되는 JSON 파일의 예입니다.

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

다음은 FreeRTOS OTA가 기존 코드 서명 작업 ID로 스트림을 생성할 수 있도록 허용하는 create-ota-update AWS CLI 명령에 전달되는 JSON 파일의 예입니다.

```
[
  {
```

```
    "fileName": "your_firmware_path_on_device",
    "fileType": 1,
    "fileVersion": "1",
    "codeSigning":{
      "awsSignerJobId": "your_signer_job_id"
    }
  }
]
```

다음은 OTA 업데이트를 생성하는 create-ota-update AWS CLI 명령에 전달되는 JSON 파일의 예입니다. 업데이트는 지정된 S3 객체에서 스트림을 생성하고 사용자 지정 코드 서명을 사용합니다.

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

OTA 업데이트 나열

list-ota-updates AWS CLI 명령을 사용하여 모든 OTA 업데이트 목록을 가져올 수 있습니다.

```
aws iot list-ota-updates
```

list-ota-updates 명령의 출력은 다음과 같습니다.

```
{
  "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
    }
  ]
}
```

OTA 업데이트에 대한 정보 가져오기

get-ota-update AWS CLI 명령을 사용하여 OTA 업데이트의 생성 또는 삭제 상태를 가져올 수 있습니다.

```
aws iot get-ota-update --ota-update-id your-ota-update-id
```

get-ota-update 명령에 대한 출력은 다음과 같습니다.

```
{
    "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"
    }
}
```

otaUpdateStatus에 대해 반환되는 값은 다음과 같습니다.

`CREATE_PENDING`

OTA 업데이트 생성이 보류 중입니다.

`CREATE_IN_PROGRESS`

OTA 업데이트가 생성되고 있습니다.

`CREATE_COMPLETE`

OTA 업데이트가 생성되었습니다.

`CREATE_FAILED`

OTA 업데이트를 생성하지 못했습니다.

`DELETE_IN_PROGRESS`

OTA 업데이트를 삭제하고 있습니다.

`DELETE_FAILED`

OTA 업데이트를 삭제하지 못했습니다.

> **Note**
>
> OTA 업데이트가 생성된 후 실행 상태를 가져오려면 describe-job-execution 명령을 사용해야 합니다. 자세한 내용은 작업 실행 설명을 참조하세요.

OTA 관련 데이터 삭제

현재는 AWS IoT 콘솔을 사용하여 스트림 또는 OTA 업데이트를 삭제할 수 없습니다. 를 AWS CLI 사용하여 스트림, OTA 업데이트 및 OTA 업데이트 중에 생성된 AWS IoT 작업을 삭제할 수 있습니다.

OTA 스트림 삭제

MQTT를 사용하는 OTA 업데이트를 생성할 때 명령줄 또는 AWS IoT 콘솔을 사용하여 펌웨어를 청크로 나누는 스트림을 생성하여 MQTT를 통해 전송할 수 있습니다. 다음 예제와 같이 delete-stream AWS CLI 명령을 사용하여이 스트림을 삭제할 수 있습니다.

```
aws iot delete-stream --stream-id your_stream_id
```

OTA 업데이트 삭제

OTA 업데이트를 생성할 때 다음과 같은 항목이 생성됩니다.

- OTA 업데이트 작업 데이터베이스 내 항목
- 업데이트를 수행하는 AWS IoT 작업입니다.
- 업데이트 중인 각 디바이스에 대한 AWS IoT 작업 실행입니다.

delete-ota-update 명령은 OTA 업데이트 작업 데이터베이스에서만 항목을 삭제합니다. delete-job 명령을 사용하여 AWS IoT 작업을 삭제해야 합니다.

delete-ota-update 명령을 사용하여 OTA 업데이트를 삭제합니다.

```
aws iot delete-ota-update --ota-update-id your_ota_update_id
```

`ota-update-id`

삭제할 OTA 업데이트의 ID

`delete-stream`

OTA 업데이트와 연결된 스트림을 삭제합니다.

`force-delete-aws-job`

OTA 업데이트와 연결된 AWS IoT 작업을 삭제합니다. 이 플래그가 설정되어 있지 않고 작업이 `In_Progress` 상태인 경우에는 작업이 삭제되지 않습니다.

OTA 업데이트용으로 생성된 IoT 작업 삭제

FreeRTOS는 OTA 업데이트를 생성할 때 AWS IoT 작업을 생성합니다. 또한 작업을 처리하는 각 디바이스에 대해 작업 실행이 생성됩니다. delete-job AWS CLI 명령을 사용하여 작업 및 관련 작업 실행을 삭제할 수 있습니다.

```
aws iot delete-job --job-id your-job-id --no-force
```

`no-force` 파라미터는 종료 상태(완료됨 또는 취소됨)인 작업만 삭제할 수 있도록 지정합니다. `force` 파라미터를 전달하여 종료 상태가 아닌 작업을 삭제할 수 있습니다. 자세한 내용은 DeleteJob API 단원을 참조하십시오.

> **Note**
>
> IN_PROGRESS 상태인 작업을 삭제하면 디바이스에서 IN_PROGRESS 상태인 작업 실행이 중단되고 디바이스가 비결정적 상태로 유지될 수 있습니다. 삭제된 작업을 실행하는 각 디바이스를 알려진 상태로 복구할 수 있는지 확인합니다.

작업에 대해 생성된 작업 실행의 수와 기타 요소에 따라 작업을 삭제하는 데 몇 분 정도 걸릴 수 있습니다. 작업이 삭제되는 동안 상태는 DELETION_IN_PROGRESS입니다. 상태가 이미 DELETION_IN_PROGRESS인 작업을 삭제하거나 취소하려고 하면 오류가 발생합니다.

delete-job-execution을 사용하여 작업 실행을 삭제할 수 있습니다. 일부 디바이스에서 작업을 처리할 수 없는 경우에 작업 실행을 삭제하려고 할 수 있습니다. 이렇게 하면 다음 예제와 같이 단일 디바이스에 대한 작업 실행이 삭제됩니다.

```
aws iot delete-job-execution --job-id your-job-id --thing-name
                    your-thing-name --execution-number your-job-execution-number --no-
force
```

delete-job AWS CLI 명령과 마찬가지로 `--force` 파라미터를에 전달delete-job-execution하여 작업 실행을 강제로 삭제할 수 있습니다. 자세한 내용은 DeleteJobExecution API 단원을 참조하십시오.

> **Note**
>
> IN_PROGRESS 상태인 작업 실행을 삭제하면 디바이스에서 IN_PROGRESS 상태인 작업 실행이 중단되고 디바이스가 비결정적 상태로 유지될 수 있습니다. 삭제된 작업을 실행하는 각 디바이스를 알려진 상태로 복구할 수 있는지 확인합니다.

OTA 업데이트 데모 애플리케이션을 사용하는 방법에 대한 자세한 내용은 OTA(Over-the-Air) 업데이트 데모 애플리케이션 단원을 참조하십시오.

OTA 업데이트 관리자 서비스

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

무선 업데이트(OTA) 관리자 서비스를 사용하여 다음을 수행할 수 있습니다.

- OTA 업데이트와 AWS IoT 작업, AWS IoT 스트림, 코드 서명을 포함하여 사용하는 리소스를 생성합니다.
- OTA 업데이트에 대한 정보를 가져옵니다.
- AWS 계정과 연결된 모든 OTA 업데이트를 나열합니다.
- OTA 업데이트를 삭제합니다.

OTA 업데이트는 OTA 업데이트 관리자 서비스에 의해 유지보수되는 데이터 구조입니다. OTA 업데이트는 다음을 포함합니다.

- OTA 업데이트 ID
- OTA 업데이트 설명(선택 사항)
- 업데이트할 디바이스 목록(대상).
- OTA 업데이트 유형: 연속 또는 스냅샷 필요한 업데이트 유형에 대한 설명은 AWS IoT 개발자 안내서의 작업 단원을 참조하세요.
- OTA 업데이트를 수행하는 데 사용되는 프로토콜은 [MQTT], [HTTP] 또는 [MQTT, HTTP]입니다. MQTT 및 HTTP를 지정하면 디바이스 설정에 따라 사용되는 프로토콜이 결정됩니다.
- 대상 디바이스로 전송할 파일 목록
- OTA 업데이트 작업을 생성하기 위해 Amazon S3, AWS IoT 작업 및 AWS 코드 서명 리소스에 대한 AWS IoT 액세스 권한을 부여하는 IAM 역할입니다.
- 사용자 정의 이름-값 페어 목록(선택 사항)

OTA 업데이트는 디바이스 펌웨어를 업데이트하도록 설계되었지만, 이를 사용하여 하나 이상의 디바이스에 등록하려는 파일을 전송할 수 있습니다 AWS IoT. 무선으로 펌웨어 업데이트를 전송할 경우 업데이트를 수신하는 디바이스에서 업데이트가 중간에 변조되지 않았음을 확인할 수 있도록 파일에 디지털 방식으로 서명하는 것이 좋습니다.

선택한 설정에 따라 HTTP 또는 MQTT 프로토콜을 사용하여 업데이트된 펌웨어 이미지를 보낼 수 있습니다. FreeRTOS용 코드 서명을 사용하여 펌웨어 업데이트에 서명하거나, 자체 코드 서명 도구를 사용할 수 있습니다.

프로세스를 더 자세히 제어하려면 CreateStream API를 사용하여 MQTT를 통해 업데이트를 전송할 때 스트림을 생성할 수 있습니다. 경우에 따라 FreeRTOS 에이전트 코드를 수정하여 보내고 받는 블록의 크기를 조정할 수 있습니다.

OTA 업데이트를 생성하면 OTA 관리자 서비스에서 디바이스에 업데이트가 사용 가능함을 알려주는 AWS IoT 작업을 생성합니다. FreeRTOS OTA 에이전트가 디바이스에서 실행되고 업데이트 메시지를 수신합니다. 업데이트를 사용할 수 있는 경우 HTTP 또는 MQTT를 통해 펌웨어 업데이트 이미지를 요청하고 파일을 로컬로 저장합니다. 다운로드한 파일의 디지털 서명을 확인하고 유효할 경우 펌웨어 업데이트를 설치합니다. FreeRTOS를 사용하지 않는 경우 업데이트를 수신 및 다운로드할 자체 OTA 에이전트를 구현하고 설치 작업을 수행해야 합니다.

애플리케이션에 OTA 에이전트 통합

Important

이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

무선(OTA) 에이전트는 OTA 업데이트 기능을 제품에 추가하기 위해 작성해야 하는 코드 양을 간소화하기 위해 설계되었습니다. 통합 작업에서는 기본적으로 OTA 에이전트를 초기화하고, 선택적으로 OTA 완료 이벤트 메시지에 응답하기 위한 사용자 지정 콜백 함수를 생성합니다.

Note

OTA 업데이트 기능을 애플리케이션에 통합하는 작업은 다소 간단하지만, OTA 업데이트 시스템에서 디바이스 코드 통합보다 더 많은 것을 이해해야 합니다. 사물, 자격 증명, 코드 서명 인증서, 프로비저닝 디바이스 및 OTA 업데이트 작업으로 AWS AWS IoT 계정을 구성하는 방법을 숙지하려면 FreeRTOS 사전 조건을 참조하세요.

연결 관리

OTA 에이전트는 AWS IoT 서비스와 관련된 모든 제어 통신 작업에 MQTT 프로토콜을 사용하지만 MQTT 연결을 관리하지는 않습니다. OTA 에이전트가 애플리케이션의 연결 관리 정책을 간섭하지 않도록 하려면 MQTT 연결(연결 해제 및 다시 연결 기능 포함)을 기본 사용자 애플리케이션에서 처리해야 합니다. 이 파일은 MQTT 또는 HTTP 프로토콜을 통해 다운로드할 수 있습니다. OTA 작업을 생성할 때 프로토콜을 선택할 수 있습니다. MQTT를 선택하면 OTA 에이전트는 제어 작업 및 파일 다운로드에 동일한 연결을 사용합니다. HTTP를 선택하면 OTA 에이전트가 HTTP 연결을 처리합니다.

간단한 OTA 데모

다음은 에이전트가 MQTT 브로커에 연결하고 OTA 에이전트를 초기화하는 방법을 보여주는 간단한 OTA 데모에서 발췌한 내용입니다. 이 예제에서는 기본 OTA 완료 콜백을 사용하고 일부 통계를 초당 하나씩 출력하도록 데모를 구성합니다. 간결하게 나타내기 위해 이 데모에서는 일부 세부 정보를 생략합니다.

또한 OTA 데모는 연결 해제 콜백을 모니터링하고 연결을 다시 설정하여 MQTT 연결을 관리하는 방법을 보여줍니다. 연결이 끊어지면 데모에서는 먼저 OTA 에이전트 작업을 일시 중단한 다음 MQTT 연결을 다시 설정하려고 시도합니다. MQTT 재연결 시도는 최대 값까지 기하급수적으로 증가하는 시간만큼 지연되며 지터도 추가됩니다. 연결이 다시 설정되면 OTA 에이전트는 작업을 계속합니다.

AWS IoT MQTT 브로커를 사용하는 작업 예제는 `demos/ota` 디렉터리의 OTA 데모 코드를 참조하세요.

OTA 에이전트는 자체 작업이므로 이 예의 의도적인 1초 지연은 이 애플리케이션에만 영향을 줍니다. 따라서 에이전트의 성능에는 영향을 주지 않습니다.

```
void vRunOTAUpdateDemo( bool awsIotMqttMode,
                        const char * pIdentifier,
                        void * pNetworkServerInfo,
                        void * pNetworkCredentialInfo,
                        const IotNetworkInterface_t * pNetworkInterface )
{
    OTA_State_t eState;
    static OTA_ConnectionContext_t xOTAConnectionCtx;

    IotLogInfo( "OTA demo version %u.%u.%u\r\n",
                xAppFirmwareVersion.u.x.ucMajor,
                xAppFirmwareVersion.u.x.ucMinor,
                xAppFirmwareVersion.u.x.usBuild );

    for( ; ; )
    {
        IotLogInfo( "Connecting to broker...\r\n" );

        /* Establish a new MQTT connection. */
        if( _establishMqttConnection( awsIotMqttMode,
                                      pIdentifier,
                                      pNetworkServerInfo,
                                      pNetworkCredentialInfo,
                                      pNetworkInterface,
```

```
                                    &_mqttConnection ) == EXIT_SUCCESS )
        {
            /* Update the connection context shared with OTA Agent.*/
            xOTAConnectionCtx.pxNetworkInterface = ( void * ) pNetworkInterface;
            xOTAConnectionCtx.pvNetworkCredentials = pNetworkCredentialInfo;
            xOTAConnectionCtx.pvControlClient = _mqttConnection;

            /* Set the base interval for connection retry.*/
            _retryInterval = OTA_DEMO_CONN_RETRY_BASE_INTERVAL_SECONDS;

            /* Update the connection available flag.*/
            _networkConnected = true;

            /* Check if OTA Agent is suspended and resume.*/
            if( ( eState = OTA_GetAgentState() ) == eOTA_AgentState_Suspended )
            {
                OTA_Resume( &xOTAConnectionCtx );
            }

            /* Initialize the OTA Agent , if it is resuming the OTA statistics will be
cleared for new connection.*/
            OTA_AgentInit( ( void * ) ( &xOTAConnectionCtx ),
                           ( const uint8_t * ) ( clientcredentialIOT_THING_NAME ),
                           App_OTACompleteCallback,
                           ( TickType_t ) ~0 );

            while( ( ( eState = OTA_GetAgentState() ) != eOTA_AgentState_Stopped ) &&
_networkConnected )
            {
                /* Wait forever for OTA traffic but allow other tasks to run and output
statistics only once per second. */
                IotClock_SleepMs( OTA_DEMO_TASK_DELAY_SECONDS * 1000 );

                IotLogInfo( "State: %s  Received: %u   Queued: %u   Processed: %u
Dropped: %u\r\n", _pStateStr[ eState ],
                            OTA_GetPacketsReceived(), OTA_GetPacketsQueued(),
OTA_GetPacketsProcessed(), OTA_GetPacketsDropped() );
            }

            /* Check if we got network disconnect callback and suspend OTA Agent.*/
            if( _networkConnected == false )
            {
                /* Suspend OTA agent.*/
                if( OTA_Suspend() == kOTA_Err_None )
```

```
                {
                    while( ( eState = OTA_GetAgentState() ) !=
 eOTA_AgentState_Suspended )
                    {
                        /* Wait for OTA Agent to process the suspend event. */
                        IotClock_SleepMs( OTA_DEMO_TASK_DELAY_SECONDS * 1000 );
                    }
                }
            }
            else
            {
                /* Try to close the MQTT connection. */
                if( _mqttConnection != NULL )
                {
                    IotMqtt_Disconnect( _mqttConnection, 0 );
                }
            }
        }
        else
        {
            IotLogError( "ERROR:  MQTT_AGENT_Connect() Failed.\r\n" );
        }

        /* After failure to connect or a disconnect, delay for retrying connection. */
        _connectionRetryDelay();
    }
}
```

다음은 이 데모 애플리케이션의 상위 수준 흐름입니다.

- MQTT 에이전트 컨텍스트를 생성합니다.
- AWS IoT 엔드포인트에 연결합니다.
- OTA 에이전트를 초기화합니다.
- OTA 업데이트 작업을 허용하고 1초에 1회 통계를 출력하는 루프입니다.
- MQTT 연결이 끊어지면 OTA 에이전트 작업을 일시 중단합니다.
- 연결을 다시 시도합니다(이때 기하급수적인 지연 및 지터가 발생함).
- 다시 연결되면 OTA 에이전트 작업을 재개합니다.
- 에이전트가 중지되면 1초 정도 지연한 다음 재연결을 시도합니다.

OTA 완료 이벤트에 대해 사용자 지정 콜백 사용

이전 예제에서는 App_OTACompleteCallback이 OTA 완료 이벤트에 대한 콜백 핸들러로 사용되었습니다. (OTA_AgentInit API 직접 호출의 세 번째 파라미터 참조) App_OTACompleteCallback은 *freertos*/demos/ota/aws_iot_ota_update_demo.c에 정의되어 있습니다. 완료 이벤트에 대해 사용자 지정 처리를 구현하려면 콜백 핸들러의 함수 주소를 OTA_AgentInit API에 전달해야 합니다. OTA 프로세스 중에 에이전트는 다음 이벤트 열거형 중 하나를 콜백 핸들러에 전송할 수 있습니다. 이벤트를 처리하는 방법과 시기는 애플리케이션 개발자가 결정해야 합니다.

```
/**
* @brief OTA Job callback events.
*
* After an OTA update image is received and authenticated, the Agent calls the user
* callback (set with the OTA_AgentInit API) with the value eOTA_JobEvent_Activate 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 eOTA_JobEvent_StartTest, 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 eOTA_JobEvent_Fail instead to allow the user to log the failure and take
* any action deemed appropriate by the user code.
*
*/
typedef enum {
    eOTA_JobEvent_Activate,  /*! OTA receive is authenticated and ready to activate. */
    eOTA_JobEvent_Fail,      /*! OTA receive failed. Unable to use this update. */
    eOTA_JobEvent_StartTest  /*! OTA job is now in self test, perform user tests. */
} OTA_JobEvent_t;
```

OTA 에이전트는 기본 애플리케이션에서 활성 처리 중에 백그라운드에서 업데이트를 수신할 수 있습니다. 이러한 이벤트를 전달하는 목적은 즉시 조치할 수 있는지, 아니면 일부 다른 애플리케이션별 처리가 완료될 때까지 지연할지 여부를 애플리케이션에서 결정할 수 있도록 하기 위한 것입니다. 그러면 활성 처리(예: vacuum 수행) 중에 디바이스의 예기치 않은 중단으로 펌웨어 업데이트 후에 재설정하는 것을 방지할 수 있습니다. 다음은 콜백 핸들러에 의해 수신되는 작업 이벤트입니다.

eOTA_JobEvent_Activate event

이 이벤트가 콜백 핸들러에 수신된 경우 즉시 디바이스를 재설정하거나 나중에 디바이스 재설정을 위한 호출을 예약할 수 있습니다. 따라서 필요한 경우 디바이스 재설정 및 자체 테스트 단계를 연기할 수 있습니다.

`eOTA_JobEvent_Fail event`

이 이벤트가 콜백 핸들러에 수신된 경우 업데이트가 실패한 것입니다. 이 경우 아무것도 수행할 필요가 없습니다. 로그 메시지를 출력하거나 애플리케이션 관련 작업을 수행할 수 있습니다.

`eOTA_JobEvent_StartTest event`

자체 테스트 단계에서는 새로 업데이트된 펌웨어를 실행하고 올바르게 작동하는지 결정하기 이전에 자체 테스트를 실시한 후 최신 영구 애플리케이션 이미지로 커밋할 수 있습니다. 새 업데이트를 수신하여 인증하고 디바이스를 재설정한 후 테스트 준비가 되면 OTA 에이전트가 `eOTA_JobEvent_StartTest` 이벤트를 콜백 함수에 전송합니다. 개발자는 업데이트 후 디바이스 펌웨어가 올바르게 작동하는지 확인하는 데 필요한 테스트를 추가할 수 있습니다. 자체 테스트에서 디바이스 펌웨어가 안정적인 것으로 간주될 경우 코드는 `OTA_SetImageState( eOTA_ImageState_Accepted )` 함수를 호출하여 펌웨어를 새 영구 이미지로 커밋해야 합니다.

디바이스에 테스트를 수행하는 데 필요한 특수 하드웨어 또는 메커니즘이 없는 경우 기본 콜백 핸들러를 사용할 수 있습니다. `eOTA_JobEvent_Activate` 이벤트가 수신되면 기본 핸들러가 즉시 디바이스를 재설정합니다.

OTA 보안

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

다음은 무선(OTA) 보안의 세 가지 측면입니다.

연결 보안

OTA 업데이트 관리자 서비스는 AWS IoT에서 사용되는 전송 계층 보안(TLS) 상호 인증과 같은 기존 보안 메커니즘을 사용합니다. OTA 업데이트 트래픽은 AWS IoT 디바이스 게이트웨이를 통과하고 보안 메커니즘을 사용합니다 AWS IoT . 디바이스 게이트웨이를 통해 수신 및 발신되는 각 HTTP 또는 MQTT 메시지는 엄격한 인증 및 권한 부여 절차를 거칩니다.

OTA 업데이트의 신뢰성 및 무결성

OTA 업데이트 이전에 펌웨어에 디지털 방식으로 서명하여 OTA 업데이트가 출처를 신뢰할 수 있고 변조되지 않았음을 보장할 수 있습니다.

FreeRTOS OTA Update Manager 서비스는 용 코드 서명을 사용하여 펌웨어 AWS IoT 에 자동으로 서명합니다. 자세한 내용은 AWS IoT용 코드 서명을 참조하십시오.

디바이스에서 실행되는 OTA 에이전트는 펌웨어가 디바이스에 수신되면 무결성 검사를 수행합니다.

운영자 보안

컨트롤 플레인 API를 통해 생성되는 모든 API 직접 호출은 표준 IAM 서명 버전 4 인증 및 권한 부여 절차를 거칩니다. 배포를 생성하려면 `CreateDeployment`, `CreateJob` 및 `CreateStream` API를 호출할 권한이 있어야 합니다. 또한 Amazon S3 버킷 정책 또는 ACL에서 스트리밍 중에 Amazon S3에 저장된 펌웨어 업데이트에 액세스할 수 있도록 AWS IoT 서비스 보안 주체에 읽기 권한을 부여해야 합니다.

에 대한 코드 서명 AWS IoT

AWS IoT 콘솔은 용 코드 서명을 AWS IoT 사용하여에서 지원하는 모든 디바이스의 펌웨어 이미지에 자동으로 서명합니다 AWS IoT.

용 코드 서명은 ACM으로 가져오는 인증서와 프라이빗 키를 AWS IoT 사용합니다. 자체 서명된 인증서를 테스트용으로 사용할 수 있지만 잘 알려진 상용 인증 기관(CA)에서 인증서를 받는 것이 좋습니다.

코드 서명 인증서는 X.509 버전 3 `Key Usage` 및 `Extended Key Usage` 확장을 사용합니다. `Key Usage` 확장은 `Digital Signature`로 설정되어 있고, `Extended Key Usage` 확장은 `Code Signing`으로 설정되어 있습니다. 코드 이미지 서명에 대한 자세한 내용은 개발자용 AWS IoT 코드 서명 안내서 및 AWS IoT API용 코드 서명 참조를 참조하세요.

> **Note**
>
> Amazon Web Services용 도구에서 AWS IoT SDK용 코드 서명을 다운로드할 수 있습니다.

OTA 문제 해결

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

다음 단원에서는 OTA 업데이트 관련 문제를 해결하는 데 유용한 정보를 제공합니다.

주제

- OTA 업데이트를 위한 Cloudwatch Logs 설정
- 를 사용하여 AWS IoT OTA API 호출 로깅 AWS CloudTrail
- 를 사용하여 CreateOTAUpdate 실패 세부 정보 가져오기 AWS CLI
- 를 사용하여 OTA 실패 코드 가져오기 AWS CLI
- 여러 디바이스의 OTA 업데이트 문제 해결
- Texas Instruments CC3220SF Launchpad를 사용하여 OTA 업데이트 문제 해결

OTA 업데이트를 위한 Cloudwatch Logs 설정

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

OTA 업데이트 서비스에서는 Amazon CloudWatch를 통한 로깅을 지원합니다. AWS IoT 콘솔을 사용하여 OTA 업데이트에 대한 Amazon CloudWatch 로깅을 활성화하고 구성할 수 있습니다. 자세한 내용은 Cloudwatch Logs를 참조하십시오.

로깅을 활성화하려면 IAM 역할을 생성하고 OTA 업데이트 로깅을 구성해야 합니다.

> **Note**
>
> OTA 업데이트 로깅을 활성화하기 전에 CloudWatch Logs 액세스 권한을 이해해야 합니다. CloudWatch 로그 액세스 권한이 있는 사용자는 디버깅 정보를 볼 수 있습니다. 자세한 내용은 Amazon CloudWatch Logs에 대한 인증 및 액세스 제어를 참조하십시오.

로깅 역할 생성 및 로깅 활성화

AWS IoT 콘솔을 사용하여 로깅 역할을 생성하고 로깅을 활성화합니다.

1. 탐색 창에서 설정을 선택합니다.
2. 로그에서 편집을 선택합니다.
3. 세부 사항 수준에서 디버깅을 선택합니다.
4. 역할 설정에서 새로 생성을 선택하여 로깅을 위한 IAM 역할을 생성합니다.
5. 이름 아래에 역할의 고유한 이름을 입력합니다. 모든 필요한 권한을 가진 역할이 생성됩니다.
6. 업데이트를 선택합니다.

OTA 업데이트 로그

OTA 업데이트 서비스에서는 다음 중 하나가 발생할 경우 사용자의 계정에 로그를 게시합니다.

- OTA 업데이트가 생성된 경우
- OTA 업데이트가 완료된 경우
- 코드 서명 작업이 생성된 경우
- 코드 서명 작업이 완료된 경우
- AWS IoT 작업이 생성됩니다.
- AWS IoT 작업이 완료되었습니다.
- 스트림이 생성된 경우

CloudWatch 콘솔에서 로그를 확인할 수 있습니다.

CloudWatch Logs에서 OTA 업데이트를 보려면

1. 탐색 창에서 로그를 선택합니다.

2. 로그 그룹에서 AWSIoTLogsV2를 선택합니다.

OTA 업데이트 로그는 다음 속성을 포함할 수 있습니다.

accountId

로그가 생성된 AWS 계정 ID입니다.

actionType

로그를 생성한 작업입니다. 이 속성은 다음 값 중 하나로 설정될 수 있습니다.

- `CreateOTAUpdate`: OTA 업데이트가 생성되었습니다.
- `DeleteOTAUpdate`: OTA 업데이트가 삭제되었습니다.
- `StartCodeSigning`: 코드 서명 작업이 시작되었습니다.
- `CreateAWSJob`: AWS IoT 작업이 생성되었습니다.
- `CreateStream`: 스트림이 생성되었습니다.
- `GetStream`: 스트림에 대한 요청이 AWS IoT MQTT 기반 파일 전송 기능으로 전송되었습니다.
- `DescribeStream`: 스트림 정보에 대한 정보 요청이 AWS IoT MQTT 기반 파일 전송 기능으로 전송되었습니다.

awsJobId

로그를 생성한 AWS IoT 작업 ID입니다.

clientId

로그를 생성한 요청을 보낸 MQTT 클라이언트 ID입니다.

clientToken

로그를 생성한 요청과 연결된 클라이언트 토큰입니다.

details

로그를 생성한 작업에 대한 추가 정보입니다.

logLevel

로그의 로깅 수준입니다. OTA 업데이트 로그의 경우 이 값은 항상 `DEBUG`로 설정됩니다.

otaUpdateId

로그를 생성한 OTA 업데이트의 ID입니다.

프로토콜

로그를 생성한 요청을 만드는 데 사용된 프로토콜입니다.

status

로그를 생성한 작업의 상태입니다. 유효한 값은 다음과 같습니다.

- Success
- 실패

streamId

로그를 생성한 AWS IoT 스트림 ID입니다.

타임스탬프

로그가 생성된 시간입니다.

topicName

로그를 생성한 요청을 만드는 데 사용된 MQTT 주제입니다.

로그 예

다음은 코드 서명 작업을 시작할 때 생성되는 로그의 예입니다.

```
{
    "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."
}
```

다음은 AWS IoT 작업이 생성될 때 생성되는 로그의 예입니다.

```
{
    "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."
}
```

다음은 OTA 업데이트를 만들 때 생성되는 로그의 예입니다.

```
{
    "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."
}
```

다음은 스트림을 만들 때 생성되는 로그의 예입니다.

```
{
    "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."
}
```

다음은 OTA 업데이트를 삭제할 때 생성되는 로그의 예입니다.

```
{
    "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."
}
```

다음은 디바이스가 MQTT 기반 파일 전송 기능에서 스트림을 요청할 때 생성되는 로그의 예입니다.

```
{
    "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."
}
```

다음은 디바이스가 `DescribeStream` API를 호출할 때 생성되는 로그의 예입니다.

```
{
    "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."
}
```

를 사용하여 AWS IoT OTA API 호출 로깅 AWS CloudTrail

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

FreeRTOS는 AWS IoT OTA API 호출을 캡처하고 지정한 Amazon S3 버킷으로 로그 파일을 전송하는 서비스인 CloudTrail과 통합됩니다. CloudTrail은 코드에서 AWS IoT OTA API로 API 호출을 캡처APIs.

CloudTrail에서 수집한 정보를 사용하여 AWS IoT OTA에 수행된 요청, 요청이 수행된 소스 IP 주소, 요청을 수행한 사람, 요청이 수행된 시간 등을 확인할 수 있습니다.

이 서비스를 구성하고 사용하는 방법을 포함한 CloudTrail에 대한 자세한 내용은 AWS CloudTrail 사용 설명서를 참조하세요.

CloudTrail의 FreeRTOS 플랜 정보

AWS 계정에서 CloudTrail 로깅이 활성화되면 AWS IoT OTA 작업에 대한 API 호출이 다른 AWS 서비스 레코드와 함께 기록되는 CloudTrail 로그 파일에서 추적됩니다. CloudTrail은 기간 및 파일 크기를 기준으로 새 파일을 만들고 기록하는 시점을 결정합니다.

CloudTrail은 다음 OTA 컨트롤 AWS IoT 플레인 작업을 로깅합니다.

- CreateStream
- DescribeStream
- ListStreams
- UpdateStream
- DeleteStream
- CreateOTAUpdate
- GetOTAUpdate
- ListOTAUpdates
- DeleteOTAUpdate

> **Note**
>
> AWS IoT OTA 데이터 영역 작업(디바이스 측)은 CloudTrail에서 로깅되지 않습니다. CloudWatch를 사용하여 이러한 작업을 모니터링합니다.

모든 로그 항목은 누가 요청을 생성했는가에 대한 정보가 들어 있습니다. 로그 항목의 사용자 신원 정보를 이용하면 다음을 쉽게 판단할 수 있습니다.

- 요청을 루트로 했는지 아니면 IAM 사용자 보안 인증 정보로 했는지 여부.
- 역할 또는 페더레이션 사용자의 임시 보안 인증을 사용하여 요청이 생성되었는지 여부.
- 요청이 다른 AWS 서비스에서 이루어졌는지 여부입니다.

자세한 내용은 AWS IoT OTA API 참조에 설명된 CloudTrail userIdentity Element.OTA 작업을 참조하세요. AWS IoT

원하는 기간만큼 Amazon S3 버킷에 로그 파일을 저장할 수 있습니다. 그러나 Amazon S3 수명 주기 규칙을 정의하여 자동으로 로그 파일을 보관하거나 삭제할 수도 있습니다. 기본적으로 로그 파일은 Amazon S3 서버 측 암호화(SSE)를 통해 암호화합니다.

로그 파일이 전송될 때 알림을 받으려면 Amazon SNS 알림을 게시하도록 CloudTrail을 구성할 수 있습니다. 자세한 내용은 CloudTrail용 Amazon SNS 알림 구성을 참조하세요.

여러 AWS 리전 및 여러 AWS 계정의 AWS IoT OTA 로그 파일을 단일 Amazon S3 버킷으로 집계할 수도 있습니다.

자세한 내용은 여러 리전에서 CloudTrail 로그 파일 받기와 여러 계정에서 CloudTrail 로그 파일 받기를 참조하세요.

FreeRTOS 로그 파일 항목 이해

CloudTrail 로그 파일에는 하나 이상의 로그 항목이 포함될 수 있습니다. 각 항목은 여러 개의 JSON 형식 이벤트를 표시합니다. 로그 항목은 어떤 소스로부터의 단일 요청을 나타내며 요청된 작업, 작업 날짜와 시간, 요청 파라미터 등에 대한 정보가 들어 있습니다. 로그 항목은 퍼블릭 API 직접 호출의 주문 스택 트레이스가 아니기 때문에 특정 순서로 표시되지 않습니다.

다음은 `CreateOTAUpdate` 작업 호출의 로그를 보여 주는 CloudTrail 로그 항목을 나타낸 예제입니다.

```
{
    "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"
}
```

를 사용하여 CreateOTAUpdate 실패 세부 정보 가져오기 AWS CLI

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

OTA 업데이트 작업 생성 프로세스가 실패할 경우 문제를 해결하기 위해 취할 수 있는 조치가 있을 수 있습니다. OTA 업데이트 작업을 생성하면 OTA 관리자 서비스가 IoT 작업을 생성하고 대상 디바이스에 대해 예약하며,이 프로세스는 계정에서 다른 유형의 AWS 리소스(코드 서명 작업, AWS IoT 스트림, Amazon S3 객체)도 생성하거나 사용합니다. 오류가 발생하면 AWS IoT 작업을 생성하지 않고 프로세스가 실패할 수 있습니다. 이 문제 해결 단원에서는 오류의 세부 정보를 검색하는 방법에 대한 지침을 제공합니다.

1. AWS CLI를 설치하고 구성합니다.
2. aws configure를 실행하고 다음 정보를 입력합니다.

   ```
   $ aws configure
   AWS Access Key ID [None]: AccessID
   AWS Secret Access Key [None]: AccessKey
   Default region name [None]: Region
   Default output format [None]: json
   ```

   자세한 내용은 aws configure를 사용한 빠른 구성을 참조하세요.
3. 실행합니다.

   ```
   aws iot get-ota-update --ota-update-id ota_update_job_001
   ```

   여기서, *ota_update_job_001*은 OTA 업데이트를 생성할 때 제공한 ID입니다.
4. 출력값은 다음과 같습니다.

   ```
   {
       "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)"
        }
    }
```

```
}
```

생성이 실패한 경우 명령 출력의 `otaUpdateStatus` 필드에 `CREATE_FAILED`가 포함되고 `errorInfo` 필드에 실패 세부 정보가 포함됩니다.

를 사용하여 OTA 실패 코드 가져오기 AWS CLI

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

1. AWS CLI를 설치하고 구성합니다.
2. aws configure를 실행하고 다음 정보를 입력합니다.

```
$ aws configure
AWS Access Key ID [None]: AccessID
AWS Secret Access Key [None]: AccessKey
Default region name [None]: Region
Default output format [None]: json
```

자세한 내용은 aws configure를 사용한 빠른 구성을 참조하세요.

3. 실행합니다.

```
aws iot describe-job-execution --job-id JobID --thing-name ThingName
```

여기서 *JobID*는 상태를 가져오려는 작업의 전체 작업 ID 문자열(생성 시 OTA 업데이트 작업과 연결됨)이고 *ThingName*은 디바이스가에 등록된 AWS IoT 사물 이름입니다. AWS IoT

4. 출력값은 다음과 같습니다.

```
{
    "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
    }
}
```

이 예제 출력에서 "detailsmap"의 "reason"에는 2개의 필드가 있습니다. “0xEEEEEEEE”로 표시된 필드에는 OTA 에이전트의 일반 오류 코드가 있고 “0xffffffff”로 표시된 필드에는 하위 코드가 있습니다. 일반 오류 코드는 https://docs.aws.amazon.com/freertos/latest/lib-ref/html1/aws__ota__agent_8h.html에 나열되어 있습니다. 접두사가 "kOTA_Err_“인 오류 코드를 참조하십시오. 하위 코드는 플랫폼별 코드이거나 일반 오류에 대한 자세한 내용을 제공할 수 있습니다.

여러 디바이스의 OTA 업데이트 문제 해결

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

동일한 펌웨어 이미지를 사용해 여러 디바이스(사물)에서 OTA를 수행하려면 비휘발성 메모리에서 clientcredentialIOT_THING_NAME을 검색하는 함수(예를 들어, getThingName())를 구현합니다. 이 함수가 사물 이름을 OTA가 덮어쓰지 않은 비휘발성 메모리의 일부에서 읽고, 사물 이름이 최초 작업 전 프로비저닝되는지 확인합니다. JITP 흐름을 사용한다면 디바이스 인증서의 일반 이름에서 사물 이름을 읽을 수 있습니다.

Texas Instruments CC3220SF Launchpad를 사용하여 OTA 업데이트 문제 해결

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

CC3220SF Launchpad 플랫폼은 소프트웨어 변조 탐지 메커니즘을 제공합니다. 이 플랫폼은 무결성 위반 시마다 증가하는 보안 알림 카운터를 사용합니다. 보안 알림 카운터가 미리 결정된 임계값(기본값 15)에 도달하면 디바이스가 잠기고 호스트는 SL_ERROR_DEVICE_LOCKED_SECURITY_ALERT 비동기 이벤트를 수신합니다. 잠긴 디바이스는 액세스가 제한됩니다. 디바이스를 복구하려면 디바이스를 다시 프로그래밍하거나 출하 시 설정으로 복원 프로세스를 사용하여 출하 시 이미지로 되돌릴 수 있습니다. network_if.c에서 비동기 이벤트 핸들러를 업데이트하여 원하는 동작을 프로그래밍해야 합니다.

FreeRTOS 라이브러리

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

FreeRTOS 라이브러리는 FreeRTOS 커널 및 내부 라이브러리에 추가 기능을 제공합니다. 임베디드 애플리케이션의 네트워킹 및 보안을 위해 FreeRTOS 라이브러리를 사용할 수 있습니다. FreeRTOS 라이브러리를 사용하면 애플리케이션이 AWS IoT 서비스와 상호 작용할 수도 있습니다. FreeRTOS에는 다음을 위한 라이브러리가 포함되어 있습니다.

- MQTT 및 디바이스 섀도우를 사용하여 디바이스를 AWS IoT 클라우드에 안전하게 연결합니다.
- AWS IoT Greengrass 코어를 검색하고 연결합니다.
- Wi-Fi 연결을 관리합니다.
- FreeRTOS 무선 업데이트(OTA)를 수신하고 처리합니다.

`libraries` 디렉터리에는 FreeRTOS 라이브러리의 소스 코드가 들어 있습니다. 라이브러리 기능의 구현을 지원하는 헬퍼 함수가 있습니다. 이 헬퍼 함수를 변경하지 않는 것이 좋습니다.

FreeRTOS 이식 라이브러리

다음 이식 라이브러리는 FreeRTOS 콘솔에서 다운로드할 수 있는 FreeRTOS 구성에 포함되어 있습니다. 이러한 라이브러리는 플랫폼에 따라 다릅니다. 라이브러리의 콘텐츠는 하드웨어 플랫폼에 따라 변경됩니다. 디바이스에 이러한 라이브러리를 이식하는 방법에 대한 자세한 내용은 FreeRTOS 이식 안내서를 참조하세요.

FreeRTOS 이식 라이브러리

라이브러리	API 참조	설명
Bluetooth Low Energy	Bluetooth Low Energy API 참조	FreeRTOS Bluetooth Low Energy 라이브러리를 사용하면 마이크로컨트롤러가 게이트웨이 디바이스를 통해 AWS IoT MQTT 브로커와 통신할 수

라이브러리	API 참조	설명
		있습니다. 자세한 내용은 Bluetooth Low Energy 라이브러리 단원을 참조하십시오.
OTA(Over-the-Air) 업데이트	OTA 에이전트 API 참조	FreeRTOS AWS IoT Over-the-Air(OTA) 에이전트 라이브러리는 FreeRTOS 디바이스를 AWS IoT OTA 에이전트에 연결합니다. 자세한 내용은 OTA 에이전트 라이브러리 단원을 참조하십시오.
FreeRTOS+POSIX	FreeRTOS+POSIX API 참조	FreeRTOS+POSIX 라이브러리를 사용하여 POSIX 호환 애플리케이션을 FreeRTOS 에코시스템에 이식할 수 있습니다. 자세한 내용은 FreeRTOS+POSIX를 참조하십시오.
보안 소켓	Secure Sockets API 참조	자세한 내용은 보안 소켓 라이브러리 단원을 참조하십시오.
FreeRTOS+TCP	FreeRTOS+TCP API 참조	FreeRTOS+TCP는 FreeRTOS를 위한 확장 가능한 오픈 소스 및 스레드 세이프 TCP/IP 스택입니다. 자세한 내용은 FreeRTOS+TCP를 참조하십시오.
Wi-Fi	Wi-Fi API 참조	FreeRTOS Wi-Fi 라이브러리를 사용하면 마이크로컨트롤러의 하위 수준 무선 스택과 연결할 수 있습니다. 자세한 내용은 Wi-Fi 라이브러리 단원을 참조하십시오.

라이브러리	API 참조	설명
corePKCS11		corePKCS11 라이브러리는 프로비저닝 및 TLS 클라이언트 인증을 지원하기 위한 퍼블릭 키 암호화 표준 #11의 참조 구현입니다. 자세한 내용은 corePKCS11 라이브러리 단원을 참조하십시오.
TLS		자세한 내용은 전송 계층 보안 단원을 참조하십시오.
공통 I/O	공통 I/O API 참조	자세한 내용은 공통 I/O 단원을 참조하십시오.

FreeRTOS 애플리케이션 라이브러리

필요에 따라 FreeRTOS 구성에 다음과 같은 독립 실행형 애플리케이션 라이브러리를 포함하여 클라우드의 AWS IoT 서비스와 상호 작용할 수 있습니다.

> **Note**
>
> 일부 애플리케이션 라이브러리는 Embedded C용 AWS IoT 디바이스 SDK의 라이브러리와 동일한 API를 사용합니다. 이러한 라이브러리의 경우 AWS IoT 디바이스 SDK C API 참조를 참조하세요. 임베디드 C용 AWS IoT 디바이스 SDK에 대한 자세한 내용은 섹션을 참조하세요AWS IoT 임베디드 C용 디바이스 SDK.

FreeRTOS 애플리케이션 라이브러리

라이브러리	API 참조	설명
AWS IoT Device Defender	Device Defender C SDK API 참조	FreeRTOS AWS IoT Device Defender 라이브러리는 FreeRTOS 디바이스를에 연결합니다 AWS IoT Device Defender.

라이브러리	API 참조	설명
		자세한 내용은 AWS IoT Device Defender 라이브러리 단원을 참조하십시오.
AWS IoT Greengrass	Greengrass API 참조	FreeRTOS AWS IoT Greengrass 라이브러리는 FreeRTOS 디바이스를에 연결합니다 AWS IoT Greengrass. 자세한 내용은 AWS IoT Greengrass 검색 라이브러리 단원을 참조하십시오.
MQTT	MQTT(v1.x.x) 라이브러리 API 참조 MQTT(v1) 에이전트 API 참조 MQTT(v2.x.x) C SDK API 참조	coreMQTT 라이브러리는 FreeRTOS 디바이스가 MQTT 주제를 게시 및 구독할 수 있도록 클라이언트를 제공합니다. MQTT는 디바이스가 상호 작용하는 데 사용하는 프로토콜입니다 AWS IoT. coreMQTT 라이브러리 버전 3.0.0에 대한 자세한 내용은 coreMQTT 라이브러리 섹션을 참조하세요.
AWS IoT 디바이스 섀도우	디바이스 섀도우 C SDK API 참조	AWS IoT 디바이스 섀도우 라이브러리를 사용하면 FreeRTOS 디바이스가 AWS IoT 디바이스 섀도우와 상호 작용할 수 있습니다. 자세한 내용은 AWS IoT 디바이스 섀도우 라이브러리 단원을 참조하십시오.

FreERTOS 공용 라이브러리

다음의 공통 라이브러리는 내장형 애플리케이션 개발을 위한 추가 데이터 구조 및 함수로 커널 기능을 확장합니다. 이러한 라이브러리는 종종 FreeRTOS 이식 및 애플리케이션 라이브러리의 종속 항목입니다.

> **Note**
>
> 임베디드 C용 AWS IoT 디바이스 SDK에는 이러한 라이브러리와 동일한 APIs 및 기능을 갖춘 공통 라이브러리가 포함되어 있습니다. API 참조의 경우 AWS IoT 디바이스 SDK C API 참조를 참조하십시오.

FreERTOS 공용 라이브러리

라이브러리	API 참조	설명
원자성 작업	원자 조작 C SDK API 참조	자세한 내용은 원자성 작업 단원을 참조하십시오.
선형 컨테이너	선형 컨테이너 C SDK API 참조	자세한 내용은 선형 컨테이너 라이브러리 단원을 참조하십시오.
로깅	로깅 C SDK API 참조	자세한 내용은 로깅 라이브러리 단원을 참조하십시오.
정적 메모리	정적 메모리 C SDK API 참조	자세한 내용은 정적 메모리 라이브러리 단원을 참조하십시오.
작업 풀	작업 풀 C SDK API 참조	자세한 내용은 작업 풀 라이브러리 단원을 참조하십시오.

FreeRTOS 라이브러리 구성

FreeRTOS 및 임베디드 C용 AWS IoT 디바이스 SDK의 구성 설정은 C 프리프로세서 상수로 정의됩니다. 전역 구성 파일을 사용하거나 `gcc`의 `-D`와 같은 컴파일러 옵션을 사용하여 구성 설정을 지정할 수 있습니다. 구성 설정은 컴파일 시간 상수로 정의되어 있기 때문에 구성 설정이 변경되면 라이브러리를 다시 빌드해야 합니다.

전역 구성 파일을 사용하여 구성 옵션을 설정하려면 이름이 `iot_config.h`인 파일을 만들고 저장한 다음 이를 포함 경로에 추가합니다. 파일 내에서 `#define` 지시문을 사용하여 FreeRTOS 라이브러리, 데모 및 테스트를 구성합니다.

지원되는 전역 구성 옵션에 대한 자세한 내용은 전역 구성 파일 참조를 참조하십시오.

공통 라이브러리

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

FreeRTOS에는 내장형 애플리케이션 개발을 위한 추가 데이터 구조 및 함수로 커널 기능을 확장하는 몇 가지 공통 라이브러리가 포함되어 있습니다. 이러한 라이브러리는 종종 다른 FreeRTOS 라이브러리의 종속 라이브러리이기도 합니다.

주제

- 원자성 작업
- 선형 컨테이너 라이브러리
- 로깅 라이브러리
- 정적 메모리 라이브러리
- 작업 풀 라이브러리

원자성 작업

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

개요

원자성 작업은 동시 프로그래밍에서 비 차단 동기화를 보장합니다. 원자성 작업을 사용하여 공유 메모리에서 작동하는 비동기 작업으로 인해 발생하는 성능 문제를 해결할 수 있습니다. FreeRTOS는 `iot_atomic.h` 헤더 파일에 구현된 것처럼 원자성 작업을 지원합니다.

`iot_atomic.h` 헤더 파일에는 원자성 작업을 위한 두 가지 구현이 포함됩니다.

- 전역적 인터럽트 비활성화.

  이 구현은 모든 FreeRTOS 플랫폼에서 사용할 수 있습니다.
- ISA 네이티브 원자성 지원.

  이 구현은 GCC, 버전 4.7.0 이상으로 컴파일되고 ISA 원자성 지원이 있는 플랫폼에서만 사용할 수 있습니다. GCC 기본 함수에 대한 자세한 내용은 Built-in Functions for Memory Model Aware Atomic Operations를 참조하십시오.

Initialization(초기화)

FreeRTOS 원자성 작업을 사용하기 전에, 사용하려는 원자성 작업의 구현을 선택해야 합니다.

1. 편집할 `FreeRTOSConfig.h` 구성 파일을 엽니다.
2. ISA 네이티브 원자성 지원 구현을 위해서는 `configUSE_ATOMIC_INSTRUCTION` 변수를 정의하고 `1`로 설정합니다.

   전역 인터럽트 구현을 비활성화하려면 `configUSE_ATOMIC_INSTRUCTION`의 정의를 해제하거나 지웁니다.

API 참조

전체 API 참조는 원자성 작업 C SDK API 참조를 참조하십시오.

선형 컨테이너 라이브러리

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

FreeRTOS 선형 컨테이너 라이브러리는 내장형 애플리케이션을 개발할 때 사용할 수 있는 목록 및 대기열을 포함한 선형 데이터 구조를 정의합니다.

API 참조

전체 API 참조는 선형 컨테이너 C SDK API 참조를 참조하십시오.

로깅 라이브러리

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

FreeRTOS 로깅 라이브러리를 사용하면 FreeRTOS 라이브러리가 디버깅을 위해 로그에 메시지를 인쇄할 수 있습니다.

API 참조

전체 API 참조는 로깅 C SDK API 참조를 참조하세요.

정적 메모리 라이브러리

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

FreeRTOS 정적 메모리 라이브러리는 정적 버퍼를 관리하기 위한 몇 가지 기능을 정의합니다. 이 라이브러리를 사용하면 정적 메모리 구성 요소를 사용하여 동적 메모리 할당 대신 정적으로 할당된 버퍼를 제공할 수 있습니다.

API 참조

전체 API 참조는 정적 메모리 C SDK API 참조를 참조하십시오.

작업 풀 라이브러리

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

개요

FreeRTOS는 FreeRTOS 태스크 풀 라이브러리를 사용한 태스크 관리를 지원합니다. 작업 풀 라이브러리를 사용하여 백그라운드 작업을 예약하고 안전하고 비동기적인 작업 예약 및 취소를 허용할 수 있습니다. 작업 풀 API로 애플리케이션의 작업을 구성하여 성능과 메모리 사용량 간의 균형을 최적화할 수 있습니다.

작업 풀 라이브러리는 작업 풀 및 작업 풀 작업의 두 가지 기본 데이터 구조로 빌드됩니다.

작업 풀(**`IotTaskPool_t`**)

작업 풀은 실행을 위해 작업 대기열을 관리하고 작업을 실행하는 작업자 스레드를 관리하는 디스패치 대기열을 포함합니다.

작업 풀 작업(**`IotTaskPoolJob_t`**)

작업 풀 작업은 백그라운드 작업 또는 시간이 지정된 백그라운드 작업으로 실행될 수 있습니다. 백그라운드 작업은 FIFO(선입선출) 방식으로 시작되며 시간 제한이 없습니다. 시간이 지정된 작업은 타이머에 따라 백그라운드 실행이 예약됩니다.

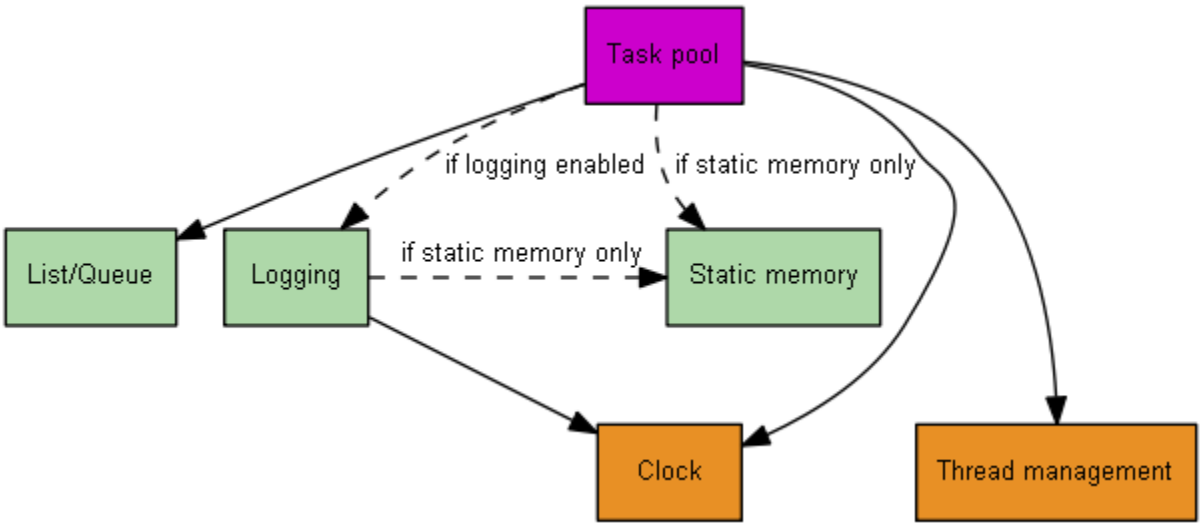
> **Note**
>
> 작업 풀은 시간이 지정된 작업이 특정 시간대가 아니라 제한 시간이 경과한 후에만 실행되도록 보장할 수 있습니다.

종속성 및 요구 사항

작업 풀 라이브러리에는 다음 종속성이 있습니다.

- 예약되거나 진행 중인 작업 풀 운영에 대한 데이터 구조를 유지 관리하는 선형 컨테이너(목록/대기열) 라이브러리

- 로깅 라이브러리(IOT_LOG_LEVEL_TASKPOOL 구성 설정이 IOT_LOG_NONE이 아닌 경우)
- 스레드 관리, 타이머, 클록 기능 등을 위해 운영 체제와의 인터페이스를 제공하는 플랫폼 계층입니다.



Features

작업 풀 라이브러리 API를 사용하여 다음을 수행할 수 있습니다.

- 라이브러리의 비 차단 API 함수로 즉시 작업과 지연 작업을 예약합니다.
- 정적 및 동적으로 할당된 작업을 만듭니다.
- 시스템 리소스를 기반으로 성능 및 공간을 확장하도록 라이브러리 설정을 구성합니다.
- 작업을 동적으로 생성할 때 캐싱을 사용자 지정하여 메모리 오버헤드를 낮춥니다.

문제 해결

작업 풀 라이브러리 함수는 오류 코드를 IotTaskPoolError_t 열거형 값으로 반환합니다. 각 오류 코드에 대한 자세한 내용은 태스크 풀 C SDK API 참조의 IotTaskPoolError_t 열거형 데이터 형식에 대한 참조 설명서를 참조하세요.

사용 제한

인터럽트 서비스 루틴(ISR)에서는 작업 풀 라이브러리를 사용할 수 없습니다.

차단 작업(특히 무기한 차단 작업)을 수행하는 작업 풀 사용자 콜백은 지양할 것을 강력히 권장합니다. 차단 작업이 오래 지속될 경우 작업 풀 스레드가 스틸되고 교착 또는 결핍이 발생할 가능성이 있습니다.

Initialization(초기화)

애플리케이션은 작업 풀을 사용하기 전에 `IotTaskPool_CreateSystemTaskPool`을 호출하여 시스템 작업 풀의 인스턴스를 초기화해야 합니다. 애플리케이션은 부팅 시퀀스에서 시스템 수준 작업 풀이 라이브러리가 작업 풀을 사용하기 전에 충분히 일찍, 그리고 애플리케이션 코드가 작업 풀에 작업을 게시하기 전에 초기화되었는지 확인해야 합니다. 부팅 직후 시스템은 모든 라이브러리가 공유할 단일 시스템 수준 작업 풀을 초기화합니다. 초기화 후에, `IOT_SYSTEM_TASKPOOL` API와 함께 사용할 작업 풀 핸들을 검색할 수 있습니다.

> **Note**
>
> `IotTaskPool_CreateSystemTaskPool`을 호출하면 작업 풀 데이터 구조 및 상태를 보유할 메모리가 할당되지 않지만 작업 풀의 스레드와 같은 종속 엔터티 및 데이터 구조를 보유하기 위해 메모리를 할당할 수 있습니다.

API 참조

전체 API 참조는 작업 풀 C SDK API 참조를 참조하십시오.

사용 예

반복되는 AWS IoT Device Defender 지표 모음을 예약해야 하고 타이머를 사용하여 MQTT 연결, 구독 및 게시 APIs. 다음 코드는 MQTT 연결에서 연결 해제되는 연결 해제 콜백과 함께 MQTT 전반의 AWS IoT Device Defender 지표를 수락하기 위한 콜백 함수를 정의합니다.

```
/* An example of a user context to pass to a callback through a task pool thread. */
typedef struct JobUserContext
{
    uint32_t counter;
} JobUserContext_t;

/* An example of a user callback to invoke through a task pool thread. */
static void ExecutionCb( IotTaskPool_t * pTaskPool, IotTaskPoolJob_t * pJob, void *
 context )
{
    ( void )pTaskPool;
```

```
    ( void )pJob;
    JobUserContext_t * pUserContext = ( JobUserContext_t * )context;
    pUserContext->counter++;
}

void TaskPoolExample( )
{
    JobUserContext_t userContext = { 0 };
    IotTaskPoolJob_t job;
    IotTaskPool_t * pTaskPool;
    IotTaskPoolError_t errorSchedule;

    /* Configure the task pool to hold at least two threads and three at the maximum.
 */
    /* Provide proper stack size and priority per the application needs. */
    const IotTaskPoolInfo_t tpInfo = { .minThreads = 2, .maxThreads = 3, .stackSize =
 512, .priority = 0 };

    /* Create a task pool. */
    IotTaskPool_Create( &tpInfo, &pTaskPool );

    /* Statically allocate one job, then schedule it. */
    IotTaskPool_CreateJob( &ExecutionCb, &userContext, &job );
    errorSchedule = IotTaskPool_Schedule( pTaskPool, &job, 0 );

    switch ( errorSchedule )
    {
    case IOT_TASKPOOL_SUCCESS:
        break;
    case IOT_TASKPOOL_BAD_PARAMETER:          // Invalid parameters, such as a NULL
 handle, can trigger this error.
    case IOT_TASKPOOL_ILLEGAL_OPERATION:      // Scheduling a job that was previously
 scheduled or destroyed could trigger this error.
    case IOT_TASKPOOL_NO_MEMORY:              // Scheduling a with flag
 #IOT_TASKPOOL_JOB_HIGH_PRIORITY could trigger this error.
    case IOT_TASKPOOL_SHUTDOWN_IN_PROGRESS:   // Scheduling a job after trying to
 destroy the task pool could trigger this error.
        // ASSERT
        break;
    default:
        // ASSERT*/
    }

    /* ............................... */
```

```
    /*  ... Perform other operations ... */
    /* ................................ */

    IotTaskPool_Destroy( pTaskPool );
}
```

backoffAlgorithm 라이브러리

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

소개

backoffAlgorithm 라이브러리는 네트워크 혼잡을 방지하기 위해 동일한 데이터 블록의 반복 재전송 간격을 지정하는 데 사용되는 유틸리티 라이브러리입니다. 이 라이브러리는 지터 포함 지수 백오프 알고리즘을 사용하여 네트워크 작업 재시도(예: 서버와의 네트워크 연결 실패)에 대한 백오프 기간을 계산합니다.

지터 포함 지수 백오프는 일반적으로 네트워크 혼잡 또는 서버 고부하로 인해 실패한 서버 연결 또는 네트워크 요청을 재시도할 때 사용됩니다. 여러 디바이스가 동시에 네트워크 연결을 시도할 때 발생하는 재시도 요청의 타이밍을 분산하는 데 사용됩니다. 연결 상태가 좋지 않은 환경에서는 클라이언트 연결이 언제든지 끊길 수 있습니다. 따라서 백오프 전략을 사용하면 실패할 가능성이 높을 때 반복적으로 재연결을 시도하지 않으므로 클라이언트가 배터리를 절약하는 데도 도움이 됩니다.

이 라이브러리는 C로 작성되었으며 ISO C90 및 MISRA C:2012를 준수하도록 설계되었습니다. 표준 C 라이브러리 이외의 추가 라이브러리에 대한 종속성이 없고 힙 할당이 없으므로 IoT 마이크로컨트롤러에 적합할 뿐만 아니라 다른 플랫폼으로도 완벽하게 이식할 수 있습니다.

이 라이브러리는 자유롭게 사용할 수 있으며 MIT 오픈 소스 라이선스에 따라 배포됩니다.

backoffAlgorithm 코드 크기(ARM Cortex-M용 GCC로 생성된 예제)		
파일	-O1 최적화	-Os 최적화
backoff_algorithm.c	0.1K	0.1K

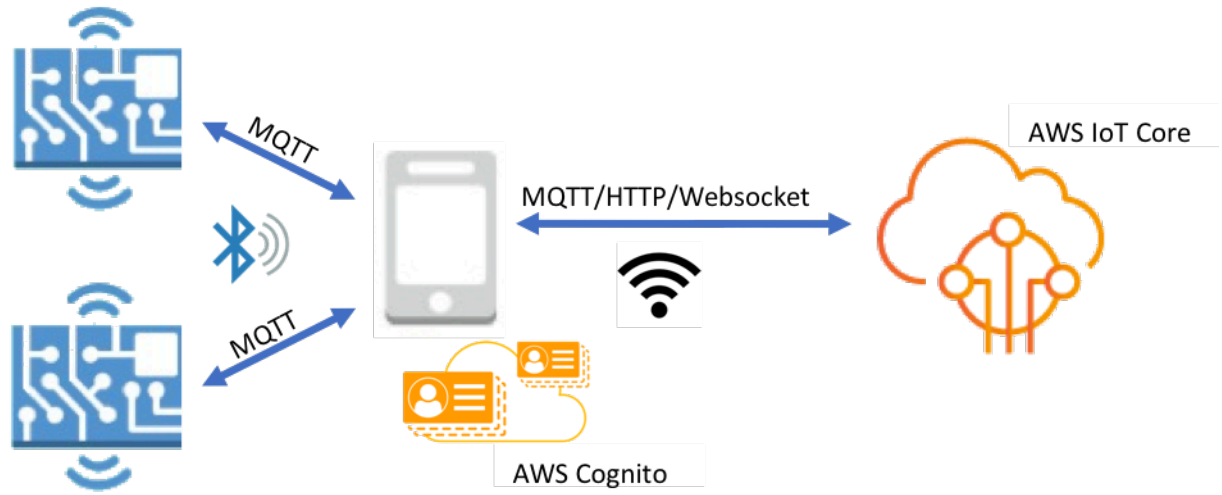
backoffAlgorithm 코드 크기(ARM Cortex-M용 GCC로 생성된 예제)		
파일	-O1 최적화	-Os 최적화
총 추정치	0.1K	0.1K

Bluetooth Low Energy 라이브러리

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

개요

FreeRTOS는 프록시 디바이스(예: 휴대폰)를 통해 Bluetooth Low Energy를 이용한 MQTT 주제 게시 및 구독을 지원합니다. FreeRTOS Bluetooth Low Energy 라이브러리를 사용하면 마이크로컨트롤러가 AWS IoT MQTT 브로커와 안전하게 통신할 수 있습니다.



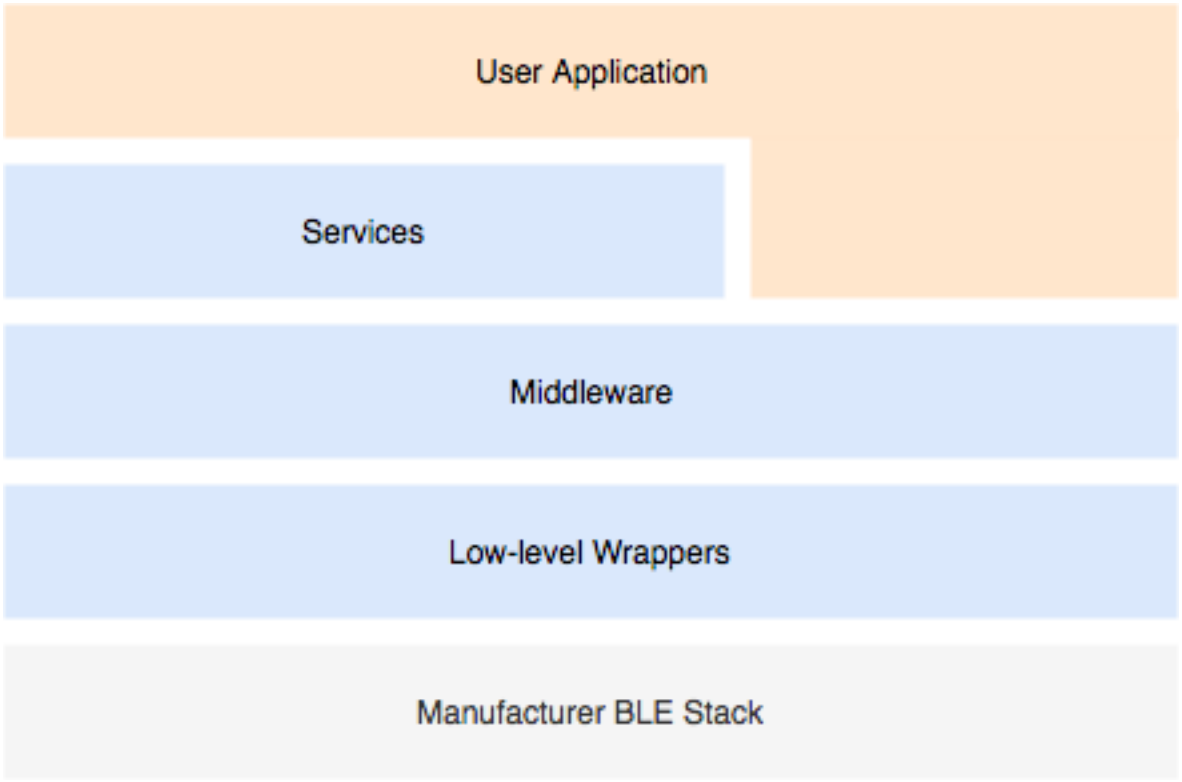
FreeRTOS Bluetooth 디바이스용 Mobile SDK를 사용하여 Bluetooth Low Energy를 통해 마이크로 컨트롤러의 내장형 애플리케이션과 통신하는 네이티브 모바일 애플리케이션을 작성할 수 있습니다. 모바일 SDK에 대한 자세한 내용은 FreeRTOS Bluetooth 디바이스용 Mobile SDK을 참조하십시오.

FreeRTOS Bluetooth Low Energy 라이브러리에는 Wi-Fi 네트워크를 구성하고 대용량 데이터를 전송하며 Bluetooth Low Energy를 통해 네트워크 추상화를 제공하는 서비스가 포함됩니다. 또한

FreeRTOS Bluetooth Low Energy 라이브러리에는 Bluetooth Low Energy 스택을 통해 더 직접적으로 제어하기 위한 하위 수준 API와 미들웨어도 포함됩니다.

아키텍처

FreeRTOS Bluetooth Low Energy 라이브러리는 서비스, 미들웨어 및 하위 수준 래퍼 계층으로 구성되어 있습니다.



서비스

FreeRTOS Bluetooth Low Energy 서비스 계층은 미들웨어 API를 활용하는 네 가지 일반 속성(GATT) 서비스인 디바이스 정보, Wi-Fi 프로비저닝, 네트워크 추상화 및 대용량 객체 전송으로 구성됩니다.

디바이스 정보

디바이스 정보 서비스는 마이크로컨트롤러에 대한 다음 정보를 수집합니다.

- 디바이스가 사용 중인 FreeRTOS의 버전
- 디바이스가 등록된 계정의 AWS IoT 엔드포인트입니다.
- Bluetooth Low Energy 최대 전송 단위(MTU)

Wi-Fi 프로비저닝

Wi-Fi 기능이 있는 마이크로컨트롤러는 Wi-Fi 프로비저닝 서비스를 통해 다음을 수행할 수 있습니다.

- 범위에 있는 네트워크 나열
- 네트워크 및 네트워크 자격 증명을 플래시 메모리에 저장
- 네트워크 우선 순위 설정
- 플래시 메모리에서 네트워크 및 네트워크 자격 증명 삭제

네트워크 추상화

네트워크 추상화 서비스는 애플리케이션의 네트워크 연결 유형을 추상화합니다. 공통 API는 애플리케이션이 여러 연결 유형과 호환될 수 있도록 디바이스의 Wi-Fi, 이더넷 및 Bluetooth Low Energy 하드웨어 스택과 상호 작용합니다.

대용량 객체 전송

대용량 객체 전송 서비스는 클라이언트와 데이터를 주고 받습니다. Wi-Fi 프로비저닝 및 네트워크 추상화와 같은 다른 서비스는 대용량 객체 전송 서비스를 사용하여 데이터를 보내고 받습니다. 또한 대용량 객체 전송 API를 사용하여 서비스와 직접 상호 작용할 수 있습니다.

미들웨어

FreeRTOS Bluetooth Low Energy 미들웨어는 하위 수준 API의 추상화입니다. 미들웨어 API는 Bluetooth Low Energy 스택에 대해 보다 사용자 친화적인 인터페이스를 구성합니다.

미들웨어 API를 사용하여 다중 계층의 여러 콜백을 단일 이벤트에 등록할 수 있습니다. Bluetooth Low Energy 미들웨어를 초기화하면 서비스가 초기화되고 광고가 시작됩니다.

유연한 콜백 구독

Bluetooth Low Energy 하드웨어가 연결 해제되었고, MQTT over Bluetooth Low Energy 서비스가 이 연결 해제를 감지해야 한다고 가정해 보겠습니다. 작성한 애플리케이션도 이 동일한 연결 해제 이벤트를 감지해야 할 수 있습니다. Bluetooth Low Energy 미들웨어는 상위 수준 계층이 하위 수준 리소스를 두고 경쟁하게 하지 않고 콜백을 등록한 코드의 여러 부분으로 이벤트를 라우팅할 수 있습니다.

하위 수준 래퍼

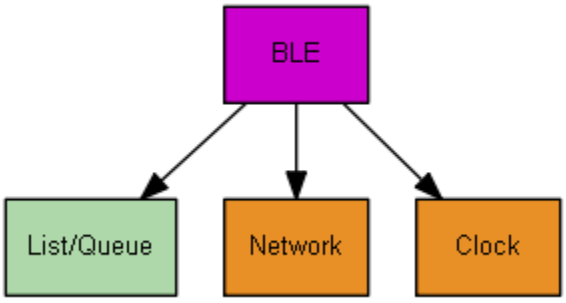
하위 수준 FreeRTOS Bluetooth Low Energy 래퍼는 제조업체의 Bluetooth Low Energy 스택의 추상화입니다. 하위 수준 래퍼는 하드웨어를 통한 직접 제어를 위해 공통 API 세트를 제공합니다. 하위 수준 API는 RAM 사용량을 최적화하지만 기능이 제한됩니다.

Bluetooth Low Energy 서비스 API를 사용하여 Bluetooth Low Energy 서비스와 상호 작용합니다. 서비스 API는 하위 수준 API보다 많은 리소스를 요구합니다.

종속성 및 요구 사항

Bluetooth Low Energy 라이브러리의 직접 종속성은 다음과 같습니다.

- 선형 컨테이너 라이브러리
- 스레드 관리, 타이머, 클록 기능 및 네트워크 액세스를 위해 운영 체제와 인터페이스하는 플랫폼 계층입니다.



Wi-Fi 프로비저닝 서비스에만 FreeRTOS 라이브러리 종속성이 있습니다.

GATT 서비스	종속성
Wi-Fi 프로비저닝	Wi-Fi 라이브러리

AWS IoT MQTT 브로커와 통신하려면 AWS 계정이 있어야 하며 디바이스를 AWS IoT 사물로 등록해야 합니다. 설정에 대한 자세한 내용은 AWS IoT 개발자 안내서를 참조하십시오.

FreeRTOS Bluetooth Low Energy는 모바일 디바이스의 사용자 인증에 Amazon Cognito를 사용합니다. MQTT 프록시 서비스를 사용하려면 Amazon Cognito 자격 증명과 사용자 풀을 생성해야 합니다. 각 Amazon Cognito 자격 증명에는 적절한 정책이 연결되어 있어야 합니다. 자세한 정보는 Amazon Cognito 개발자 안내서를 참조하세요.

라이브러리 구성 파일

FreeRTOS MQTT over Bluetooth Low Energy 서비스를 사용하는 애플리케이션은 구성 파라미터가 정의된 `iot_ble_config.h` 헤더 파일을 제공해야 합니다. 정의되지 않은 구성 파라미터는 `iot_ble_config_defaults.h`에 지정된 기본값을 사용합니다.

중요한 구성 파라미터는 다음과 같습니다.

`IOT_BLE_ADD_CUSTOM_SERVICES`

사용자가 고유한 서비스를 생성할 수 있습니다.

`IOT_BLE_SET_CUSTOM_ADVERTISEMENT_MSG`

사용자가 광고를 사용자 지정하고 응답 메시지를 검색할 수 있습니다.

자세한 내용은 Bluetooth Low Energy API 참조를 참조하십시오.

최적화

보드의 성능을 최적화할 때 다음을 고려하십시오.

- 하위 수준 API는 RAM을 덜 사용하지만 제한된 기능을 제공합니다.
- `iot_ble_config.h` 헤더 파일에서 `bleconfigMAX_NETWORK` 파라미터를 더 작은 값으로 설정하여 사용되는 스택 양을 줄일 수 있습니다.
- MTU 크기를 최대값으로 늘려 메시지 버퍼링을 제한하고, 코드를 더 빠르게 실행하며 RAM을 덜 사용할 수 있습니다.

사용 제한

기본적으로 FreeRTOS Bluetooth Low Energy 라이브러리는 `eBTpropertySecureConnectionOnly` 속성을 TRUE로 설정하여 디바이스를 보안 연결만 허용 모드로 설정합니다. Bluetooth 핵심 규격 v5.0, Vol 3, Part C, 10.2.4에 지정되어 있듯이 디바이스가 Secure Connections Only(보안 연결만 허용) 모드로 설정된 경우 최저 LE 보안 모드 1 수준, 수준 1보다 높은 권한이 있는 속성에 액세스하려면 최고 LE 보안 모드 1 수준, 수준 4가 필요합니다. LE 보안 모드 1 수준 4에서 디바이스에는 숫자 비교를 위해 입력 및 출력 기능이 있어야 합니다.

다음은 지원되는 모드와 이에 관련된 속성입니다.

모드 1, 수준 1(보안 없음)

```
/* 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 )
```

모드 1, 수준 2(암호화되었지만 페어링 인증되지 않음)

```
#define IOT_BLE_ENABLE_NUMERIC_COMPARISON        ( 0 )
#define IOT_BLE_ENABLE_SECURE_CONNECTION         ( 0 )
#define IOT_BLE_INPUT_OUTPUT                     ( eBTIONone )
```

모드 1, 수준 3(암호화 및 페어링 인증됨)

이 모드는 지원되지 않습니다.

모드 1, 수준 4(암호화 및 LE 보안 연결 페어링 인증됨)

이 모드는 기본적으로 지원됩니다.

LE 보안 모드에 대한 자세한 내용은 Bluetooth 핵심 규격 v5.0, Vol 3, Part C, 10.2.1을 참조하십시오.

Initialization(초기화)

애플리케이션이 미들웨어를 통해 Bluetooth Low Energy 스택과 상호 작용하는 경우 미들웨어만 초기화하면 됩니다. 미들웨어는 더 낮은 계층의 스택 초기화를 처리합니다.

미들웨어

미들웨어를 초기화하려면

1. Bluetooth Low Energy 미들웨어 API를 호출하기 전에 Bluetooth Low Energy 하드웨어 드라이버를 초기화합니다.
2. Bluetooth Low Energy를 활성화합니다.
3. `IotBLE_Init()`를 사용하여 미들웨어를 초기화합니다.

> **Note**
>
> AWS 데모를 실행하는 경우이 초기화 단계는 필요하지 않습니다. 데모 초기화는 *freertos*/demos/network_manager에 있는 네트워크 관리자에 의해 처리됩니다.

하위 수준 API

FreeRTOS Bluetooth Low Energy GATT 서비스를 사용하지 않으려는 경우 미들웨어를 무시하고 하위 수준 API와 직접 상호 작용하여 리소스를 저장할 수 있습니다.

하위 수준 API를 초기화하려면

1. API를 호출하기 전에 모든 Bluetooth Low Energy 하드웨어 드라이버를 초기화합니다. 드라이버 초기화는 Bluetooth Low Energy 하위 수준 API의 일부가 아닙니다.

2. Bluetooth Low Energy 하위 수준 API는 전력 및 리소스를 최적화하기 위해 Bluetooth Low Energy 스택에 활성화/비활성화 호출을 제공합니다. API를 호출하기 전에 Bluetooth Low Energy를 활성화해야 합니다.

```
const BTInterface_t * pxIface = BTGetBluetoothInterface();
xStatus = pxIface->pxEnable( 0 );
```

3. Bluetooth 관리자에는 Bluetooth Low Energy와 Bluetooth classic 모두에 공통인 API가 포함되어 있습니다. 공통 관리자에 대한 콜백은 두 번째로 초기화해야 합니다.

```
xStatus = xBTInterface.pxBTInterface->pxBtManagerInit( &xBTManagerCb );
```

4. Bluetooth Low Energy 어댑터는 공통 API의 상단에 맞습니다. 공통 API를 초기화한 것과 마찬가지로 콜백을 초기화해야 합니다.

```
xBTInterface.pxBTLeAdapterInterface = ( BTBleAdapter_t * )
 xBTInterface.pxBTInterface->pxGetLeAdapter();
xStatus = xBTInterface.pxBTLeAdapterInterface-
>pxBleAdapterInit( &xBTBleAdapterCb );
```

5. 새 사용자 애플리케이션을 등록합니다.

```
xBTInterface.pxBTLeAdapterInterface->pxRegisterBleApp( pxAppUuid );
```

6. GATT 서버로의 콜백을 초기화합니다.

```
xBTInterface.pxGattServerInterface = ( BTGattServerInterface_t * )
 xBTInterface.pxBTLeAdapterInterface->ppvGetGattServerInterface();
xBTInterface.pxGattServerInterface->pxGattServerInit( &xBTGattServerCb );
```

Bluetooth Low Energy 어댑터를 초기화한 후 GATT 서버를 추가할 수 있습니다. GATT 서버는 한 번에 하나만 등록할 수 있습니다.

```
xStatus = xBTInterface.pxGattServerInterface->pxRegisterServer( pxAppUuid );
```

7. 애플리케이션 속성(예: 보안 연결만 허용, MTU 크기 등)을 설정합니다.

```
xStatus = xBTInterface.pxBTInterface-
>pxSetDeviceProperty( &pxProperty[ usIndex ] );
```

API 참조

전체 API 참조는 Bluetooth Low Energy API 참조를 참조하십시오.

사용 예

아래의 예는 새로운 서비스를 광고하고 생성하기 위해 Bluetooth Low Energy 라이브러리를 사용하는 방법을 보여줍니다. 전체 FreeRTOS Bluetooth Low Energy 데모 애플리케이션은 Bluetooth Low Energy 데모 애플리케이션을 참조하세요.

광고

1. 애플리케이션에서 광고 UUID를 설정합니다.

```
static const BTUuid_t _advUUID =
{
    .uu.uu128 = IOT_BLE_ADVERTISING_UUID,
    .ucType   = eBTuuidType128
};
```

2. 그런 다음 `IotBle_SetCustomAdvCb` 콜백 함수를 정의합니다.

```
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;
}
```

이 콜백은 광고 메시지의 UUID와 스캔 응답의 전체 이름을 보냅니다.

3. vendors/*vendor*/boards/*board*/aws_demos/config_files/iot_ble_config.h를 열고 IOT_BLE_SET_CUSTOM_ADVERTISEMENT_MSG를 1로 설정합니다. 이렇게 하면 IotBle_SetCustomAdvCb 콜백이 트리거됩니다.

새 서비스 추가

서비스의 전체 예제는 *freertos*/.../ble/services를 참조하십시오.

1. 서비스 특성 및 설명자에 대한 UUID를 생성합니다.

```
#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. 특성 및 설명자의 핸들을 등록할 버퍼를 생성합니다.

```
static uint16_t usHandlesBuffer[egattDemoNbAttributes];
```

3. 속성 테이블을 생성합니다. 일부 RAM을 저장하려면 테이블을 const로 정의합니다.

> **Important**
>
> 항상 속성을 순서대로 생성하십시오. 첫 번째 속성은 서비스입니다.

```
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. 콜백 배열을 생성합니다. 이 콜백 배열은 위에 정의된 테이블 배열과 동일한 순서를 따라야 합니다.

예를 들어 xCharCounterUUID_TYPE을 액세스할 때 vReadCounter가 트리거되고, xCharControlUUID_TYPE을 액세스할 때 vWriteCommand가 트리거되는 경우 다음과 같이 배열을 정의합니다.

```
static const IotBleAttributeEventCallback_t pxCallBackArray[egattDemoNbAttributes]
 =
    {
  NULL,
  vReadCounter,
  vEnableNotification,
  vWriteCommand
};
```

5. 서비스를 만듭니다.

```
static const BTService_t xGattDemoService =
{
  .xNumberOfAttributes = egattDemoNbAttributes,
  .ucInstId = 0,
  .xType = eBTServiceTypePrimary,
  .pusHandlesBuffer = usHandlesBuffer,
  .pxBLEAttributes = (BTAttribute_t *)pxAttributeTable
};
```

6. 이전 단계에서 생성한 구조로 API IotBle_CreateService를 호출합니다. 미들웨어는 모든 서비스의 생성을 동기화하므로 IotBle_AddCustomServicesCb 콜백이 트리거될 때 새로운 서비스가 이미 정의되어 있어야 합니다.

   a. vendors/*vendor*/boards/*board*/aws_demos/config_files/iot_ble_config.h에서 IOT_BLE_ADD_CUSTOM_SERVICES를 1로 설정합니다.

   b. 애플리케이션에서 IotBle_AddCustomServicesCb를 생성합니다.

```
void IotBle_AddCustomServicesCb(void)
{
    BTStatus_t xStatus;
    /* Select the handle buffer. */
    xStatus = IotBle_CreateService( (BTService_t *)&xGattDemoService,
 (IotBleAttributeEventCallback_t *)pxCallBackArray );
}
```

이식

사용자 입력 및 출력 주변 장치

보안 연결에는 숫자 비교를 위한 입력 및 출력이 모두 필요합니다.
eBLENumericComparisonCallback 이벤트는 이벤트 관리자를 사용하여 등록할 수 있습니다.

```
xEventCb.pxNumericComparisonCb = &prvNumericComparisonCb;
xStatus = BLE_RegisterEventCb( eBLENumericComparisonCallback, xEventCb );
```

주변 장치는 숫자 패스 키를 표시하고 비교 결과를 입력으로 사용해야 합니다.

API 구현 이식

FreeRTOS를 새 대상으로 이식하려면 Wi-Fi 프로비저닝 서비스와 Bluetooth Low Energy 기능을 위한 몇 가지 API를 구현해야 합니다.

Bluetooth Low Energy API

FreeRTOS Bluetooth Low Energy 미들웨어를 사용하려면 몇 가지 API를 구현해야 합니다.

Bluetooth Classic용 GAP와 Bluetooth Low Energy용 GAP의 공통 API

- pxBtManagerInit
- pxEnable
- pxDisable
- pxGetDeviceProperty
- pxSetDeviceProperty(eBTpropertyRemoteRssi와 eBTpropertyRemoteVersionInfo를 제외한 모든 옵션은 필수)
- pxPair
- pxRemoveBond
- pxGetConnectionState
- pxPinReply
- pxSspReply
- pxGetTxpower
- pxGetLeAdapter

- pxDeviceStateChangedCb
- pxAdapterPropertiesCb
- pxSspRequestCb
- pxPairingStateChangedCb
- pxTxPowerCb

Bluetooth Low Energy용 GAP 전용 API

- pxRegisterBleApp
- pxUnregisterBleApp
- pxBleAdapterInit
- pxStartAdv
- pxStopAdv
- pxSetAdvData
- pxConnParameterUpdateRequest
- pxRegisterBleAdapterCb
- pxAdvStartCb
- pxSetAdvDataCb
- pxConnParameterUpdateRequestCb
- pxCongestionCb

GATT 서버

- 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

FreeRTOS Bluetooth Low Energy를 플랫폼으로 이식하는 방법에 대한 자세한 내용은 FreeRTOS 이식 안내서의 Bluetooth Low Energy 라이브러리 이식을 참조하세요.

FreeRTOS Bluetooth 디바이스용 Mobile SDK

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

FreeRTOS Bluetooth 디바이스용 Mobile SDK를 사용하여 Bluetooth Low Energy를 통해 마이크로컨트롤러와 상호 작용하는 모바일 애플리케이션을 빌드할 수 있습니다. Mobile SDKs는 사용자 인증에 Amazon Cognito를 사용하여 AWS 서비스와 통신할 수도 있습니다.

FreeRTOS Bluetooth 디바이스용 Android SDK

FreeRTOS Bluetooth 디바이스용 Android SDK를 사용하여 Bluetooth Low Energy를 통해 마이크로컨트롤러와 상호 작용하는 Android 모바일 애플리케이션을 빌드합니다. SDK는 GitHub에서 사용 가능합니다.

FreeRTOS Bluetooth 디바이스용 Android SDK를 설치하려면 프로젝트의 README.md 파일에 있는 'SDK 설정' 지침을 따릅니다.

SDK에 포함된 데모 모바일 애플리케이션 설정 및 실행에 대한 정보는 사전 조건 및 FreeRTOS Bluetooth Low Energy Mobile SDK 데모 애플리케이션을 참조하십시오.

FreeRTOS Bluetooth 디바이스용 iOS SDK

FreeRTOS Bluetooth 디바이스용 iOS SDK를 사용하여 Bluetooth Low Energy를 통해 마이크로컨트롤러와 상호 작용하는 iOS 모바일 애플리케이션을 빌드합니다. SDK는 GitHub에서 사용 가능합니다.

iOS SDK를 설치하려면

1. CocoaPods를 다음과 같이 설치합니다.

```
$ gem install cocoapods
$ pod setup
```

> **Note**
>
> sudo를 사용해 CocoaPods를 설치할 수 있습니다.

2. CocoaPods를 사용하여 SDK 설치(사용자의 Podfile에 추가):

```
$ pod 'FreeRTOS', :git => 'https://github.com/aws/amazon-freertos-ble-ios-sdk.git'
```

SDK에 포함된 데모 모바일 애플리케이션 설정 및 실행에 대한 정보는 사전 조건 및 FreeRTOS Bluetooth Low Energy Mobile SDK 데모 애플리케이션을 참조하십시오.

공통 I/O

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

개요

일반적으로 디바이스 드라이버는 기본 운영 체제와 독립적이며 지정된 하드웨어 구성에 따라 다릅니다. HAL(하드웨어 추상화 계층)은 드라이버와 상위 수준 애플리케이션 코드 간에 공통 인터페이스를 제공합니다. HAL은 특정 드라이버의 작동 방식에 대한 세부 정보를 추상화하고 이러한 디바이스를 제어하는 일관된 API를 제공합니다. 동일한 API를 사용하여 여러 마이크로컨트롤러(MCU) 기반 참조 보드에서 다양한 디바이스 드라이버에 액세스할 수 있습니다.

FreeRTOS 공통 I/O는 이 하드웨어 추상화 계층 역할을 수행합니다. 지원되는 참조 보드에서 공통 직렬 디바이스에 액세스하기 위한 표준 API 세트를 제공합니다. 이러한 공통 API를 사용하면 이러한 주변 장치와 통신하고 상호 작용하며 코드가 플랫폼 간에 작동하도록 활성화됩니다. 공통 I/O가 없는 경우 저수준 디바이스로 작업하도록 코드를 작성하는 방법은 실리콘 공급업체에 따라 다릅니다.

지원되는 주변 장치

- UART
- SPI
- I2C

지원 기능

- 동기식 읽기/쓰기 - 요청된 양의 데이터가 전송될 때까지 함수가 반환되지 않습니다.
- 비동기식 읽기/쓰기 - 함수는 즉시 반환되고 데이터 전송은 비동기적으로 발생합니다. 작업이 완료되면 등록된 사용자 콜백이 호출됩니다.

주변 장치별

- I2C - 하나의 트랜잭션으로 여러 작업을 결합합니다. 한 트랜잭션에서 작업을 읽은 다음 쓰기 작업을 수행하는 데 사용됩니다.
- SPI - 기본 및 보조 간에 데이터를 전송합니다. 즉, 쓰기 및 읽기가 동시에 수행됩니다.

이식

자세한 내용은 FreeRTOS 이식 안내서를 참조하세요.

AWS IoT Device Defender 라이브러리

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

소개

AWS IoT Device Defender 라이브러리를 사용하여 IoT 디바이스에서 보안 지표를 보낼 수 있습니다 AWS IoT Device Defender. AWS IoT Device Defender 를 사용하면 디바이스에서 이러한 보안 지표를 지속적으로 모니터링하여 각 디바이스에 정의된 적절한 동작과 편차가 있는지 확인할 수 있습니다. 문제가 있는 것으로 보이면에서 알림을 AWS IoT Device Defender 보내 문제를 해결하기 위한 조치를 취할 수 있도록 합니다. 와의 상호 작용은 경량 게시-구독 프로토콜인 MQTT를 AWS IoT Device Defender 사용합니다. 이 라이브러리는 AWS IoT Device Defender에서 사용하는 MQTT 주제 문자열을 작성하고 인식하는 API를 제공합니다.

자세한 내용은 AWS IoT 개발자 안내서의 AWS IoT Device Defender 섹션을 참조하세요.

이 라이브러리는 C로 작성되었으며 ISO C90 및 MISRA C:2012를 준수하도록 설계되었습니다. 이 라이브러리는 표준 C 라이브러리 이외의 추가 라이브러리에 대한 종속성이 없습니다. 또한 스레딩 또는 동기화와 같은 플랫폼 종속성도 없습니다. 모든 MQTT 라이브러리 및 모든 JSON 또는 CBOR 라이브러리와 함께 사용할 수 있습니다. 이 라이브러리에는 안전한 메모리 사용과 힙 할당 없음을 보여주는 증거가 있어 IoT 마이크로컨트롤러에 적합할 뿐만 아니라 다른 플랫폼으로도 완벽하게 이식할 수 있습니다.

AWS IoT Device Defender 라이브러리는 자유롭게 사용할 수 있으며 MIT 오픈 소스 라이선스에 따라 배포됩니다.

AWS IoT Device Defender의 코드 크기(ARM Cortex-M용 GCC로 생성된 예)		
파일	-O1 최적화	-Os 최적화
defender.c	1.1K	0.6K

AWS IoT Device Defender의 코드 크기(ARM Cortex-M용 GCC로 생성된 예)		
파일	-O1 최적화	-Os 최적화
총 추정치	1.1K	0.6K

AWS IoT Greengrass 검색 라이브러리

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

개요

AWS IoT Greengrass Discovery 라이브러리는 마이크로 컨트롤러 디바이스가 네트워크에서 Greengrass 코어를 검색하는 데 사용됩니다. AWS IoT Greengrass Discovery API를 사용하면 디바이스가 코어의 엔드포인트를 찾은 후 Greengrass 코어에 메시지를 보낼 수 있습니다.

종속성 및 요구 사항

Greengrass Discovery 라이브러리를 사용하려면 인증서 및 정책을 AWS IoT포함하여에 사물을 생성해야 합니다. 자세한 내용은 AWS IoT 시작하기를 참조하십시오.

freertos/demos/include/aws_clientcredential.h 파일에서 다음 상수의 값을 설정해야 합니다.

`clientcredentialMQTT_BROKER_ENDPOINT`

AWS IoT 엔드포인트.

`clientcredentialIOT_THING_NAME`

IoT 사물의 이름입니다.

`clientcredentialWIFI_SSID`

Wi-Fi 네트워크의 SSID입니다.

`clientcredentialWIFI_PASSWORD`

Wi-Fi 암호입니다.

`clientcredentialWIFI_SECURITY`

Wi-Fi 네트워크에서 사용되는 보안 유형입니다.

또한 *freertos*`/demos/include/aws_clientcredential_keys.h` 파일에서 다음 상수의 값을 설정해야 합니다.

`keyCLIENT_CERTIFICATE_PEM`

사물과 연결된 인증서 PEM입니다.

`keyCLIENT_PRIVATE_KEY_PEM`

사물과 연결된 프라이빗 키 PEM입니다.

콘솔에서 Greengrass 그룹 및 코어 디바이스를 설정해야 합니다. 자세한 내용은 시작하기를 참조하세요 AWS IoT Greengrass.

coreMQTT 라이브러리가 Greengrass 연결에 필요하지는 않지만 이 라이브러리를 설치하는 것이 좋습니다. Greengrass 코어를 검색한 후 이 라이브러리를 사용하여 해당 코어와 통신할 수 있습니다.

API 참조

전체 API 참조는 Greengrass API 참조를 참조하십시오.

사용 예

Greengrass 워크플로우

MCU 디바이스는 Greengrass 코어 연결 파라미터가 포함된 AWS IoT JSON 파일에서를 요청하여 검색 프로세스를 시작합니다. JSON 파일에서 Greengrass 코어 연결 파라미터를 검색하는 두 가지 방법이 있습니다.

- 자동 선택에서는 JSON 파일에 나열된 모든 Greengrass 코어를 반복한 후 사용 가능한 첫 번째 코어에 연결합니다.
- 수동 선택에서는 `aws_ggd_config.h`의 정보를 사용하여 지정된 Greengrass 코어에 연결합니다.

Greengrass API를 사용하는 방법

Greengrass API에 대한 모든 기본 구성 옵션은 aws_ggd_config_defaults.h에 정의되어 있습니다.

Greengrass 코어가 하나만 존재하는 경우 GGD_GetGGCIPandCertificate를 호출하여 Greengrass 코어 연결 정보로 JSON 파일을 요청합니다. GGD_GetGGCIPandCertificate가 반환되는 경우 pcBuffer 파라미터에는 JSON 파일의 텍스트가 포함되어 있습니다. pxHostAddressData 파라미터에는 연결할 수 있는 Greengrass 코어의 IP 주소와 포트가 포함되어 있습니다.

인증서 동적 할당과 같은 추가 사용자 지정 옵션을 보려면 다음 API를 호출해야 합니다.

GGD_JSONRequestStart

Greengrass 코어를 검색 AWS IoT 하기 위한 검색 요청을 시작하기 위해에 HTTP GET 요청을 수행합니다. GD_SecureConnect_Send는 요청을 보내는 데 사용됩니다 AWS IoT.

GGD_JSONRequestGetSize

HTTP 응답에서 JSON 파일의 크기를 가져옵니다.

GGD_JSONRequestGetFile

JSON 객체 문자열을 가져옵니다. GGD_JSONRequestGetSize 및 GGD_JSONRequestGetFile은 GGD_SecureConnect_Read를 사용하여 소켓에서 JSON 데이터를 가져옵니다. AWS IoT에서 JSON 데이터를 수신하려면 GGD_JSONRequestStart, GGD_SecureConnect_Send, GGD_JSONRequestGetSize를 호출해야 합니다.

GGD_GetIPandCertificateFromJSON

JSON 데이터에서 IP 주소 및 Greengrass 코어 인증서를 추출합니다. xAutoSelectFlag를 True로 설정하여 자동 선택을 켤 수 있습니다. 자동 선택에서는 FreeRTOS 디바이스를 연결할 수 있는 첫 번째 코어 디바이스를 찾습니다. Greengrass 코어에 연결하려면 GGD_SecureConnect_Connect 함수를 호출하여 코어 디바이스의 IP 주소, 포트 및 인증서를 전달합니다. 수동 선택을 사용하려면 HostParameters_t 파라미터의 다음 필드를 설정합니다.

pcGroupName

코어가 속한 Greengrass 그룹의 ID입니다. aws greengrass list-groups CLI 명령을 사용하여 Greengrass 그룹의 ID를 찾을 수 있습니다.

pcCoreAddress

연결할 Greengrass 코어의 ARN입니다.

coreHTTP 라이브러리

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

소형 IoT 디바이스(MCU 또는 소형 MPU)용 HTTP C 클라이언트 라이브러리

소개

coreHTTP 라이브러리는 HTTP/1.1 표준 하위 집합의 클라이언트 구현입니다. HTTP 표준은 TCP/IP를 기반으로 실행되는 상태 비저장 프로토콜을 제공하며 분산형 협업 하이퍼텍스트 정보 시스템에서 주로 사용됩니다.

coreHTTP 라이브러리는 HTTP/1.1 프로토콜 표준의 하위 집합을 구현합니다. 이 라이브러리는 메모리 사용량을 줄이도록 최적화되었습니다. 또한 애플리케이션이 동시성을 완전히 관리할 수 있도록 완전 동기식 API를 제공합니다. 고정 버퍼만 사용하므로 애플리케이션이 메모리 할당 전략을 완벽하게 제어할 수 있습니다.

이 라이브러리는 C로 작성되었으며 ISO C90 및 MISRA C:2012를 준수하도록 설계되었습니다. 이 라이브러리의 종속 항목은 표준 C 라이브러리와 Node.js의 http-parser LTS 버전(v12.19.1)뿐입니다. 이 라이브러리에는 안전한 메모리 사용과 힙 할당 없음을 보여주는 증거가 있어 IoT 마이크로컨트롤러에 적합할 뿐만 아니라 다른 플랫폼으로도 완벽하게 이식할 수 있습니다.

IoT 애플리케이션에서 HTTP 연결을 사용할 때는 coreHTTP 데모(상호 인증)에 설명된 대로 TLS 프로토콜을 사용하는 것과 같은 보안 전송 인터페이스를 사용하는 것이 좋습니다.

이 라이브러리는 자유롭게 사용할 수 있으며 MIT 오픈 소스 라이선스에 따라 배포됩니다.

coreHTTP 코드 크기(ARM Cortex-M용 GCC로 생성된 예제)		
파일	-O1 최적화	-Os 최적화
core_http_client.c	3.2K	2.6K
http_parser.c(http-parser)	15.7K	13.0K

coreHTTP 코드 크기(ARM Cortex-M용 GCC로 생성된 예제)		
파일	-O1 최적화	-Os 최적화
총 추정치	18.9K	15.6K

coreJSON 라이브러리

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

소개

JSON(JavaScript 객체 표기법)은 사람이 읽을 수 있는 데이터 직렬화 형식입니다. AWS IoT 디바이스 섀도우 서비스와 같이 데이터를 교환하는 데 널리 사용되며 GitHub REST API와 같은 여러 API의 일부입니다. JSON은 Ecma International이 표준으로 유지 관리합니다.

coreJSON 라이브러리는 ECMA-404 표준 JSON 데이터 교환 구문을 엄격하게 적용하면서 키 조회를 지원하는 파서를 제공합니다. C로 작성되었으며 ISO C90 및 MISRA C:2012를 준수하도록 설계되었습니다. 이 라이브러리에는 안전한 메모리 사용과 힙 할당 없음을 보여주는 증거가 있어 IoT 마이크로컨트롤러에 적합할 뿐만 아니라 다른 플랫폼으로도 완벽하게 이식할 수 있습니다.

메모리 사용

coreJSON 라이브러리는 내부 스택을 사용하여 JSON 문서의 중첩 구조를 추적합니다. 스택은 단일 함수 직접 호출 기간 동안 존재하며 보존되지 않습니다. 스택 크기는 매크로 `JSON_MAX_DEPTH`를 정의하여 지정할 수 있으며, 기본값은 32레벨입니다. 각 레벨은 1바이트를 소비합니다.

coreJSON 코드 크기(ARM Cortex-M용 GCC로 생성된 예제)		
파일	-O1 최적화	-Os 최적화
core_json.c	2.9K	2.4K
총 추정치	2.9K	2.4K

coreMQTT 라이브러리

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

소개

coreMQTT 라이브러리는 MQTT(Message Queue Telemetry Transport) 표준의 클라이언트 구현입니다. MQTT 표준은 TCP/IP를 기반으로 실행되는 간단한 게시/구독(또는 PubSub) 메시징 프로토콜을 제공하며 M2M(시스템 간) 및 사물 인터넷(IoT) 사용 사례에서 주로 사용됩니다.

coreMQTT 라이브러리는 MQTT 3.1.1 프로토콜 표준을 준수합니다. 이 라이브러리는 메모리 사용량을 줄이도록 최적화되었습니다. 이 라이브러리의 설계에는 QoS 0 MQTT PUBLISH 메시지만 사용하는 리소스가 제한된 플랫폼부터 TLS(전송 계층 보안) 연결을 통한 QoS 2 MQTT PUBLISH를 사용하는 리소스가 풍부한 플랫폼에 이르기까지 다양한 사용 사례가 포함됩니다. 이 라이브러리는 특정 사용 사례의 요구 사항에 정확히 맞도록 선택 및 결합할 수 있는 구성 가능한 함수 메뉴를 제공합니다.

C로 작성되었으며 ISO C90 및 MISRA C:2012를 준수하도록 설계되었습니다. 이 MQTT 라이브러리는 다음을 제외하고 추가 라이브러리에 종속되지 않습니다.

- 표준 C 라이브러리
- 고객이 구현한 네트워크 전송 인터페이스
- (선택 사항) 사용자가 구현한 플랫폼 시간 함수

이 라이브러리는 간단한 송수신 전송 인터페이스 사양을 제공하여 기본 네트워크 드라이버와 분리됩니다. 애플리케이션 작성자는 기존 전송 인터페이스를 선택하거나 해당 애플리케이션에 맞게 자체 전송 인터페이스를 구현할 수 있습니다.

이 라이브러리는 MQTT 브로커에 연결하고, 주제를 구독/구독 취소하고, 주제에 메시지를 게시하고, 수신 메시지를 수신할 수 있는 고급 API를 제공합니다. 이 API는 위에서 설명한 전송 인터페이스를 파라미터로 사용하고 이 인터페이스를 통해 MQTT 브로커와 메시지를 주고 받습니다.

또한 라이브러리는 하위 수준 serializer/deserializer API를 제공합니다. 이 API는 다른 오버헤드 없이 필요한 MQTT 기능의 하위 집합으로만 구성된 간단한 IoT 애플리케이션을 구축하는 데 사용할 수 있습

니다. serializer/deserializer API는 소켓과 같은 사용 가능한 모든 전송 계층 API와 함께 사용하여 브로커와 메시지를 주고 받을 수 있습니다.

IoT 애플리케이션에서 MQTT 연결을 사용할 때는 TLS 프로토콜을 사용하는 것과 같은 보안 전송 인터페이스를 사용하는 것이 좋습니다.

이 MQTT 라이브러리는 스레딩 또는 동기화와 같은 플랫폼 종속성이 없습니다. 이 라이브러리에는 안전한 메모리 사용과 힙 할당 없음을 보여주는 증거가 있어 IoT 마이크로컨트롤러에 적합할 뿐만 아니라 다른 플랫폼으로도 완벽하게 이식할 수 있습니다. 이 라이브러리는 자유롭게 사용할 수 있으며 MIT 오픈 소스 라이선스에 따라 배포됩니다.

coreMQTT 코드 크기(ARM Cortex-M용 GCC로 생성된 예제)		
파일	-O1 최적화	-Os 최적화
core_mqtt.c	3.0K	2.6K
core_mqtt_state.c	1.4K	1.1K
core_mqtt_serializer.c	2.5M	2.0K
총 추정치	6.9K	5.7K

OTA 에이전트 라이브러리

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

개요

OTA 에이전트 라이브러리를 사용하면 HTTP 또는 MQTT를 프로토콜로 사용하는 FreeRTOS 디바이스에 대한 펌웨어 업데이트의 알림, 다운로드 및 검증을 관리할 수 있습니다. OTA 에이전트 라이브러리를 사용하여 디바이스에서 실행 중인 애플리케이션과 펌웨어 업데이트를 논리적으로 분리할 수 있습니다. OTA 에이전트는 네트워크 연결을 애플리케이션과 공유할 수 있습니다. 네트워크 연결을 공유

하여 잠재적으로 많은 양의 RAM을 절약할 수 있습니다. 또한 OTA 에이전트 라이브러리를 사용하여 펌웨어 업데이트를 테스트, 커밋 또는 롤백하기 위한 애플리케이션별 로직을 정의할 수 있습니다.

FreeRTOS를 통한 OTA 업데이트 설정에 대한 자세한 내용은 FreeRTOS 무선 업데이트(OTA) 섹션을 참조하세요.

Features

전체 OTA 에이전트 인터페이스는 다음과 같습니다.

`OTA_AgentInit`

OTA 에이전트를 초기화합니다. 호출자가 메시징 프로토콜 컨텍스트, 선택적 콜백 및 제한 시간을 제공합니다.

`OTA_AgentShutdown`

OTA 에이전트를 사용한 이후에 리소스를 정리합니다.

`OTA_GetAgentState`

OTA 에이전트의 현재 상태를 가져옵니다.

`OTA_ActivateNewImage`

OTA를 통해 수신된 최신 마이크로 컨트롤러 펌웨어 이미지를 활성화합니다. 이제 상세 작업 상태가 자체 테스트됩니다.

`OTA_SetImageState`

현재 실행 중인 마이크로 컨트롤러 펌웨어 이미지의 확인 상태(테스트 중, 수락됨, 거부됨)를 설정합니다.

`OTA_GetImageState`

현재 실행 중인 마이크로 컨트롤러 펌웨어 이미지의 상태(테스트 중, 수락됨, 거부됨)를 가져옵니다.

`OTA_CheckForUpdate`

OTA 업데이트 서비스에서 사용 가능한 다음 OTA 업데이트를 요청합니다.

`OTA_Suspend`

모든 OTA 에이전트 작업을 일시 중단합니다.

`OTA_Resume`

OTA 에이전트 작업을 재개합니다.

API 참조

자세한 내용은 OTA 에이전트 API 참조를 참조하세요.

사용 예

MQTT 프로토콜을 사용하는 일반적인 OTA 지원 디바이스 애플리케이션은 다음과 같은 API 호출 시퀀스를 사용하여 OTA 에이전트를 구동합니다.

1. AWS IoT MQTT 브로커에 연결합니다. 자세한 내용은 coreMQTT 라이브러리 단원을 참조하십시오.
2. `OTA_AgentInit`을 호출하여 OTA 에이전트를 초기화합니다. 애플리케이션에서 사용자 지정 OTA 콜백 함수를 정의하거나 NULL 콜백 함수 포인터를 지정하여 기본 콜백을 사용할 수 있습니다. 또한 초기화 시간 제한을 입력해야 합니다.

   콜백은 OTA 업데이트 작업을 완료한 이후에 실행되는 애플리케이션별 로직을 구현합니다. 시간 제한은 초기화를 완료하는 동안 대기하는 시간을 정의합니다.
3. 에이전트가 준비되기 이전에 `OTA_AgentInit`이 시간 초과된 경우 `OTA_GetAgentState`를 호출하여 에이전트가 초기화되고 예상한 대로 작동하는지 확인할 수 있습니다.
4. OTA 업데이트가 완료되면 FreeRTOS는 `accepted`, `rejected`, `self test` 이벤트 중 하나를 사용하여 작업 완료 콜백을 호출합니다.
5. 유효성 검사 오류 등으로 인해 새 펌웨어 이미지가 거부된 경우 애플리케이션에서는 일반적으로 알림을 무시하고 다음 업데이트를 대기할 수 있습니다.
6. 업데이트가 유효하고 수락됨으로 표시된 경우 `OTA_ActivateNewImage`를 호출하여 디바이스를 재설정하고 새 펌웨어 이미지를 부팅합니다.

이식

OTA 기능을 플랫폼으로 이식하는 방법에 대한 자세한 내용은 FreeRTOS 이식 안내서의 OTA 라이브러리 이식을 참조하세요.

메모리 사용

AWS IoT OTA의 코드 크기(ARM Cortex-M용 GCC로 생성된 예)		
파일	-O1 최적화	-Os 최적화
ota.c	8.0K	7.2K
ota_interface.c	0.1K	0.1K
ota_base64.c	0.6K	0.6K
ota_mqtt.c	2.3K	2.2K
ota_cbor.c	0.8K	0.6K
ota_http.c	0.3K	0.3K
총 추정치	12.1K	11.0K

corePKCS11 라이브러리

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

개요

퍼블릭 키 암호화 표준 #11은 암호화 토큰을 관리하고 사용하기 위한 플랫폼 독립적 API를 정의합니다. PKCS #11은 표준에서 정의한 API와 표준 자체를 의미합니다. PKCS #11 암호화 API는 키 스토리지, 암호화 객체에 대한 get/set 속성 및 세션 의미 체계를 추상화합니다. 이는 일반적인 암호화 객체를 조작하는 데 널리 사용되며, 지정한 함수를 통해 애플리케이션 소프트웨어가 암호화 객체를 애플리케이션 메모리에 노출하지 않고도 사용, 생성, 수정 및 삭제할 수 있기 때문에 중요합니다. 예를 들어 FreeRTOS AWS 참조 통합은 PKCS #11 API의 작은 하위 집합을 사용하여 애플리케이션이 키를 '보지' 않고 TLS(전송 계층 보안) 프로토콜로 인증되고 보호되는 네트워크 연결을 생성하는 데 필요한 보안 암호(프라이빗) 키에 액세스합니다.

corePKCS11 라이브러리에는 Mbed TLS에서 제공하는 암호화 기능을 사용하는 PKCS #11 인터페이스(API)의 소프트웨어 기반 모의 구현이 포함되어 있습니다. 소프트웨어 모의를 사용하면 개발 속도 및 유연성이 향상되지만, 모의 객체를 프로덕션 디바이스에 사용되는 보안 키 스토리지 전용 구현으로 대체해야 합니다. 일반적으로 TPM(신뢰 플랫폼 모듈), HSM(하드웨어 보안 모듈), Secure Element와 같은 보안 암호화 프로세서 또는 기타 보안 하드웨어 엔클레이브의 공급업체는 하드웨어와 함께 PKCS #11 구현을 배포합니다. 따라서 corePKCS11 소프트웨어 전용 모의 라이브러리의 목적은 프로덕션 디바이스에서 암호화 프로세서 전용 PKCS #11 구현으로 전환하기 전에 신속한 프로토타이핑 및 개발을 가능하게 하는 비 하드웨어 전용 PKCS #11 구현을 제공하는 것입니다.

비대칭 키, 난수 생성, 해싱 등 PKCS #11 표준의 일부만 구현됩니다. 대상 사용 사례에는 소형 임베디드 디바이스에서의 TLS 인증을 위한 인증서 및 키 관리, 코드 서명 확인 등이 포함됩니다. FreeRTOS 소스 코드 리포지토리에서 OASIS 표준 기구로부터 가져온 `pkcs11.h` 파일을 참조하세요. FreeRTOS 참조 구현에서는 TLS 헬퍼 인터페이스가 `SOCKETS_Connect` 중에 TLS 클라이언트 인증을 위해 PKCS #11 API를 호출합니다. 1회 개발자 프로비저닝 워크플로우에서 인증을 위한 프라이빗 키와 TLS 클라이언트 인증서를 AWS IoT MQTT 브로커로 가져오기 위해서도 PKCS#11 API를 호출합니다. 이 두 사용 사례(프로비저닝 및 TLS 클라이언트 인증)에서는 PKCS#11 인터페이스 표준의 일부만 구현하면 됩니다.

Features

PKCS#11의 다음 하위 집합이 사용됩니다. 이 목록은 프로비저닝, TLS 클라이언트 인증 및 정리를 지원하기 위해 루틴을 호출하는 순서와 거의 동일합니다. 함수에 대한 자세한 내용은 표준 위원회에서 제공하는 PKCS#11 문서를 참조하세요.

일반 설정 및 제거 API

- `C_Initialize`
- `C_Finalize`
- `C_GetFunctionList`
- `C_GetSlotList`
- `C_GetTokenInfo`
- `C_OpenSession`
- `C_CloseSession`
- `C_Login`

프로비저닝 API

- C_CreateObject CKO_PRIVATE_KEY(디바이스 프라이빗 키용)
- C_CreateObject CKO_CERTIFICATE(디바이스 인증서 및 코드 확인 인증서용)
- C_GenerateKeyPair
- C_DestroyObject

클라이언트 인증

- C_GetAttributeValue
- C_FindObjectsInit
- C_FindObjects
- C_FindObjectsFinal
- C_GenerateRandom
- C_SignInit
- C_Sign
- C_VerifyInit
- C_Verify
- C_DigestInit
- C_DigestUpdate
- C_DigestFinal

비대칭 암호화 지원

FreeRTOS 참조 구현은 PKCS #11 2048비트 RSA(서명 전용) 및 NIST P-256 곡선의 ECDSA를 사용합니다. 다음 지침에서는 P-256 클라이언트 인증서를 기반으로 AWS IoT 사물을 생성하는 방법을 설명합니다.

AWS CLI 및 OpenSSL의 다음(또는 최신) 버전을 사용하고 있는지 확인합니다.

```
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
```

다음 절차에서는 aws configure 명령을 사용하여 AWS CLI를 구성했다고 가정합니다. 자세한 내용은 AWS Command Line Interface 사용 설명서에서 aws configure를 사용한 빠른 구성을 참조하세요.

P-256 클라이언트 인증서를 기반으로 AWS IoT 사물을 생성하려면

1. AWS IoT 사물을 생성합니다.

   ```
   aws iot create-thing --thing-name thing-name
   ```

2. OpenSSL을 사용해 P-256 키를 생성합니다.

   ```
   openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -pkeyopt
    ec_param_enc:named_curve -outform PEM -out thing-name.key
   ```

3. 2단계에서 생성한 키로 서명된 인증서 등록 요청을 생성합니다.

   ```
   openssl req -new -nodes -days 365 -key thing-name.key -out thing-name.req
   ```

4. 인증서 등록 요청을에 제출합니다 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. (이전 명령에서 출력된 ARN에서 참조되는) 인증서를 사물에 연결합니다.

   ```
   aws iot attach-thing-principal --thing-name thing-name \
     --principal "arn:aws:iot:us-
   east-1:123456789012:cert/
   86e41339a6d1bbc67abf31faf455092cdebf8f21ffbc67c4d238d1326c7de729"
   ```

6. 정책을 생성합니다. (이 정책은 지나치게 허용적입니다. 개발 목적으로만 사용해야 합니다.)

   ```
   aws iot create-policy --policy-name FullControl --policy-document file://
   policy.json
   ```

   다음은 `create-policy` 명령에 지정되는 policy.json 파일 목록입니다. Greengrass 연결 및 검색을 위해 FreeRTOS 데모를 실행하지 않으려면 `greengrass:*` 작업을 생략할 수 있습니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:*",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "greengrass:*",
      "Resource": "*"
    }
  ]
}
```

7. 보안 주체(인증서) 및 정책을 사물에 연결합니다.

```
aws iot attach-principal-policy --policy-name FullControl \
  --principal "arn:aws:iot:us-
east-1:123456789012:cert/
86e41339a6d1bbc67abf31faf455092cdebf8f21ffbc67c4d238d1326c7de729"
```

이제 이 안내서의 AWS IoT 시작하기 단원에 나오는 단계를 따릅니다. 생성한 인증서 및 프라이빗 키를 `aws_clientcredential_keys.h` 파일에 복사해야 합니다. 사물 이름을 `aws_clientcredential.h`에 복사합니다.

> **Note**
>
> 인증서와 프라이빗 키는 데모 용도로만 하드 코딩됩니다. 프로덕션 수준 애플리케이션은 이러한 파일을 보안 위치에 저장해야 합니다.

이식

corePKCS11 라이브러리를 플랫폼으로 이식하는 방법에 대한 자세한 내용은 FreeRTOS 이식 안내서의 corePKCS11 라이브러리 이식을 참조하세요.

메모리 사용

corePKCS11 코드 크기(ARM Cortex-M용 GCC로 생성된 예제)		
파일	-O1 최적화	-Os 최적화
core_pkcs11.c	0.8K	0.8K
core_pki_utils.c	0.5K	0.3K
core_pkcs11_mbedtls.c	8.7K	7.3K
총 추정치	10.0K	8.4K

보안 소켓 라이브러리

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

개요

FreeRTOS 보안 소켓 라이브러리를 사용하여 안전하게 통신하는 임베디드 애플리케이션을 생성할 수 있습니다. 이 라이브러리는 다양한 네트워크 프로그래밍 배경의 소프트웨어 개발자가 쉽게 온보딩할 수 있도록 설계되었습니다.

FreeRTOS 보안 소켓 라이브러리는 Berkeley 소켓 인터페이스를 기반으로 하며, TLS 프로토콜을 사용한 추가 보안 통신 옵션도 있습니다. FreeRTOS 보안 소켓 라이브러리와 Berkeley 소켓 인터페이스 간의 차이점에 대한 자세한 내용은 보안 소켓 API 참조의 `SOCKETS_SetSockOpt` 섹션을 참조하세요.

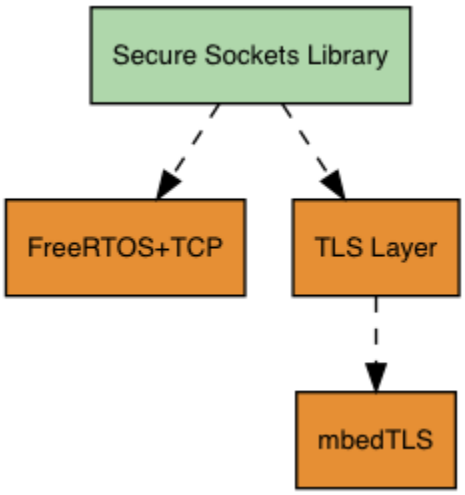
> **Note**
>
> 현재, FreeRTOS 보안 소켓에는 클라이언트 API 및 경량 IP(lwIP) 구현의 서버 측 `Bind` API만 지원됩니다.

종속성 및 요구 사항

FreeRTOS 보안 소켓 라이브러리는 TCP/IP 스택 및 TLS 구현에 종속됩니다. FreeRTOS의 포트는 다음 세 가지 방법 중 하나로 이러한 종속성을 충족합니다.

- TCP/IP 및 TLS의 사용자 지정 구현
- TCP/IP의 사용자 지정 구현 및 mbedTLS를 사용한 FreeRTOS TLS 계층
- FreeRTOS+TCP 및 mbedTLS를 사용한 FreeRTOS TLS 계층

아래 종속성 다이어그램은 FreeRTOS 보안 소켓 라이브러리에 포함된 참조 구현을 보여줍니다. 이 참조 구현은 FreeRTOS+TCP와 mbedTLS를 종속성으로 하여 이더넷과 Wi-Fi를 통한 TLS 및 TCP/IP를 지원합니다. FreeRTOS TLS 계층에 대한 자세한 내용은 전송 계층 보안 섹션을 참조하세요.



Features

FreeRTOS 보안 소켓 라이브러리 기능은 다음과 같습니다.

- 표준 Berkeley 소켓 기반 인터페이스
- 데이터 전송 및 수신을 위한 스레드 세이프 API
- 활성화하기 쉬운 TLS

문제 해결

오류 코드

FreeRTOS 보안 소켓 라이브러리가 반환하는 오류 코드는 음수 값입니다. 각 오류 코드에 대한 자세한 내용은 보안 소켓 API 참조의 보안 소켓 오류 코드를 참조하십시오.

> **Note**
>
> FreeRTOS 보안 소켓 API가 오류 코드를 반환하는 경우, FreeRTOS 보안 소켓 라이브러리에 종속된 coreMQTT 라이브러리는 오류 코드 `AWS_IOT_MQTT_SEND_ERROR`를 반환합니다.

개발자 지원

FreeRTOS 보안 소켓 라이브러리에는 IP 주소를 처리하기 위한 헬퍼 매크로 두 개가 포함되어 있습니다.

`SOCKETS_inet_addr_quick`

이 매크로는 네 개의 개별 숫자 옥텟으로 표현된 IP 주소를 네트워크 바이트 순서의 32비트 숫자로 표현된 IP 주소로 변환합니다.

`SOCKETS_inet_ntoa`

이 매크로는 네트워크 바이트 순서의 32비크 숫자로 표현된 IP 주소를 10진수 점 표기법의 문자열로 변환합니다.

사용 제한

TCP 소켓만 FreeRTOS 보안 소켓 라이브러리에서 지원됩니다. UDP 소켓은 지원되지 않습니다.

서버 API는 경량 IP(lwIP) 구현의 서버 측 `Bind` API를 제외하고 FreeRTOS 보안 소켓 라이브러리에서 지원되지 않습니다. 클라이언트 API는 지원됩니다.

Initialization(초기화)

FreeRTOS 보안 소켓 라이브러리를 사용하려면 라이브러리와 관련 종속성을 초기화해야 합니다. 보안 소켓 라이브러리를 초기화하려면 애플리케이션에서 다음 코드를 사용합니다.

```
BaseType_t xResult = pdPASS;
```

```
xResult = SOCKETS_Init();
```

종속 라이브러리는 따로 초기화해야 합니다. 예를 들어 FreeRTOS+TCP가 종속성인 경우 애플리케이션에서 FreeRTOS_IPInit도 호출해야 합니다.

API 참조

전체 API 참조는 보안 소켓 API 참조를 참조하세요.

사용 예

다음 코드는 클라이언트를 서버에 연결합니다.

```
#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 );
}
```

전체 예제는 보안 소켓 에코 클라이언트 데모 단원을 참조하십시오.

이식

FreeRTOS 보안 소켓은 TCP/IP 스택 및 TLS 구현에 종속됩니다. 스택에 따라 보안 소켓 라이브러리를 이식하려면 다음 중 일부를 이식해야 할 수 있습니다.

- FreeRTOS+TCP TCP/IP 스택
- corePKCS11 라이브러리은
- 전송 계층 보안은

이식에 대한 자세한 내용은 FreeRTOS 이식 안내서의 보안 소켓 라이브러리 이식을 참조하세요.

AWS IoT 디바이스 섀도우 라이브러리

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

소개

AWS IoT Device Shadow 라이브러리를 사용하여 등록된 모든 디바이스의 현재 상태(섀도우)를 저장하고 검색할 수 있습니다. 디바이스의 섀도우는 디바이스의 영구적 가상 표현으로, 디바이스가 오프라인 상태인 경우에도 웹 애플리케이션에서 상호 작용할 수 있습니다. 디바이스 상태는 JSON 문서에 섀도우로 캡처됩니다. MQTT 또는 HTTP를 통해 AWS IoT Device Shadow 서비스에 명령을 전송하여 알려진 최신 디바이스 상태를 쿼리하거나 상태를 변경할 수 있습니다. 각 디바이스의 섀도우는 해당 사물의 이름, 즉 AWS 클라우드 상의 특정 디바이스 또는 논리적 엔티티의 표현으로 고유하게 식별됩니다. 자세한 내용은 AWS IoT를 사용하여 디바이스 관리를 참조하세요. 섀도우에 대한 자세한 내용은 AWS IoT 설명서에서 확인할 수 있습니다.

AWS IoT 디바이스 섀도우 라이브러리에는 표준 C 라이브러리 이외의 추가 라이브러리에 대한 종속성이 없습니다. 또한 스레딩 또는 동기화와 같은 플랫폼 종속성도 없습니다. 모든 MQTT 라이브러리 및 모든 JSON 라이브러리와 함께 사용할 수 있습니다.

이 라이브러리는 자유롭게 사용할 수 있으며 MIT 오픈 소스 라이선스에 따라 배포됩니다.

AWS IoT 디바이스 섀도우의 코드 크기(ARM Cortex-M용 GCC로 생성된 예)		
파일	-O1 최적화	-Os 최적화
shadow.c	1.2K	0.9K
총 추정치	1.2K	0.9K

AWS IoT 작업 라이브러리

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

소개

AWS IoT 작업은 하나 이상의 연결된 디바이스에 보류 중인 작업을 알리는 서비스입니다. 작업을 사용하여 디바이스 플릿을 관리하거나, 디바이스의 펌웨어 및 보안 인증서를 업데이트하거나, 디바이스 재시작 및 진단과 같은 관리 작업을 수행할 수 있습니다. 자세한 내용은 AWS IoT 개발자 가이드의 작업

섹션을 참조하세요. AWS IoT 작업 서비스와의 상호 작용은 경량 게시-구독 프로토콜인 MQTT를 사용합니다. 이 라이브러리는 AWS IoT Jobs 서비스에서 사용하는 MQTT 주제 문자열을 작성하고 인식하는 API를 제공합니다.

AWS IoT 작업 라이브러리는 C로 작성되며 ISO C90 및 MISRA C:2012를 준수하도록 설계되었습니다. 이 라이브러리는 표준 C 라이브러리 이외의 추가 라이브러리에 대한 종속성이 없습니다. 모든 MQTT 라이브러리 및 모든 JSON 라이브러리와 함께 사용할 수 있습니다. 이 라이브러리에는 안전한 메모리 사용과 힙 할당 없음을 보여주는 증거가 있어 IoT 마이크로컨트롤러에 적합할 뿐만 아니라 다른 플랫폼으로도 완벽하게 이식할 수 있습니다.

이 라이브러리는 자유롭게 사용할 수 있으며 MIT 오픈 소스 라이선스에 따라 배포됩니다.

AWS IoT 작업의 코드 크기(ARM Cortex-M용 GCC로 생성된 예)		
파일	-O1 최적화	-Os 최적화
jobs.c	1.8K	1.5K
총 추정치	1.8K	1.5K

전송 계층 보안

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

FreeRTOS 전송 계층 보안(TLS) 인터페이스는 프로토콜 스택의 상위에 있는 Secure Sockets Layer(SSL) 인터페이스로부터 암호화 구현 세부 정보를 추출하는 데 사용되는 선택적 씬 래퍼입니다. TLS 인터페이스의 목적은 현재 소프트웨어 암호화 라이브러리인 mbed TLS를 TLS 프로토콜 협상 및 암호화 프리미티브를 위한 대체 구현과 쉽게 교체할 수 있도록 만드는 것입니다. SSL 인터페이스를 변경할 필요 없이 TLS 인터페이스를 교체할 수 있습니다. FreeRTOS 소스 코드 리포지토리의 `iot_tls.h`를 참조하세요.

SSL에서 암호화 라이브러리로 직접 인터페이스할 수 있으므로 TLS 인터페이스는 선택 사항입니다. 이 인터페이스는 네트워크 전송 및 TLS의 전체 스택 오프로드 구현을 포함하는 MCU 솔루션에는 사용되지 않습니다.

TLS 인터페이스 이식에 대한 자세한 내용은 FreeRTOS 이식 안내서의 TLS 라이브러리 이식을 참조하세요.

Wi-Fi 라이브러리

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

개요

FreeRTOS Wi-Fi 라이브러리는 포트별 Wi-Fi 구현을 공통 API로 추상화하여 Wi-Fi 기능을 갖춘 모든 FreeRTOS 적격 보드의 애플리케이션 개발 및 이식을 간소화합니다. 애플리케이션은 이 공통 API를 사용하여 공통 인터페이스를 통해 하위 수준의 무선 스택과 통신할 수 있습니다.

종속성 및 요구 사항

FreeRTOS Wi-Fi 라이브러리에는 FreeRTOS+TCP 코어가 필요합니다.

Features

Wi-Fi 라이브러리에는 다음 기능이 있습니다.

- WEP, WPA, WPA2 및 WPA3 인증 지원
- 액세스 포인트 검색
- 전력 관리
- 네트워크 프로파일링

Wi-Fi 라이브러리의 기능에 대한 자세한 내용은 아래를 참조하십시오.

Wi-Fi 모드

Wi-Fi 디바이스는 스테이션, 액세스 포인트 또는 P2P 등의 세 가지 모드 중 하나일 수 있습니다. `WIFI_GetMode`를 호출하여 Wi-Fi 디바이스의 현재 모드를 가져올 수 있습니다. `WIFI_SetMode`를 호출하여 디바이스의 Wi-Fi 모드를 설정할 수 있습니다. `WIFI_SetMode`를 호출하여 모드를 전환하면 네트워크에 이미 연결된 디바이스의 연결이 해제됩니다.

스테이션 모드

보드를 기존 액세스 포인트에 연결하려면 디바이스를 스테이션 모드로 설정합니다.

액세스 포인트(AP) 모드

디바이스를 다른 디바이스를 연결하기 위한 액세스 포인트로 만들려면 디바이스를 AP 모드로 설정합니다. 디바이스가 AP 모드에 있는 경우, 다른 디바이스를 FreeRTOS 디바이스에 연결하고 새 Wi-Fi 자격 증명을 구성할 수 있습니다. AP 모드를 구성하려면 `WIFI_ConfigureAP`를 호출합니다. 디바이스를 AP 모드로 전환하려면 `WIFI_StartAP`를 호출합니다. AP 모드를 끄려면 `WIFI_StopAP`를 호출합니다.

> **Note**
>
> FreeRTOS 라이브러리는 AP 모드에서 Wi-Fi 프로비저닝을 제공하지 않습니다. AP 모드를 완벽하게 지원하려면 DHCP 및 HTTP 서버 기능을 비롯한 추가 기능을 제공해야 합니다.

P2P 모드

액세스 포인트 없이 여러 디바이스가 서로 직접 연결할 수 있게 하려면 디바이스를 P2P 모드로 설정합니다.

보안

Wi-Fi API는 WEP, WPA, WPA2 및 WPA3 보안 유형을 지원합니다. 디바이스가 스테이션 모드에 있는 경우, `WIFI_ConnectAP` 함수를 호출할 때 네트워크 보안 유형을 지정해야 합니다. 디바이스가 AP 모드에 있는 경우 지원되는 보안 유형을 사용하도록 디바이스를 구성할 수 있습니다.

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

검색 및 연결

인근의 액세스 포인트를 검색하려면 디바이스를 스테이션 모드로 설정하고 `WIFI_Scan` 함수를 호출합니다. 스캔에서 원하는 네트워크를 찾은 경우 `WIFI_ConnectAP`를 호출하고 네트워크 자격 증명을

제공하여 네트워크에 연결할 수 있습니다. `WIFI_Disconnect`를 호출하여 네트워크에서 Wi-Fi 디바이스를 연결 해제할 수 있습니다. 검색 및 연결에 대한 자세한 내용은 사용 예 및 API 참조를 참조하십시오.

전력 관리

애플리케이션과 사용 가능한 전원에 따라 Wi-Fi 디바이스마다 전력 요구 사항이 다릅니다. 디바이스를 항상 켜서 지연 시간을 줄이거나, 디바이스를 간헐적으로 연결한 후 Wi-Fi가 필요하지 않을 때 저전력 모드로 전환할 수 있습니다. 인터페이스 API는 항상 연결, 저전력, 일반 모드 등 다양한 전력 관리 모드를 지원합니다. `WIFI_SetPMMode` 함수를 사용하여 디바이스에 대한 전력 모드를 설정합니다. `WIFI_GetPMMode` 함수를 호출하여 디바이스의 현재 전력 모드를 가져올 수 있습니다.

네트워크 프로필

Wi-Fi 라이브러리를 사용하여 네트워크 프로필을 디바이스의 비휘발성 메모리에 저장할 수 있습니다. 따라서 디바이스가 Wi-Fi 네트워크에 다시 연결될 때 검색할 수 있도록 네트워크 설정을 저장할 수 있습니다. 그러면 네트워크에 연결된 이후에 디바이스를 다시 프로비저닝할 필요가 없습니다. `WIFI_NetworkAdd`는 네트워크 프로필을 추가하고, `WIFI_NetworkGet`은 네트워크 프로필을 검색하고, `WIFI_NetworkDel`은 네트워크 프로필을 삭제합니다. 저장할 수 있는 프로필 수는 플랫폼에 따라 다릅니다.

구성

Wi-Fi 라이브러리를 사용하려면 구성 파일에서 여러 식별자를 정의해야 합니다. 이러한 식별자에 대한 자세한 내용은 API 참조를 참조하십시오.

> **Note**
>
> 이 라이브러리에는 필요한 구성 파일이 포함되어 있지 않으므로 하나를 생성해야 합니다. 구성 파일을 생성할 때 보드에 필요한 보드 관련 구성 식별자를 포함시켜야 합니다.

Initialization(초기화)

Wi-Fi 라이브러리를 사용하기 전에 FreeRTOS 구성 요소 외에도 일부 보드 관련 구성 요소를 초기화해야 합니다. `vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/application_code/main.c` 파일을 초기화를 위한 템플릿으로 사용하여 다음을 수행합니다.

1. 애플리케이션이 Wi-Fi 연결을 처리하는 경우 `main.c`에서 샘플 Wi-Fi 연결 로직을 제거합니다. 다음 `DEMO_RUNNER_RunDemos()` 함수 호출을

```
if( SYSTEM_Init() == pdPASS )
    {
    ...
        DEMO_RUNNER_RunDemos();
    ...
        }
```

자체 애플리케이션 호출로 바꿉니다.

```
if( SYSTEM_Init() == pdPASS )
    {
    ...
        // This function should create any tasks
        // that your application requires to run.
        YOUR_APP_FUNCTION();
    ...
        }
```

2. `WIFI_On()`을 호출하여 Wi-Fi 칩을 초기화하고 전원을 켭니다.

> **Note**
>
> 일부 보드에는 추가 하드웨어 초기화가 필요할 수 있습니다.

3. 구성된 `WIFINetworkParams_t` 구조를 `WIFI_ConnectAP()`로 전달하여 보드를 사용 가능한 Wi-Fi 네트워크에 연결합니다. `WIFINetworkParams_t` 구조에 대한 자세한 내용은 사용 예 및 API 참조를 참조하세요.

API 참조

전체 API 참조는 Wi-Fi API 참조를 참조하십시오.

사용 예

알려진 AP에 연결

```
#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
}
```

인근 AP 검색

```
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);
}
```

이식

`iot_wifi.c` 구현은 `iot_wifi.h`에 정의된 함수를 구현해야 합니다. 적어도 이 구현은 필수적이지 않거나 지원되지 않는 함수에 대해 `eWiFiNotSupported`를 반환해야 합니다.

Wi-Fi 라이브러리 이식에 대한 자세한 내용은 FreeRTOS 이식 안내서의 Wi-Fi 라이브러리 이식을 참조하세요.

FreeRTOS 데모

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

FreeRTOS는 주 FreeRTOS 디렉터리의 demos 폴더에 몇 개의 데모 애플리케이션을 포함하고 있습니다. FreeRTOS에서 실행할 수 있는 모든 예제는 demos의 common 폴더에 있습니다. 또한 demos 폴더에는 FreeRTOS 적격 플랫폼별로 폴더가 있습니다.

데모 애플리케이션을 사용하기 전에 FreeRTOS 시작하기에서 자습서를 완료하는 것이 좋습니다. coreMQTT 상호 인증 데모를 설정하고 실행하는 방법을 소개합니다.

FreeRTOS 데모 실행

다음 주제에서는 FreeRTOS 데모를 설정하고 실행하는 방법을 설명합니다.

- Bluetooth Low Energy 데모 애플리케이션
- Microchip Curiosity PIC32MZEF용 데모 부트로더
- AWS IoT Device Defender 데모
- AWS IoT Greengrass 검색 데모 애플리케이션
- coreHTTP 데모
- AWS IoT 작업 라이브러리 데모
- coreMQTT 데모
- OTA(Over-the-Air) 업데이트 데모 애플리케이션
- 보안 소켓 에코 클라이언트 데모
- AWS IoT 디바이스 섀도우 데모 애플리케이션

freertos/demos/demo_runner/iot_demo_runner.c 파일에 있는 DEMO_RUNNER_RunDemos 함수는 단일 데모 애플리케이션이 실행되는 분리된 스레드를 초기화합니다. 기본적으로 DEMO_RUNNER_RunDemos는 coreMQTT 상호 인증 데모만 호출하고 시작합니다. FreeRTOS를 다운로드할 때 선택한 구성에 따라 또는 FreeRTOS를 다운로드한 위치에 따라 다른 예제 실행기 함수가 기본적으로 시작될 수 있습니다. 데모 애플리케이션을 활성화하려면 *freertos*/vendors/*vendor*/

boards/*board*/aws_demos/config_files/aws_demo_config.h 파일을 열고 실행할 데모를 정의합니다.

> **Note**
>
> 모든 예제 조합이 함께 작용하는 것은 아닙니다. 조합에 따라, 메모리 제한으로 인해 선택한 대상에서 소프트웨어를 실행하지 못할 수도 있습니다. 한 번에 한 개의 데모를 실행할 것을 권장합니다.

데모 구성

이 데모는 사용자가 빠르게 시작할 수 있도록 구성되었습니다. 해당 프로젝트에 맞게 일부 구성을 변경하여 해당 플랫폼에서 실행하는 버전을 만들 수도 있습니다. 구성 파일은 vendors/*vendor*/boards/*board*/aws_demos/config_files에서 찾을 수 있습니다.

Bluetooth Low Energy 데모 애플리케이션

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

개요

FreeRTOS Bluetooth Low Energy에는 세 개의 데모 애플리케이션이 포함되어 있습니다.

- MQTT over Bluetooth Low Energy 데모

  이 애플리케이션은 MQTT over Bluetooth Low Energy 서비스를 사용하는 방법을 보여 줍니다.
- Wi-Fi 프로비저닝 데모

  이 애플리케이션은 Bluetooth Low Energy Wi-Fi 프로비저닝 서비스를 사용하는 방법을 보여 줍니다.
- 일반 속성 서버 데모

  이 애플리케이션은 FreeRTOS Bluetooth Low Energy 미들웨어 API를 사용하여 간단한 GATT 서버를 생성하는 방법을 보여 줍니다.

> **Note**
>
> FreeRTOS 데모를 설정하고 실행하려면 FreeRTOS 시작하기의 단계를 따릅니다.

사전 조건

이 데모를 따라 수행하려면 Bluetooth Low Energy 기능을 지원하는 마이크로 컨트롤러가 필요합니다. 또한 FreeRTOS Bluetooth 디바이스용 iOS SDK 또는 FreeRTOS Bluetooth 디바이스용 Android SDK이 필요합니다.

FreeRTOS Bluetooth Low Energy용 AWS IoT 및 Amazon Cognito 설정

MQTT AWS IoT 에서 디바이스를에 연결하려면 AWS IoT 및 Amazon Cognito를 설정해야 합니다.

설정하려면 AWS IoT

1. https://aws.amazon.com/ AWS 계정을 설정합니다.
2. AWS IoT 콘솔을 열고 탐색 창에서 관리를 선택한 후 사물을 선택합니다.
3. 생성을 선택한 후 단일 사물 생성을 선택합니다.
4. 디바이스에 대해 기억하기 쉬운 이름을 입력한 후 다음을 선택합니다.
5. 모바일 디바이스를 통해 클라우드에 마이크로 컨트롤러를 연결하는 경우 인증서없이 사물 생성을 선택합니다. Mobile SDK는 디바이스 인증에 Amazon Cognito를 사용하기 때문에 Bluetooth Low Energy를 사용하는 데모용 디바이스 인증서를 생성할 필요가 없습니다.

   WiFi를 통해 마이크로 컨트롤러를 클라우드에 직접 연결하는 경우 인증서 생성을 선택하고 활성화를 선택한 후 사물 인증서, 퍼블릭 키, 프라이빗 키를 다운로드합니다.
6. 등록된 사물 목록에서 방금 생성한 사물을 선택한 다음 사물 페이지에서 상호 작용을 선택합니다. AWS IoT REST API 엔드포인트를 기록해 둡니다.

설정에 대한 자세한 내용은 시작하기를 참조하세요 AWS IoT.

Amazon Cognito 사용자 풀을 생성하려면

1. Amazon Cognito 콘솔로 이동하여 사용자 풀 관리를 선택합니다.
2. 사용자 풀 생성을 선택합니다.
3. 사용자 풀에 이름을 지정한 다음 기본값 검토를 선택합니다.

4. 왼쪽 탐색 창에서 앱 클라이언트를 선택한 다음 앱 클라이언트 추가를 선택합니다.
5. 앱 클라이언트의 이름을 입력한 후 앱 클라이언트 생성을 선택합니다.
6. 왼쪽 탐색 창에서 검토를 선택한 후 풀 생성을 선택합니다.

   사용자 풀의 일반 설정 페이지에 나타나는 풀 ID를 기록해 둡니다.
7. 왼쪽 탐색 창에서 앱 클라이언트를 선택한 다음 세부 정보 표시를 선택합니다. 앱 클라이언트 ID와 앱 클라이언트 암호를 메모합니다.

Amazon Cognito 자격 증명 풀을 생성하려면

1. Amazon Cognito 콘솔에서 자격 증명 풀 관리를 선택합니다.
2. 자격 증명 풀의 이름을 입력합니다.
3. 인증 공급자를 확장하고 Cognito 탭을 선택한 후 사용자 풀 ID와 앱 클라이언트 ID를 입력합니다.
4. 풀 생성을 선택합니다.
5. 세부 정보 보기를 확장하고 2개의 IAM 역할 이름을 적어 둡니다. 허용을 선택하여 Amazon Cognito에 액세스할 인증된 자격 증명 및 인증되지 않은 자격 증명을 생성합니다.
6. 자격 증명 풀 편집을 선택합니다. 자격 증명 풀 ID를 메모합니다. `us-west-2:12345678-1234-1234-1234-123456789012` 형식이어야 합니다.

Amazon Cognito 설정에 대한 자세한 내용은 Amazon Cognito 시작하기를 참조하세요.

IAM 정책을 생성하여 인증된 자격 증명에 연결하려면

1. IAM 콘솔을 열고 탐색 창에서 역할을 선택합니다.
2. 인증된 자격 증명의 역할을 검색 및 선택하고 Attach policies(정책 연결)를 선택한 후 Add inline policy(인라인 정책 추가)를 선택합니다.
3. JSON 탭을 선택하고 다음 JSON을 붙여 넣습니다.

```
{
   "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. 정책 검토를 선택하고 정책 이름을 입력한 후 정책 생성을 선택합니다.

AWS IoT 및 Amazon Cognito 정보를 보관합니다. AWS 클라우드로 모바일 애플리케이션을 인증하려면 엔드포인트와 IDs가 필요합니다.

Bluetooth Low Energy용 FreeRTOS 환경 설정

환경을 설정하려면 마이크로컨트롤러에 Bluetooth Low Energy 라이브러리와 함께 FreeRTOS를 다운로드해야 하며 모바일 디바이스에 FreeRTOS Bluetooth 디바이스용 Mobile SDK를 다운로드하고 구성해야 합니다.

FreeRTOS Bluetooth Low Energy로 마이크로컨트롤러 환경을 설정하려면

1. GitHub에서 FreeRTOS를 복제하거나 다운로드합니다. 자세한 내용은 README.md 파일을 참조하십시오.
2. 마이크로컨트롤러에 FreeRTOS를 설정합니다.

   FreeRTOS 적격 마이크로컨트롤러에서 FreeRTOS를 시작하는 방법에 대한 자세한 내용은 FreeRTOS 시작하기에서 해당 보드에 대한 안내서를 참조하세요.

   > **Note**
   >
   > FreeRTOS 및 이식된 FreeRTOS Bluetooth Low Energy 라이브러리를 지원하는 모든 Bluetooth Low Energy 사용 마이크로컨트롤러에서 데모를 실행할 수 있습니다. 현재 FreeRTOS MQTT over Bluetooth Low Energy 데모 프로젝트는 다음 Bluetooth Low Energy 사용 디바이스에 완벽하게 이식됩니다.

- Espressif ESP32-DevKitC 및 ESP-WROVER-KIT
- Nordic nRF52840-DK

공통 구성 요소

FreeRTOS 데모 애플리케이션에는 두 개의 공통 요소가 있습니다.

- Network Manager
- Bluetooth Low Energy Mobile SDK 데모 애플리케이션

Network Manager

Network Manager는 마이크로 컨트롤러의 네트워크 연결을 관리하며, `demos/network_manager/aws_iot_network_manager.c`의 FreeRTOS 디렉터리에 있습니다. Network Manager를 Wi-Fi와 Bluetooth Low Energy 모두에 대해 활성화한 경우 데모는 기본적으로 Bluetooth Low Energy로 시작합니다. Bluetooth Low Energy 연결이 끊어지고 보드가 Wi-Fi를 지원하는 경우 Network Manager는 사용 가능한 Wi-Fi 연결로 전환하여 네트워크 연결이 끊어지지 않도록 보호합니다.

Network Manager를 사용하여 네트워크 연결 유형을 활성화하려면 `vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/aws_iot_network_config.h`의 `configENABLED_NETWORKS` 파라미터에 네트워크 연결 유형을 추가합니다. 여기서 *`vendor`*는 공급업체의 이름이고 *`board`*는 데모를 실행하는 데 사용하는 보드 이름입니다.

예를 들어, Bluetooth Low Energy와 Wi-Fi를 모두 활성화한 경우 `aws_iot_network_config.h`에서 `#define configENABLED_NETWORKS`로 시작하는 행은 다음과 같습니다.

```
#define  configENABLED_NETWORKS  ( AWSIOT_NETWORK_TYPE_BLE | AWSIOT_NETWORK_TYPE_WIFI )
```

현재 지원되는 네트워크 연결 유형의 목록을 가져오려면 `aws_iot_network.h`에서 `#define AWSIOT_NETWORK_TYPE`으로 시작하는 줄을 참조하십시오.

FreeRTOS Bluetooth Low Energy Mobile SDK 데모 애플리케이션

FreeRTOS Bluetooth Low Energy Mobile SDK 데모 애플리케이션은 GitHub에서 Android SDK for FreeRTOS Bluetooth Devices 아래의 `amazon-freertos-ble-android-sdk/app` 및 iOS SDK for FreeRTOS Bluetooth Devices 아래의 `amazon-freertos-ble-ios-sdk/Example/`

AmazonFreeRTOSDemo에 있습니다. 이 예에서는 iOS 버전의 데모 모바일 애플리케이션의 스크린샷을 사용합니다.

> **Note**
>
> iOS 디바이스를 사용하는 경우 데모 모바일 애플리케이션을 빌드하려면 Xcode가 필요합니다. Android 디바이스를 사용하는 경우 Android Studio를 사용하여 데모 모바일 애플리케이션을 빌드할 수 있습니다.

iOS SDK 데모 애플리케이션을 구성하려면

구성 변수를 정의할 경우 구성 파일에 제공된 자리 표시자 값의 형식을 사용합니다.

1. FreeRTOS Bluetooth 디바이스용 iOS SDK이 설치되어 있는지 확인합니다.
2. amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/에서 다음 명령을 실행합니다.

   ```
   $ pod install
   ```

3. Xcode를 사용하여 amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/AmazonFreeRTOSDemo.xcworkspace 프로젝트를 열고 서명 개발자 계정을 사용자 계정으로 변경합니다.
4. 리전에서 AWS IoT 정책을 생성합니다(아직 생성하지 않은 경우).

   > **Note**
   >
   > 이 정책은 Amazon Cognito 인증 자격 증명에 대해 생성된 IAM 정책과 다릅니다.

   a. AWS IoT 콘솔을 엽니다.
   b. 탐색 창에서 Secure(보안)를 선택하고 Policies(정책)를 선택한 다음 Create(생성)를 선택합니다. 정책을 식별할 이름을 입력합니다. Add statements(설명문 추가) 섹션에서 Advanced mode(고급 모드)를 선택합니다. 다음 JSON을 복사하여 정책 편집기 창에 붙여 넣습니다. *aws-region* 및 *aws-account*를 AWS 리전 및 계정 ID로 바꿉니다.

      ```
      {
          "Version": "2012-10-17",
          "Statement": [
      ```

```
        {
            "Effect": "Allow",
            "Action": "iot:Connect",
            "Resource":"arn:aws:iot:region:account-id:*"
        },
        {
            "Effect": "Allow",
            "Action": "iot:Publish",
            "Resource": "arn:aws:iot:region:account-id:*"
        },
        {
             "Effect": "Allow",
             "Action": "iot:Subscribe",
             "Resource": "arn:aws:iot:region:account-id:*"
        },
        {
             "Effect": "Allow",
             "Action": "iot:Receive",
             "Resource": "arn:aws:iot:region:account-id:*"
        }
    ]
}
```

c. 생성(Create)을 선택합니다.

5. `amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/AmazonFreeRTOSDemo/Amazon/AmazonConstants.swift`를 열고 다음 변수를 재정의합니다.

   - `region`: AWS 리전.
   - `iotPolicyName`: AWS IoT 정책 이름입니다.
   - `mqttCustomTopic`: 게시하려는 MQTT 주제.

6. `amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/AmazonFreeRTOSDemo/Support/awsconfiguration.json`를 엽니다.

   `CognitoIdentity` 아래에서 다음 변수를 재정의합니다.

   - `PoolId`: Amazon Cognito 자격 증명 풀 ID.
   - `Region`: AWS 리전.

   `CognitoUserPool` 아래에서 다음 변수를 재정의합니다.

- `PoolId`: Amazon Cognito 사용자 풀 ID.
- `AppClientId`: 해당 앱 클라이언트 ID.
- `AppClientSecret`: 해당 앱 클라이언트 암호.
- `Region`: AWS 리전.

Android SDK 데모 애플리케이션을 구성하려면

구성 변수를 정의할 경우 구성 파일에 제공된 자리 표시자 값의 형식을 사용합니다.

1. FreeRTOS Bluetooth 디바이스용 Android SDK이 설치되어 있는지 확인합니다.
2. 리전에서 AWS IoT 정책을 생성합니다(아직 생성하지 않은 경우).

> **Note**
>
> 이 정책은 Amazon Cognito 인증 자격 증명에 대해 생성된 IAM 정책과 다릅니다.

   a. AWS IoT 콘솔을 엽니다.

   b. 탐색 창에서 Secure(보안)를 선택하고 Policies(정책)를 선택한 다음 Create(생성)를 선택합니다. 정책을 식별할 이름을 입력합니다. Add statements(설명문 추가) 섹션에서 Advanced mode(고급 모드)를 선택합니다. 다음 JSON을 복사하여 정책 편집기 창에 붙여 넣습니다. *aws-region* 및 *aws-account*를 AWS 리전 및 계정 ID로 바꿉니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:Connect",
            "Resource":"arn:aws:iot:region:account-id:*"
        },
        {
            "Effect": "Allow",
            "Action": "iot:Publish",
            "Resource": "arn:aws:iot:region:account-id:*"
        },
        {
             "Effect": "Allow",
```

```
            "Action": "iot:Subscribe",
            "Resource": "arn:aws:iot:region:account-id:*"
        },
        {
            "Effect": "Allow",
            "Action": "iot:Receive",
            "Resource": "arn:aws:iot:region:account-id:*"
        }
    ]
}
```

c. 생성(Create)을 선택합니다.

3. https://github.com/aws/amazon-freertos-ble-android-sdk/blob/master/app/src/main/java/software/amazon/freertos/demo/DemoConstants.java를 열고 다음 변수를 재정의합니다.

   - `AWS_IOT_POLICY_NAME`: AWS IoT 정책 이름입니다.
   - `AWS_IOT_REGION`: AWS 리전.

4. https://github.com/aws/amazon-freertos-ble-android-sdk/blob/master/app/src/main/res/raw/awsconfiguration.json을 엽니다.

   `CognitoIdentity` 아래에서 다음 변수를 재정의합니다.

   - `PoolId`: Amazon Cognito 자격 증명 풀 ID.
   - `Region`: AWS 리전.

   `CognitoUserPool` 아래에서 다음 변수를 재정의합니다.

   - `PoolId`: Amazon Cognito 사용자 풀 ID.
   - `AppClientId`: 해당 앱 클라이언트 ID.
   - `AppClientSecret`: 해당 앱 클라이언트 암호.
   - `Region`: AWS 리전.

Bluetooth Low Energy를 통해 마이크로 컨트롤러와의 보안 연결을 찾아보고 설정하려면

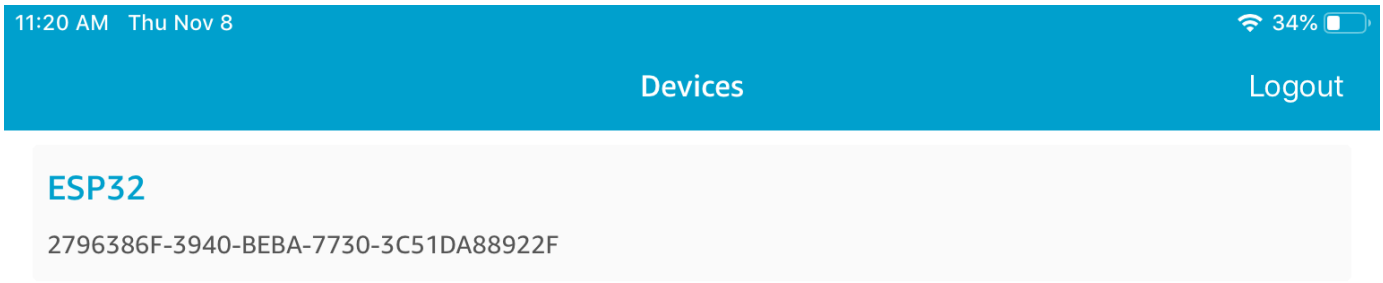
1. 마이크로컨트롤러와 모바일 디바이스를 안전하게 페어링하려면(6단계) 입력 및 출력 기능을 모두 갖춘 직렬 터미널 에뮬레이터(예: TeraTerm)가 필요합니다. 터미널 에뮬레이터 설치의 지침에 따라 터미널을 직렬 연결로 보드에 연결하도록 구성합니다.
2. 마이크로 컨트롤러에서 Bluetooth Low Energy 데모 프로젝트를 실행합니다.

3. 모바일 디바이스에서 Bluetooth Low Energy Mobile SDK 데모 애플리케이션을 실행합니다.

   명령줄에서 Android SDK의 데모 애플리케이션을 시작하려면 다음 명령을 실행하십시오.

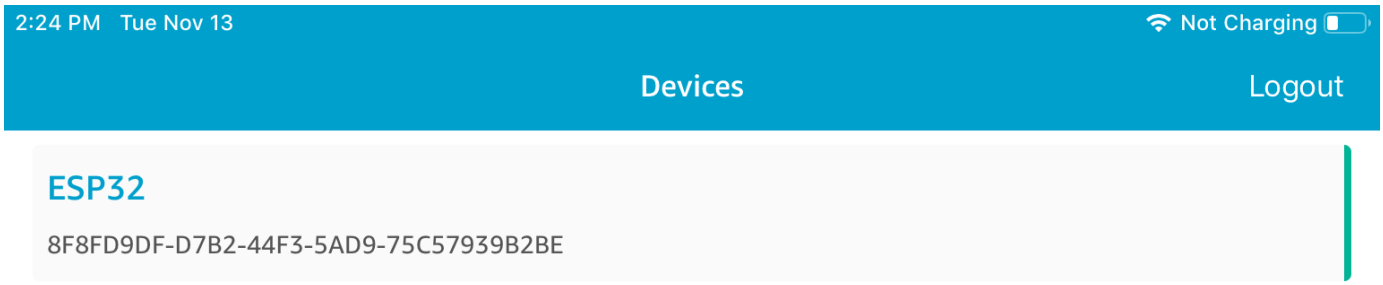
   ```
   $ ./gradlew installDebug
   ```

4. Bluetooth Low Energy Mobile SDK 데모 앱에서 디바이스 아래에 마이크로 컨트롤러가 표시되는지 확인합니다.

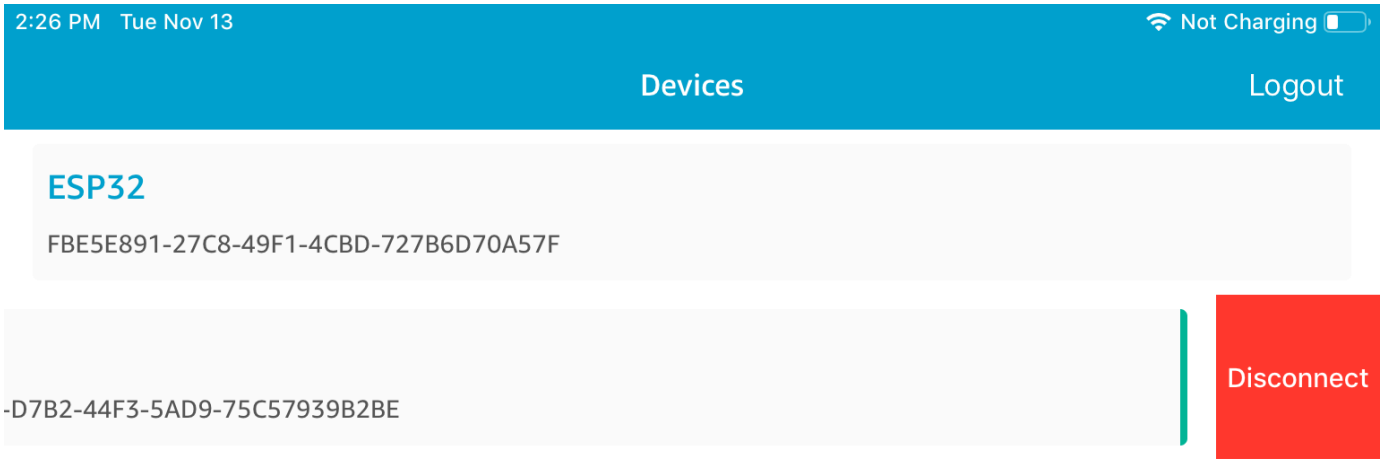


   > **Note**
   >
   > FreeRTOS를 지원하는 모든 디바이스와 범위 내에 있는 디바이스 정보 서비스(*freertos*/.../device_information)가 목록에 나타납니다.

5. 디바이스 목록에서 마이크로 컨트롤러를 선택합니다. 애플리케이션에서 보드와의 연결을 설정하고 연결된 디바이스 옆에 녹색 선이 나타납니다.

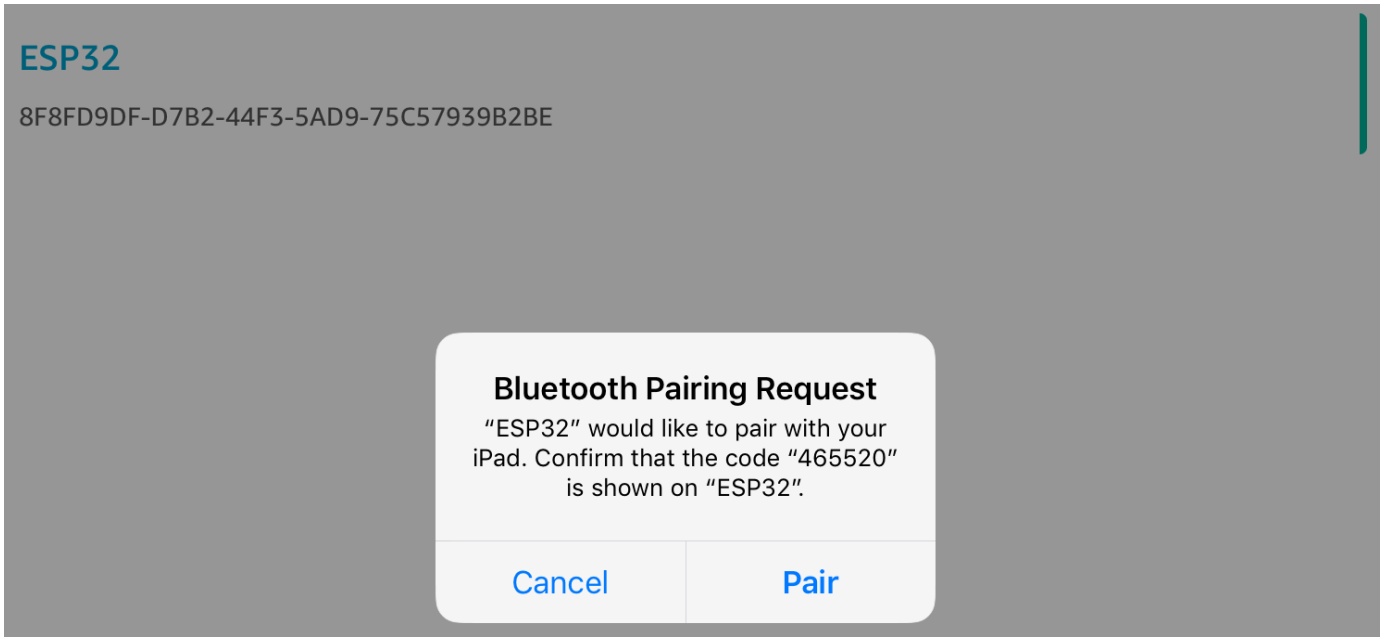


   선을 왼쪽으로 끌어서 마이크로 컨트롤러를 연결 해제할 수 있습니다.



6. 메시지가 표시되면 마이크로컨트롤러와 모바일 디바이스를 페어링합니다.

```
24 261107 [Btc_task]  Disconnected from BLE device. Stopping the counter update
25 261108 [Btc_task] Disconnect received for MQTT service instance 0
26 261108 [Btc_task] BLE disconnected with remote device, start advertisement
27 261108 [Btc_task] Started advertisement. Listening for a BLE Connection.
28 261289 [Btc_task] BLE Connected to remote device, connId = 0
E (2614110) BT_GATT: gatts_write_attr_perm_check - GATT_INSUF_AUTHENTICATION: MITM required
E (2614420) BT_SMP: Value for numeric comparison = 465520
29 261412 [uTask] Numeric comparison:465520
30 261412 [uTask] Press 'y' to confirm
```



두 디바이스의 수 비교를 위한 코드가 동일한 경우 디바이스를 연결합니다.

> **Note**
>
> Bluetooth Low Energy Mobile SDK 데모 애플리케이션에서는 사용자 인증을 위해 Amazon Cognito를 사용합니다. Amazon Cognito 사용자 및 자격 증명 풀을 설정하고 IAM 정책을 인증된 자격 증명에 연결했는지 확인합니다.

MQTT over Bluetooth Low Energy

Bluetooth Low Energy를 통한 MQTT 데모에서 마이크로컨트롤러는 MQTT 프록시를 통해 AWS 클라우드에 메시지를 게시합니다.

데모 MQTT 주제를 구독하려면

1. AWS IoT 콘솔에 로그인합니다.
2. 탐색 창에서 Test(테스트)를 선택하여 MQTT 클라이언트를 엽니다.
3. 구독 주제에 ***thing-name*/example/topic1**을 입력한 다음 주제 구독을 선택합니다.

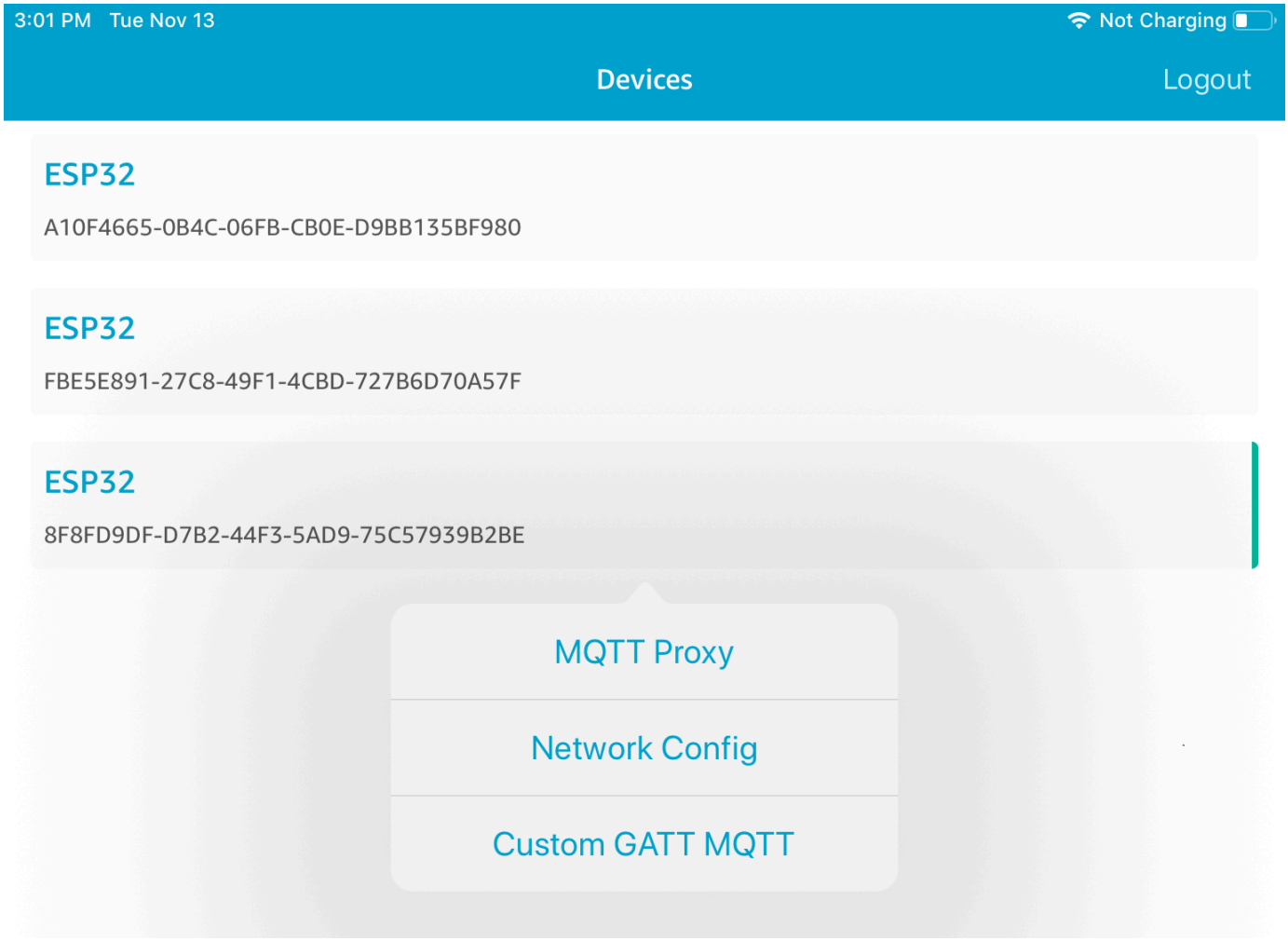
Bluetooth Low Energy를 사용하여 마이크로 컨트롤러를 모바일 디바이스와 연결하는 경우 MQTT 메시지는 모바일 디바이스의 Bluetooth Low Energy Mobile SDK 데모 애플리케이션을 통해 라우팅됩니다.

Bluetooth Low Energy를 통해 데모를 활성화하려면

- `vendors/vendor/boards/board/aws_demos/config_files/aws_demo_config.h`를 열고 `CONFIG_MQTT_BLE_TRANSPORT_DEMO_ENABLED`를 정의합니다.

데모를 실행하려면

1. 마이크로 컨트롤러에서 데모 프로젝트를 빌드한 후 실행합니다.
2. FreeRTOS Bluetooth Low Energy Mobile SDK 데모 애플리케이션을 사용하여 보드와 모바일 디바이스를 연결했는지 확인합니다.
3. 데모 모바일 앱의 디바이스 목록에서 마이크로 컨트롤러를 선택한 다음 MQTT Proxy(MQTT 프록시)를 선택하여 MQTT 프록시 설정을 엽니다.



4. MQTT 프록시를 활성화한 후에는 MQTT 메시지가 *thing-name*/example/topic1 주제에 나타나고 데이터가 UART 터미널에 인쇄됩니다.

Wi-Fi 프로비저닝

Wi-Fi 프로비저닝은 Bluetooth Low Energy를 통해 모바일 디바이스에서 마이크로컨트롤러로 Wi-Fi 네트워크 보안 인증 정보를 안전하게 전송할 수 있도록 해주는 FreeRTOS Bluetooth Low Energy 서비스입니다. Wi-Fi 프로비저닝 서비스를 위한 소스 코드는 *freertos*/.../wifi_provisioning에서 찾을 수 있습니다.

> **Note**
>
> Wi-Fi 프로비저닝 데모는 현재 Espressif ESP32-DevKitC에서 지원됩니다.

데모를 활성화하려면

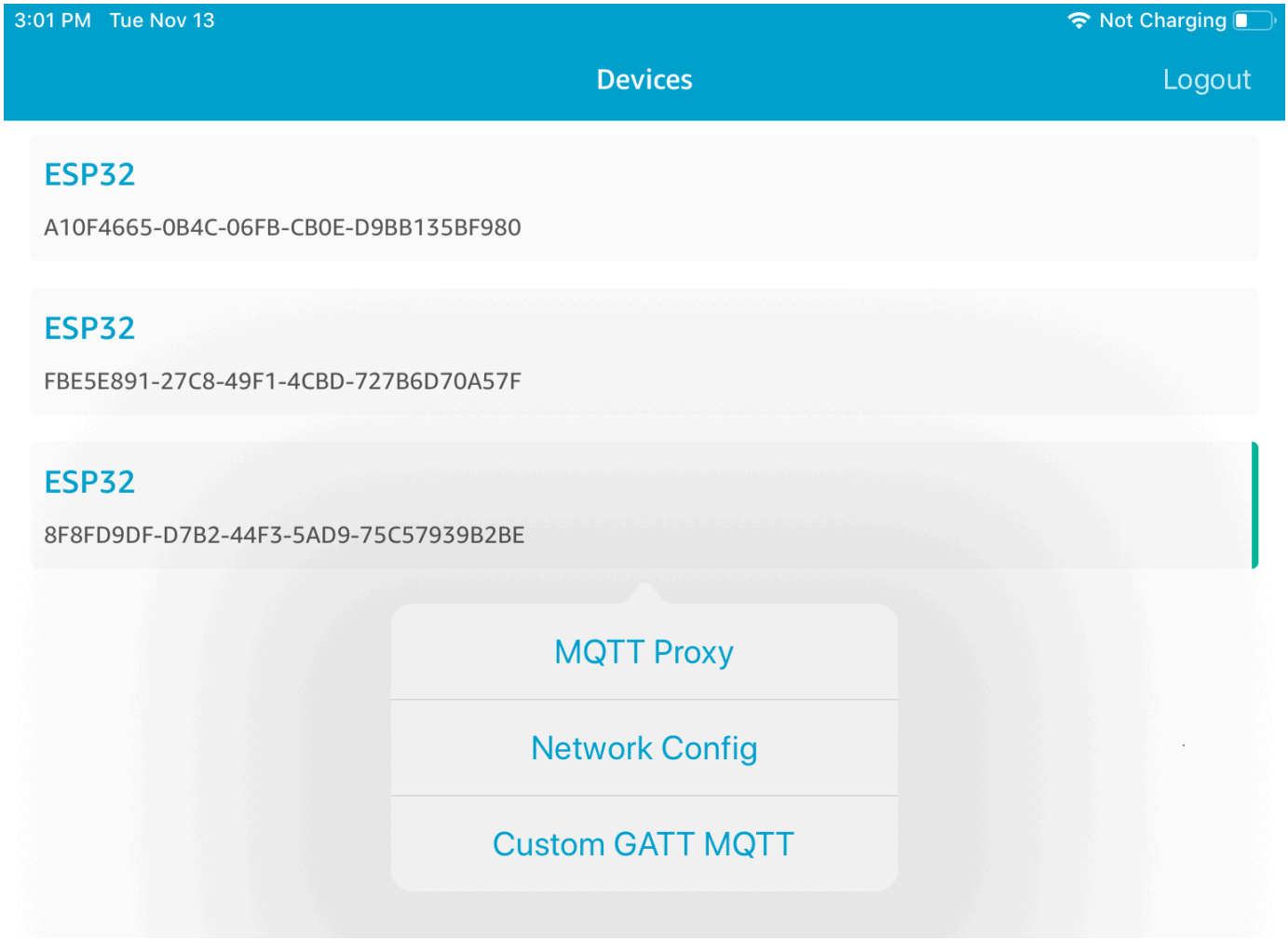
1. Wi-Fi 프로비저닝 서비스를 활성화합니다. `vendors/vendor/boards/board/aws_demos/config_files/iot_ble_config.h`를 열고 `#define IOT_BLE_ENABLE_WIFI_PROVISIONING`을 1로 설정합니다. 여기서 *vendor*는 공급업체의 이름이고 *board*는 데모를 실행하는 데 사용하는 보드 이름입니다.

   > **Note**
   >
   > Wi-Fi 프로비저닝 서비스는 기본적으로 비활성화되어 있습니다.

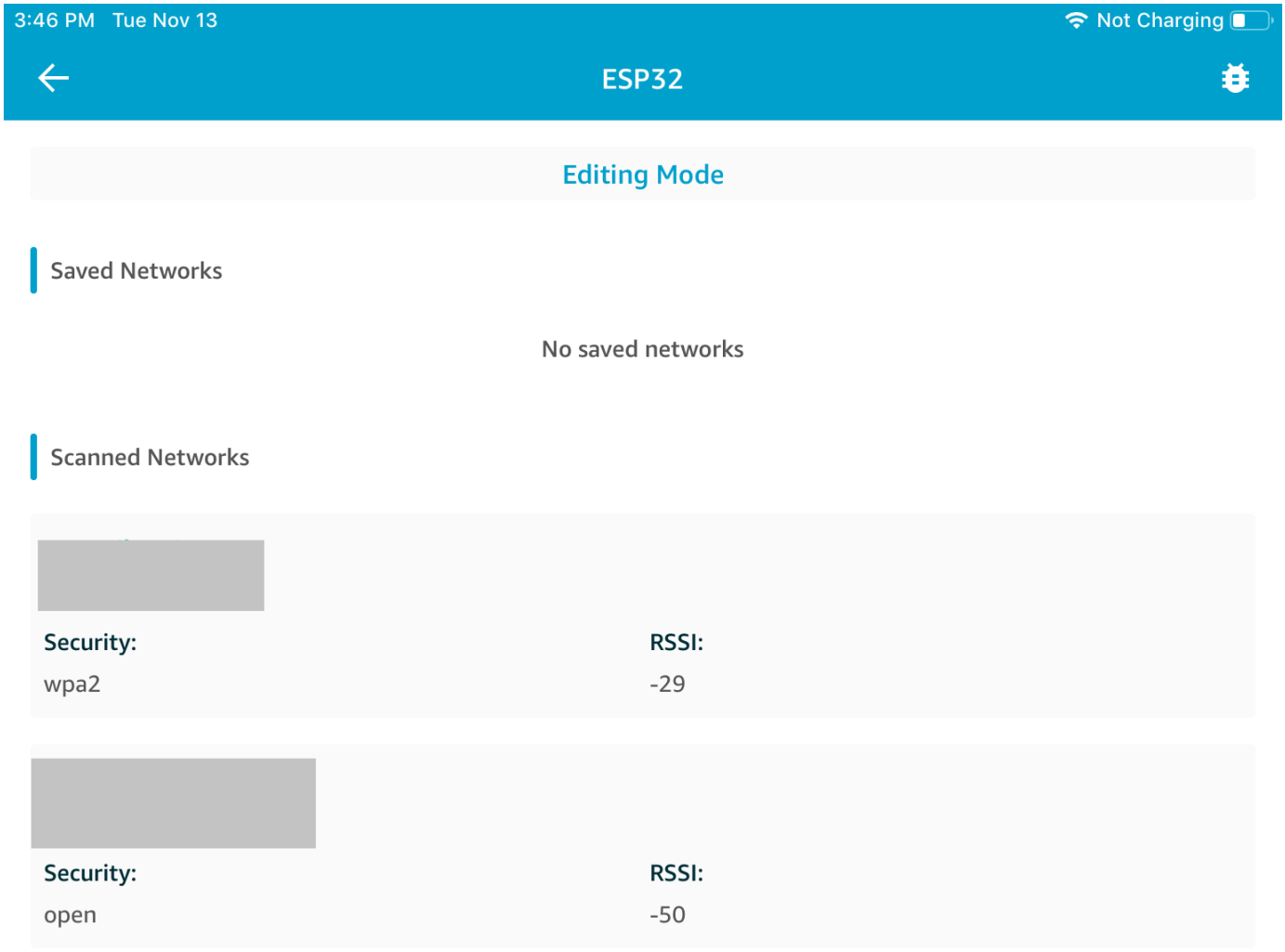
2. Bluetooth Low Energy와 Wi-Fi를 모두 활성화하도록 Network Manager를 구성합니다.

데모를 실행하려면

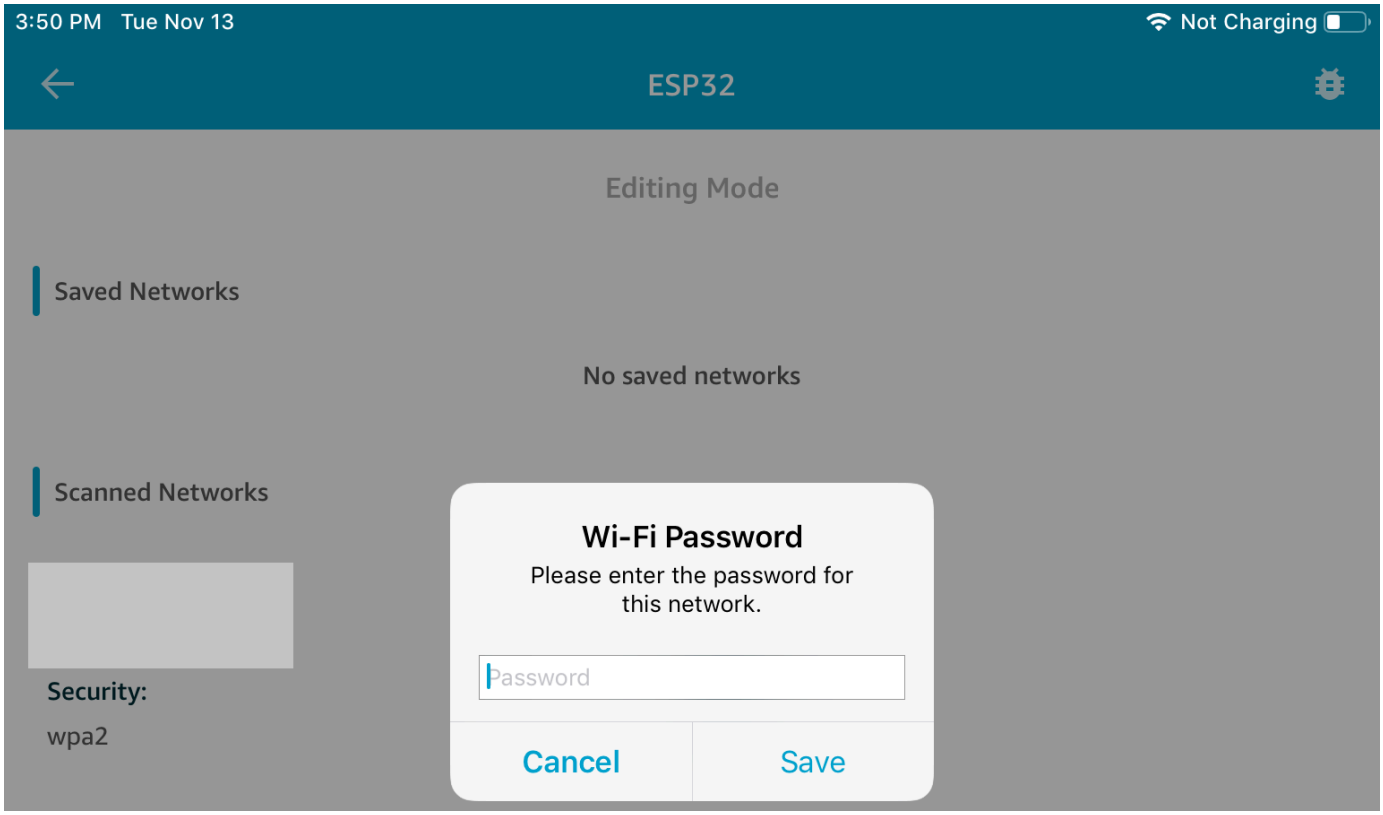
1. 마이크로 컨트롤러에서 데모 프로젝트를 빌드한 후 실행합니다.
2. FreeRTOS Bluetooth Low Energy Mobile SDK 데모 애플리케이션을 사용하여 마이크로 컨트롤러와 모바일 디바이스를 연결했는지 확인합니다.
3. 데모 모바일 앱의 디바이스 목록에서 마이크로 컨트롤러를 선택한 다음 네트워크 구성을 선택하여 네트워크 구성 설정을 엽니다.



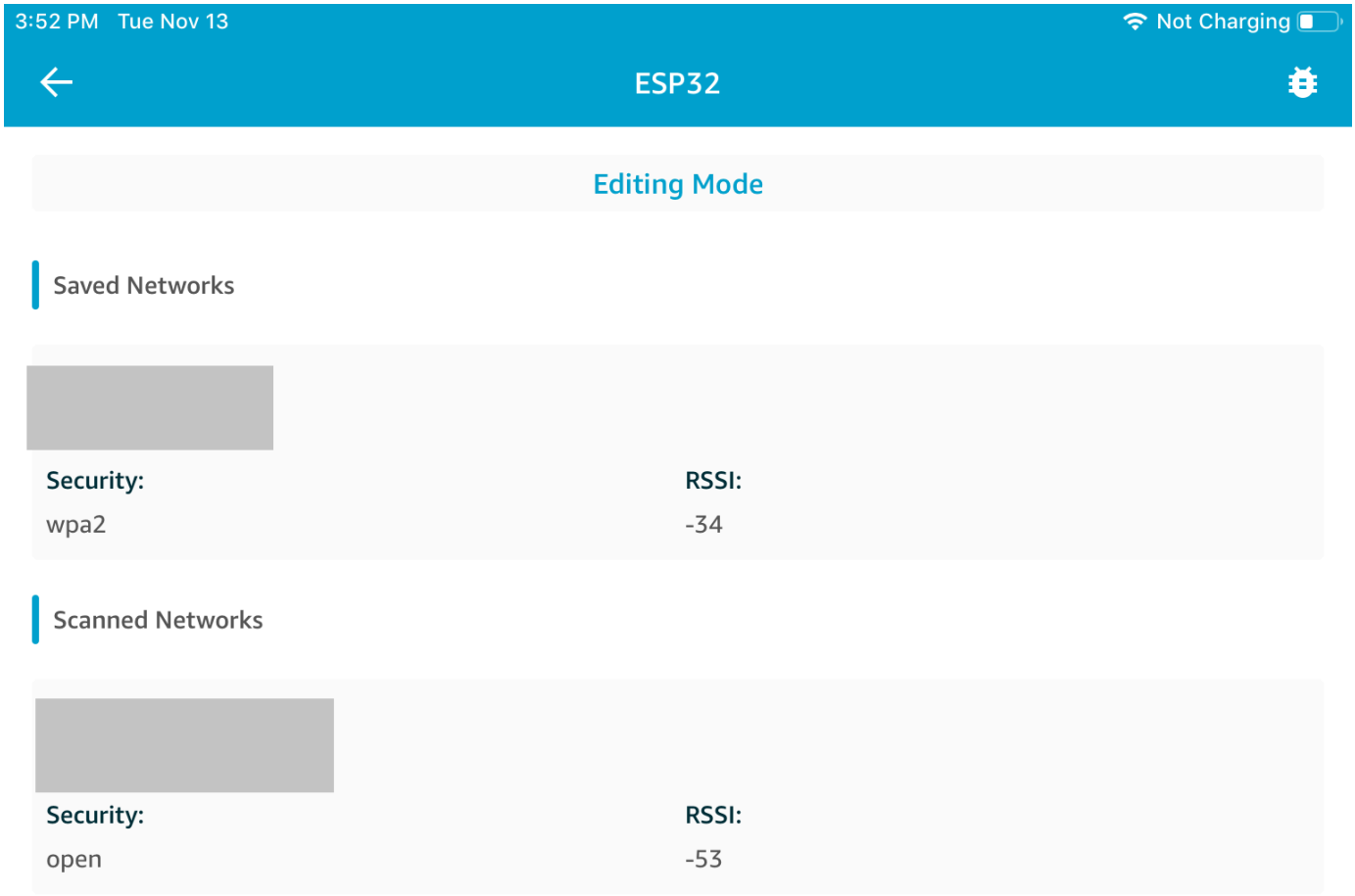
4. 보드에 대한 Network Config(네트워크 구성)를 선택하면 마이크로 컨트롤러는 근처에 있는 네트워크 목록을 모바일 디바이스로 전송합니다. 사용 가능한 Wi-Fi 네트워크가 Scanned Networks(스캔된 네트워크) 아래 목록에 표시됩니다.



Scanned Networks(스캔된 네트워크) 목록에서 네트워크를 선택한 후 필요한 경우 SSID 및 암호를 입력합니다.

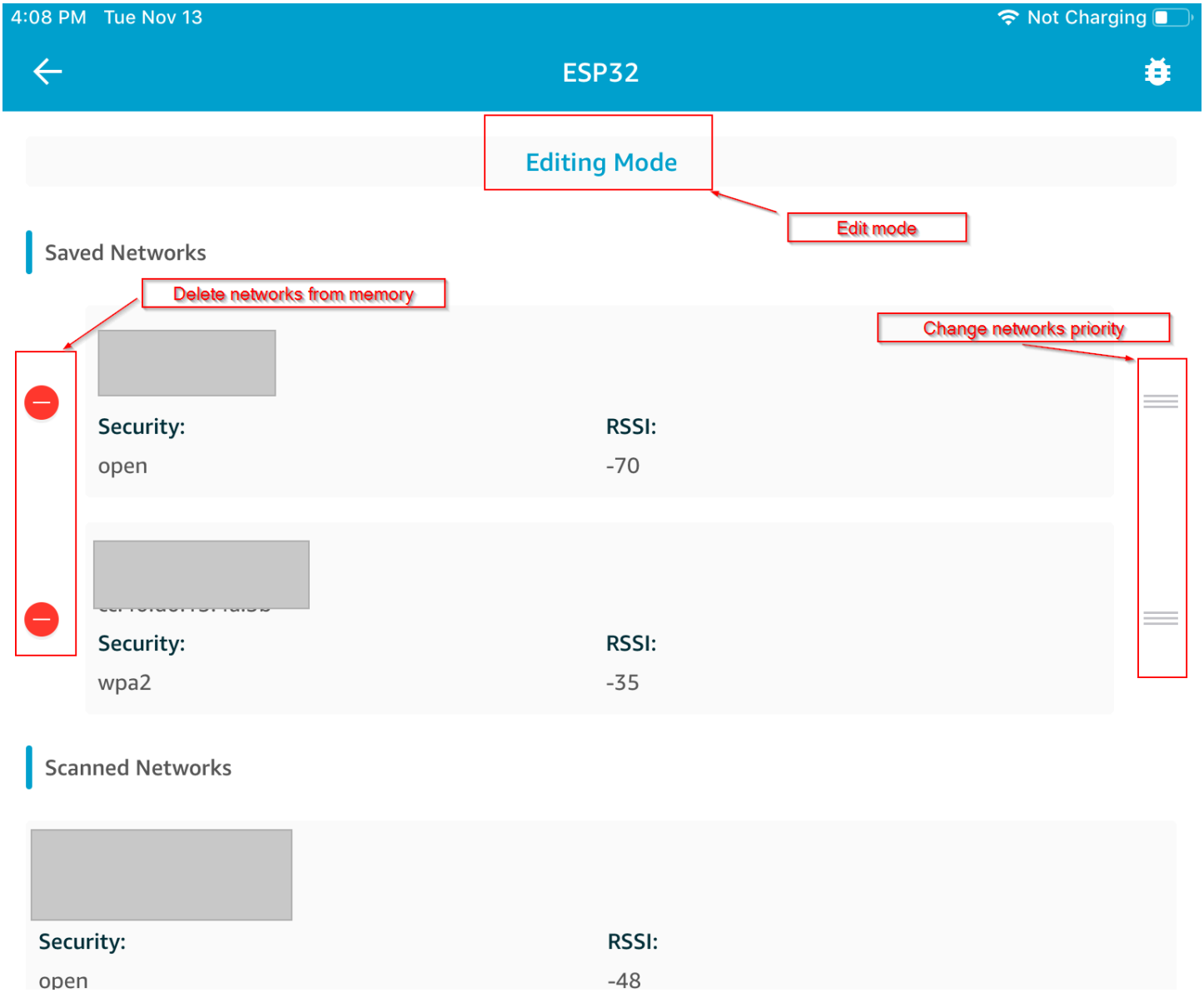


마이크로 컨트롤러가 네트워크에 연결되고 네트워크를 저장합니다. 네트워크가 Saved Networks(저장된 네트워크) 아래에 나타납니다.



데모 모바일 앱에서 여러 네트워크를 저장할 수 있습니다. 애플리케이션 및 데모를 재시작하면 마이크로 컨트롤러가 Saved Networks(저장된 네트워크) 목록의 맨 위에서부터 첫 번째 사용 가능한 저장 네트워크에 연결됩니다.

네트워크 우선 순위를 변경하거나 네트워크를 삭제하려면 Network Configuration(네트워크 구성) 페이지에서 Editing Mode(모드 편집)를 선택합니다. 네트워크 우선 순위를 변경하려면 우선 순위를 조정하려는 네트워크의 오른쪽을 선택하고 네트워크를 위 또는 아래로 끌어 놓습니다. 네트워크를 삭제하려면 삭제할 네트워크의 왼쪽에 있는 빨간색 버튼을 선택합니다.



일반 속성 서버

이 예에서 마이크로 컨트롤러에 대한 데모 일반 속성(GATT) 서버 애플리케이션은 단순 카운터 값을 FreeRTOS Bluetooth Low Energy Mobile SDK 데모 애플리케이션에 전송합니다.

Bluetooth Low Energy Mobile SDK를 사용하여 마이크로 컨트롤러의 GATT 서버에 연결되고 데모 모바일 애플리케이션과 병렬로 실행되는 모바일 디바이스를 위한 고유 GATT 클라이언트를 생성할 수 있습니다.

데모를 활성화하려면

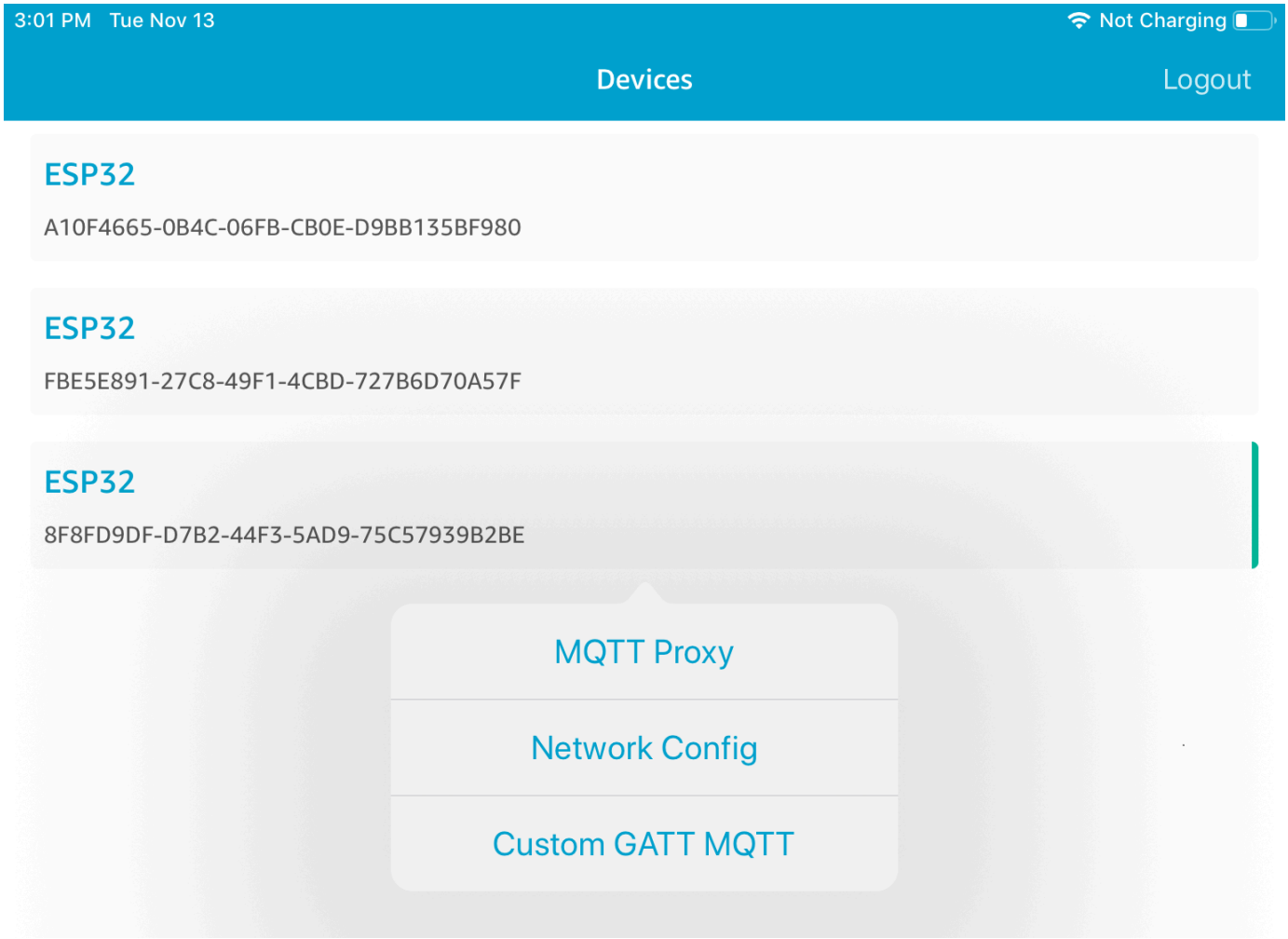
1. Bluetooth Low Energy GATT 데모를 활성화하려면 vendors/*vendor*/boards/*board*/aws_demos/config_files/iot_ble_config.h에서(여기서 *vendor*는 공급업체의 이름이고 *board*는 데모를 실행하는 데 사용하는 보드 이름임) #define IOT_BLE_ADD_CUSTOM_SERVICES (1)를 정의문 목록에 추가합니다.

   > **Note**
   >
   > Bluetooth Low Energy GATT 데모는 기본적으로 비활성화되어 있습니다.

2. *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h를 열고 #define CONFIG_MQTT_DEMO_ENABLED를 주석으로 처리한 다음 CONFIG_BLE_GATT_SERVER_DEMO_ENABLED를 정의합니다.

데모를 실행하려면

1. 마이크로 컨트롤러에서 데모 프로젝트를 빌드한 후 실행합니다.
2. FreeRTOS Bluetooth Low Energy Mobile SDK 데모 애플리케이션을 사용하여 보드와 모바일 디바이스를 연결했는지 확인합니다.
3. 앱의 디바이스 목록에서 보드를 선택한 다음 MQTT Proxy(MQTT 프록시)를 선택하여 MQTT 프록시 옵션을 엽니다.



4. 디바이스 목록으로 돌아가서 보드를 선택한 다음 Custom GATT MQTT(사용자 지정 GATT MQTT)를 선택하여 사용자 지정 GATT 서비스 옵션을 엽니다.
5. Start Counter(카운터 시작)를 선택하여 ***your-thing-name*/example/topic** MQTT 주제에 대한 데이터 게시를 시작합니다.

   MQTT 프록시를 활성화한 후에는 Hello World 및 증분 카운터 메시지가 *your-thing-name*/example/topic 주제에 나타납니다.

Microchip Curiosity PIC32MZEF용 데모 부트로더

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

이 데모 부트로더는 펌웨어 버전 확인, 암호화 서명 검증, 애플리케이션 셀프 테스트를 수행합니다. 이들 기능은 FreeRTOS에 대한 펌웨어 무선 업데이트(OTA)를 지원합니다.

펌웨어 검증에는 원격(OTA)으로 수신한 새 펌웨어의 신뢰성 및 무결성 검증이 포함됩니다. 부트로더는 부팅 전에 애플리케이션의 암호화 서명을 검증합니다. 데모에서는 SHA-256을 통한 ECDSA(Elliptic-Curve Digital Signature Algorithm)를 사용합니다. 제공된 유틸리티를 사용하여 디바이스에 플래시할 수 있는 서명된 애플리케이션을 생성할 수 있습니다.

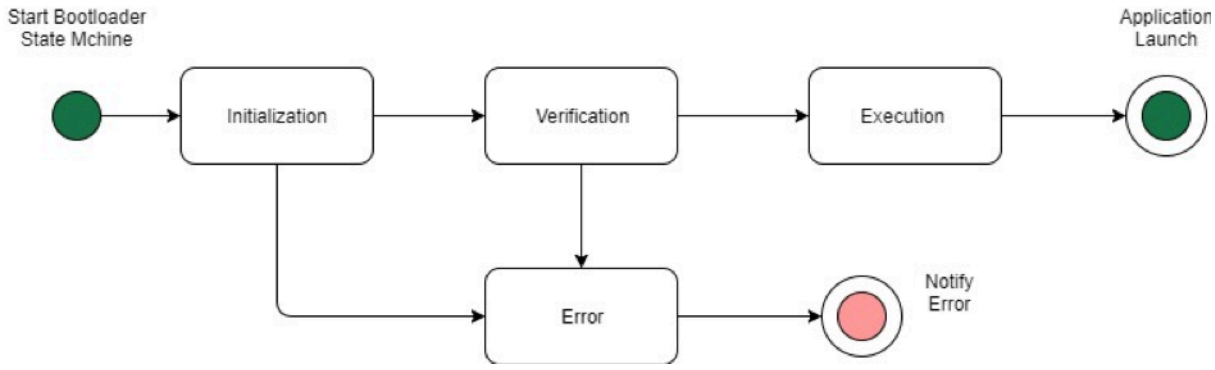
부트로더는 OTA에 필요한 다음 기능을 지원합니다.

- 디바이스에 애플리케이션 이미지를 유지하여 이미지 간에 전환합니다.
- 수신된 OTA 이미지의 셀프 테스트 실행 및 실패 시 롤백을 허용합니다.
- OTA 업데이트 이미지의 서명과 버전을 확인합니다.

> **Note**
>
> FreeRTOS 데모를 설정하고 실행하려면 FreeRTOS 시작하기의 단계를 따릅니다.

부트로더 상태

부트로더 프로세스는 다음과 같은 상태 시스템으로 표시됩니다.



다음 표에는 부트로더 상태에 대한 설명이 나와 있습니다.

부트로더 상태	설명
Initialization(초기화)	부트로더가 초기화 상태입니다.

부트로더 상태	설명
확인	부트로더가 디바이스에 있는 이미지를 확인 중입니다.
Execute Image(이미지 실행)	부트로더가 선택한 이미지를 시작 중입니다.
Execute Default(기본 실행)	부트로더가 기본 이미지를 시작 중입니다.
오류	부트로더가 오류 상태입니다.

위 다이어그램에서 `Execute Image`와 `Execute Default`는 `Execution` 상태로 표시됩니다.

부트로더 실행 상태

부트로더가 `Execution` 상태이고 확인된 선택 이미지를 시작할 준비가 되었습니다. 애플리케이션은 항상 하위 뱅크용으로 빌드되기 때문에 상위 뱅크에서 이미지를 시작할 경우 이미지를 실행하기 전에 뱅크가 바뀝니다.

부트로더 기본 실행 상태

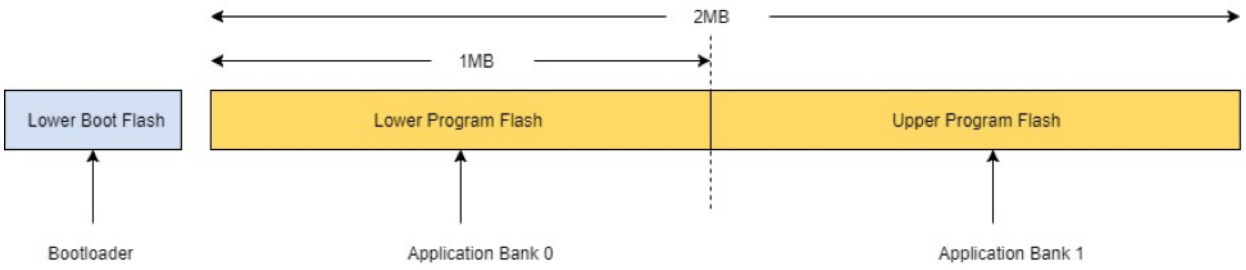
기본 이미지를 실행하는 구성 옵션이 활성화된 경우 부트로더는 기본 실행 주소에서 애플리케이션을 시작합니다. 이 옵션은 디버깅할 때를 제외하고는 비활성화해야 합니다.

부트로더 오류 상태

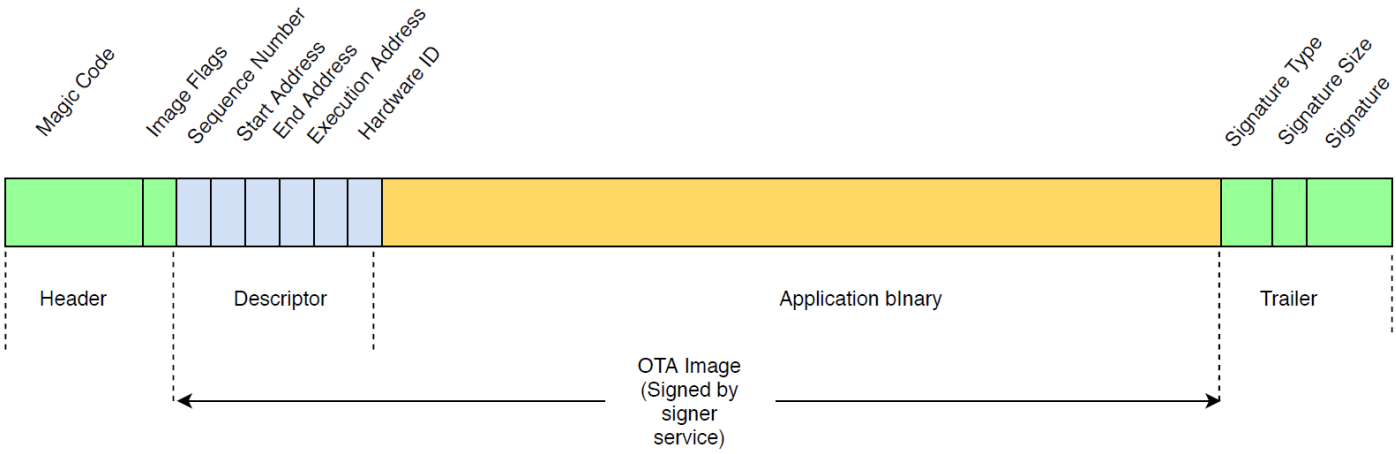
부트로더가 오류 상태이고, 디바이스에 유효한 이미지가 없습니다. 부트로더가 사용자에게 알려야 합니다. 기본 구현은 콘솔에 로그 메시지를 전송하고, 보드의 LED를 계속 빠르게 깜박이는 것입니다.

플래시 디바이스

Microchip Curiosity PIC32MZEF 플랫폼에는 두 개의 뱅크로 나뉜 2MB의 내부 프로그램 플래시가 있습니다. 이 플래시는 이 뱅크 두 개와 실시간 업데이트 간에 메모리 맵 스왑을 지원합니다. 데모 부트로더는 별도의 하위 부트 플래시 영역에서 프로그래밍됩니다.



애플리케이션 이미지 구조



이 다이어그램은 디바이스의 각 뱅크에 저장된 애플리케이션 이미지의 기본 구성 요소를 보여 줍니다.

구성 요소	크기(바이트)
이미지 헤더	8 bytes
이미지 설명자	24바이트
애플리케이션 이진 파일	< 1MB - (324)
트레일러	292바이트

이미지 헤더

디바이스의 애플리케이션 이미지는 매직 코드와 이미지 플래그로 구성된 헤더로 시작해야 합니다.

헤더 필드	크기(바이트)
매직 코드	7바이트
이미지 플래그	1바이트

매직 코드

플래시 디바이스의 이미지는 매직 코드로 시작해야 합니다. 기본 매직 코드는 @AFRTOS입니다. 부트로더는 이미지를 부팅하기 전에 유효한 매직 코드가 있는지 검사합니다. 이것이 검증의 첫 단계입니다.

이미지 플래그

이미지 플래그는 애플리케이션 이미지의 상태를 저장하는 데 사용됩니다. 플래그는 OTA 프로세스에서 사용됩니다. 두 뱅크의 이미지 플래그는 디바이스의 상태를 결정합니다. 실행 이미지가 커밋 대기 중으로 표시되면 디바이스는 OTA 셀프 테스트 단계입니다. 디바이스의 이미지가 유효한 것으로 표시되더라도 부팅할 때마다 동일한 검증 단계를 거칩니다. 이미지가 새 이미지로 표시된 경우, 부트로더는 해당 이미지를 커밋 대기 중으로 표시하고 검증 후 셀프 테스트를 위해 이미지를 시작합니다. 또한 부트로더는 감시 타이머를 초기화하고 시작함으로써 새 OTA 이미지가 셀프 테스트에 실패할 경우 디바이스가 재부팅되고, 부트로더는 이미지를 삭제하여 거부하고, 이전의 유효한 이미지를 실행하도록 합니다.

디바이스에는 유효한 이미지가 한 개만 있어야 합니다. 다른 이미지는 새 OTA 이미지이거나 커밋 대기 중(셀프 테스트) 상태일 수 있습니다. OTA 업데이트가 성공하면 디바이스에서 이전 이미지가 삭제됩니다.

상태 표시기	값	설명
새로운 이미지	0xFF	애플리케이션 이미지가 새 이미지이며 실행되지 않습니다.
커밋 대기 중	0xFE	애플리케이션 이미지가 테스트 실행용으로 표시되었습니다.
유효함	0xFC	애플리케이션 이미지가 유효로 표시되고 커밋되었습니다.

상태 표시기	값	설명
잘못됨	0xF8	애플리케이션 이미지가 잘못됨으로 표시되었습니다.

이미지 설명자

플래시 디바이스의 애플리케이션 이미지는 이미지 헤더 뒤에 이미지 설명자를 포함해야 합니다. 이미지 설명자는 구성 파일(`ota-descriptor.config`)을 사용하여 해당 설명자를 생성하고 이 설명자를 애플리케이션 이진 파일에 추가하는 포스트 빌드 유틸리티를 통해 생성됩니다. 이 빌드 후 단계의 출력은 이진 이미지이며 OTA에 사용할 수 있습니다.

설명자 필드	크기(바이트)
시퀀스 번호	4 bytes
시작 주소	4 bytes
종료 주소	4 bytes
실행 주소	4 bytes
하드웨어 ID	4 bytes
예약	4 bytes

시퀀스 번호

시퀀스 번호가 증가한 뒤에 새 OTA 이미지를 빌드해야 합니다. `ota-descriptor.config` 파일을 참조하십시오. 부트로더는 이 번호로 부팅할 이미지를 결정합니다. 유효한 값은 1~4294967295입니다.

시작 주소

디바이스에 있는 애플리케이션 이미지의 시작 주소입니다. 이미지 설명자가 애플리케이션 이진 파일에 추가되기 때문에 이 주소는 이미지 설명자의 시작입니다.

종료 주소

디바이스에 있는 애플리케이션 이미지의 종료 주소입니다(이미지 트레일러 제외).

실행 주소

이미지의 실행 주소입니다.

하드웨어 ID

OTA 이미지가 올바른 플랫폼용으로 빌드되었는지 확인하기 위해 부트로더가 사용하는 고유한 하드웨어 ID입니다.

예약

나중에 사용하기 위해 예약되어 있습니다.

이미지 트레일러

이미지 트레일러는 애플리케이션 이진 파일에 추가됩니다. 서명 유형 문자열, 서명 크기, 이미지의 서명 등이 들어 있습니다.

트레일러 필드	크기(바이트)
서명 유형	32바이트
서명 크기	4 bytes
서명	256 bytes

서명 유형

이 서명 유형은 사용되는 암호화 알고리즘을 나타내며, 트레일러를 위한 마커 역할을 합니다. 부트로더는 ECDSA(Elliptic-Curve Digital Signature Algorithm)를 지원합니다. 기본값은 sig-sha256-ecdsa입니다.

서명 크기

암호화 서명의 크기(바이트)입니다.

Signature

이미지 설명자와 함께 추가된 애플리케이션 이진 파일의 암호화 서명입니다.

부트로더 구성

기본 부트로더 구성 옵션은 *freertos*/vendors/microchip/boards/curiosity_pic32mzef/bootloader/config_files/aws_boot_config.h에 제공됩니다. 일부 옵션은 디버그용으로만 제공됩니다.

기본 시작 활성화

기본 주소에서 애플리케이션을 실행할 수 있도록 하고, 디버그용으로만 활성화합니다. 이미지는 검증 없이 기본 주소에서 실행됩니다.

Enable Crypto Signature Verification(암호화 서명 검증 활성화)

부팅 시 암호화 서명 검증을 활성화합니다. 실패한 이미지는 디바이스에서 삭제됩니다. 이 옵션은 디버그용으로만 제공되며, 프로덕션 환경에서 활성 상태로 유지해야 합니다.

Erase Invalid Image(잘못된 이미지 삭제)

뱅크에서 이미지 검증이 실패할 경우 전체 뱅크를 삭제합니다. 이 옵션은 디버그용이며, 프로덕션 환경에서 활성 상태여야 합니다.

Enable Hardware ID Verification(하드웨어 ID 검증 활성화)

OTA 이미지의 설명자에 있는 하드웨어 ID와 부트로더에 프로그래밍된 하드웨어 ID를 검증합니다. 선택 사항이며, 하드웨어 검증이 필요하지 않은 경우 비활성화할 수 있습니다.

Enable Address Verification(주소 검증 활성화)

OTA 이미지의 설명자에 있는 시작, 종료, 실행 주소를 검증합니다. 이 옵션을 항상 활성화해 놓는 것이 좋습니다.

부트로더 빌드

데모 부트로더는 FreeRTOS 소스 코드 리포지토리의 *freertos*/vendors/microchip/boards/curiosity_pic32mzef/aws_demos/mplab/에 있는 aws_demos 프로젝트에 로드 가능한 프로젝트로 포함되어 있습니다. 빌드된 aws_demos 프로젝트는 먼저 부트로더를 빌드한 다음 애플리케이션을 빌드합니다. 최종 출력은 부트로더와 애플리케이션을 포함하여 통합 16진 이미지입니다. 암호화 서명이 있는 통합 16진 이미지를 생성할 수 있도록 factory_image_generator.py 유틸리티가 제공됩니다. 부트로더 유틸리티 스크립트는 *freertos*/demos/ota/bootloader/utility/에 있습니다.

부트로더 빌드 전 절차

이 빌드 전 절차는 코드 서명 인증서에서 퍼블릭 키를 추출하는 `codesigner_cert_utility.py` 유틸리티 스크립트를 실행하고, 퍼블릭 키를 포함하는 C 헤더 파일을 ASN.1(Abstract Syntax Notation One) 인코딩 형식으로 생성합니다. 이 헤더는 부트로더 프로젝트에 컴파일됩니다. 생성된 헤더에는 퍼블릭 키 배열과 키 길이, 두 개의 상수가 들어 있습니다. `aws_demos`를 사용하지 않고 부트로더 프로젝트를 빌드할 수도 있으며 일반 애플리케이션처럼 디버깅할 수 있습니다.

AWS IoT Device Defender 데모

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

소개

이 데모에서는 AWS IoT Device Shadow 라이브러리를 사용하여에 연결하는 방법을 보여줍니다AWS IoT Device Defender. 데모는 coreMQTT 라이브러리를 사용하여 MQTT 브로커 및 coreJSON 라이브러리에 대한 TLS(상호 인증)와 AWS IoT MQTT 연결을 설정하여 수신된 응답을 검증하고 구문 분석합니다 AWS IoT Device Defender. 데모에서는 디바이스에서 수집한 지표를 사용하여 JSON 형식의 보고서를 생성하는 방법과 이 보고서를 AWS IoT Device Defender에 제출하는 방법을 보여줍니다. 또한 이 데모에서는가 보고서가 수락되었는지 거부되었는지 확인하기 위해 AWS IoT Device Defender 보내는 응답을 처리하기 위해 coreMQTT 라이브러리에 콜백 함수를 등록하는 방법을 보여줍니다.

> **Note**
>
> FreeRTOS 데모를 설정하고 실행하려면 FreeRTOS 시작하기의 단계를 따릅니다.

기능

이 데모에서는 지표를 수집하고, JSON 형식으로 디바이스 방어기 보고서를 구성하고, AWS IoT MQTT 브로커에 대한 보안 MQTT 연결을 AWS IoT Device Defender 통해에 제출하는 방법을 보여주는 단일 애플리케이션 작업을 생성합니다.

지표를 수집하는 방법은 사용 중인 TCP/IP 스택에 따라 다릅니다. FreeRTOS+TCP 및 지원되는 lwIP 구성의 경우, 디바이스에서 실제 지표를 수집하여 AWS IoT Device Defender 보고서로 제출하는 지표 수집 구현을 제공합니다. FreeRTOS+TCP 및 lwIP에 대한 구현은 GitHub에서 찾을 수 있습니다.

다른 TCP/IP 스택을 사용하는 보드의 경우 모든 지표를 0으로 반환하는 지표 수집 함수의 스텁 정의가 제공됩니다. 이 스텁 구현을 사용하여 보드의 실제 지표를 전송하려면 네트워크 스택에 *freertos*/demos/device_defender_for_aws/metrics_collector/stub/metrics_collector.c의 함수를 구현합니다. 이 파일은 GitHub 웹 사이트에서도 사용할 수 있습니다.

ESP32의 경우 기본 lwIP 구성에서는 코어 잠금을 사용하지 않으므로 데모에서 스텁 지표를 사용합니다. 참조 lwIP 지표 수집 구현을 사용하려면 lwiopts.h에서 다음 매크로를 정의합니다.

```
#define LINK_SPEED_OF_YOUR_NETIF_IN_BPS 0
#define LWIP_TCPIP_CORE_LOCKING         1
#define LWIP_STATS                      1
#define MIB2_STATS                      1
```

다음은 데모를 실행하면 생성되는 출력의 예입니다.

```
15 2026 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:632] MQTT connection successfully established with broker.


16 2026 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:660] A clean MQTT connection is established. Cleaning up all the
 stored outgoing publishes.


17 2026 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:751] SUBSCRIBE topic $aws/things/MyThingName/defender/metrics/js
on/accepted to broker.


18 2086 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=3.
19 2086 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:463] MQTT_PACKET_TYPE_SUBACK.


20 2686 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:751] SUBSCRIBE topic $aws/things/MyThingName/defender/metrics/js
on/rejected to broker.


21 2746 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=3.
22 2746 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:463] MQTT_PACKET_TYPE_SUBACK.


23 3346 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:864] the published payload:{"header": {"report_id": 1499,"versio
n": "1.0"},"metrics": {"listening_tcp_ports": {"ports": [{"port": 26023}],"total": 1},"listening_udp_ports": {"ports": [
],"total": 0},"network_stats": {"bytes_in": 0,"bytes_out": 0,"packets_in":
24 3346 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:892] PUBLISH sent for topic $aws/things/MyThingName/defender/met
rics/json to broker with packet ID 3.


25 3506 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=2.
26 3506 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1132] Ack packet deserialized with result: MQTTSuccess.
27 3506 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1145] State record updated. New state=MQTTPublishDone.
28 3506 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:484] PUBACK received for packet id 3.


29 3506 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:448] Cleaned up outgoing publish packet with packet id 3.


30 3506 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=145.
31 3506 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1015] De-serialized incoming PUBLISH packet: DeserializerResult=MQTTSucc
ess.
32 3506 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1028] State record updated. New state=MQTTPublishDone.
33 3906 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:814] UNSUBSCRIBE sent topic $aws/things/MyThingName/defender/met
rics/json/accepted to broker.


34 3986 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=2.
35 3986 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:469] MQTT_PACKET_TYPE_UNSUBACK.


36 4586 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:814] UNSUBSCRIBE sent topic $aws/things/MyThingName/defender/met
rics/json/rejected to broker.


37 4646 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=2.
38 4646 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:469] MQTT_PACKET_TYPE_UNSUBACK.


39 5246 [iot_thread] [INFO] [MQTT] [core_mqtt.c:2118] Disconnected from the broker.
40 5246 [iot_thread] [INFO ][DEMO][5246] memory_metrics::freertos_heap::before::bytes::2088152
[INFO ][DEMO][5246] memory_metrics::freertos_heap::after::bytes::1986576
41 5246 [iot_thread] [INFO ][DEMO][5246] memory_metrics::demo_task_stack::before::bytes::1908
[INFO ][DEMO][5246] memory_metrics::demo_task_stack::after::bytes::1908
42 6246 [iot_thread] [INFO ][DEMO][6246] Demo completed successfully.
[INFO ][INIT][6248] SDK cleanup done.
43 6248 [iot_thread] [INFO ][DEMO][6248] -------DEMO FINISHED-------
```

보드가 FreerTOS+TCP 또는 지원되는 lwIP 구성을 사용하지 않는 경우 출력은 다음과 같습니다.

```
15 1528 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:632] MQTT connection successfully established with broker.


16 1528 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:660] A clean MQTT connection is established. Cleaning up all the
 stored outgoing publishes.


17 1528 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:751] SUBSCRIBE topic $aws/things/MyThingName/defender/metrics/js
on/accepted to broker.


18 1568 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=3.
19 1568 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:463] MQTT_PACKET_TYPE_SUBACK.


20 2168 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:751] SUBSCRIBE topic $aws/things/MyThingName/defender/metrics/js
on/rejected to broker.


21 2248 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=3.
22 2248 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:463] MQTT_PACKET_TYPE_SUBACK.


23 2848 [iot_thread] [ERROR] [Device_Defender_Demo] [metrics_collector.c:82] Using stub definition of GetNetworkStats! P
lease implement for your network stack to get correct metrics.
24 2848 [iot_thread] [ERROR] [Device_Defender_Demo] [metrics_collector.c:110] Using stub definition of GetOpenTcpPorts!
Please implement for your network stack to get correct metrics.
25 2848 [iot_thread] [ERROR] [Device_Defender_Demo] [metrics_collector.c:146] Using stub definition of GetOpenUdpPorts!
Please implement for your network stack to get correct metrics.
26 2848 [iot_thread] [ERROR] [Device_Defender_Demo] [metrics_collector.c:183] Using stub definition of GetEstablishedCon
nections! Please implement for your network stack to get correct metrics.
27 2848 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:864] the published payload:{"header": {"report_id": 1093,"versio
n": "1.0"},"metrics": {"listening_tcp_ports": {"ports": [],"total": 0},"listening_udp_ports": {"ports": [],"total": 0},"
network_stats": {"bytes_in": 0,"bytes_out": 0,"packets_in": 0,"packets_out"
28 2848 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:892] PUBLISH sent for topic $aws/things/MyThingName/defender/met
rics/json to broker with packet ID 3.


29 2968 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=2.
30 2968 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1132] Ack packet deserialized with result: MQTTSuccess.
31 2968 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1145] State record updated. New state=MQTTPublishDone.
32 2968 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:484] PUBACK received for packet id 3.


33 2968 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:448] Cleaned up outgoing publish packet with packet id 3.


34 2968 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=145.
35 2968 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1015] De-serialized incoming PUBLISH packet: DeserializerResult=MQTTSucc
ess.
36 2968 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1028] State record updated. New state=MQTTPublishDone.
37 3368 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:814] UNSUBSCRIBE sent topic $aws/things/MyThingName/defender/met
rics/json/accepted to broker.


38 3408 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=2.
39 3408 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:469] MQTT_PACKET_TYPE_UNSUBACK.


40 4008 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:814] UNSUBSCRIBE sent topic $aws/things/MyThingName/defender/met
rics/json/rejected to broker.


41 4088 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=2.
42 4088 [iot_thread] [INFO] [MQTT] [mqtt_demo_helpers.c:469] MQTT_PACKET_TYPE_UNSUBACK.


43 4688 [iot_thread] [INFO] [MQTT] [core_mqtt.c:2118] Disconnected from the broker.
44 4688 [iot_thread] [INFO ][DEMO][4688] memory_metrics::freertos_heap::before::bytes::2088152
[INFO ][DEMO][4688] memory_metrics::freertos_heap::after::bytes::1986760
45 4688 [iot_thread] [INFO ][DEMO][4688] memory_metrics::demo_task_stack::before::bytes::1908
[INFO ][DEMO][4688] memory_metrics::demo_task_stack::after::bytes::1908
46 5688 [iot_thread] [INFO ][DEMO][5688] Demo completed successfully.
[INFO ][INIT][5690] SDK cleanup done.
47 5690 [iot_thread] [INFO ][DEMO][5690] -------DEMO FINISHED-------
```

데모의 소스 코드는 *freertos*/demos/device_defender_for_aws/ 디렉터리 또는 GitHub 웹 사이트에서 다운로드할 수 있습니다.

AWS IoT Device Defender 주제 구독

subscribeToDefenderTopics 함수는 게시된 Device Defender 보고서에 대한 응답을 수신할 수 있는 MQTT 주제를 구독합니다. 매크로 `DEFENDER_API_JSON_ACCEPTED`를 사용하여 수락된 Device Defender 보고서에 대한 응답을 수신할 주제 문자열을 구성합니다. 매크로 `DEFENDER_API_JSON_REJECTED`를 사용하여 거부된 Device Defender 보고서에 대한 응답을 수신할 주제 문자열을 구성합니다.

디바이스 지표 수집

collectDeviceMetrics 함수는 `metrics_collector.h`에 정의된 함수를 사용하여 네트워킹 지표를 수집합니다. 수집되는 지표는 송수신된 바이트 및 패킷 수, 열린 TCP 포트, 열린 UDP 포트, 설정된 TCP 연결입니다.

AWS IoT Device Defender 보고서 생성

generateDeviceMetricsReport 함수는 `report_builder.h`에 정의된 함수를 사용하여 Device Defender 보고서를 생성합니다. 이 함수는 네트워킹 지표와 버퍼를 가져 AWS IoT Device Defender 와서에서 예상한 형식으로 JSON 문서를 생성하고 제공된 버퍼에 기록합니다. 에서 예상되는 JSON 문서의 형식은 AWS IoT 개발자 안내서의 디바이스 측 지표에 지정 AWS IoT Device Defender 되어 있습니다.

AWS IoT Device Defender 보고서 게시

보고서는 JSON AWS IoT Device Defender 보고서 게시를 위한 MQTT 주제에 게시됩니다 AWS IoT Device Defender . 보고서는 GitHub 웹 사이트의 이 코드 조각에 나와 있는 매크로 `DEFENDER_API_JSON_PUBLISH`를 사용하여 구성됩니다.

응답 처리를 위한 콜백

publishCallback 함수는 들어오는 MQTT 게시 메시지를 처리합니다. AWS IoT Device Defender 라이브러리의 `Defender_MatchTopic` API를 사용하여 수신 MQTT 메시지가 AWS IoT Device Defender 서비스에서 왔는지 확인합니다. 메시지가 발신인 경우 수신된 JSON 응답을 AWS IoT Device Defender구문 분석하고 응답에서 보고서 ID를 추출합니다. 그런 다음 보고서 ID가 보고서에서 전송된 ID와 동일한지 확인합니다.

AWS IoT Greengrass 검색 데모 애플리케이션

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

FreeRTOS용 AWS IoT Greengrass Discovery 데모를 실행하기 전에 AWS AWS IoT Greengrass및를 설정해야 합니다 AWS IoT. 설정하려면의 지침을 AWS따르세요AWS 계정 및 권한 설정. 설정하려면 Greengrass 그룹을 생성한 다음 Greengrass 코어를 추가 AWS IoT Greengrass해야 합니다. 설정에 대한 자세한 내용은 시작하기를 AWS IoT Greengrass참조하세요. AWS IoT Greengrass

AWS 및를 설정한 후에는에 대한 몇 가지 추가 권한을 구성 AWS IoT Greengrass해야 합니다 AWS IoT Greengrass.

AWS IoT Greengrass 권한을 설정하려면

1. IAM 콘솔로 이동합니다.
2. 탐색 창에서 역할을 선택한 후 Greengrass_ServiceRole을 찾아 선택합니다.
3. 정책 연결을 선택하고 AmazonS3FullAccess와 AWSIoTFullAccess를 선택한 후 정책 연결을 선택합니다.
4. AWS IoT 콘솔로 이동합니다.
5. 탐색 창에서 Greengrass를 선택하고 그룹을 선택한 후 앞에서 만든 Greengrass 그룹을 선택합니다.
6. 설정을 선택한 후 역할 추가를 선택합니다.
7. Greengrass_ServiceRole을 선택한 후 저장을 선택합니다.

보드를 AWS IoT 에 연결하고 FreeRTOS 데모를 구성합니다.

1. 에 MCU 보드 등록 AWS IoT

   보드를 등록한 후에는 새로운 Greengrass 정책을 생성하여 디바이스 인증서에 연결해야 합니다.

   새 AWS IoT Greengrass 정책을 생성하려면

   1. AWS IoT 콘솔로 이동합니다.

2. 탐색 창에서 Secure(보안)를 선택하고 Policies(정책)를 선택한 다음 Create(생성)를 선택합니다.
3. 정책을 식별할 이름을 입력합니다.
4. Add statements(설명문 추가) 섹션에서 Advanced mode(고급 모드)를 선택합니다. 다음 JSON을 복사하여 정책 편집기 창에 붙여넣습니다.

```
{
      "Effect": "Allow",
      "Action": [
        "greengrass:*"
      ],
      "Resource": "*"
}
```

   이 정책은 모든 리소스에 AWS IoT Greengrass 권한을 부여합니다.
5. 생성(Create)을 선택합니다.

디바이스의 인증서에 AWS IoT Greengrass 정책을 연결하려면

1. AWS IoT 콘솔로 이동합니다.
2. 탐색 창에서 관리를 선택하고 사물을 선택한 후 앞에서 만든 사물을 선택합니다.
3. 보안을 선택한 후 디바이스에 연결된 인증서를 선택합니다.
4. 정책을 선택하고 작업을 선택한 후 정책 연결을 선택합니다.
5. 앞에서 만든 Greengrass 정책을 선택한 후 연결을 선택합니다.

2. FreeRTOS 다운로드

> **Note**
>
> FreeRTOS 콘솔에서 FreeRTOS를 다운로드하는 경우 AWS IoT Greengrass#### 연결 대신 AWS IoT#### 연결을 선택합니다.

3. FreeRTOS 데모 구성.

freertos/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h를 열고 #define CONFIG_MQTT_DEMO_ENABLED를 주석으로 처리한 다음 CONFIG_GREENGRASS_DISCOVERY_DEMO_ENABLED를 정의합니다.

AWS IoT 및를 설정하고 FreeRTOS를 다운로드 AWS IoT Greengrass하고 구성한 후 디바이스에서 Greengrass 데모를 빌드, 플래시 및 실행할 수 있습니다. 보드의 하드웨어 및 소프트웨어 개발 환경을 설정하려면 보드별 시작 안내서의 지침을 수행해야 합니다.

Greengrass 데모는 일련의 메시지를 Greengrass 코어와 AWS IoT MQTT 클라이언트에 게시합니다. AWS IoT MQTT 클라이언트에서 메시지를 보려면 AWS IoT 콘솔을 열고 테스트를 선택한 다음에 구독을 추가합니다`freertos/demos/ggd`.

MQTT 클라이언트에서 다음 문자열이 표시되어야 합니다.

```
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
```

Amazon EC2 인스턴스 사용

Amazon EC2 인스턴스로 작업하는 경우

1. Amazon EC2 인스턴스와 연결된 퍼블릭 DNS(IPv4) 찾기 - Amazon EC2 콘솔로 이동한 다음 왼쪽 탐색 패널에서 인스턴스를 선택합니다. Amazon EC2 인스턴스를 선택한 다음 설명 패널을 선택합니다. Public DNS (IPv4)에 대한 항목을 찾아 기록해 둡니다.
2. 보안 그룹에 대한 항목을 찾아 Amazon EC2 인스턴스에 연결된 보안 그룹을 선택합니다.
3. Inbound rules(인바운드 규칙) 탭을 선택한 다음 Edit inbound rules(인바운드 규칙 편집)을 선택하고 다음 규칙을 추가합니다.

   인바운드 규칙

Type	프로토콜	포트 범위	소스	설명 - 선택 사항
HTTP	TCP	80	0.0.0.0/0	-
HTTP	TCP	80	::/0	-
SSH	TCP	22	0.0.0.0/0	-
사용자 지정 TCP	TCP	8883	0.0.0.0/0	MQTT 커뮤니케이션

Type	프로토콜	포트 범위	소스	설명 - 선택 사항
사용자 지정 TCP	TCP	8883	::/0	MQTT 커뮤니케이션
HTTPS	TCP	443	0.0.0.0/0	-
HTTPS	TCP	443	::0/0	-
모든 ICMP - IPv4	ICMP	모두	0.0.0.0/0	-
모든 ICMP - IPv4	ICMP	모두	::0/0	-

4. AWS IoT 콘솔에서 Greengrass, 그룹, 이전에 생성한 Greengrass 그룹을 차례로 선택합니다. 설정을 선택합니다. Local connection detection(로컬 연결 감지)을 Manually manage connection information(수동으로 연결 정보 관리)으로 변경합니다.
5. 탐색 창에서 Cores(코어)를 선택한 다음 그룹 코어를 선택합니다.
6. Connectivity(연결)를 선택하고 코어 엔드포인트가 하나만 있는지(나머지는 모두 삭제) 확인하고 IP 주소(변경될 수 있으므로)가 아닌지 확인합니다. 가장 좋은 방법은 첫 번째 단계에서 기록해 두었던 Public DNS(IPv4)를 사용하는 것입니다.
7. 생성한 FreeRTOS IoT 사물을 GG 그룹에 추가합니다.

   a. 뒤로 화살표를 선택하여 AWS IoT Greengrass 그룹 페이지로 돌아갑니다. 탐색 창에서 Devices(디바이스)를 선택한 다음 Add Device(디바이스 추가)를 선택합니다.

   b. Select an IoT Thing(IoT 사물 선택)을 선택합니다. 디바이스를 선택한 다음 Finish(완료)를 선택합니다.
8. 필요한 구독 추가 - Greengrass 그룹 페이지에서 구독을 선택한 다음, 구독 추가를 선택하고 여기에 표시된 대로 정보를 입력합니다.

   구독

소스	대상	주제
TIGG1	IoT Cloud	freertos/demos/ggd

여기서 "Source"는 보드를 등록할 때 AWS IoT 콘솔에서 생성된 AWS IoT 사물에 지정된 이름입니다. 여기에 제공된 예제에서 "TIGG1"입니다.

9. AWS IoT Greengrass 그룹 배포를 시작하고 배포가 성공했는지 확인합니다. 이제 AWS IoT Greengrass 검색 데모를 성공적으로 실행할 수 있습니다.

coreHTTP 데모

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

이러한 데모는 coreHTTP 라이브러리 사용 방법을 알아보는 데 유용하게 활용할 수 있습니다.

주제

- coreHTTP 상호 인증 데모
- coreHTTP 기본 Amazon S3 업로드 데모
- coreHTTP 기본 S3 다운로드 데모
- coreHTTP 기본 다중 스레드 데모

coreHTTP 상호 인증 데모

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

소개

coreHTTP(상호 인증) 데모 프로젝트에서는 클라이언트와 서버 간 상호 인증을 통한 TLS를 사용하여 HTTP 서버에 연결하는 방법을 보여줍니다. 이 데모에서는 mbedTLS 기반 전송 인터페이스 구현을 사용하여 서버 및 클라이언트 인증 TLS 연결을 설정하고 HTTP의 요청 응답 워크플로를 보여줍니다.

> **Note**
>
> FreeRTOS 데모를 설정하고 실행하려면 FreeRTOS 시작하기의 단계를 따릅니다.

기능

이 데모는 다음 작업을 완료하는 방법을 보여주는 예제가 포함된 단일 애플리케이션 태스크를 생성합니다.

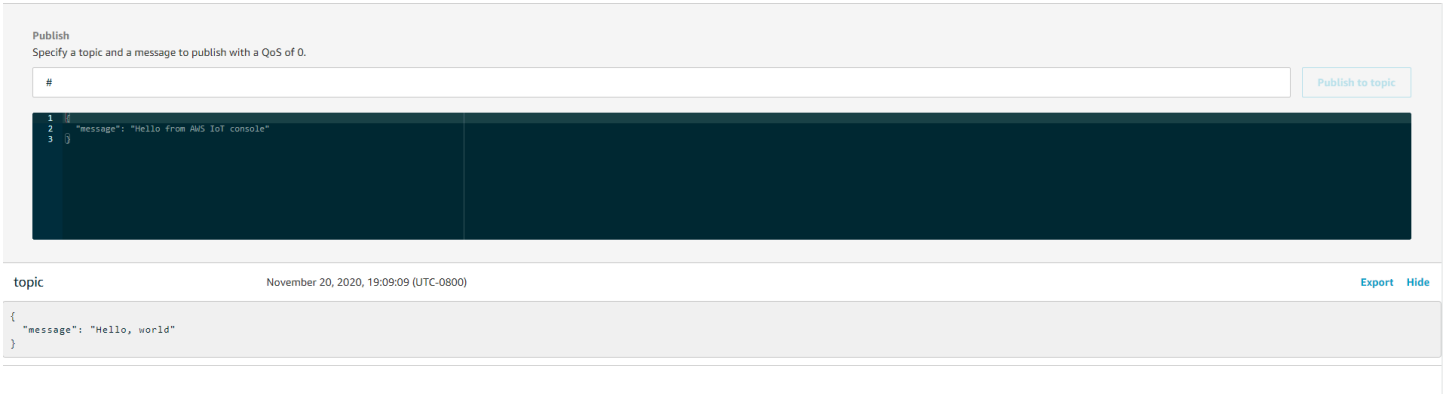
- AWS IoT 엔드포인트의 HTTP 서버에 연결합니다.
- POST 요청 전송
- 응답 수신
- 서버 연결 해제

이러한 단계를 완료하면 데모에서 다음 스크린샷과 비슷한 출력이 생성됩니다.

```
9 1565 [iot_thread] [INFO ][DEMO][1565] ---------STARTING DEMO---------

10 1566 [iot_thread] [INFO ][INIT][1566] SDK successfully initialized.
11 1566 [iot_thread] [INFO ][DEMO][1566] Successfully initialized the demo. Network type for the demo: 4
12 1566 [iot_thread] [INFO] [HTTPDemo] [http_demo_mutual_auth.c:319] 13 1566 [iot_thread] Establishing a TLS session to a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com:8443.14 1566 [iot_thread]
15 1622 [iot_thread] DNS[0x68F5]: The answer to 'a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com' (35.166.102.88) will be stored
16 1622 [iot_thread] DNS[0x68F5]: The answer to 'a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com' (35.166.2.12) will be stored
17 1622 [iot_thread] DNS[0x68F5]: The answer to 'a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com' (44.239.154.164) will be stored
18 1622 [iot_thread] DNS[0x68F5]: The answer to 'a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com' (44.239.97.33) will be stored
19 1622 [iot_thread] DNS[0x68F5]: The answer to 'a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com' (44.238.43.70) will be stored
20 1622 [iot_thread] DNS[0x68F5]: The answer to 'a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com' (52.32.144.134) will be stored
21 2042 [iot_thread] [INFO] [HTTPDemo] [http_demo_mutual_auth.c:393] 22 2042 [iot_thread] Sending HTTP POST request to a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com/topics/topic?qos=1...23 2042 [iot_thread]
24 2082 [iot_thread] [INFO] [HTTPDemo] [http_demo_mutual_auth.c:418] 25 2082 [iot_thread] Received HTTP response from a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com/topics/topic?qos=1...
26 2082 [iot_thread]
27 2082 [iot_thread] [INFO] [HTTPDemo] [http_demo_mutual_auth.c:283] 28 2082 [iot_thread] Demo completed successfully.29 2082 [iot_thread]
30 2082 [iot_thread] [INFO ][DEMO][2082] memory_metrics::freertos_heap::before::bytes::2088152
31 2082 [iot_thread] [INFO ][DEMO][2082] memory_metrics::freertos_heap::after::bytes::1990104
32 2082 [iot_thread] [INFO ][DEMO][2082] memory_metrics::demo_task_stack::before::bytes::1908
33 2082 [iot_thread] [INFO ][DEMO][2082] memory_metrics::demo_task_stack::after::bytes::1908
34 3082 [iot_thread] [INFO ][DEMO][3082] Demo completed successfully.
35 3084 [iot_thread] [INFO ][INIT][3084] SDK cleanup done.
36 3084 [iot_thread] [INFO ][DEMO][3084] -------DEMO FINISHED-------
```

AWS IoT 콘솔은 다음 스크린샷과 유사한 출력을 생성합니다.



소스 코드 구성

데모 소스 파일은 이름이 `http_demo_mutual_auth.c`이며 *`freertos`*`/demos/coreHTTP/` 디렉터리 및 GitHub 웹 사이트에서 찾을 수 있습니다.

AWS IoT HTTP 서버에 연결

connectToServerWithBackoffRetries 함수는 AWS IoT HTTP 서버에 상호 인증된 TLS 연결을 시도합니다. 연결이 실패하면 제한 시간이 경과한 후 다시 시도합니다. 최대 시도 횟수에 도달하거나 최대 제한 시간 값에 도달할 때까지 제한 시간 값은 기하급수적으로 증가합니다. `RetryUtils_BackoffAndSleep` 함수는 기하급수적으로 증가하는 제한 시간 값을 제공하고 최대 시도 횟수에 도달하면 `RetryUtilsRetriesExhausted`를 반환합니다. `connectToServerWithBackoffRetries` 함수는 구성된 시도 횟수 이후에도 브로커에 대한 TLS 연결을 설정할 수 없는 경우 실패 상태를 반환합니다.

HTTP 요청 전송 및 응답 수신

prvSendHttpRequest 함수는 AWS IoT HTTP 서버로 POST 요청을 보내는 방법을 보여줍니다. 에서 REST API를 요청하는 방법에 대한 자세한 내용은 디바이스 통신 프로토콜 - HTTPS를 AWS IoT참조하세요. 응답은 동일한 coreHTTP API 직접 호출 `HTTPClient_Send`로 수신됩니다.

coreHTTP 기본 Amazon S3 업로드 데모

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

소개

이 예제에서는 Amazon Simple Storage Service(S3) HTTP 서버에 PUT 요청을 전송하고 소용량 파일을 업로드하는 방법을 보여줍니다. 업로드 후 파일 크기를 확인하기 위한 GET 요청도 수행합니다. 이 예제에서는 mbedTLS를 사용하는 네트워크 전송 인터페이스를 사용하여 coreHTTP를 실행하는 IoT 디바이스 클라이언트와 Amazon S3 HTTP 서버 간에 상호 인증된 연결을 설정합니다.

> **Note**
>
> FreeRTOS 데모를 설정하고 실행하려면 FreeRTOS 시작하기의 단계를 따릅니다.

단일 스레드와 다중 스레드

coreHTTP 사용 모델에는 단일 스레드와 다중 스레드(멀티태스킹)의 두 가지가 있습니다. 이 섹션의 데모는 하나의 스레드에서 HTTP 라이브러리를 실행하지만 실제로는 단일 스레드 환경에서 coreHTTP를 사용하는 방법을 보여줍니다. 이 데모에서는 한 가지 태스크만 HTTP API를 사용합니다. 단일 스레드 애플리케이션은 HTTP 라이브러리를 반복적으로 직접 호출해야 하지만, 다중 스레드 애플리케이션은 대신 에이전트(또는 대몬(daemon)) 태스크 내에서 백그라운드로 HTTP 요청을 전송할 수 있습니다.

소스 코드 구성

데모 소스 파일은 이름이 `http_demo_s3_upload.c`이며 *`freertos`*`/demos/coreHTTP/` 디렉터리 및 GitHub 웹 사이트에서 찾을 수 있습니다.

Amazon S3 HTTP 서버 연결 구성

이 데모는 미리 서명된 URL을 사용하여 Amazon S3 HTTP 서버에 연결하고 다운로드할 객체에 대한 액세스 권한을 부여합니다. Amazon S3 HTTP 서버의 TLS 연결은 서버 인증만 사용합니다. 애플리케이션 수준에서는 객체 액세스가 미리 서명된 URL 쿼리의 파라미터를 사용하여 인증됩니다. 아래 단계에 따라 AWS에 대한 연결을 구성합니다.

1. AWS 계정 설정:

   a. 아직 생성하지 않았다면 AWS 계정을 생성합니다.

   b. 계정 및 권한은 AWS Identity and Access Management (IAM)을 사용하여 설정됩니다. IAM을 사용하여 계정의 각 사용자에 대한 권한을 관리합니다. 기본적으로 사용자는 루트 소유자가 부여할 때까지는 아무 권한도 없습니다.

      i. AWS 계정에 IAM 사용자를 추가하려면 IAM 사용 설명서를 참조하세요.

      ii. 이 정책을 추가하여 AWS 계정에 FreeRTOS 및에 액세스할 수 AWS IoT 있는 권한을 부여합니다.

      - AmazonS3FullAccess

2. Amazon Simple Storage Service 사용 설명서의 S3 버킷을 생성하려면 어떻게 해야 합니까?의 단계에 따라 Amazon S3에서 버킷을 생성합니다.
3. S3 버킷에 파일 및 폴더를 업로드하려면 어떻게 해야 합니까?의 단계에 따라 파일을 Amazon S3에 업로드합니다.
4. `FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/presigned_urls_gen.py` 파일에 있는 스크립트를 사용하여 미리 서명된 URL을 생성합니다.

사용 지침은 FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generato/README.md 파일을 참조하세요.

기능

이 데모는 먼저 TLS 서버 인증을 사용하여 Amazon S3 HTTP 서버에 연결합니다. 그런 다음 democonfigDEMO_HTTP_UPLOAD_DATA에 지정된 데이터를 업로드하기 위한 HTTP 요청을 생성합니다. 파일을 업로드한 후 파일 크기를 요청하여 파일이 성공적으로 업로드되었는지 확인합니다. 데모의 소스 코드는 GitHub 웹 사이트에서 찾을 수 있습니다.

Amazon S3 HTTP 서버에 연결

connectToServerWithBackoffRetries 함수는 HTTP 서버에 TCP 연결을 시도합니다. 연결이 실패하면 제한 시간이 경과한 후 다시 시도합니다. 최대 시도 횟수에 도달하거나 최대 제한 시간 값에 도달할 때까지 제한 시간 값은 기하급수적으로 증가합니다. connectToServerWithBackoffRetries 함수는 구성된 시도 횟수 이후에도 브로커에 대한 TCP 연결을 설정할 수 없는 경우 실패 상태를 반환합니다.

prvConnectToServer 함수는 서버 인증만 사용하여 Amazon S3 HTTP 서버에 연결하는 방법을 보여줍니다. FreeRTOS-Plus/Source/Application-Protocols/network_transport/freertos_plus_tcp/using_mbedtls/using_mbedtls.c 파일에 구현된 mbedTLS 기반 전송 인터페이스를 사용합니다. prvConnectToServer의 정의는 GitHub 웹 사이트에서 찾을 수 있습니다.

데이터 업로드

prvUploadS3ObjectFile 함수는 PUT 요청을 생성하고 업로드할 파일을 지정하는 방법을 보여줍니다. 파일이 업로드되는 Amazon S3 버킷과 업로드할 파일의 이름은 미리 서명된 URL에 지정됩니다. 메모리를 절약하기 위해 요청 헤더와 응답 수신 모두에 동일한 버퍼가 사용됩니다. 응답은 HTTPClient_Send API 함수를 사용하여 동기식으로 수신됩니다. Amazon S3 HTTP 서버에서 200 OK 응답 상태 코드를 전송해야 합니다. 다른 모든 상태 코드는 오류입니다.

prvUploadS3ObjectFile()의 소스 코드는 GitHub 웹 사이트에서 찾을 수 있습니다.

업로드 확인

prvVerifyS3ObjectFileSize 함수는 prvGetS3ObjectFileSize를 호출하여 S3 버킷에 있는 객체의 크기를 검색합니다. Amazon S3 HTTP 서버는 현재 미리 서명된 URL을 사용하는 HEAD 요청을 지원하지 않으므로 0번째 바이트가 요청됩니다. 파일 크기는 응답의 Content-Range 헤더 필드에

포함되어 있습니다. 서버에서 `206 Partial Content` 응답을 전송해야 합니다. 다른 모든 응답 상태 코드는 오류입니다.

`prvGetS3ObjectFileSize()`의 소스 코드는 GitHub 웹 사이트에서 찾을 수 있습니다.

coreHTTP 기본 S3 다운로드 데모

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

소개

이 데모는 범위 요청을 사용하여 Amazon S3 HTTP 서버에서 파일을 다운로드하는 방법을 보여줍니다. HTTP 요청을 생성하는 데 `HTTPClient_AddRangeHeader`를 사용하면 coreHTTP API에서 범위 요청이 기본적으로 지원됩니다. 마이크로컨트롤러 환경에서는 범위 요청을 적극 권장합니다. 큰 파일을 단일 요청 대신 별도의 범위로 다운로드하면 네트워크 소켓을 차단하지 않고 파일의 각 섹션을 처리할 수 있습니다. 범위 요청을 사용하면 TCP 연결에서 재전송이 필요한 패킷 손실의 위험이 줄어들어 디바이스의 전력 소비가 향상됩니다.

이 예제에서는 mbedTLS를 사용하는 네트워크 전송 인터페이스를 사용하여 coreHTTP를 실행하는 IoT 디바이스 클라이언트와 Amazon S3 HTTP 서버 간에 상호 인증된 연결을 설정합니다.

> **Note**
>
> FreeRTOS 데모를 설정하고 실행하려면 FreeRTOS 시작하기의 단계를 따릅니다.

단일 스레드와 다중 스레드

coreHTTP 사용 모델에는 단일 스레드와 다중 스레드(멀티태스킹)의 두 가지가 있습니다. 이 섹션의 데모는 하나의 스레드에서 HTTP 라이브러리를 실행하지만 실제로는 단일 스레드 환경에서 coreHTTP를 사용하는 방법을 보여줍니다(데모에서는 하나의 태스크만 HTTP API를 사용함). 단일 스레드 애플리케이션은 HTTP 라이브러리를 반복적으로 직접 호출해야 하지만, 다중 스레드 애플리케이션은 대신 에이전트(또는 대몬(daemon)) 태스크 내에서 백그라운드로 HTTP 요청을 전송할 수 있습니다.

소스 코드 구성

데모 프로젝트는 이름이 `http_demo_s3_download.c`이며 *`freertos`*`/demos/coreHTTP/` 디렉터리 및 GitHub 웹 사이트에서 찾을 수 있습니다.

Amazon S3 HTTP 서버 연결 구성

이 데모는 미리 서명된 URL을 사용하여 Amazon S3 HTTP 서버에 연결하고 다운로드할 객체에 대한 액세스 권한을 부여합니다. Amazon S3 HTTP 서버의 TLS 연결은 서버 인증만 사용합니다. 애플리케이션 수준에서는 객체 액세스가 미리 서명된 URL 쿼리의 파라미터를 사용하여 인증됩니다. 아래 단계에 따라 AWS에 대한 연결을 구성합니다.

1. AWS 계정 설정:

   a. 아직 생성하지 않은 경우 AWS 계정을 생성하고 활성화합니다.

   b. 계정 및 권한은 AWS Identity and Access Management (IAM)을 사용하여 설정됩니다. IAM을 사용하면 계정의 각 사용자에게 부여된 권한을 관리할 수 있습니다. 기본적으로 사용자는 루트 소유자가 부여할 때까지는 아무 권한도 없습니다.

      i. AWS 계정에 IAM 사용자를 추가하려면 IAM 사용 설명서를 참조하세요.

      ii. 다음 정책을 추가하여 AWS 계정에 FreeRTOS 및에 액세스할 수 AWS IoT 있는 권한을 부여합니다.

      - AmazonS3FullAccess

2. Amazon Simple Storage Service 사용 설명서의 S3 버킷을 생성하려면 어떻게 해야 합니까?의 단계에 따라 S3에서 버킷을 생성합니다.
3. S3 버킷에 파일 및 폴더를 업로드하려면 어떻게 해야 합니까?의 단계에 따라 파일을 S3에 업로드합니다.
4. `FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/presigned_urls_gen.py` 파일에 있는 스크립트를 사용하여 미리 서명된 URL을 생성합니다. 사용 지침은 `FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generato/README.md` 섹션을 참조하세요.

기능

데모는 먼저 파일의 크기를 검색합니다. 그런 다음 democonfigRANGE_REQUEST_LENGTH의 범위 크기를 사용하여 루프에서 각 바이트 범위를 순차적으로 요청합니다.

데모의 소스 코드는 GitHub 웹 사이트에서 찾을 수 있습니다.

Amazon S3 HTTP 서버에 연결

connectToServerWithBackoffRetries() 함수는 HTTP 서버에 TCP 연결을 시도합니다. 연결이 실패하면 제한 시간이 경과한 후 다시 시도합니다. 최대 시도 횟수에 도달하거나 최대 제한 시간 값에 도달할 때까지 제한 시간 값은 기하급수적으로 증가합니다. `connectToServerWithBackoffRetries()` 함수는 구성된 시도 횟수 이후에도 서버에 대한 TCP 연결을 설정할 수 없는 경우 실패 상태를 반환합니다.

`prvConnectToServer()` 함수는 서버 인증만 사용하여 Amazon S3 HTTP 서버에 연결하는 방법을 보여줍니다. FreeRTOS-Plus/Source/Application-Protocols/network_transport/freertos_plus_tcp/using_mbedtls/using_mbedtls.c 파일에 구현된 mbedTLS 기반 전송 인터페이스를 사용합니다.

`prvConnectToServer()`의 소스 코드는 GitHub에서 찾을 수 있습니다.

범위 요청 생성

`HTTPClient_AddRangeHeader()` API 함수는 바이트 범위를 HTTP 요청 헤더로 직렬화하여 범위 요청을 구성할 수 있도록 지원합니다. 이 데모에서는 범위 요청을 사용하여 파일 크기를 검색하고 파일의 각 섹션을 요청합니다.

`prvGetS3ObjectFileSize()` 함수는 S3 버킷에 있는 파일의 크기를 검색합니다. Amazon S3에 대한 이 첫 번째 요청에 `Connection: keep-alive` 헤더가 추가되어 응답이 전송된 후에도 연결을 열린 상태로 유지합니다. S3 HTTP 서버는 현재 미리 서명된 URL을 사용하는 HEAD 요청을 지원하지 않으므로 0번째 바이트가 요청됩니다. 파일 크기는 응답의 `Content-Range` 헤더 필드에 포함되어 있습니다. 서버에서 `206 Partial Content` 응답을 전송해야 하며, 수신된 다른 모든 응답 상태 코드는 오류입니다.

`prvGetS3ObjectFileSize()`의 소스 코드는 GitHub에서 찾을 수 있습니다.

이 데모는 파일 크기를 검색한 후 다운로드할 파일의 각 바이트 범위에 대해 새 범위 요청을 생성합니다. 파일의 각 섹션에 대해 `HTTPClient_AddRangeHeader()`를 사용합니다.

범위 요청 전송 및 응답 수신

`prvDownloadS3ObjectFile()` 함수는 전체 파일이 다운로드될 때까지 루프에서 범위 요청을 전송합니다. `HTTPClient_Send()` API 함수는 요청을 전송하고 동기식으로 응답을 수신합니다. 함수가 반환되면 응답이 `xResponse`로 수신됩니다. 그런 다음 상태 코드가 `206 Partial Content`인지 확인되고 지금까지 다운로드한 바이트 수가 `Content-Length` 헤더 값만큼 증가합니다.

`prvDownloadS3ObjectFile()`의 소스 코드는 GitHub에서 찾을 수 있습니다.

coreHTTP 기본 다중 스레드 데모

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

소개

이 데모는 FreeRTOS의 스레드 안전 대기열을 사용하여 처리 대기 중인 요청 및 응답을 보관합니다. 이 데모에서 주목해야 할 세 가지 태스크가 있습니다.

- 메인 태스크는 요청 대기열에 요청이 표시되기를 기다립니다. 네트워크를 통해 해당 요청을 전송한 다음 응답을 응답 대기열에 배치합니다.
- 요청 태스크는 서버에 전송할 HTTP 라이브러리 요청 객체를 생성하여 요청 대기열에 배치합니다. 각 요청 객체는 애플리케이션이 다운로드하도록 구성한 S3 파일의 바이트 범위를 지정합니다.
- 응답 태스크는 응답 대기열에 응답이 표시되기를 기다립니다. 그런 다음 수신한 모든 응답을 로그합니다.

이 기본 다중 스레드 데모는 서버 인증만 포함된 TLS 연결을 사용하도록 구성되어 있으며, 이는 Amazon S3 HTTP 서버에 필요합니다. 애플리케이션 계층 인증은 미리 서명된 URL 쿼리의 서명 버전 4 파라미터를 사용하여 수행됩니다.

소스 코드 구성

데모 프로젝트는 이름이 `http_demo_s3_download_multithreaded.c`이며 *`freertos`*`/demos/coreHTTP/` 디렉터리 및 GitHub 웹 사이트에서 찾을 수 있습니다.

데모 프로젝트 빌드

이 데모 프로젝트는 Visual Studio의 무료 Community 에디션을 사용합니다. 데모를 빌드하려면

1. Visual Studio IDE 내에서 `mqtt_multitask_demo.sln` Visual Studio 솔루션 파일을 엽니다.
2. IDE의 빌드 메뉴에서 솔루션 빌드를 선택합니다.

> **Note**
>
> Microsoft Visual Studio 2017 또는 이전 버전을 사용하는 경우 프로젝트 -> RTOSDemos 속성 -> 플랫폼 도구 세트에서 현재 버전과 호환되는 플랫폼 도구 세트를 선택해야 합니다.

데모 프로젝트 구성

이 데모는 FreeRTOS+TCP TCP/IP 스택을 사용하므로 TCP/IP 스타터 프로젝트에 제공된 지침을 따라 다음을 수행합니다.

1. 필수 구성 요소(예: WinPCap)를 설치합니다.
2. 선택적으로 고정 또는 동적 IP 주소, 게이트웨이 주소 및 넷마스크를 설정합니다.
3. 선택적으로 MAC 주소를 설정합니다.
4. 호스트 시스템에서 이더넷 네트워크 인터페이스를 선택합니다.
5. 중요한 것은 HTTP 데모를 실행하기 전에 네트워크 연결을 테스트하는 것입니다.

Amazon S3 HTTP 서버 연결 구성

coreHTTP 기본 다운로드 데모의 Amazon S3 HTTP 서버 연결 구성 지침을 따릅니다.

기능

이 데모에서 총 세 가지 태스크가 생성됩니다.

- 네트워크를 통해 요청을 전송하고 응답을 수신하는 태스크.
- 전송할 요청을 생성하는 태스크.
- 수신된 응답을 처리하는 태스크.

이 데모에서, 메인 태스크:

1. 요청 및 응답 대기열을 생성합니다.
2. 서버와의 연결을 생성합니다.
3. 요청 및 응답 태스크를 생성합니다.
4. 요청 대기열이 네트워크를 통해 요청을 전송할 때까지 기다립니다.

5. 네트워크를 통해 수신한 응답을 응답 대기열에 배치합니다.

요청 태스크:

1. 각 범위 요청을 생성합니다.

응답 태스크:

1. 수신된 각 응답을 처리합니다.

Typedefs

이 데모는 다중 스레딩을 지원하기 위해 다음 구조를 정의합니다.

요청 항목

다음 구조는 요청 대기열에 배치할 요청 항목을 정의합니다. 요청 태스크가 HTTP 요청을 생성하면 요청 항목이 대기열에 복사됩니다.

```
/**
 * @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;
```

응답 항목

다음 구조는 응답 대기열에 배치할 응답 항목을 정의합니다. 메인 HTTP 태스크가 네트워크를 통해 응답을 수신하면 응답 항목이 대기열에 복사됩니다.

```
/**
```

```
 * @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;
```

메인 HTTP 전송 태스크

메인 애플리케이션 태스크:

1. 호스트 주소의 미리 서명된 URL을 파싱하여 Amazon S3 HTTP 서버와의 연결을 설정합니다.
2. S3 버킷 내 객체에 대한 경로의 미리 서명된 URL을 파싱합니다.
3. 서버 인증 TLS를 사용하여 Amazon S3 HTTP 서버에 연결합니다.
4. 요청 및 응답 대기열을 생성합니다.
5. 요청 및 응답 태스크를 생성합니다.

`prvHTTPDemoTask()` 함수가 이 설정을 수행하고 데모 상태를 제공합니다. 이 함수의 소스 코드는 GitHub에서 찾을 수 있습니다.

`prvDownloadLoop()` 함수에서 메인 태스크는 요청 대기열의 요청을 차단하고 대기합니다. 요청을 수신하면 `HTTPClient_Send()` API 함수를 사용하여 전송합니다. API 함수가 성공하면 응답이 응답 대기열에 배치합니다.

`prvDownloadLoop()`의 소스 코드는 GitHub에서 찾을 수 있습니다.

HTTP 요청 태스크

요청 태스크는 `prvRequestTask` 함수에 지정됩니다. 이 함수의 소스 코드는 GitHub에서 찾을 수 있습니다.

요청 태스크는 Amazon S3 버킷에서 파일 크기를 검색합니다. 이 작업은 `prvGetS3ObjectFileSize` 함수로 수행됩니다. Amazon S3에 대한 이 첫 번째 요청에 'Connection: keep-alive' 헤더가 추가되어 응답이 전송된 후에도 연결을 열린 상태로 유지합니다. Amazon S3 HTTP

서버는 현재 미리 서명된 URL을 사용하는 HEAD 요청을 지원하지 않으므로 0번째 바이트가 요청됩니다. 파일 크기는 응답의 `Content-Range` 헤더 필드에 포함되어 있습니다. 서버에서 `206 Partial Content` 응답을 전송해야 하며, 수신된 다른 모든 응답 상태 코드는 오류입니다.

`prvGetS3ObjectFileSize`의 소스 코드는 GitHub에서 찾을 수 있습니다.

요청 태스크는 파일 크기를 검색한 후 파일의 각 범위를 요청합니다. 각 범위 요청은 기본 태스크가 전송할 수 있도록 요청 대기열에 배치됩니다. 파일 범위는 데모 사용자가 매크로 `democonfigRANGE_REQUEST_LENGTH`에서 구성합니다. HTTP 클라이언트 라이브러리 API에서 `HTTPClient_AddRangeHeader` 함수를 사용하는 범위 요청이 기본적으로 지원됩니다. `prvRequestS3ObjectRange` 함수는 `HTTPClient_AddRangeHeader()` 사용 방법을 보여줍니다.

`prvRequestS3ObjectRange` 함수의 소스 코드는 GitHub에서 찾을 수 있습니다.

HTTP 응답 태스크

응답 태스크는 응답 대기열에서 네트워크를 통해 수신되는 응답을 대기합니다. 메인 태스크는 HTTP 응답을 성공적으로 수신하면 응답 대기열을 채웁니다. 이 태스크는 상태 코드, 헤더, 본문을 로그하여 응답을 처리합니다. 예를 들어 실제 애플리케이션은 응답 본문을 플래시 메모리에 쓰는 방식으로 응답을 처리할 수 있습니다. 응답 상태 코드가 `206 partial content`가 아닌 경우 태스크는 데모가 실패한다고 메인 태스크에 알립니다. 응답 태스크는 `prvResponseTask` 함수에 지정됩니다. 이 함수의 소스 코드는 GitHub에서 찾을 수 있습니다.

AWS IoT 작업 라이브러리 데모

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

소개

AWS IoT 작업 라이브러리 데모에서는 MQTT 연결을 통해 AWS IoT 작업 서비스에 연결하고,에서 작업을 검색하고 AWS IoT, 디바이스에서 처리하는 방법을 보여줍니다. AWS IoT Jobs 데모 프로젝트는 FreeRTOS Windows 포트를 사용하므로 Windows에서 Visual Studio 커뮤니티 버전으로 빌드하고 평가할 수 있습니다. 마이크로컨트롤러 하드웨어는 필요하지 않습니다. 데모는 AWS IoT MQTT 상호 인증 데모와 동일한 방식으로 TLS를 사용하여 MQTT 브로커에 대한 보안 연결을 설정합니다.

> **Note**
>
> FreeRTOS 데모를 설정하고 실행하려면 FreeRTOS 시작하기의 단계를 따릅니다.

소스 코드 구성

데모 코드는 `jobs_demo.c` 파일에 있으며 GitHub 웹 사이트 또는 *`freertos`*`/demos/jobs_for_aws/` 디렉터리에서 찾을 수 있습니다.

AWS IoT MQTT 브로커 연결 구성

이 데모에서는 AWS IoT MQTT 브로커에 대한 MQTT 연결을 사용합니다. 이 연결은 MQTT 상호 인증 데모와 동일한 방식으로 구성됩니다.

기능

데모는에서 작업을 수신 AWS IoT 하고 디바이스에서 처리하는 데 사용되는 워크플로를 보여줍니다. 데모는 대화형이며 AWS IoT 콘솔 또는 AWS Command Line Interface ()를 사용하여 작업을 생성해야 합니다AWS CLI. 작업 생성에 대한 자세한 내용은 AWS CLI 명령 참조의 create-job을 참조하세요. 이 데모는 메시지를 콘솔에 출력할 수 있도록 작업 문서에서 `action` 키가 `print`로 설정되어야 합니다.

이 작업 문서의 형식은 다음과 같습니다.

```
{
    "action": "print",
    "message": "ADD_MESSAGE_HERE"
}
```

AWS CLI 를 사용하여 다음 예제 명령과 같이 작업을 생성할 수 있습니다.

```
aws iot create-job \
    --job-id t12 \
    --targets arn:aws:iot:region:123456789012:thing/device1 \
    --document '{"action":"print","message":"hello world!"}'
```

또한 이 데모는 메시지를 주제에 다시 게시할 수 있도록 `action` 키가 `publish`로 설정된 작업 문서도 사용합니다. 이 작업 문서의 형식은 다음과 같습니다.

```
{
    "action": "publish",
    "message": "ADD_MESSAGE_HERE",
    "topic": "topic/name/here"
}
```

데모는 데모를 종료하도록 action 키가 exit로 설정된 작업 문서를 수신할 때까지 루프를 반복합니다. 이 작업 문서의 형식은 다음과 같습니다.

```
{
    "action: "exit"
}
```

Jobs 데모의 진입점

Jobs 데모 진입점 함수의 소스 코드는 GitHub에서 찾을 수 있습니다. 이 함수는 다음 작업을 수행합니다.

1. mqtt_demo_helpers.c의 헬퍼 함수를 사용하여 MQTT 연결을 설정합니다.
2. mqtt_demo_helpers.c의 헬퍼 함수를 사용하여 NextJobExecutionChanged API에 대한 MQTT 주제를 구독합니다. 주제 문자열은 AWS IoT 작업 라이브러리에서 정의한 매크로를 사용하여 이전에 어셈블됩니다.
3. mqtt_demo_helpers.c의 헬퍼 함수를 사용하여 StartNextPendingJobExecution API에 대한 MQTT 주제에 게시합니다. 주제 문자열은 AWS IoT 작업 라이브러리에서 정의한 매크로를 사용하여 이전에 어셈블됩니다.
4. prvEventCallback를 반복적으로 직접 호출하여 처리를 위해 MQTT_ProcessLoop에 전달되는 메시지를 수신합니다.
5. 데모가 종료 작업을 수신하면 mqtt_demo_helpers.c 파일의 헬퍼 함수를 사용하여 MQTT 주제 구독을 취소하고 연결을 끊습니다.

수신된 MQTT 메시지에 대한 콜백

prvEventCallback 함수는 AWS IoT 작업 라이브러리Jobs_MatchTopic에서를 호출하여 수신 MQTT 메시지를 분류합니다. 메시지 유형이 새 작업에 해당하는 경우 prvNextJobHandler()가 호출됩니다.

PrvNextJobHandler 함수 및 이 함수가 호출하는 함수는 JSON 형식의 메시지에서 작업 문서를 파싱하고 작업에 지정된 작업을 실행합니다. 특히 주목할 것은 prvSendUpdateForJob 함수입니다.

실행 중인 작업에 대한 업데이트 전송

prvSendUpdateForJob() 함수는 Jobs 라이브러리에서 `Jobs_Update()`를 호출하여 바로 이어지는 MQTT 게시 작업에 사용되는 주제 문자열을 채웁니다.

coreMQTT 데모

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

이러한 데모는 coreMQTT 라이브러리 사용 방법을 알아보는 데 유용하게 활용할 수 있습니다.

주제

- coreMQTT 상호 인증 데모
- coreMQTT 연결 공유 데모

coreMQTT 상호 인증 데모

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

소개

coreMQTT 상호 인증 데모 프로젝트는 클라이언트와 서버 간 상호 인증 TLS를 사용하여 MQTT 브로커에 연결하는 방법을 보여줍니다. 이 데모에서는 mbedTLS 기반 전송 인터페이스 구현을 사용하여 서버 및 클라이언트 인증 TLS 연결을 설정하고 QoS 1 수준에서 MQTT의 구독-게시 워크플로를 보여줍니다. 이 데모는 주제 필터를 구독하고, 필터와 일치하는 주제에 게시하고, QoS 1 수준에서 서버로부터 해당 메시지가 다시 수신될 때까지 대기합니다. 브로커에 게시하고 브로커로부터 동일한 메시지를 다시 수신하는 이 주기는 무한정 반복됩니다. 이 데모의 메시지는 QoS 1 수준에서 전송되므로 MQTT 사양에 따라 최소 한 번의 전송이 보장됩니다.

> **Note**
>
> FreeRTOS 데모를 설정하고 실행하려면 FreeRTOS 시작하기의 단계를 따릅니다.

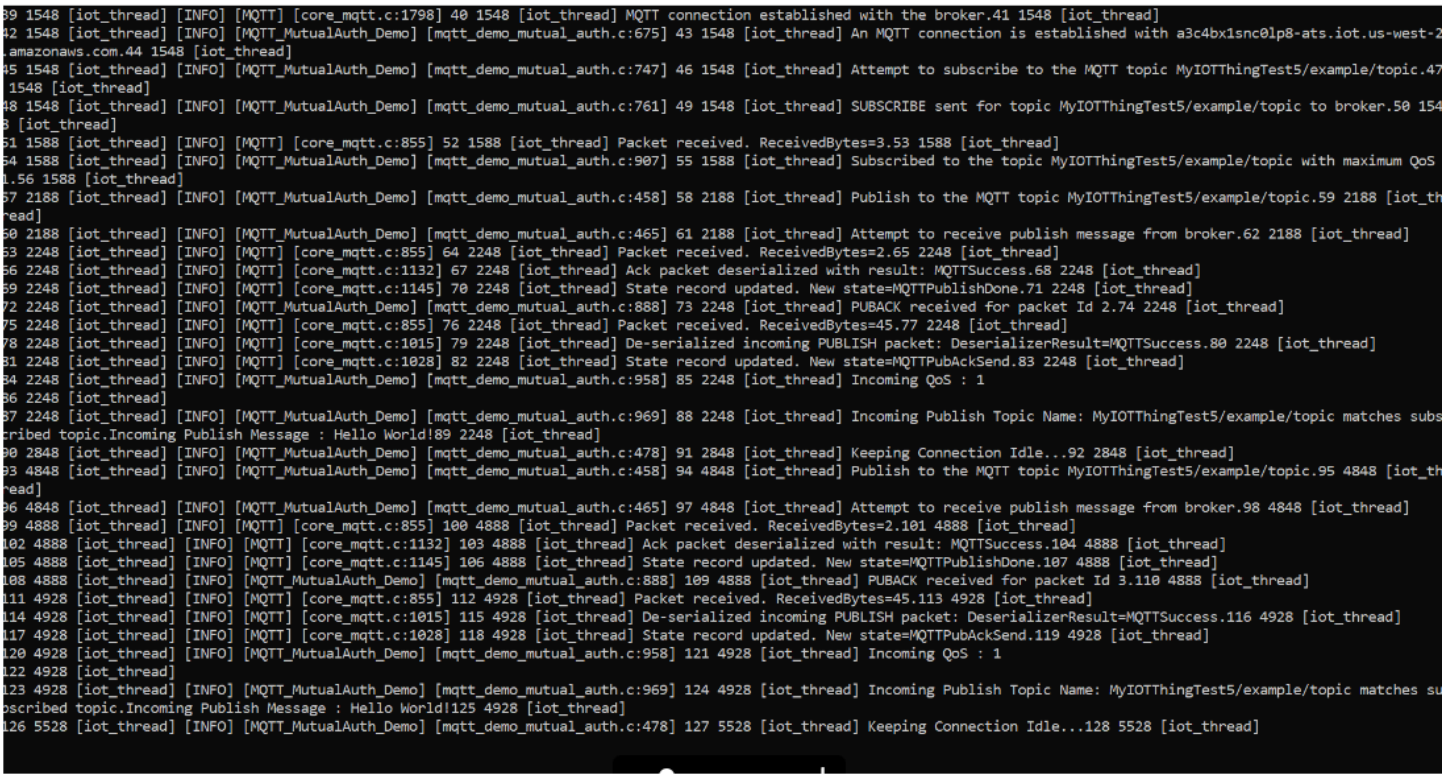
소스 코드

데모 소스 파일은 이름이 `mqtt_demo_mutual_auth.c`이며 *`freertos`*`/demos/coreMQTT/` 디렉터리 및 GitHub 웹 사이트에서 찾을 수 있습니다.

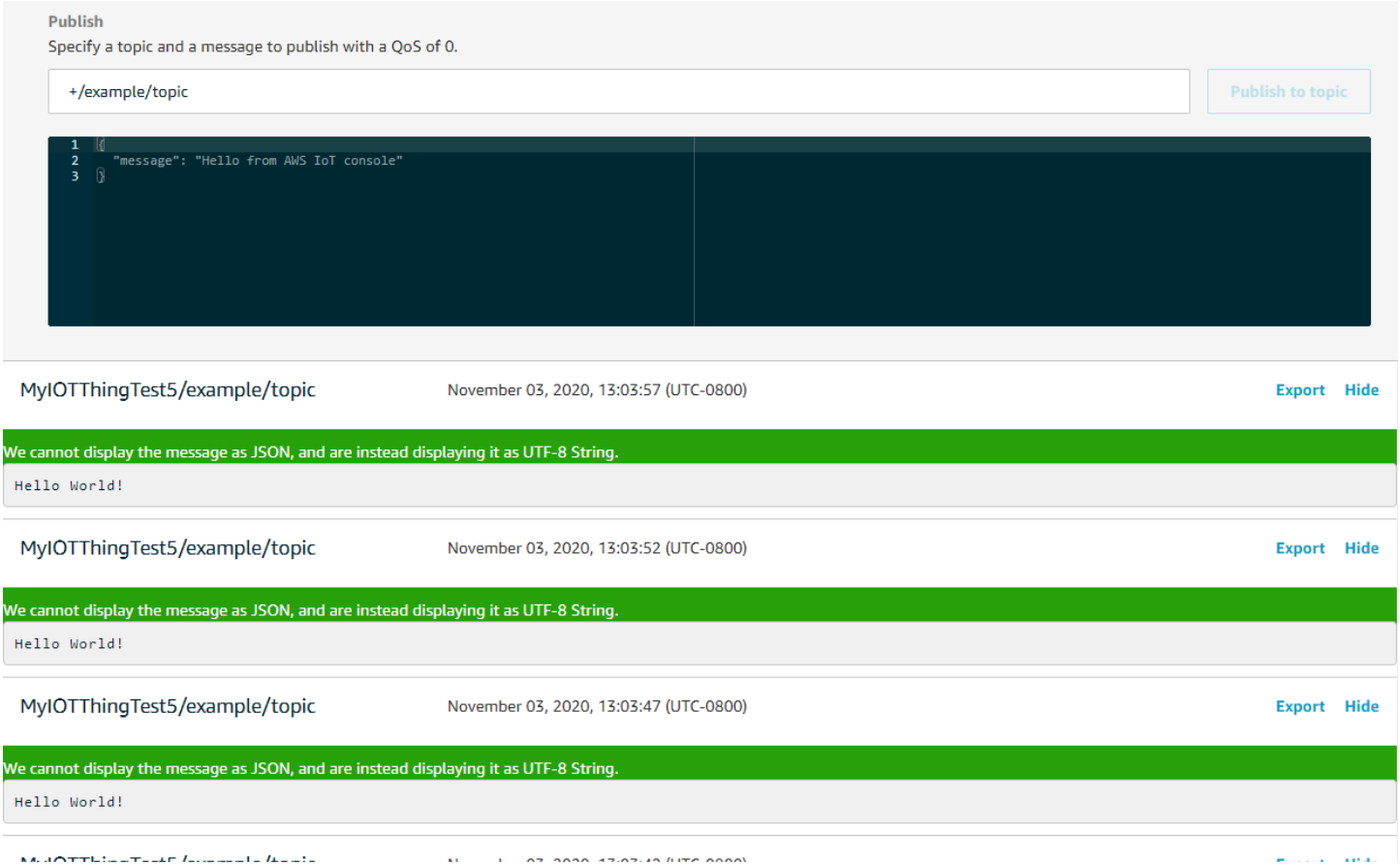
기능

이 데모는 브로커에 연결하고, 브로커의 주제를 구독하고, 브로커의 주제에 게시한 다음, 마지막으로 브로커와의 연결을 끊는 방법을 보여주는 일련의 예제를 반복하는 단일 애플리케이션 작업을 생성합니다. 데모 애플리케이션은 동일한 주제를 구독하고 게시합니다. 데모가 MQTT 브로커에 메시지를 게시할 때마다 브로커는 동일한 메시지를 데모 애플리케이션에 다시 보냅니다.

데모를 성공적으로 완료하면 다음 이미지와 비슷한 출력이 생성됩니다.



AWS IoT 콘솔은 다음 이미지와 유사한 출력을 생성합니다.



지수 백오프 및 지터를 사용한 재시도 로직

prvBackoffForRetry 함수는 서버와의 실패한 네트워크 작업(예: TLS 연결 또는 MQTT 구독 요청)을 지수 백오프 및 지터를 사용하여 재시도할 수 있는 방법을 보여줍니다. 이 함수는 다음 재시도의 백오프 기간을 계산하고, 재시도 횟수가 모두 소진되지 않은 경우 백오프 지연을 수행합니다. 백오프 기간을 계산하려면 난수를 생성해야 하므로 함수는 PKCS11 모듈을 사용하여 난수를 생성합니다. 공급업체 플랫폼에서 지원하는 경우 PKCS11 모듈을 사용하면 순수 난수 생성기(TRNG)에 액세스할 수 있습니다. 연결 재시도 시 디바이스와의 충돌 가능성이 완화되도록 디바이스별 엔트로피 소스를 난수 생성기에 시드하는 것이 좋습니다.

MQTT 브로커에 연결

prvConnectToServerWithBackoffRetries 함수는 MQTT 브로커에 상호 인증된 TLS 연결을 시도합니다. 연결이 실패하면 백오프 기간 후에 다시 시도합니다. 최대 시도 횟수에 도달하거나 최대 백오프 기간에 도달할 때까지 백오프 기간은 기하급수적으로 증가합니다. `BackoffAlgorithm_GetNextBackoff` 함수는 기하급수적으로 증가하는 백오프 값을 제공하고 최대 시도 횟수에 도달하면 `RetryUtilsRetriesExhausted`를 반환합니다.

prvConnectToServerWithBackoffRetries 함수는 구성된 시도 횟수 이후에도 브로커에 대한 TLS 연결을 설정할 수 없는 경우 실패 상태를 반환합니다.

prvCreateMQTTConnectionWithBroker 함수는 클린 세션을 통해 MQTT 브로커에 대한 MQTT 연결을 설정하는 방법을 보여줍니다. 이 함수는 FreeRTOS-Plus/Source/Application-Protocols/platform/freertos/transport/src/tls_freertos.c 파일에 구현된 TLS 전송 인터페이스를 사용합니다. xConnectInfo에서 브로커의 유지 기간(초)을 설정하고 있다는 점을 염두에 두세요.

다음 함수는 MQTT_Init 함수를 사용하여 MQTT 컨텍스트에서 TLS 전송 인터페이스와 시간 함수를 설정하는 방법을 보여줍니다. 또한 이벤트 콜백 함수 포인터(prvEventCallback)가 어떻게 설정되는지도 보여줍니다. 이 콜백은 들어오는 메시지를 보고하는 데 사용됩니다.

MQTT 주제 구독

prvMQTTSubscribeWithBackoffRetries 함수는 MQTT 브로커에서 주제 필터를 구독하는 방법을 보여줍니다. 이 예제에서는 하나의 주제 필터를 구독하는 방법을 보여 주지만, 동일한 subscribe API 직접 호출에서 주제 필터 목록을 전달하여 둘 이상의 주제 필터를 구독할 수도 있습니다. 또한 MQTT 브로커가 구독 요청을 거부하는 경우 구독은 지수 백오프를 사용하여 RETRY_MAX_ATTEMPTS에 설정된 횟수만큼 재시도됩니다.

주제에 게시

prvMQTTPublishToTopic 함수는 MQTT 브로커에 주제 필터를 게시하는 방법을 보여줍니다.

들어오는 메시지 수신

애플리케이션은 앞서 설명한 대로 브로커에 연결하기 전에 이벤트 콜백 함수를 등록합니다. prvMQTTDemoTask 함수는 MQTT_ProcessLoop 함수를 호출하여 들어오는 메시지를 수신합니다. 함수는 들어오는 MQTT 메시지를 수신하면 애플리케이션이 등록한 이벤트 콜백 함수를 호출합니다. prvEventCallback 함수는 이러한 이벤트 콜백 함수의 한 예입니다. prvEventCallback은 들어오는 패킷 유형을 검사하고 적절한 핸들러를 호출합니다. 아래 예제에서 함수는 들어오는 게시 메시지를 처리하기 위해 prvMQTTProcessIncomingPublish()를 호출하거나 확인(ACK)을 처리하기 위해 prvMQTTProcessResponse()를 호출합니다.

들어오는 MQTT 게시 패킷 처리

prvMQTTProcessIncomingPublish 함수는 MQTT 브로커의 게시 패킷을 처리하는 방법을 보여줍니다.

주제 구독 취소

워크플로의 마지막 단계는 브로커가 mqttexampleTOPIC에서 게시된 메시지를 보내지 않도록 주제 구독을 취소하는 것입니다. 다음은 prvMQTTUnsubscribeFromTopic 함수의 정의입니다.

데모에 사용된 루트 CA 변경

기본적으로 FreeRTOS 데모는 Amazon Root CA 1 인증서(RSA 2048비트 키)를 사용하여 AWS IoT Core 서버로 인증합니다. Amazon 루트 CA 3 인증서(ECC 256비트 키)를 비롯한 다른 CA 인증서를 서버 인증에 사용할 수 있습니다. coreMQTT 상호 인증 데모에서 루트 CA를 변경하려면

1. 텍스트 편집기에서 *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/mqtt_demo_mutual_auth_config.h 파일을 엽니다.
2. 파일에서 다음 줄을 찾습니다.

   ```
   * #define democonfigROOT_CA_PEM    "...insert here..."
   ```

   이 줄의 주석 처리를 해제하고 필요한 경우 주석 블록 끝 */를 붙여넣습니다.
3. 사용하려는 CA 인증서를 복사하여 "...insert here..." 텍스트에 붙여넣습니다. 결과는 다음 예제와 같아야 합니다.

   ```
   #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. (선택 사항) 다른 데모의 루트 CA를 변경할 수 있습니다. 각 *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/*demo-name*_config.h 파일마다 1~3단계를 반복합니다.

coreMQTT 연결 공유 데모

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

소개

coreMQTT 연결 공유 데모 프로젝트에서는 다중 스레드 애플리케이션을 사용하여 클라이언트와 서버 간의 상호 인증과 함께 TLS를 사용하여 AWS MQTT 브로커에 대한 연결을 설정하는 방법을 보여줍니다. 이 데모에서는 mbedTLS 기반 전송 인터페이스 구현을 사용하여 서버 및 클라이언트 인증 TLS 연결을 설정하고 QoS 1 수준에서 MQTT의 구독-게시 워크플로를 보여줍니다. 이 데모는 주제 필터를 구독하고, 필터와 일치하는 주제에 게시하고, QoS 1 수준에서 서버로부터 해당 메시지가 다시 수신될 때까지 대기합니다. 브로커에 게시하고 브로커로부터 동일한 메시지를 다시 수신하는 이 주기는 무한정 반복됩니다. 이 데모의 메시지는 QoS 1 수준에서 전송되므로 MQTT 사양에 따라 최소 한 번의 전송이 보장됩니다.

> **Note**
>
> FreeRTOS 데모를 설정하고 실행하려면 FreeRTOS 시작하기의 단계를 따릅니다.

이 데모는 스레드 안전 대기열을 사용하여 MQTT API와 상호 작용하는 명령을 보관합니다. 이 데모에서 주목해야 할 네 가지 태스크가 있습니다.

- 명령(기본) 태스크는 명령 대기열에서 명령을 가져와 처리합니다. 다른 태스크는 처리할 명령을 이 대기열에 배치합니다. 이 태스크는 루프를 시작하여 명령을 처리합니다. 종료 명령을 수신하면 이 태스크는 루프를 중단합니다.
- 동기 게시자 태스크는 일련의 게시 작업을 생성하여 명령 대기열로 푸시합니다. 그러면 명령 태스크가 이러한 작업을 실행합니다. 이 태스크는 동기 게시를 사용합니다. 즉, 이 태스크는 각 게시 작업이 완료될 때까지 기다렸다가 다음 작업을 예약합니다.
- 비동기 게시자 태스크는 일련의 게시 작업을 생성하여 명령 대기열로 푸시합니다. 그러면 명령 태스크가 이러한 작업을 실행합니다. 이 태스크와 이전 태스크의 차이점은 게시 작업이 완료될 때까지 기다리지 않고 다음 작업을 예약한다는 것입니다. 모든 게시 작업이 대기열에 추가된 후 각 게시 작업의 상태를 확인합니다. 동기 게시와 비동기 게시는 이러한 태스크의 동작에서만 차이가 있습니다. 실제 게시 명령에는 차이가 없습니다.

- 구독자 태스크는 두 게시자 태스크가 게시하는 모든 메시지의 주제와 일치하는 주제 필터에 대한 MQTT 구독을 생성합니다. 이 태스크는 루프를 시작하여 다른 태스크가 게시한 메시지를 다시 수신할 때까지 기다립니다.

태스크에는 수신한 메시지를 보관하는 대기열이 있을 수 있습니다. 명령 태스크는 들어오는 메시지를 들어오는 주제를 구독하는 모든 태스크의 대기열로 푸시합니다.

이 데모에서는 상호 인증 TLS 연결을 사용하여 AWS에 연결합니다. 데모 도중 예기치 않게 네트워크 연결이 끊어지면 클라이언트는 지수 백오프 로직을 사용하여 재연결을 시도합니다. 클라이언트가 성공적으로 재연결되었지만 브로커가 이전 세션을 재개할 수 없는 경우 클라이언트는 이전 세션과 동일한 주제를 다시 구독합니다.

단일 스레드와 다중 스레드

coreMQTT 사용 모델에는 단일 스레드와 다중 스레드(멀티태스킹)의 두 가지가 있습니다. 이 데모에서는 자체 다중 스레딩 체계를 만드는 방법을 보여줍니다. 에이전트(또는 대몬(daemon)) 태스크 내에서 MQTT 프로토콜을 백그라운드에서 실행하는 또 다른 다중 스레드 예제도 있습니다. 자세한 내용은 MWTT Agent and Demos using coreMQTT를 참조하세요. 에이전트 태스크에서 MQTT 프로토콜을 실행하는 경우 MQTT 상태를 명시적으로 관리하거나 `MQTT_ProcessLoop` 함수를 직접 호출할 필요가 없습니다. 또한 에이전트 태스크를 사용하면 뮤텍스와 같은 동기화 프리미티브 없이도 여러 애플리케이션 태스크가 단일 MQTT 연결을 공유할 수 있습니다.

소스 코드

데모 소스 파일은 이름이 `mqtt_demo_connection_sharing.c`이며 *`freertos`*`/demos/coreMQTT/` 디렉터리 및 GitHub 웹 사이트에서 찾을 수 있습니다.

기능

이 데모는 모두 네 개의 태스크를 생성합니다. 세 개는 MQTT API 호출을 요청하는 태스크이고 다른 하나는 이러한 요청을 처리하는 메인 태스크입니다. 이 데모에서는 메인 태스크가 세 개의 서브 태스크를 생성하는 루프를 시작하고, 처리 루프를 호출하고, 나중에 정리됩니다. 메인 태스크는 브로커에 대한 단일 MQTT 연결을 생성하고, 이 연결은 서브 태스크 간에 공유됩니다. 두 개의 서브 태스크는 브로커에 메시지를 게시하고, 세 번째 서브 태스크는 게시된 메시지의 모든 주제와 일치하는 주제 필터에 대한 MQTT 구독을 사용하여 메시지를 다시 수신합니다.

Typedefs

이 데모에서는 다음과 같은 구조, 열거형, 함수 포인터를 정의합니다.

명령

태스크는 MQTT API를 직접 호출하는 대신 Command_t 구조를 사용하여 적합한 API 작업을 대신 직접 호출하도록 메인 태스크에 지시하는 명령을 생성합니다. 명령의 유형은 다음과 같습니다.

- PROCESSLOOP
- PUBLISH
- SUBSCRIBE
- UNSUBSCRIBE
- PING
- DISCONNECT
- RECONNECT
- TERMINATE

TERMINATE 명령에는 해당하는 MQTT API 작업이 없습니다. 이 명령은 데모에서 명령 처리를 중지하고 정리 작업을 시작하도록 메인 태스크에 지시하는 데 사용됩니다. MQTT_Publish, MQTT_Subscribe, MQTT_Unsubscribe와 같은 일부 MQTT 명령에는 정보 게시 또는 구독과 같은 일부 추가 정보가 필요하기 때문에 CommandContext_t 필드를 사용합니다. 이 필드는 이 세 가지 명령에는 필수이며 나머지 명령에는 선택 사항입니다.

이러한 명령에는 이 컨텍스트가 필요하므로 명령이 대기열에 배치된 후 명령이 완료될 때까지 이 값을 변경하지 마세요. 명령이 완료되면 선택적 콜백을 간접 호출할 수 있습니다. 이 데모에서는 콜백을 사용하여 호출 태스크에 명령이 완료되었음을 알리는 태스크 알림을 생성합니다. 확인이 필요한 MQTT 작업의 경우(QoS가 0보다 큰 구독, 구독 취소 및 게시) 확인을 수신하면 명령이 완료된 것으로 간주됩니다. 그렇지 않으면 해당 MQTT API 직접 호출이 반환되면 명령이 완료됩니다.

mqtt_demo_connection_sharing.c 파일에서 다음 정의를 찾을 수 있습니다.

- Command_t 구조
- CommandContext_t 구조
- CommandType_t 열거형
- CommandCallback_t 함수 포인터

확인

일부 MQTT 작업에는 확인이 필요하기 때문에 예상되는 확인의 패킷 식별자가 포함된 AckInfo_t 배열과 이를 예상하는 원래 명령을 사용하여 완료 콜백을 호출할 수 있습니다.

구독

이 데모는 각 태스크의 구독을 추적할 수 있습니다. 이렇게 하려면 구독을 요청하는 각 태스크가 게시된 메시지(SubscriptionElement_t)를 다시 수신할 메시지 대기열을 제공해야 합니다. 여러 태스크가 별도의 응답 대기열을 사용해야 하므로 동일한 주제 필터를 구독할 수 있습니다.

수신된 게시 메시지

태스크는 메인 태스크와 병렬로 실행되므로 메인 태스크가 각 구독 태스크에서 수신된 게시 메시지를 읽을 때까지 기다려야 하는 것은 어렵고 시간이 많이 걸릴 수 있습니다. 따라서 수신된 각 메시지는 게시된 메시지의 주제(PublishElement_T)를 구독하는 모든 태스크의 응답 대기열에 복사됩니다. MQTT 클라이언트에서 수신한 게시 패킷에는 클라이언트의 네트워크 버퍼에 대한 포인터가 포함되어 있으므로 수신 메시지의 페이로드와 주제 이름은 응답 대기열에 삽입되기 전에 별도의 버퍼에 복사됩니다. 이렇게 하면 MQTT 클라이언트가 네트워크 버퍼를 지운 후에도 구독 태스크는 수신된 정보를 계속 읽을 수 있습니다.

메인 태스크

메인 애플리케이션 태스크 RunCoreMQTTConnectionSharingDemo는 영구 MQTT 세션을 설정하고, 세 개의 서브 태스크를 생성하고, 종료 명령을 수신할 때까지 처리 루프를 실행합니다. 영구 세션이 사용되므로 예기치 않게 네트워크 연결이 끊어져도 데모가 백그라운드에서 브로커에 다시 연결되므로 구독 또는 브로커로부터 수신되는 게시 메시지가 손실되지 않습니다. 실행할 때마다 새로운 영구 세션을 생성하기 위해 데모는 `clean session` 플래그가 설정된 상태로 브로커에 연결한 다음 플래그가 설정되지 않은 상태에서 연결을 끊었다가 다시 연결합니다. 처리 루프가 종료되면 브로커와의 연결이 끊기고 네트워크 재연결이 설정된 지점부터 다시 루프를 시작합니다.

데모를 성공적으로 완료하면 다음 이미지와 비슷한 출력이 생성됩니다.

```
128 11020 [AsyncPublisher] [INFO] [MQTTDemo] [prvAsyncPublishTask:1960] 129 11020 [AsyncPublisher] Publish operation queued. Sleeping for 100 ms.
130 11020 [AsyncPublisher]
131 11036 [MQTTDemo] [DEBUG] [MQTTDemo] [prvCommandLoop:1788] 132 11036 [MQTTDemo] Processed 2 operations.133 11036 [MQTTDemo]
134 11036 [MQTTDemo] [DEBUG] [MQTTDemo] [prvProcessCommand:1433] 135 11036 [MQTTDemo] Publishing message to filter/sync/1.136 11036 [MQTTDemo]
137 11056 [MQTTDemo] [DEBUG] [MQTTDemo] [prvCommandLoop:1788] 138 11056 [MQTTDemo] Processed 3 operations.139 11056 [MQTTDemo]
140 11056 [MQTTDemo] [DEBUG] [MQTTDemo] [prvProcessCommand:1433] 141 11056 [MQTTDemo] Publishing message to filter/async/1.142 11056 [MQTTDemo]
143 11056 [MQTTDemo] [DEBUG] [MQTTDemo] [prvHandleIncomingPublish:1571] 144 11056 [MQTTDemo] Adding publish to response queue for filter/+/+
145 11056 [MQTTDemo]
146 11056 [Subscriber] [INFO] [MQTTDemo] [prvSubscribeTask:2059] 147 11056 [Subscriber] Received publish on topic filter/sync/1
Message payload: Hello World! Sync: 1
148 11056 [Subscriber]
149 11056 [SyncPublisher] [INFO] [MQTTDemo] [prvSyncPublishTask:1888] 150 11057 [SyncPublisher] Publish operation complete. Sleeping for 100 ms.
151 11057 [SyncPublisher]
152 11122 [AsyncPublisher] [INFO] [MQTTDemo] [prvAsyncPublishTask:1954] 153 11122 [AsyncPublisher] Adding publish operation for message Hello World! Async: 4
on topic filter/async/4154 11122 [AsyncPublisher]
155 11122 [AsyncPublisher] [INFO] [MQTTDemo] [prvAsyncPublishTask:1960] 156 11122 [AsyncPublisher] Publish operation queued. Sleeping for 100 ms.
157 11122 [AsyncPublisher]
158 11157 [SyncPublisher] [INFO] [MQTTDemo] [prvSyncPublishTask:1873] 159 11157 [SyncPublisher] Adding publish operation for message Hello World! Sync: 2
on topic filter/sync/2160 11157 [SyncPublisher]
161 11157 [SyncPublisher] [INFO] [MQTTDemo] [prvSyncPublishTask:1878] 162 11157 [SyncPublisher] Waiting for publish 2 to complete.163 11157 [SyncPublisher]
164 11196 [MQTTDemo] [DEBUG] [MQTTDemo] [prvCommandLoop:1788] 165 11196 [MQTTDemo] Processed 4 operations.166 11196 [MQTTDemo]
167 11196 [MQTTDemo] [DEBUG] [MQTTDemo] [prvProcessCommand:1433] 168 11196 [MQTTDemo] Publishing message to filter/async/2.169 11196 [MQTTDemo]
170 11196 [MQTTDemo] [DEBUG] [MQTTDemo] [prvHandleIncomingPublish:1571] 171 11196 [MQTTDemo] Adding publish to response queue for filter/+/+
172 11196 [MQTTDemo]
173 11196 [Subscriber] [INFO] [MQTTDemo] [prvSubscribeTask:2059] 174 11196 [Subscriber] Received publish on topic filter/async/1
Message payload: Hello World! Async: 1
175 11196 [Subscriber]
176 11222 [AsyncPublisher] [INFO] [MQTTDemo] [prvAsyncPublishTask:1954] 177 11222 [AsyncPublisher] Adding publish operation for message Hello World! Async: 5
on topic filter/async/5178 11222 [AsyncPublisher]
179 11222 [AsyncPublisher] [INFO] [MQTTDemo] [prvAsyncPublishTask:1960] 180 11222 [AsyncPublisher] Publish operation queued. Sleeping for 100 ms.
```

명령 루프

명령 루프 prvCommandLoop는 명령이 명령 대기열에 배치될 때까지 기다렸다가 적절한 MQTT API를 호출합니다. DISCONNECT 및 TERMINATE를 제외한 모든 명령에서 MQTT_ProcessLoop도 호출됩니다. 이 데모는 소켓에서 데이터를 사용할 수 있을 때 대기열에 PROCESSLOOP 명령을 추가하도록 소켓 웨이크업 콜백을 설정합니다. 하지만 이 시점에서 대기열에 이미 많은 명령이 있을 수 있습니다. 다른 명령이 처리되는 동안 들어오는 데이터를 무시하지 않도록 하기 위해 각 명령 다음에 MQTT_ProcessLoop가 단일 반복에 대해 호출됩니다.

처리 명령

prvProcessCommand 함수를 참조하세요.

동기 게시자 태스크

동기 게시자 태스크 prvSyncPublishTask는 동기식으로 PUBLISH 작업을 생성하고 각 작업이 완료될 때까지 기다렸다가 다음 작업을 예약합니다. 이 데모에서는 QoS 1을 사용하여 메시지를 게시하므로 이러한 작업은 게시 확인 패킷을 수신할 때까지 완료된 것으로 간주되지 않습니다.

비동기 게시자 태스크

비동기 게시자 태스크 prvAsyncPublishTask는 게시가 완료될 때까지 기다리지 않고 다음 게시를 대기열에 추가합니다. 이는 작업을 재개하기 전에 MQTT 작업이 완료될 때까지 기다릴 필요가 없다는 것을 보여줍니다. 각 게시 명령에는 고유한 컨텍스트 구조가 필요하므로 이 태스크는 동기 게시자 태스크처럼 단일 컨텍스트 구조를 재사용할 수 없습니다. 이전 명령에서 여전히 필요할 수 있기 때문입니다. 따

라서 각 컨텍스트 구조에 메모리를 할당한 다음 게시할 모든 메시지가 대기열에 배치된 후 할당된 메모리가 모두 해제될 때까지 기다립니다.

구독자 태스크

구독자 태스크 prvSubscribeTask는 동기 및 비동기 태스크에서 게시된 메시지의 모든 주제와 일치하는 주제 필터를 구독합니다. 그런 다음 게시된 메시지가 모두 다시 수신되기를 기다렸다가 구독을 취소합니다. 또한 이 태스크는 메인 태스크에 명령 루프를 종료하라는 신호를 보내는 TERMINATE 작업을 생성하는 역할도 합니다.

OTA(Over-the-Air) 업데이트 데모 애플리케이션

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

FreeRTOS에는 무선 업데이트(OTA) 라이브러리의 기능을 시연하는 데모 애플리케이션이 포함되어 있습니다. OTA 데모 애플리케이션은 *freertos*/demos/ota/aws_iot_ota_update_demo.c 파일에 위치합니다.

OTA 데모 애플리케이션은 다음을 수행합니다.

1. FreeRTOS 네트워크 스택과 MQTT 버퍼 풀을 초기화합니다.
2. vRunOTAUpdateDemo()를 사용하여 OTA 라이브러리 연습을 위한 태스크를 생성합니다.
3. _establishMqttConnection()를 사용하여 MQTT 클라이언트를 생성합니다.
4. 를 사용하여 AWS IoT MQTT 브로커에 연결하고 MQTT 연결 해제 콜백을 IotMqtt_Connect() 등록합니다prvNetworkDisconnectCallback.
5. OTA_AgentInit()를 호출하여 OTA 작업을 생성하고, OTA 작업이 완료될 때 사용할 콜백을 등록합니다.
6. xOTAConnectionCtx.pvControlClient = _mqttConnection;을 사용하여 MQTT 연결을 재사용합니다.
7. MQTT 연결이 끊어지면 애플리케이션은 OTA 에이전트를 일시 중단하고 지수 지연 및 지터를 사용하여 재연결을 시도한 다음 OTA 에이전트를 다시 시작합니다.

OTA 업데이트를 사용하려면 먼저 FreeRTOS 무선 업데이트(OTA)에 명시된 모든 사전 조건을 갖추십시오.

OTA 업데이트를 위한 설정을 완료한 경우 OTA 기능을 지원하는 플랫폼에서 FreeRTOS OTA 데모를 다운로드, 빌드, 플래시 및 실행합니다. 디바이스별 데모 지침은 다음 FreeRTOS 적격 디바이스에서 사용할 수 있습니다.

- Texas Instruments CC3220SF-LAUNCHXL
- Microchip Curiosity PIC32MZEF
- Espressif ESP32
- Renesas RX65N에서 FreeRTOS OTA 데모를 다운로드, 빌드, 플래시, 실행합니다.

디바이스에서 OTA 데모 애플리케이션을 빌드, 플래시 및 실행한 후 AWS IoT 콘솔 또는를 사용하여 OTA 업데이트 작업을 AWS CLI 생성할 수 있습니다. OTA 업데이트 작업을 생성했으면, 터미널 에뮬레이터를 연결하여 OTA 업데이트의 진행 상황을 확인합니다. 프로세스 중에 발생한 오류를 기록해 둡니다.

OTA 업데이트 작업이 성공하면 다음과 같은 출력이 표시됩니다. 간결하게 하기 위해 이 예제에서는 일부 행을 목록에서 제거했습니다.

```
313 267848 [OTA] [OTA] Queued: 1   Processed: 1   Dropped: 0
314 268733 [OTA Task] [OTA] Set job doc parameter [ jobId:
 fe18c7ec_8c31_4438_b0b9_ad55acd95610 ]
315 268734 [OTA Task] [OTA] Set job doc parameter [ streamname: 327 ]
316 268734 [OTA Task] [OTA] Set job doc parameter [ filepath: /sys/mcuflashimg.bin ]
317 268734 [OTA Task] [OTA] Set job doc parameter [ filesize: 130388 ]
318 268735 [OTA Task] [OTA] Set job doc parameter [ fileid: 126 ]
319 268735 [OTA Task] [OTA] Set job doc parameter [ attr: 0 ]
320 268735 [OTA Task] [OTA] Set job doc parameter [ certfile: tisigner.crt.der ]
321 268737 [OTA Task] [OTA] Set job doc parameter [ sig-sha1-rsa:
 Q56qxHRq3Lxv6KkorvilVs4AyGJbWsJd ]
322 268737 [OTA Task] [OTA] Job was accepted. Attempting to start transfer.
323 268737 [OTA Task] Sending command to MQTT task.
324 268737 [MQTT] Received message 50000 from queue.
325 268848 [OTA] [OTA] Queued: 2   Processed: 1   Dropped: 0
326 269039 [MQTT] MQTT Subscribe was accepted. Subscribed.
327 269039 [MQTT] Notifying task.
328 269040 [OTA Task] Command sent to MQTT task passed.
329 269041 [OTA Task] [OTA] Subscribed to topic: $aws/things/TI-LaunchPad/streams/327
```

```
330 269848 [OTA] [OTA] Queued: 2   Processed: 1   Dropped: 0
... // Output removed for brevity
346 284909 [OTA Task] [OTA] file token: 74594452
.. // Output removed for brevity
363 301327 [OTA Task] [OTA] file ready for access.
364 301327 [OTA Task] [OTA] Returned buffer to MQTT Client.
365 301328 [OTA Task] Sending command to MQTT task.
366 301328 [MQTT] Received message 60000 from queue.
367 301328 [MQTT] Notifying task.
368 301329 [OTA Task] Command sent to MQTT task passed.
369 301329 [OTA Task] [OTA] Published file request to $aws/bin/things/TI-LaunchPad/
streams/327/get
370 301632 [OTA Task] [OTA] Received file block 0, size 1024
371 301647 [OTA Task] [OTA] Remaining: 127
... // Output removed for brevity
508 304622 [OTA Task] Sending command to MQTT task.
509 304622 [MQTT] Received message 70000 from queue.
510 304622 [MQTT] Notifying task.
511 304623 [OTA Task] Command sent to MQTT task passed.
512 304623 [OTA Task] [OTA] Published file request to $aws/bin/things/TI-LaunchPad/
streams/327/get
513 304860 [OTA] [OTA] Queued: 47   Processed: 47   Dropped: 83
514 304926 [OTA Task] [OTA] Received file block 4, size 1024
515 304941 [OTA Task] [OTA] Remaining: 82
... // Output removed for brevity
797 315047 [MQTT] MQTT Publish was successful.
798 315048 [MQTT] Notifying task.
799 315048 [OTA Task] Command sent to MQTT task passed.
800 315049 [OTA Task] [OTA] Published 'IN_PROGRESS' status to $aws/things/TI-LaunchPad/
jobs/fe18c7ec_8c31_4438_b0b9_ad55acd9561801 315049 [OTA Task] Sending command to MQTT
 task.
802 315049 [MQTT] Received message d0000 from queue.
803 315150 [MQTT] MQTT Unsubscribe was successful.
804 315150 [MQTT] Notifying task.
805 315151 [OTA Task] Command sent to MQTT task passed.
806 315152 [OTA Task] [OTA] Un-subscribed from topic: $aws/things/TI-LaunchPad/
streams/327

807 315172 [OTA Task] Sending command to MQTT task.
808 315172 [MQTT] Received message e0000 from queue.
809 315273 [MQTT] MQTT Unsubscribe was successful.
810 315273 [MQTT] Notifying task.
811 315274 [OTA Task] Command sent to MQTT task passed.
```

```
812 315274 [OTA Task] [OTA] Un-subscribed from topic: $aws/things/TI-LaunchPad/
streams/327

813 315275 [OTA Task] [OTA] Resetting MCU to activate new image.
0 0 [Tmr Svc] Starting Wi-Fi Module ...
1 0 [Tmr Svc] Simple Link task created

Device came up in Station mode

2 137 [Tmr Svc] Wi-Fi module initialized.
3 137 [Tmr Svc] Starting key provisioning...
4 137 [Tmr Svc] Write root certificate...
5 243 [Tmr Svc] Write device private key...
6 339 [Tmr Svc] Write device certificate...
7 436 [Tmr Svc] Key provisioning done...
Device disconnected from the AP on an ERROR..!!

[WLAN EVENT] STA Connected to the AP: Guest , BSSID: 44:48:c1:ba:b2:c3

[NETAPP EVENT] IP acquired by the device


Device has connected to Guest

Device IP Address is 192.168.3.72


8 1443 [Tmr Svc] Wi-Fi connected to AP Guest.
9 1444 [Tmr Svc] IP Address acquired 192.168.3.72
10 1444 [OTA] OTA demo version 0.9.1
11 1445 [OTA] Creating MQTT Client...
12 1445 [OTA] Connecting to broker...
13 1445 [OTA] Sending command to MQTT task.
14 1445 [MQTT] Received message 10000 from queue.
15 2910 [MQTT] MQTT Connect was accepted. Connection established.
16 2910 [MQTT] Notifying task.
17 2911 [OTA] Command sent to MQTT task passed.
18 2912 [OTA] Connected to broker.
19 2913 [OTA Task] Sending command to MQTT task.
20 2913 [MQTT] Received message 20000 from queue.
21 3014 [MQTT] MQTT Subscribe was accepted. Subscribed.
22 3014 [MQTT] Notifying task.
23 3015 [OTA Task] Command sent to MQTT task passed.
```

```
24 3015 [OTA Task] [OTA] Subscribed to topic: $aws/things/TI-LaunchPad/jobs/$next/get/
accepted

25 3028 [OTA Task] Sending command to MQTT task.
26 3028 [MQTT] Received message 30000 from queue.
27 3129 [MQTT] MQTT Subscribe was accepted. Subscribed.
28 3129 [MQTT] Notifying task.
29 3130 [OTA Task] Command sent to MQTT task passed.
30 3138 [OTA Task] [OTA] Subscribed to topic: $aws/things/TI-LaunchPad/jobs/notify-next

31 3138 [OTA Task] [OTA] Check For Update #0
32 3138 [OTA Task] Sending command to MQTT task.
33 3138 [MQTT] Received message 40000 from queue.
34 3241 [MQTT] MQTT Publish was successful.
35 3241 [MQTT] Notifying task.
36 3243 [OTA Task] Command sent to MQTT task passed.
37 3245 [OTA Task] [OTA] Set job doc parameter [ clientToken: 0:TI-LaunchPad ]
38 3245 [OTA Task] [OTA] Set job doc parameter [ jobId:
 fe18c7ec_8c31_4438_b0b9_ad55acd95610 ]
39 3245 [OTA Task] [OTA] Identified job doc parameter [ self_test ]
40 3246 [OTA Task] [OTA] Set job doc parameter [ updatedBy: 589827 ]
41 3246 [OTA Task] [OTA] Set job doc parameter [ streamname: 327 ]
42 3246 [OTA Task] [OTA] Set job doc parameter [ filepath: /sys/mcuflashimg.bin ]
43 3247 [OTA Task] [OTA] Set job doc parameter [ filesize: 130388 ]
44 3247 [OTA Task] [OTA] Set job doc parameter [ fileid: 126 ]
45 3247 [OTA Task] [OTA] Set job doc parameter [ attr: 0 ]
46 3247 [OTA Task] [OTA] Set job doc parameter [ certfile: tisigner.crt.der ]
47 3249 [OTA Task] [OTA] Set job doc parameter [ sig-sha1-rsa:
 Q56qxHRq3Lxv6KkorvilVs4AyGJbWsJd ]
48 3249 [OTA Task] [OTA] Job is ready for self test.
49 3250 [OTA Task] Sending command to MQTT task.
51 3351 [MQTT] MQTT Publish was successful.
52 3352 [MQTT] Notifying task.
53 3352 [OTA Task] Command sent to MQTT task passed.
54 3353 [OTA Task] [OTA] Published 'IN_PROGRESS' status to $aws/things/TI-LaunchPad/
jobs/fe18c7ec_8c31_4438_b0b9_ad55acd95610/u55 3353 [OTA Task] Sending command to MQTT
 task.
56 3353 [MQTT] Received message 60000 from queue.
57 3455 [MQTT] MQTT Unsubscribe was successful.
58 3455 [MQTT] Notifying task.
59 3456 [OTA Task] Command sent to MQTT task passed.
60 3456 [OTA Task] [OTA] Un-subscribed from topic: $aws/things/TI-LaunchPad/streams/327

61 3456 [OTA Task] [OTA] Accepted final image. Commit.
```

```
62 3578 [OTA Task] Sending command to MQTT task.
63 3578 [MQTT] Received message 70000 from queue.
64 3779 [MQTT] MQTT Publish was successful.
65 3780 [MQTT] Notifying task.
66 3780 [OTA Task] Command sent to MQTT task passed.
67 3781 [OTA Task] [OTA] Published 'SUCCEEDED' status to $aws/things/TI-LaunchPad/jobs/
fe18c7ec_8c31_4438_b0b9_ad55acd95610/upd68 3781 [OTA Task] [OTA] Returned buffer to
 MQTT Client.
69 4251 [OTA] [OTA] Queued: 1   Processed: 1   Dropped: 0
70 4381 [OTA Task] [OTA] Missing job parameter: execution
71 4382 [OTA Task] [OTA] Missing job parameter: jobId
72 4382 [OTA Task] [OTA] Missing job parameter: jobDocument
73 4382 [OTA Task] [OTA] Missing job parameter: ts_ota
74 4382 [OTA Task] [OTA] Missing job parameter: files
75 4382 [OTA Task] [OTA] Missing job parameter: streamname
76 4382 [OTA Task] [OTA] Missing job parameter: certfile
77 4382 [OTA Task] [OTA] Missing job parameter: filepath
78 4383 [OTA Task] [OTA] Missing job parameter: filesize
79 4383 [OTA Task] [OTA] Missing job parameter: sig-sha1-rsa
80 4383 [OTA Task] [OTA] Missing job parameter: fileid
81 4383 [OTA Task] [OTA] Missing job parameter: attr
82 4383 [OTA Task] [OTA] Returned buffer to MQTT Client.
83 5251 [OTA] [OTA] Queued: 2   Processed: 2   Dropped: 0
```

무선(OTA) 데모 구성

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

OTA 데모 구성은 `aws_iot_ota_update_demo.c`에서 제공하는 데모별 구성 옵션입니다. 이러한 구성은 OTA 라이브러리 구성 파일에 제공된 OTA 라이브러리 구성과 다릅니다.

OTA_DEMO_KEEP_ALIVE_SECONDS

MQTT 클라이언트의 경우 이 구성은 한 제어 패킷의 전송을 완료한 후 다음 제어 패킷의 전송을 시작하는 데 소요될 수 있는 최대 시간 간격입니다. 제어 패킷이 없는 경우 PINGREQ가 전송됩니다. 브로커는 이 유지 기간의 1.5배 이내에 메시지 또는 PINGREQ 패킷을 전송하지 않는 클라이언트와의 연결을 끊어야 합니다. 이 구성은 애플리케이션의 요구 사항에 따라 조정해야 합니다.

OTA_DEMO_CONN_RETRY_BASE_INTERVAL_SECONDS

네트워크 연결을 재시도하기 전의 기본 간격(초)입니다. OTA 데모는 이 기본 시간 간격 이후에 재연결을 시도합니다. 시도가 실패할 때마다 간격이 두 배로 늘어납니다. 이 기본 지연의 최대값까지의 임의 지연도 간격에 추가됩니다.

OTA_DEMO_CONN_RETRY_MAX_INTERVAL_SECONDS

네트워크 연결을 재시도하기 전의 최대 간격(초)입니다. 재연결 지연은 시도가 실패할 때마다 두 배로 늘어나지만 이 최대값에 같은 간격의 지터를 더한 값까지만 지연될 수 있습니다.

Texas Instruments CC3220SF-LAUNCHXL에 대한 FreeRTOS OTA 데모 다운로드, 빌드, 플래시 및 실행

Important

이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

FreeRTOS 및 OTA 데모 코드를 다운로드하려면

- GitHub 사이트 https://github.com/FreeRTOS/FreeRTOS에서 소스 코드를 다운로드할 수 있습니다.

데모 애플리케이션을 빌드하려면

1. 의 지침에 따라 aws_demos 프로젝트를 Code Composer Studio로 가져오FreeRTOS 시작하기고, AWS IoT 엔드포인트, Wi-Fi SSID 및 암호, 보드의 프라이빗 키 및 인증서를 구성합니다.
2. *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h를 열고 #define CONFIG_MQTT_DEMO_ENABLED를 주석으로 처리한 다음 CONFIG_OTA_UPDATE_DEMO_ENABLED를 정의합니다.
3. 솔루션을 빌드하고 오류 없이 빌드되는지 확인합니다.
4. 터미널 에뮬레이터를 시작하고 다음 설정을 사용하여 보드에 연결합니다.
   - 전송 속도: 115200
   - 데이터 비트: 8

- 패리티: 없음
- 정지 비트: 1

5. 보드에서 프로젝트를 실행하여 Wi-Fi 및 AWS IoT MQTT 메시지 브로커에 연결할 수 있는지 확인합니다.

실행되면 터미널 에뮬레이터에 다음과 같은 텍스트가 표시됩니다.

```
0 0 [Tmr Svc] Starting Wi-Fi Module ...
1 0 [Tmr Svc] Simple Link task created
Device came up in Station mode
2 142 [Tmr Svc] Wi-Fi module initialized.
3 142 [Tmr Svc] Starting key provisioning...
4 142 [Tmr Svc] Write root certificate...
5 243 [Tmr Svc] Write device private key...
6 340 [Tmr Svc] Write device certificate...
7 433 [Tmr Svc] Key provisioning done...
[WLAN EVENT] STA Connected to the AP: Mobile , BSSID: 24:de:c6:5d:32:a4
[NETAPP EVENT] IP acquired by the device

Device has connected to Mobile
Device IP Address is 192.168.111.12

8 2666 [Tmr Svc] Wi-Fi connected to AP Mobile.
9 2666 [Tmr Svc] IP Address acquired 192.168.111.12
10 2667 [OTA] OTA demo version 0.9.2
11 2667 [OTA] Creating MQTT Client...
12 2667 [OTA] Connecting to broker...
13 3512 [OTA] Connected to broker.
14 3715 [OTA Task] [prvSubscribeToJobNotificationTopics] OK: $aws/things/OtaGA/jobs/
$next/get/accepted
15 4018 [OTA Task] [prvSubscribeToJobNotificationTopics] OK: $aws/things/OtaGA/jobs/
notify-next
16 4027 [OTA Task] [prvPAL_GetPlatformImageState] xFileInfo.Flags = 0250
17 4027 [OTA Task] [prvPAL_GetPlatformImageState] eOTA_PAL_ImageState_Valid
18 4034 [OTA Task] [OTA_CheckForUpdate] Request #0
19 4248 [OTA] [OTA_AgentInit] Ready.
20 4249 [OTA Task] [prvParseJSONbyModel] Extracted parameter [ clientToken: 0:OtaGA ]
21 4249 [OTA Task] [prvParseJSONbyModel] parameter not present: execution
22 4249 [OTA Task] [prvParseJSONbyModel] parameter not present: jobId
23 4249 [OTA Task] [prvParseJSONbyModel] parameter not present: jobDocument
24 4249 [OTA Task] [prvParseJSONbyModel] parameter not present: afr_ota
```

```
25 4250 [OTA Task] [prvParseJSONbyModel] parameter not present: streamname
26 4250 [OTA Task] [prvParseJSONbyModel] parameter not present: files
27 4250 [OTA Task] [prvParseJSONbyModel] parameter not present: filepath
28 4250 [OTA Task] [prvParseJSONbyModel] parameter not present: filesize
29 4250 [OTA Task] [prvParseJSONbyModel] parameter not present: fileid
30 4250 [OTA Task] [prvParseJSONbyModel] parameter not present: certfile
31 4251 [OTA Task] [prvParseJSONbyModel] parameter not present: sig-sha1-rsa
32 4251 [OTA Task] [prvParseJobDoc] Ignoring job without ID.
33 4251 [OTA Task] [prvOTA_Close] Context->0x2001b2c4
34 5248 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
35 6248 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
36 7248 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
37 8248 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
38 9248 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
```

Microchip Curiosity PIC32MZEF에 대한 FreeRTOS OTA 데모 다운로드, 빌드, 플래시 및 실행

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

FreeRTOS OTA 데모 코드를 다운로드하려면

- GitHub 사이트 https://github.com/FreeRTOS/FreeRTOS에서 소스 코드를 다운로드할 수 있습니다.

OTA 업데이트 데모 애플리케이션을 빌드하려면

1. 의 지침에 따라 `aws_demos` 프로젝트를 MPLAB X IDE로 FreeRTOS 시작하기 가져오고, AWS IoT 엔드포인트, Wi-Fi SSID 및 암호, 보드의 프라이빗 키 및 인증서를 구성합니다.
2. `demos/include/aws_ota_codesigner_certificate.h`를 엽니다.
3. 코드 서명 인증서의 내용을 `static const char signingcredentialSIGNING_CERTIFICATE_PEM` 변수에 붙여 넣습니다. `aws_clientcredential_keys.h`와 동일한 형식에 따라 각 줄은 줄 바꿈 문자('\n')로 끝나고 인용 부호로 묶여 있어야 합니다.

예를 들어, 인증서는 다음과 비슷합니다.

```
"-----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. Python 3 이상을 설치합니다.
5. `pip install pyopenssl`을 실행하여 pyOpenSSL을 설치합니다.
6. 코드 서명 인증서를 demos/ota/bootloader/utility/codesigner_cert_utility/ 경로에 .pem 형식으로 복사합니다. 인증서 파일의 이름을 `aws_ota_codesigner_certificate.pem`으로 바꿉니다.
7. *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/ `aws_demo_config.h`를 열고 `#define CONFIG_MQTT_DEMO_ENABLED`를 주석으로 처리한 다음 `CONFIG_OTA_UPDATE_DEMO_ENABLED`를 정의합니다.
8. 솔루션을 빌드하고 오류 없이 빌드되는지 확인합니다.
9. 터미널 에뮬레이터를 시작하고 다음 설정을 사용하여 보드에 연결합니다.
   - 전송 속도: 115200
   - 데이터 비트: 8
   - 패리티: 없음
   - 정지 비트: 1
10. 보드에서 디버거를 분리하고 보드에서 프로젝트를 실행하여 Wi-Fi 및 AWS IoT MQTT 메시지 브로커에 연결할 수 있는지 확인합니다.

프로젝트를 실행할 때 MPLAB X IDE에서 출력 창을 열어야 합니다. ICD4 탭이 선택되는지 확인합니다. 다음과 같이 출력되어야 합니다.

```
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
```

터미널 에뮬레이터에 다음과 같은 텍스트가 표시됩니다.

```
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
```

이 출력은 Microchip Curiosity PIC32MZEF가 OTA 업데이트에 필요한 MQTT 주제에 연결하고 AWS IoT 구독할 수 있음을 보여줍니다. 대기 중인 OTA 업데이트 작업이 없으므로 `Missing job parameter` 메시지가 필요합니다.

Espressif ESP32에 대한 FreeRTOS OTA 데모 다운로드, 빌드, 플래시 및 실행

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

1. GitHub에서 FreeRTOS 소스를 다운로드합니다. 자세한 내용은 README.md 파일을 참조하십시오. 모든 필요한 소스와 라이브러리가 포함된 IDE에서 프로젝트를 생성합니다.
2. Espressif 시작하기의 지침에 따라 필요한 GCC 기반 도구 체인을 설정합니다.
3. *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h를 열고 #define CONFIG_MQTT_DEMO_ENABLED를 주석으로 처리한 다음 CONFIG_OTA_UPDATE_DEMO_ENABLED를 정의합니다.
4. vendors/espressif/boards/esp32/aws_demos 디렉터리에서 make를 실행하여 데모 프로젝트를 빌드합니다. Espressif 시작하기에 설명된 대로 make flash monitor를 실행하여 데모 프로그램을 플래시하고 출력을 확인할 수 있습니다.
5. OTA 업데이트 데모를 실행하기 전에:
   - *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h를 열고 #define CONFIG_MQTT_DEMO_ENABLED를 주석으로 처리한 다음 CONFIG_OTA_UPDATE_DEMO_ENABLED를 정의합니다.
   - SHA-256/ECDSA 코드 서명 인증서가 demos/include/aws_ota_codesigner_certificate.h에 복사되는지 확인합니다.

Renesas RX65N에서 FreeRTOS OTA 데모를 다운로드, 빌드, 플래시, 실행합니다.

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

이 장에서는 Renesas RX65N에서 FreeRTOS OTA 데모 애플리케이션을 다운로드, 빌드, 플래시, 실행하는 방법을 보여줍니다.

주제

- 운영 환경 설정
- AWS 리소스 설정
- 헤더 파일을 가져오고 구성하고 aws_demos 및 boot_loader를 빌드합니다.

운영 환경 설정

이 섹션의 절차에서는 다음 환경을 사용합니다.

- IDE: e^2 studio 7.8.0, e^2 studio 2020-07
- 도구 체인: CCRX Compiler v3.0.1
- 대상 디바이스: RSKRX65N-2MB
- 디버거: E^2, E^2 Lite 에뮬레이터
- 소프트웨어: Renesas Flash Programmer, Renesas Secure Flash Programmer.exe, Tera Term

하드웨어를 설정하려면

1. E^2 Lite 에뮬레이터 및 USB 직렬 포트를 RX65N 보드 및 PC에 연결합니다.
2. 전원을 RX65N에 연결합니다.

AWS 리소스 설정

1. FreeRTOS 데모를 실행하려면 AWS IoT 서비스에 액세스할 수 있는 권한이 있는 AWS Identity and Access Management (IAM) 사용자가 있는 AWS 계정이 있어야 합니다. 아직 AWS 계정 및 권한 설정의 단계를 완료하지 않았다면 지금 수행합니다.
2. OTA 업데이트를 설정하려면 OTA 업데이트 사전 조건의 단계를 따릅니다. 특히 MQTT를 사용한 OTA 업데이트 사전 조건의 단계를 따르십시오.
3. AWS IoT 콘솔을 엽니다.
4. 왼쪽 탐색 창에서 관리를 선택한 후 사물을 선택하여 사물을 하나 생성합니다.

   AWS IoT에서 사물이란 특정 디바이스 또는 논리적 엔터티의 표현입니다. 사물은 물리적 장치 또는 센서일 수 있습니다(예: 전구 또는 벽면 스위치). 또한에 연결되지 AWS IoT는 않지만에 연결된

디바이스(예: 엔진 센서 또는 제어판이 있는 자동차)와 관련된 애플리케이션 인스턴스 또는 물리적 개체와 같은 논리적 개체일 수도 있습니다.는 사물을 관리하는 데 도움이 되는 사물 레지스트리를 AWS IoT 제공합니다.

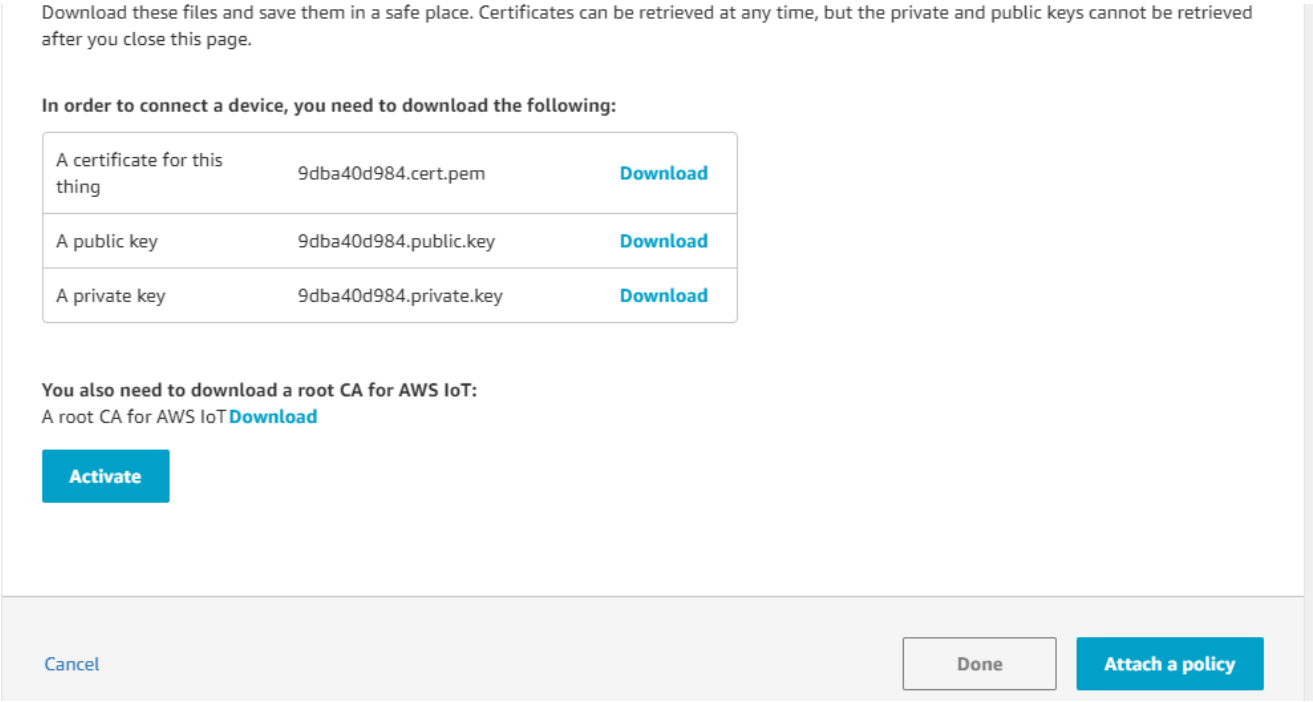
a. 생성을 선택하고 단일 사물 생성을 선택합니다.

b. 사물의 이름을 입력하고 다음을 선택합니다.

c. [Create certificate]를 선택합니다.

d. 생성된 세 개의 파일을 다운로드한 다음 활성화를 선택합니다.

e. 정책 연결을 선택합니다.



f. 디바이스 정책에서 생성한 정책을 선택합니다.

MQTT를 사용하여 OTA 업데이트를 수신하는 각 디바이스는에 사물로 등록해야 하며 AWS IoT 사물에는 나열된 것과 같은 연결된 정책이 있어야 합니다. `"Action"` 및 `"Resource"` 객체의 항목에 대한 자세한 내용은 AWS IoT 핵심 정책 작업 및 AWS IoT 핵심 작업 리소스에서 확인할 수 있습니다.

Notes

- `iot:Connect` 권한을 통해 디바이스가 MQTT를 AWS IoT 통해에 연결할 수 있습니다.
- AWS IoT 작업(`.../jobs/*`) 항목에 대한 `iot:Subscribe` 및 `iot:Publish` 권한을 사용하면 연결된 디바이스에서 작업 알림과 작업 문서를 수신하고 작업 실행 완료 상태를 게시할 수 있습니다.

- AWS IoT OTA 스트림(`.../streams/*`) 주제에 대한 `iot:Subscribe` 및 `iot:Publish` 권한을 통해 연결된 디바이스가 OTA 업데이트 데이터를 가져올 수 있습니다 AWS IoT. MQTT를 통해 펌웨어 업데이트를 수행하려면 이러한 권한이 필요합니다.
- `iot:Receive` 권한을 통해 AWS IoT Core 는 연결된 디바이스에 해당 주제에 대한 메시지를 게시할 수 있습니다. MQTT 메시지를 전송할 때마다 이 권한을 확인합니다. 이 권한을 사용하면 주제를 현재 구독 중인 클라이언트에 대한 액세스 권한을 취소할 수 있습니다.

5. 코드 서명 프로파일을 생성하고 코드 서명 인증서를 등록하려면 AWS

   a. 키와 인증을 생성하려면 Renesas MCU Firmware Update Design Policy의 섹션 7.3 'Generating ECDSA-SHA256 Key Pairs with OpenSSL'을 참조하세요.

   b. AWS IoT 콘솔을 엽니다. 왼쪽 탐색 창에서 관리를 선택하고 작업을 선택합니다. 작업 생성을 선택하고 OTA 업데이트 작업 생성을 선택합니다.

   c. 업데이트할 디바이스 선택에서 선택을 선택하고 이전에 생성한 사물을 선택합니다. 다음을 선택합니다.

   d. FreeRTOS OTA 업데이트 작업 생성 페이지에서 다음을 수행합니다.

      i. 펌웨어 이미지 전송을 위한 프로토콜 선택에서 MQTT를 선택합니다.

      ii. 펌웨어 이미지 선택 및 서명에서 나를 위해 새 펌웨어 이미지에 서명을 선택합니다.

      iii. 코드 서명 프로필에서 생성을 선택합니다.

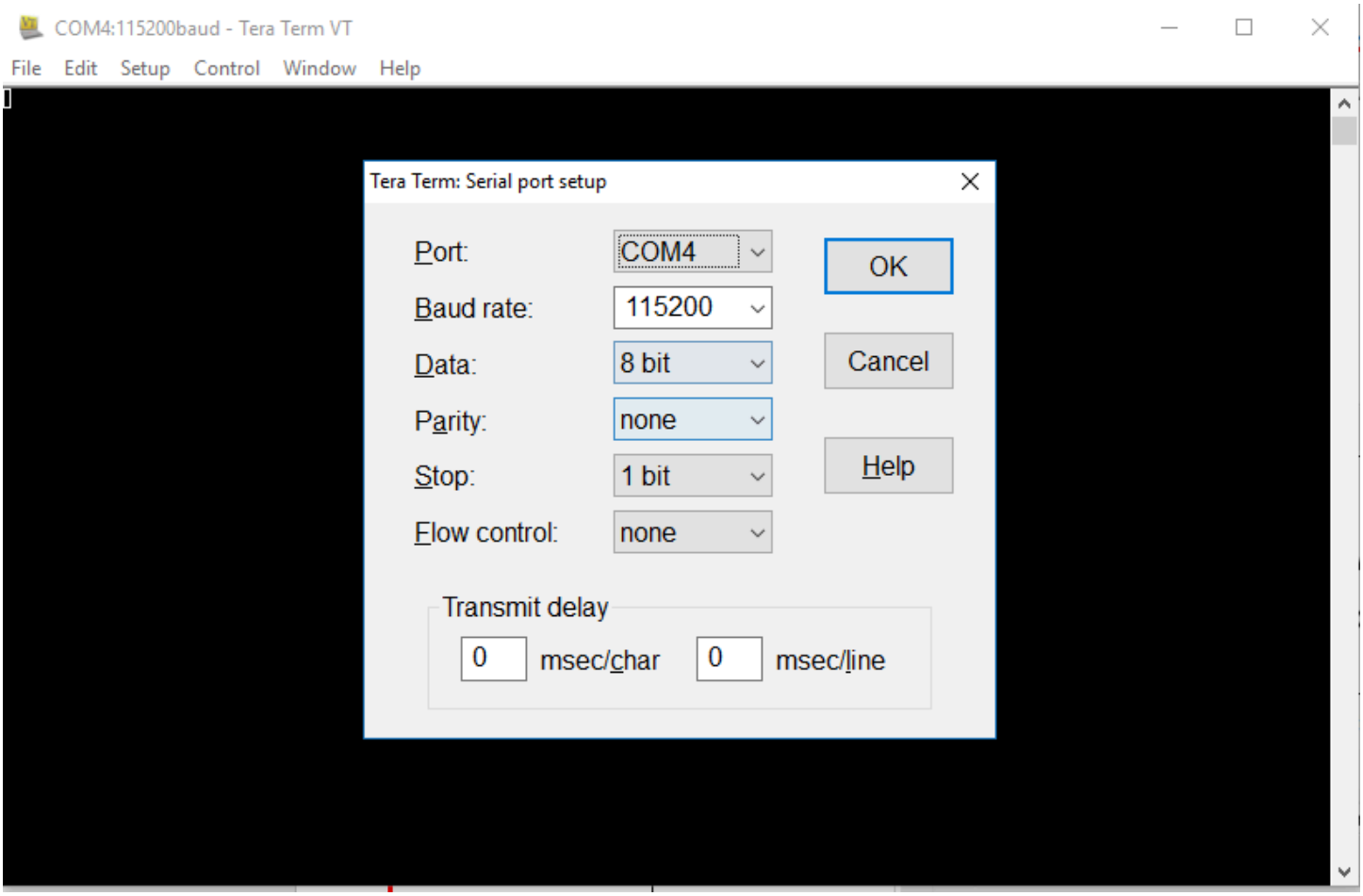
      iv. 코드 서명 프로필 생성 창에서 프로필 이름을 입력합니다. 디바이스 하드웨어 플랫폼에서 Windows 시뮬레이터를 선택합니다. 코드 서명 인증서에서 가져오기를 선택합니다.

      v. 인증서(`secp256r1.crt`), 인증서 프라이빗 키(`secp256r1.key`) 및 인증서 체인(`ca.crt`)을 찾아 선택합니다.

      vi. 디바이스에 있는 코드 서명 인증서의 경로 이름을 입력합니다. 그런 다음 생성을 선택합니다.

6. 에 대한 코드 서명에 대한 액세스 권한을 부여하려면의 단계를 AWS IoT따릅니다에 대한 코드 서명에 대한 액세스 권한 부여 AWS IoT.

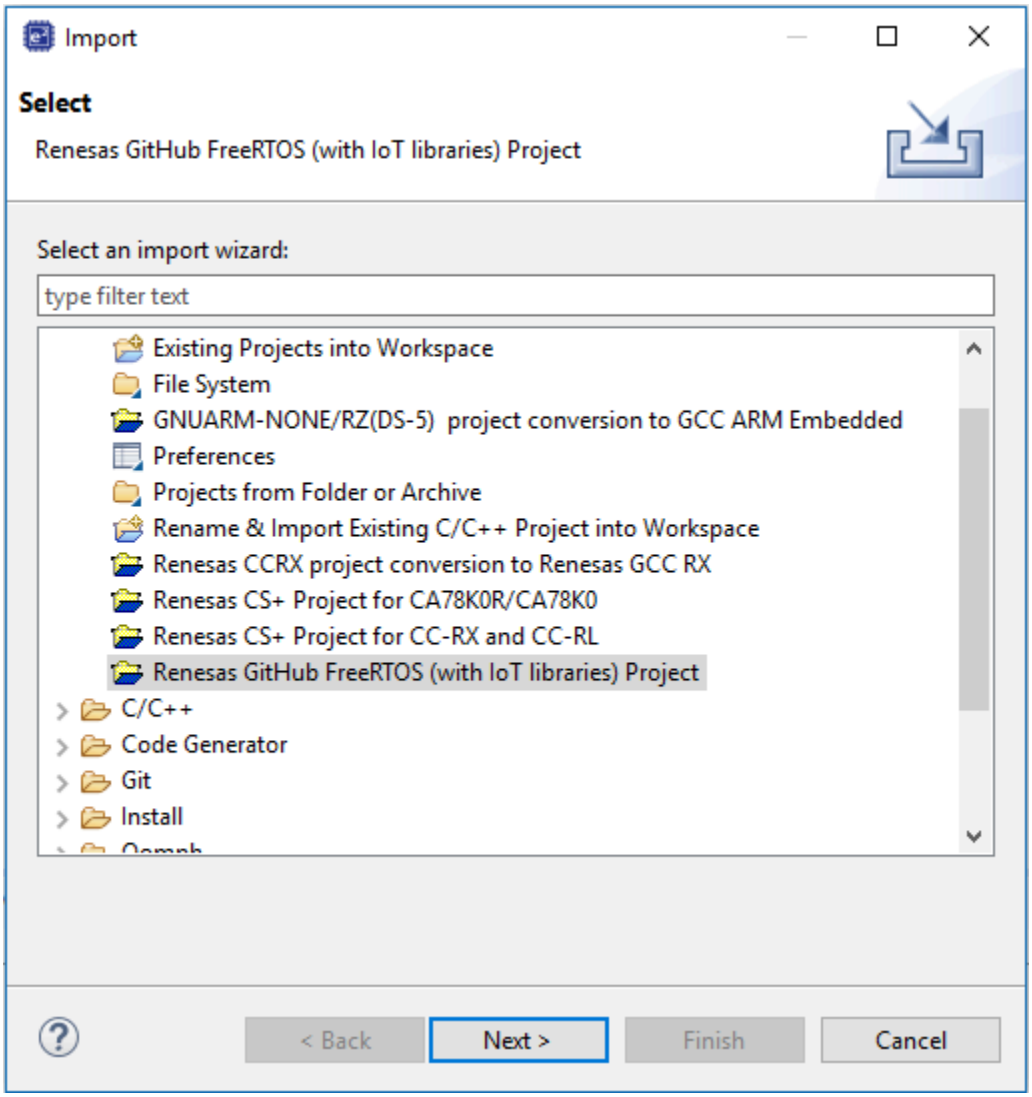
PC에 Tera Term이 설치되어 있지 않은 경우 https://ttssh2.osdn.jp/index.html.en에서 다운로드하여 여기에 표시된 대로 설정할 수 있습니다. 디바이스의 USB 직렬 포트를 PC에 연결했는지 확인하세요.



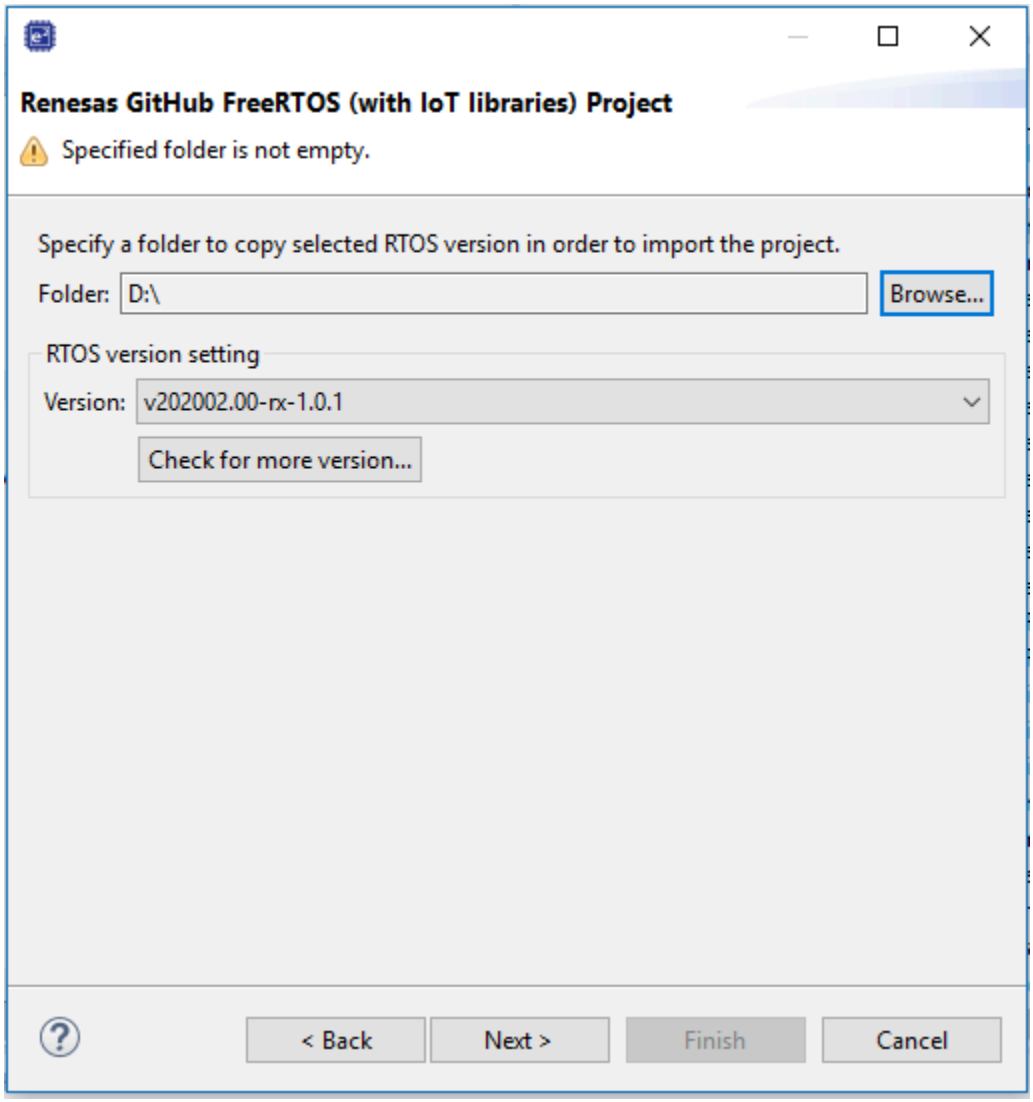
헤더 파일을 가져오고 구성하고 aws_demos 및 boot_loader를 빌드합니다.

시작하려면 최신 버전의 FreeRTOS 패키지를 선택합니다. 그러면 이 패키지가 GitHub에서 다운로드되고 자동으로 프로젝트로 가져와집니다. 그러므로 FreeRTOS 구성과 애플리케이션 코드 작성에 집중할 수 있습니다.

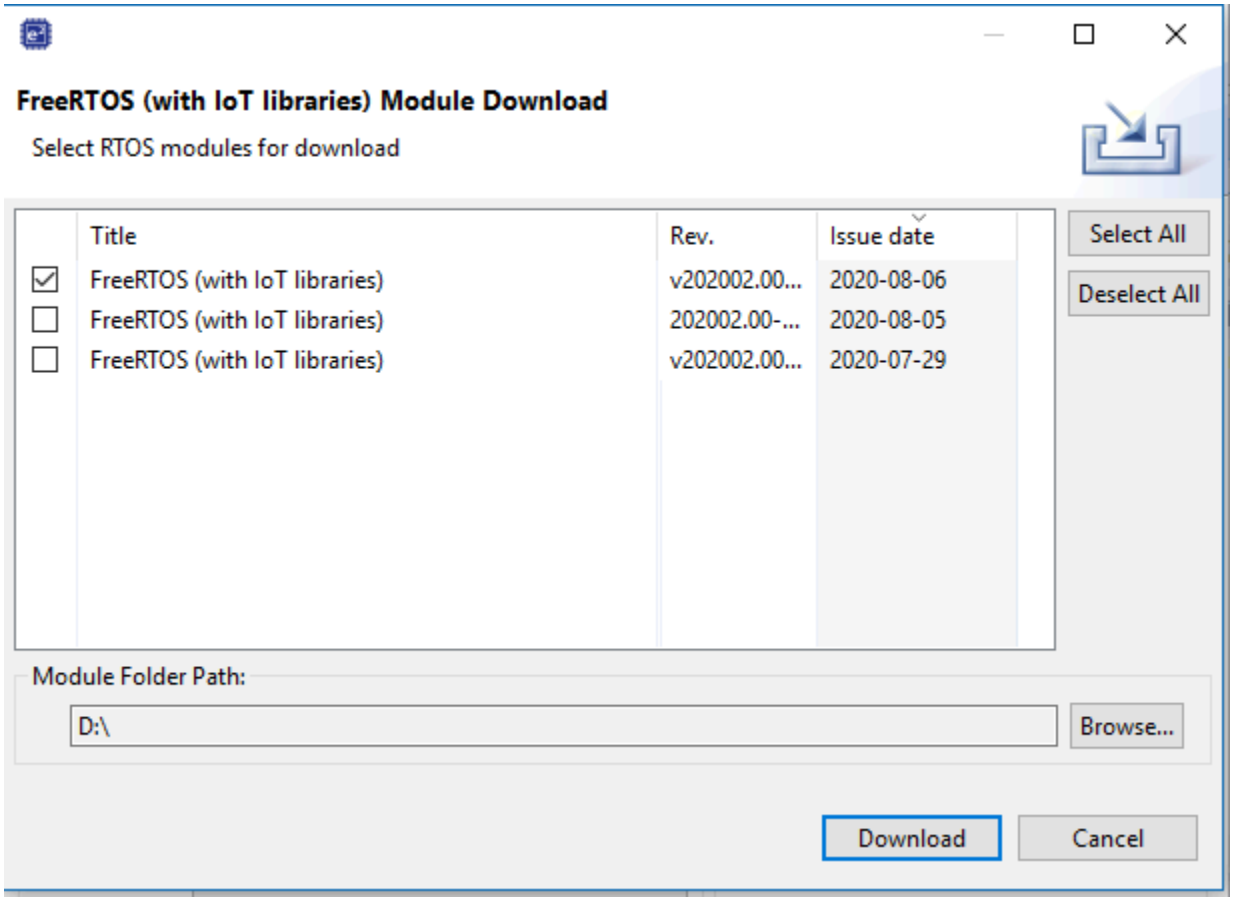
1. e^2 studio 시작.
2. 파일을 선택하고 가져오기...를 선택합니다.
3. Renesas GitHub FreeRTOS(IoT 라이브러리 포함) 프로젝트를 선택합니다.



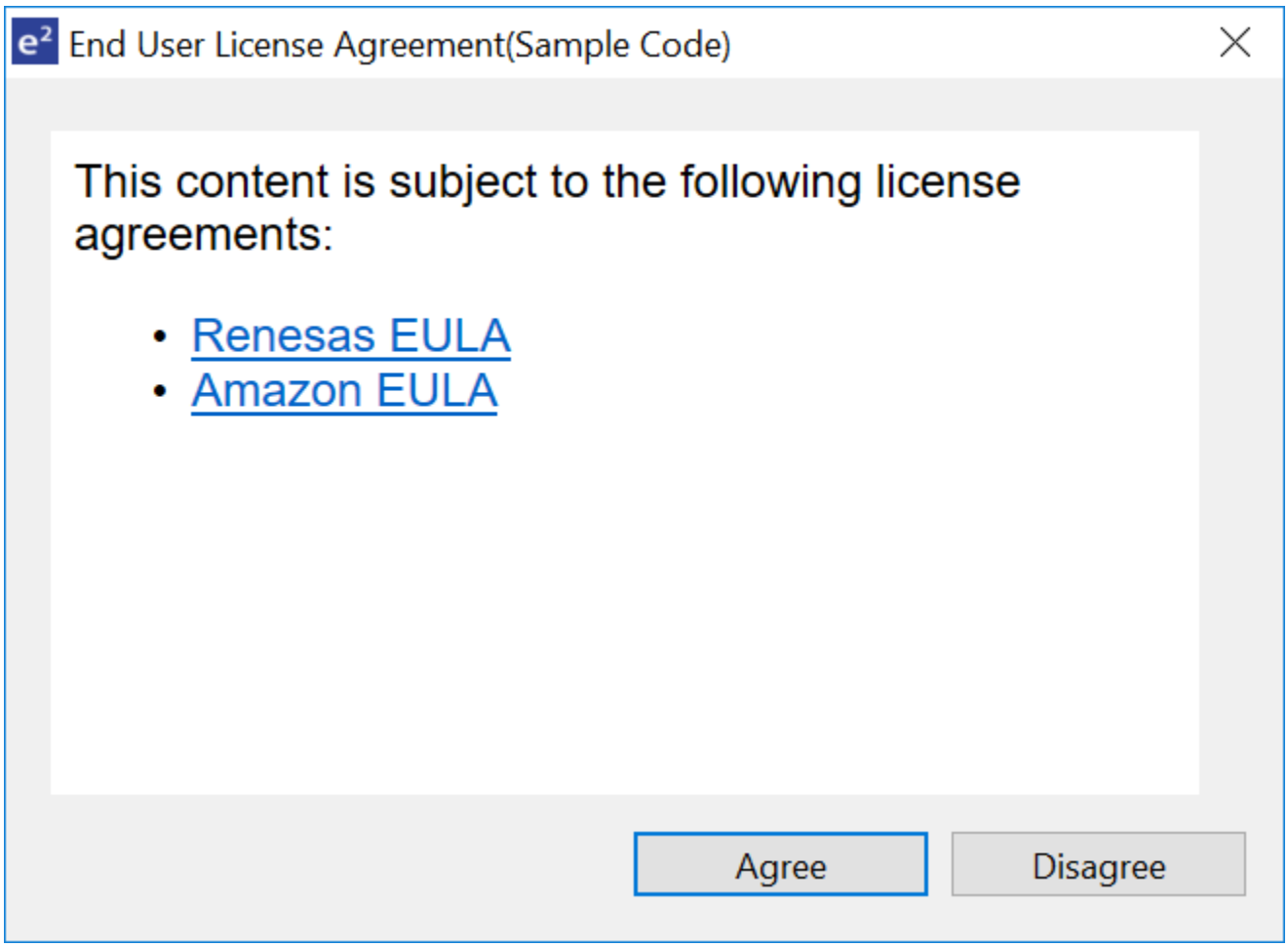
4. 추가 버전 확인...을 선택하여 다운로드 대화 상자를 표시합니다.



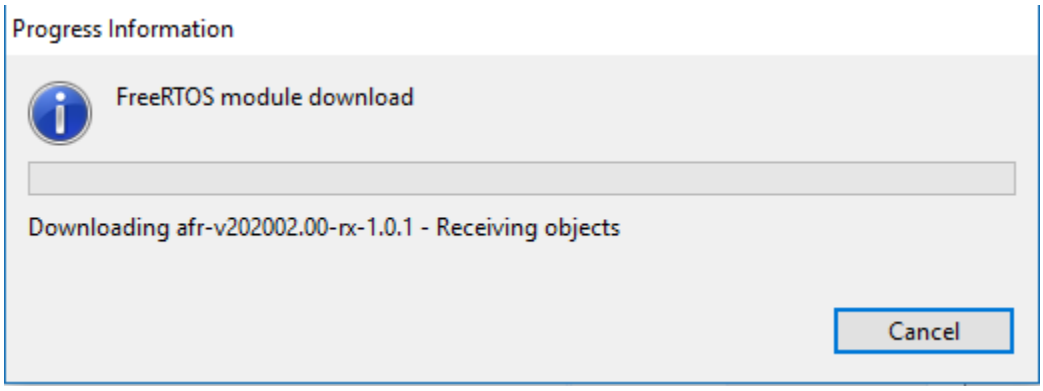
5. 최신 패키지를 선택합니다.



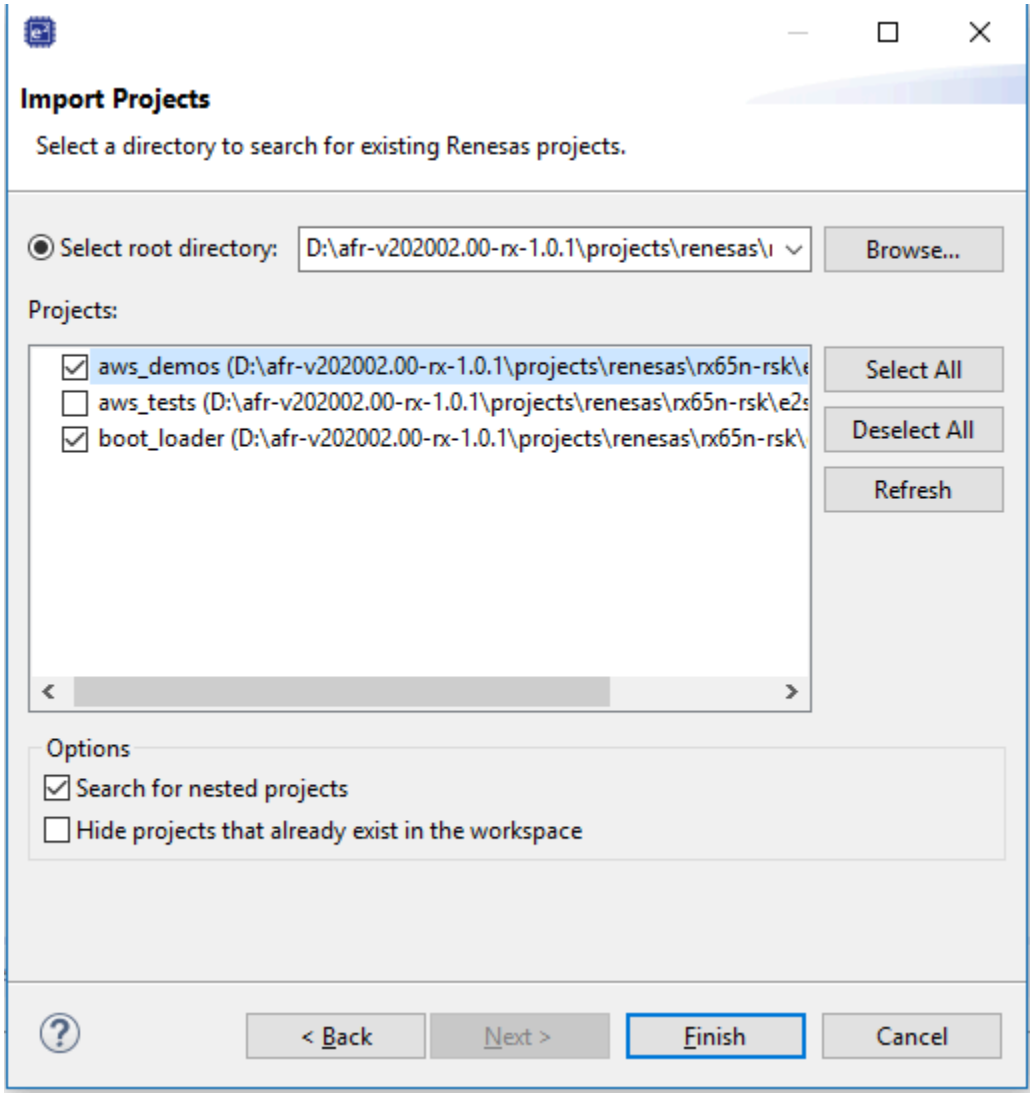
6. 동의를 선택하여 최종 사용자 사용권 계약을 수락합니다.



7. 다운로드가 완료될 때까지 기다립니다.



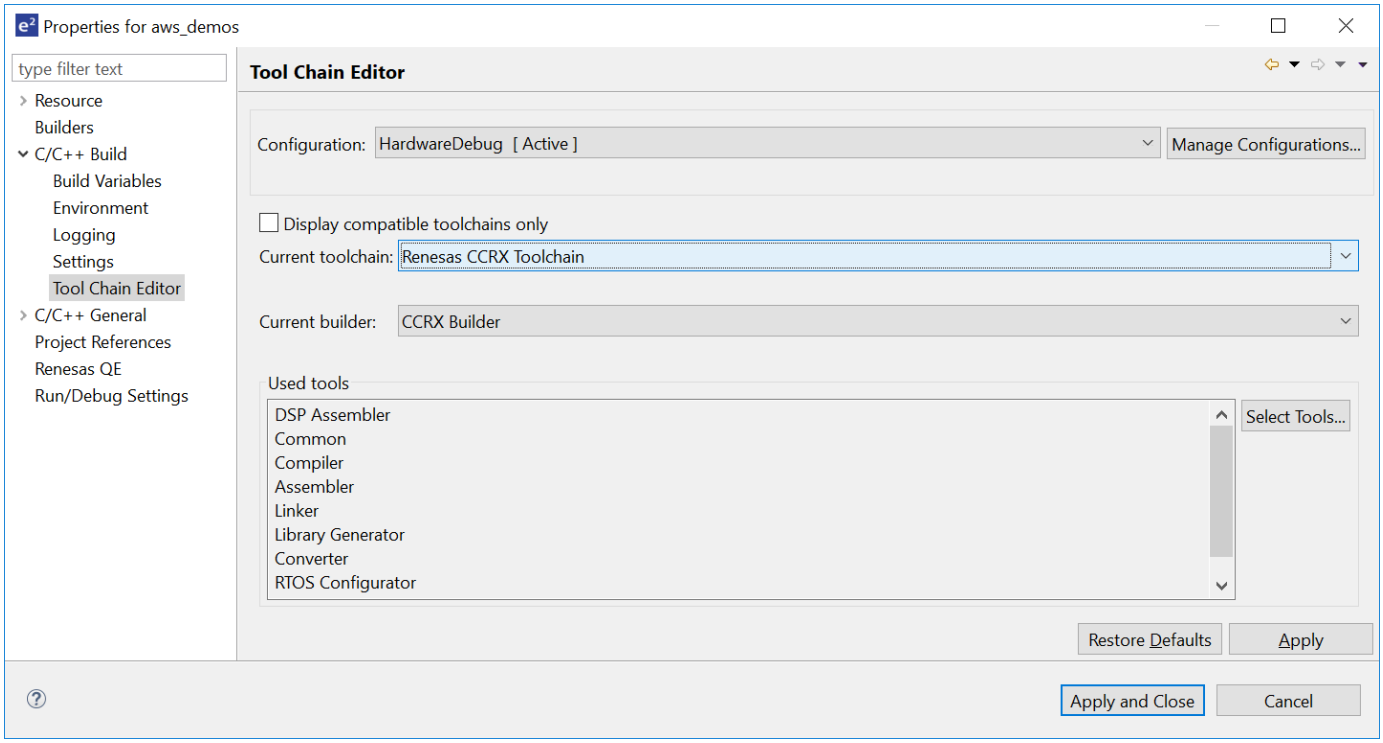
8. aws_demos 및 boot_loader 프로젝트를 선택한 다음 마침을 선택하여 프로젝트를 가져옵니다.



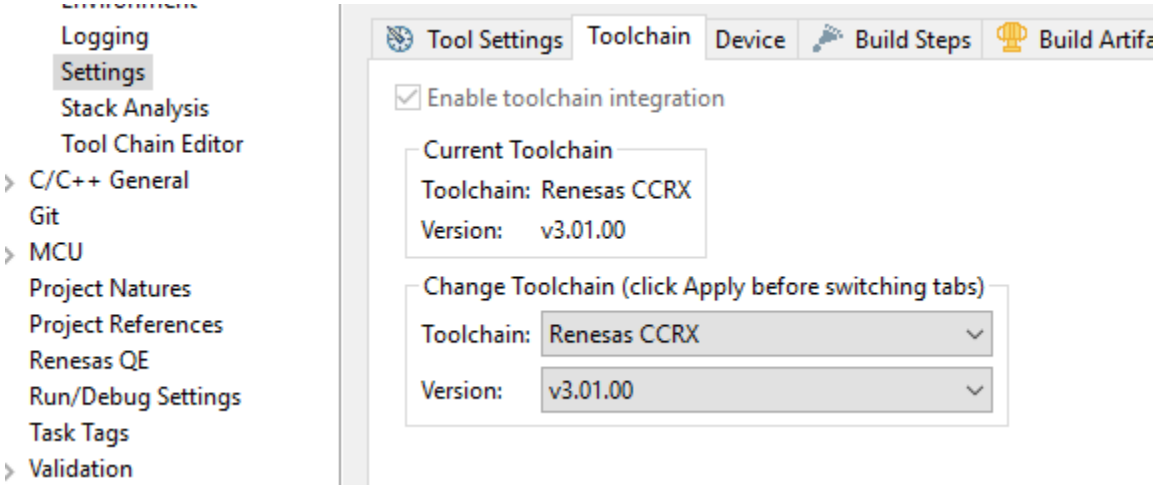
9. 두 프로젝트 모두 프로젝트 속성을 엽니다. 탐색 창에서 도구 체인 편집기를 선택합니다.

   a. 현재 도구 체인을 선택합니다.

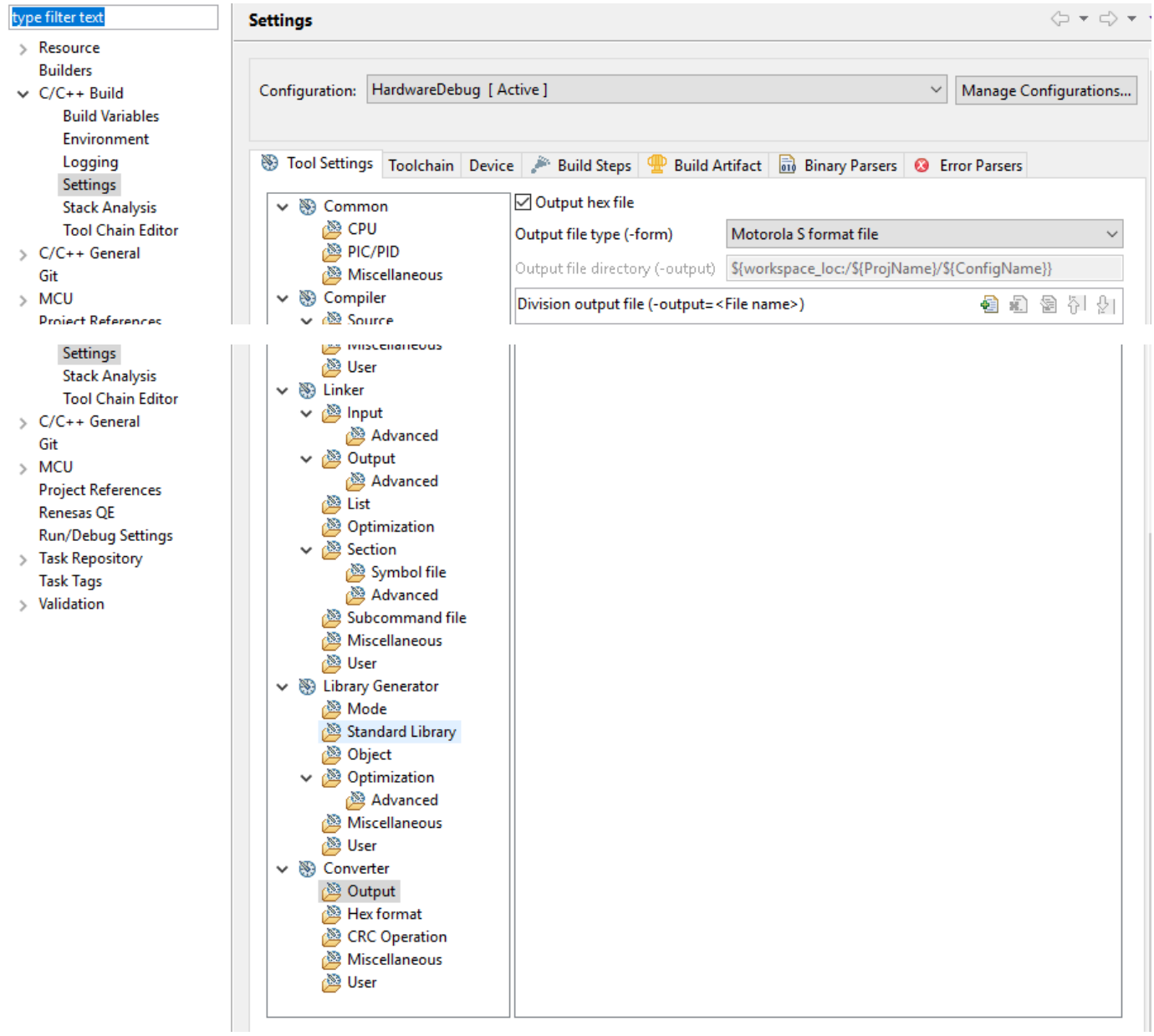
   b. 현재 빌더를 선택합니다.



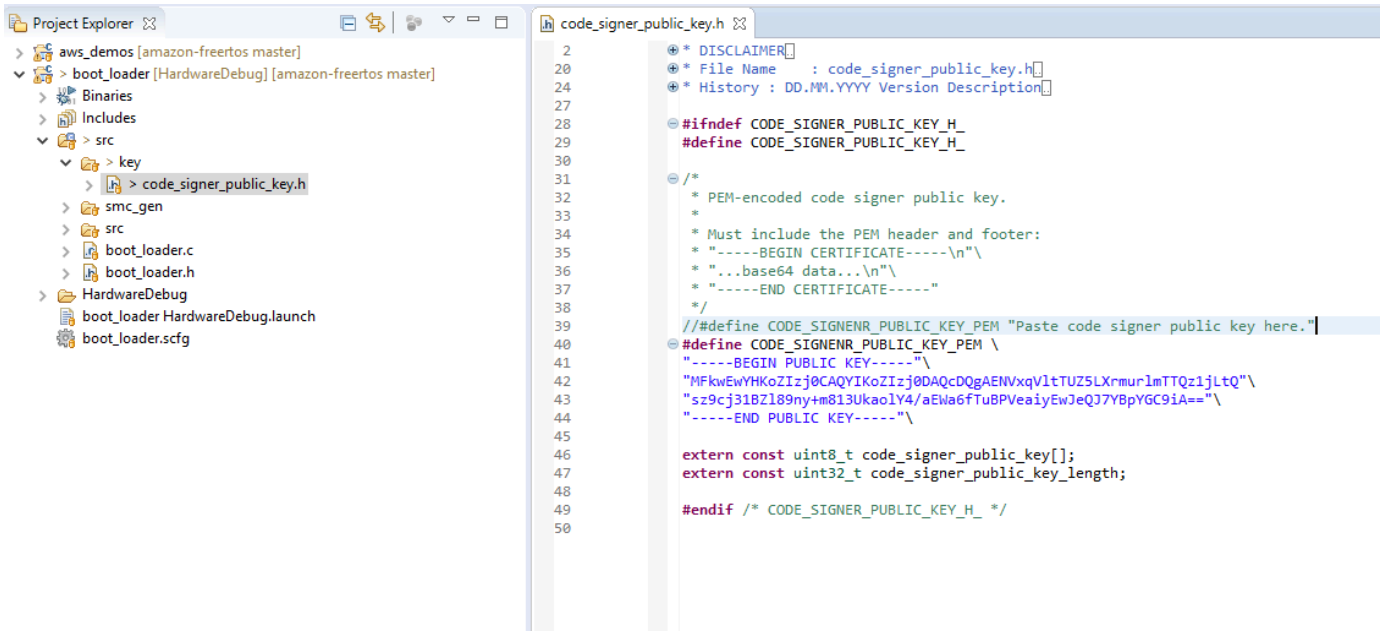
10. 탐색 창에서 설정을 선택합니다. 도구 체인 탭을 선택하고 버전을 선택합니다.



도구 설정 탭을 선택하고 변환기를 확장한 다음 출력을 선택합니다. 기본 창에서 출력 16진수 파일이 선택되어 있는지 확인한 다음 출력 파일 유형을 선택합니다.



11. 부트 로더 프로젝트에서 `projects\renesas\rx65n-rsk\e2studio\boot_loader\src\key\code_signer_public_key.h`를 열고 퍼블릭 키를 입력합니다. 퍼블릭 키를 생성하는 방법에 대한 자세한 내용은 How to implement FreeRTOS OTA by using Amazon Web Services on RX65N 및 Renesas MCU Firmware Update Design Policy 섹션 7.3 ‘Generating ECDSA-SHA256 Key Pairs with OpenSSL’을 참조하세요.



그런 다음 프로젝트를 빌드하여 `boot_loader.mot`를 생성합니다.

12. `aws_demos` 프로젝트를 엽니다.

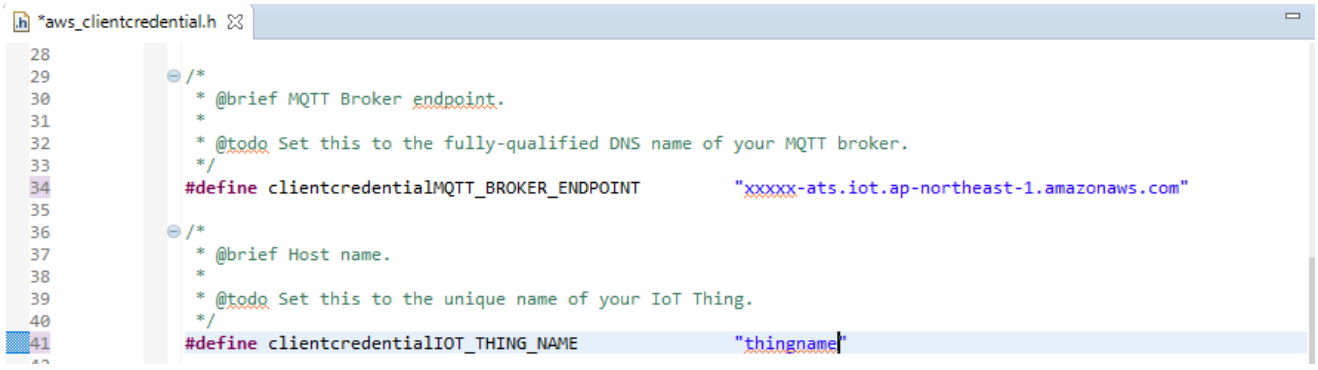
    a. AWS IoT 콘솔을 엽니다.

    b. 왼쪽 탐색 창에서 설정을 선택합니다. 사용자 지정 엔드포인트를 적어둡니다.

    c. 관리를 선택하고 사물을 선택합니다. 보드의 AWS IoT 사물 이름을 기록해 둡니다.

    d. `aws_demos` 프로젝트에서 `demos/include/aws_clientcredential.h`를 열고 다음 값을 지정합니다.

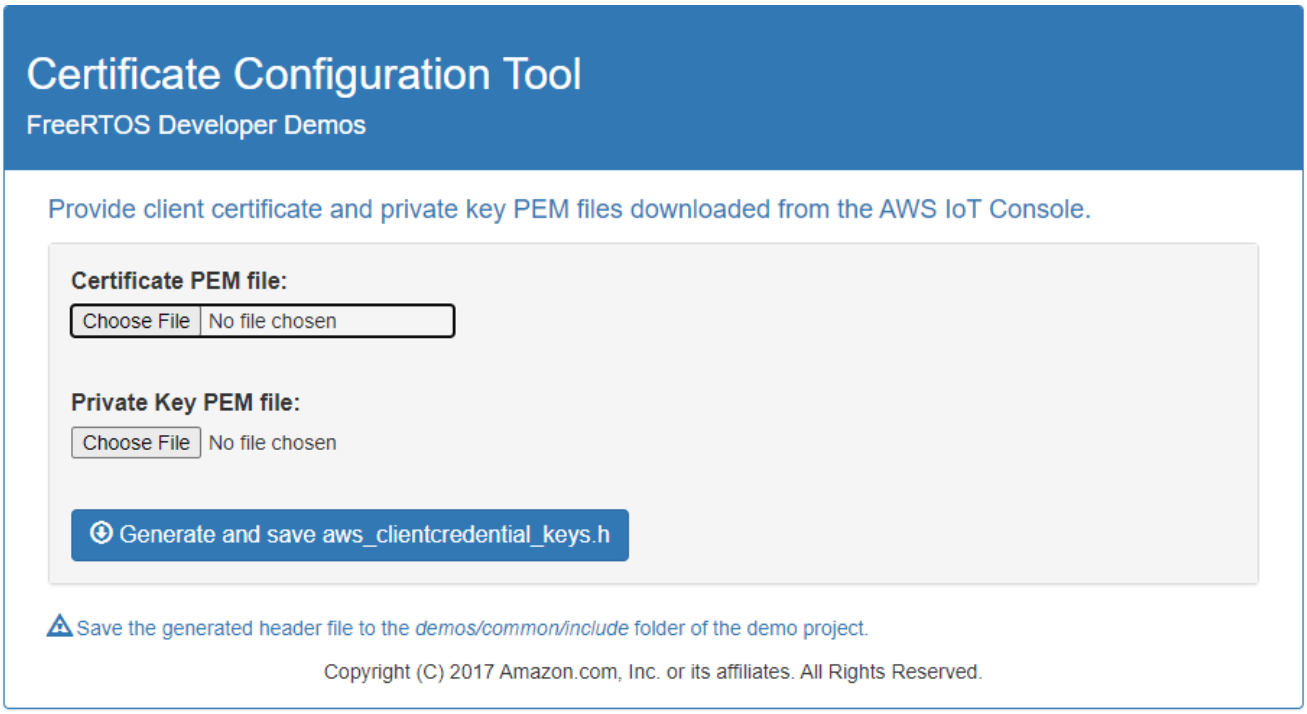
```
#define clientcredentialMQTT_BROKER_ENDPOINT[] = "Your AWS IoT endpoint";
#define clientcredentialIOT_THING_NAME "The AWS IoT thing name of your board"
```



    e. `tools/certificate_configuration/CertificateConfigurator.html` 파일을 엽니다.

f. 이전에 다운로드한 인증서 PEM 파일과 프라이빗 키 PEM 파일을 가져옵니다.

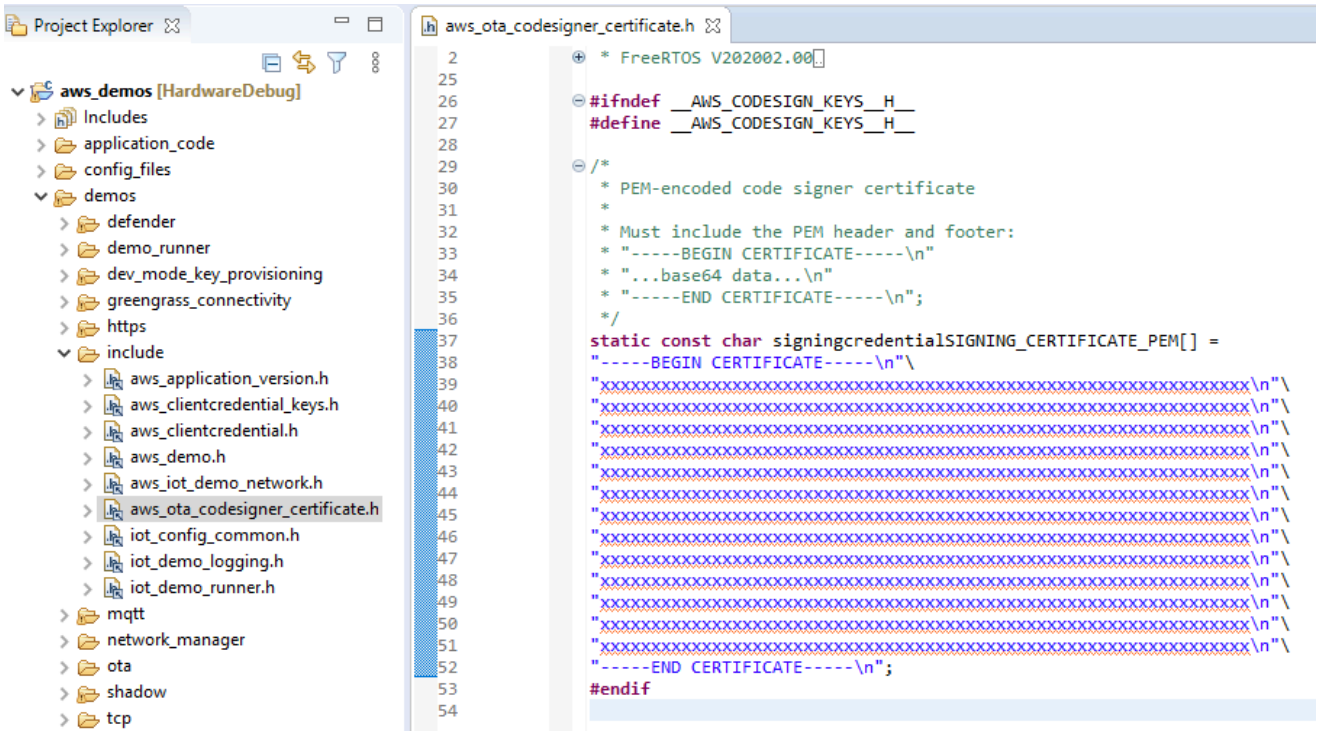
g. aws_clientcredential_keys.h 생성 및 저장을 선택하고 `demos/include/` 디렉터리에서 이 파일로 바꿉니다.



h. `demos/include/aws_ota_codesigner_certificate.h` 파일을 열고 다음 값을 지정합니다.

```
#define signingcredentialSIGNING_CERTIFICATE_PEM [] = "your-certificate-key";
```

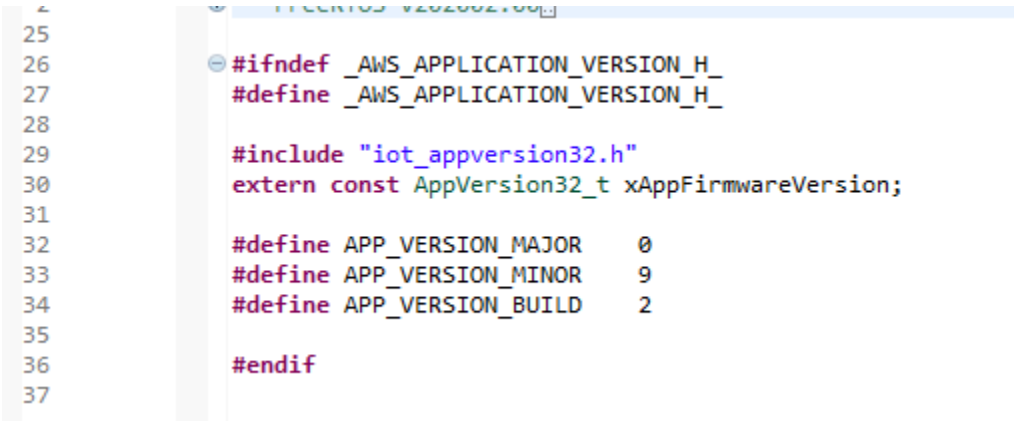
여기서 *your-certificate-key*는 `secp256r1.crt` 파일의 값입니다. 인증서의 각 줄 뒤에 '\'를 추가해야 합니다. `secp256r1.crt` 파일을 생성하는 방법에 대한 자세한 내용은 How to implement FreeRTOS OTA by using Amazon Web Services on RX65N 및 Renesas MCU Firmware Update Design Policy 섹션 7.3 'Generating ECDSA-SHA256 Key Pairs with OpenSSL'을 참조하세요.



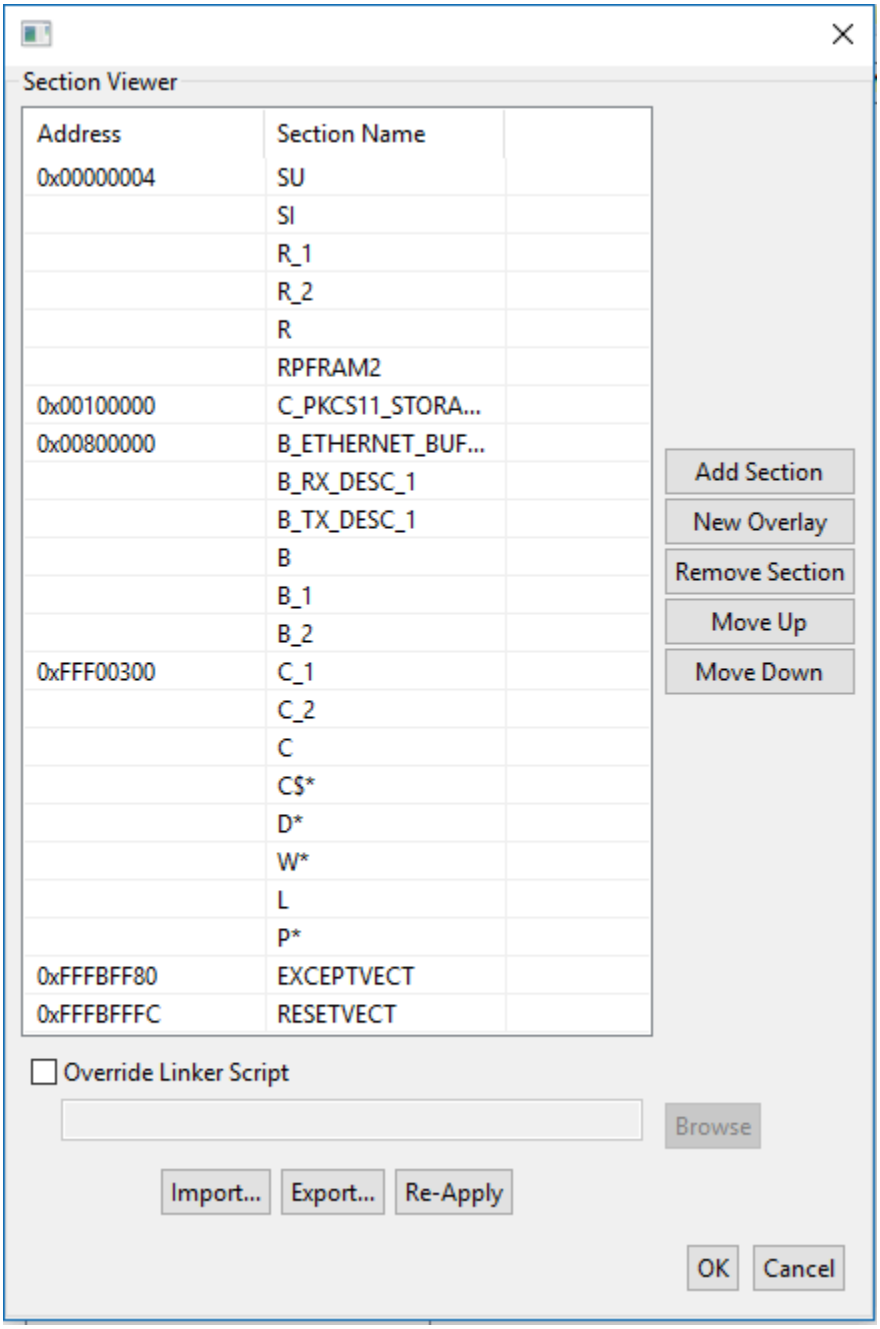
13. 작업 A: 펌웨어의 초기 버전 설치

    a. vendors/renesas/boards/board/aws_demos/config_files/ aws_demo_config.h 파일을 열고 #define CONFIG_MQTT_DEMO_ENABLED를 주석으로 처리한 다음 CONFIG_OTA_UPDATE_DEMO_ENABLED를 정의합니다.

    b. demos/include/ aws_application_version.h 파일을 열고 펌웨어의 초기 버전을 0.9.2로 설정합니다.



    c. 섹션 뷰어에서 다음 설정을 변경합니다.



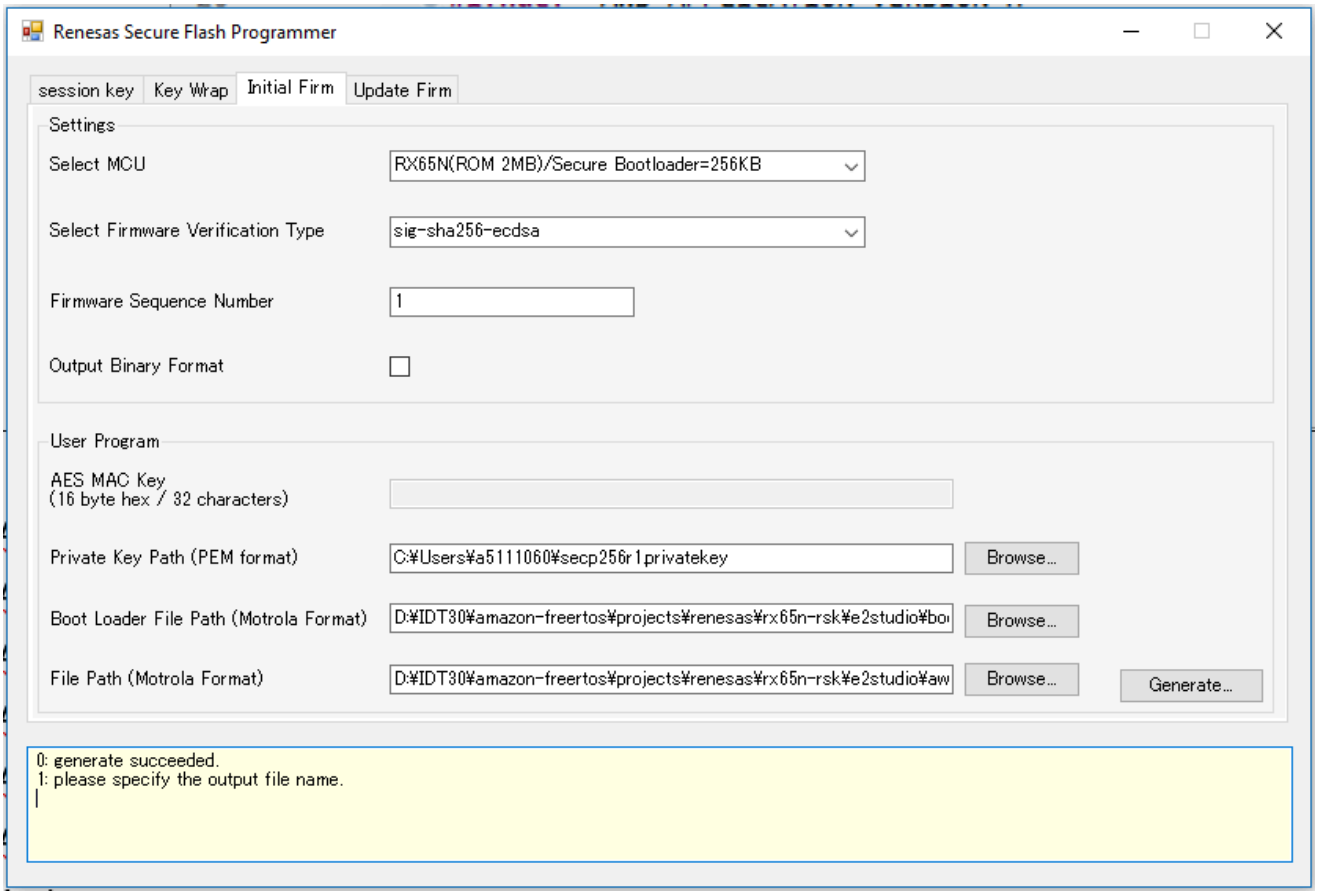
d. 빌드를 선택하여 `aws_demos.mot` 파일을 생성합니다.

14. Renesas Secure Flash Programmer를 사용하여 `userprog.mot` 파일을 생성합니다. `userprog.mot`는 `aws_demos.mot` 및 `boot_loader.mot`의 조합입니다. 이 파일을 RX65N-RSK로 플래시하여 초기 펌웨어를 설치할 수 있습니다.

    a. https://github.com/renesas/Amazon-FreeRTOS-Tools를 다운로드하고 `Renesas Secure Flash Programmer.exe`를 엽니다.

    b. 초기 펌웨어 탭을 선택한 후 다음 파라미터를 설정합니다.

- 프라이빗 키 경로 - `secp256r1.privatekey`의 위치입니다.
- 부트 로더 파일 경로 - `boot_loader.mot(projects\renesas\rx65n-rsk\e2studio\boot_loader\HardwareDebug)`의 위치입니다.
- 파일 경로 - `aws_demos.mot(projects\renesas\rx65n-rsk\e2studio\aws_demos\HardwareDebug)`의 위치입니다.



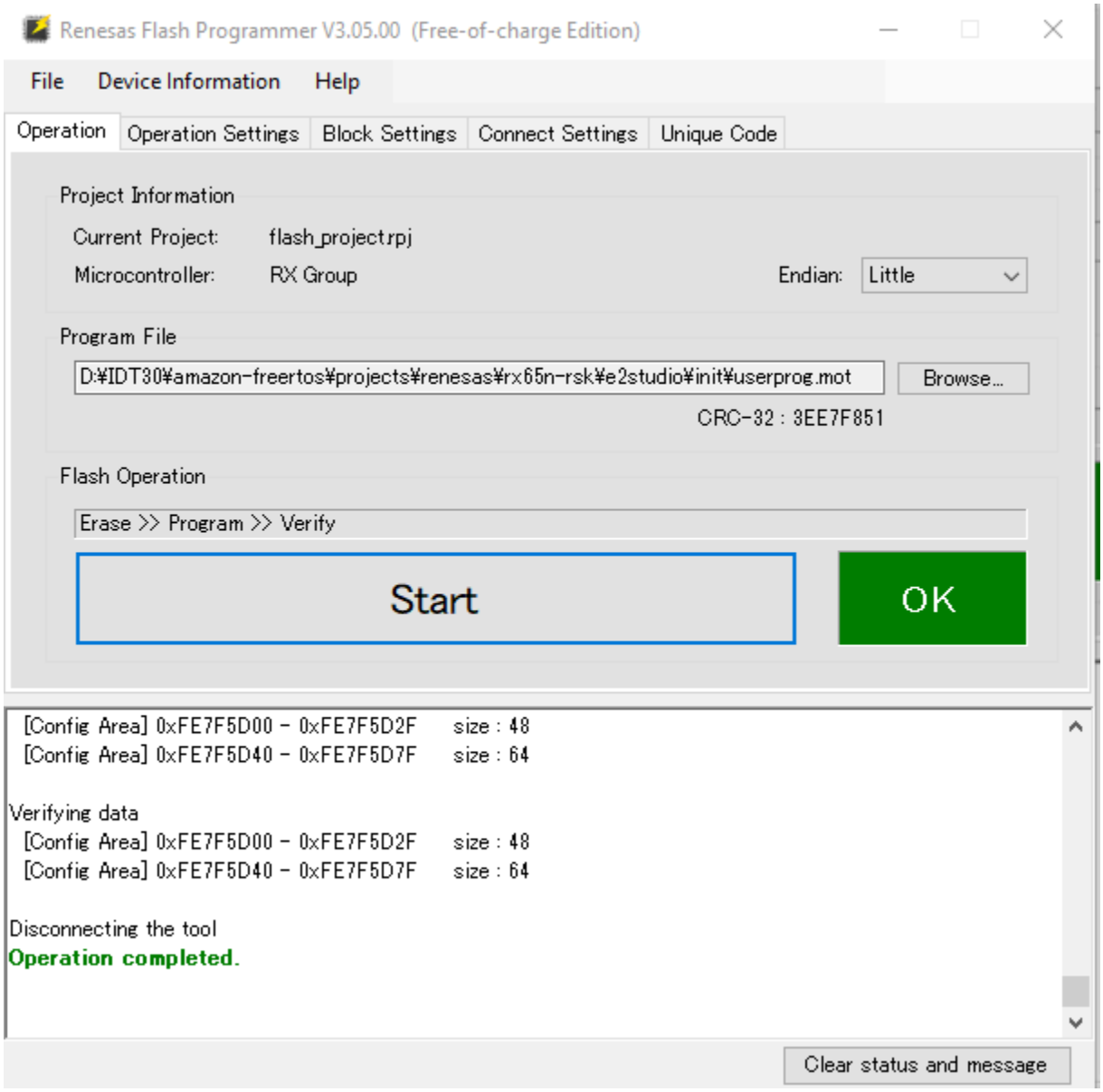
c. `init_firmware`라는 이름의 디렉터리를 생성합니다. `userprog.mot`를 생성하여 `init_firmware` 디렉터리에 저장합니다. 생성이 성공했는지 확인합니다.

15. RX65N-RSK에서 초기 펌웨어를 플래시합니다.

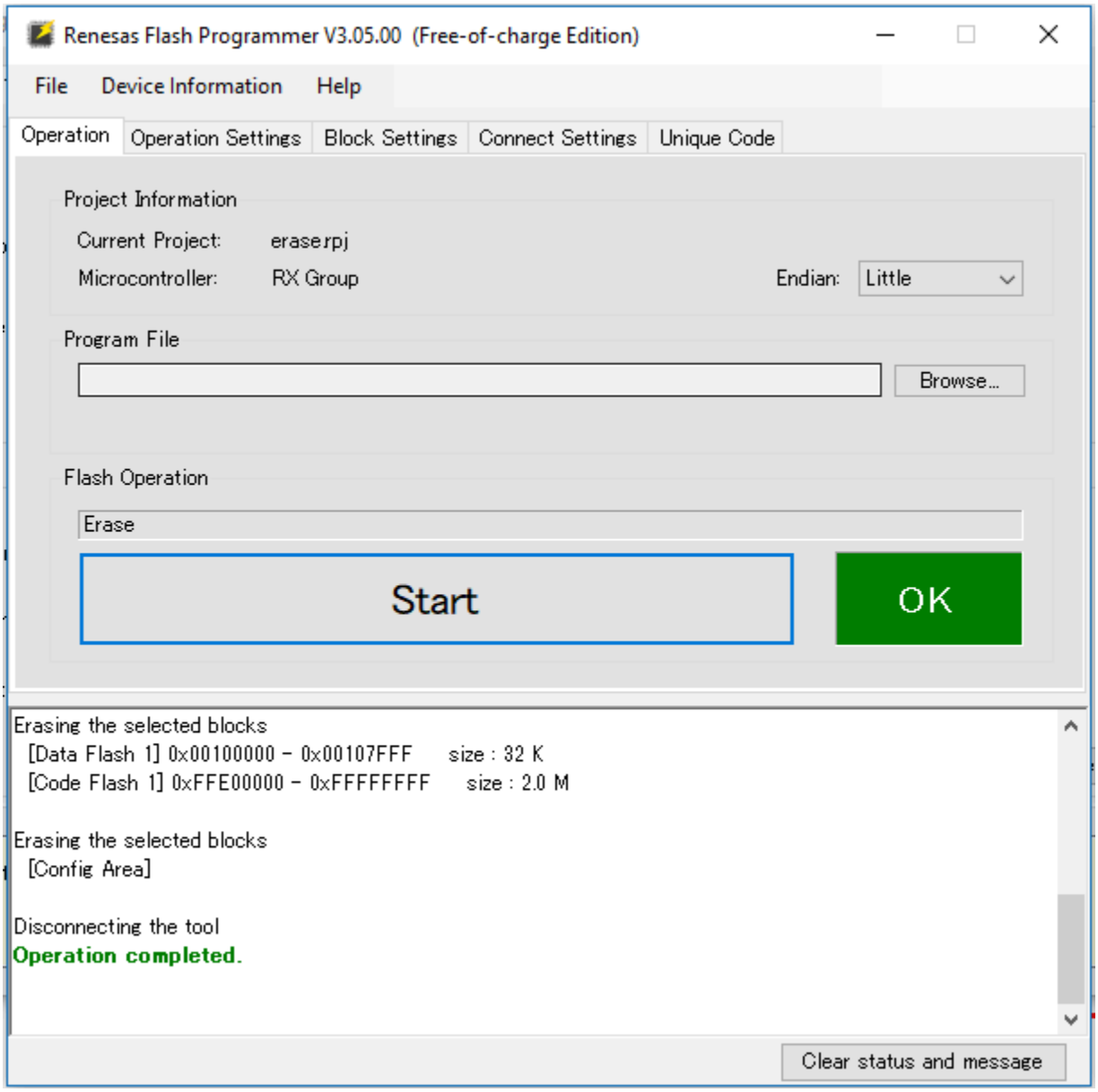
a. https://www.renesas.com/tw/en/products/software-tools/tools/programmer/renesas-flash-programmer-programming-gui.html에서 최신 버전의 Renesas Flash Programmer(프로그래밍 GUI)를 다운로드합니다.

b. `vendors\renesas\rx_mcu_boards\boards\rx65n-rsk\aws_demos\flash_project\erase_from_bank\ erase.rpj` 파일을 열어 뱅크에 있는 데이터를 지웁니다.

c. 시작을 선택하여 뱅크를 지웁니다.



d. `userprog.mot`를 플래시하려면 찾아보기...를 선택하고 `init_firmware` 디렉터리로 이동한 다음, `userprog.mot` 파일을 선택하고 시작을 선택합니다.



16. RX65N-RSK에 펌웨어 버전 0.9.2(초기 버전)가 설치되었습니다. 이제 RX65N-RSK 보드에서 OTA 업데이트를 수신합니다. PC에서 Tera Term을 연 경우 초기 펌웨어가 실행될 때 다음과 같은 내용이 표시됩니다.

```
-----------------------------------------------
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. 작업 B: 펌웨어 버전 업데이트

    a. demos/include/aws_application_version.h 파일을 열고 APP_VERSION_BUILD 토큰 값을 0.9.3으로 높입니다.

b. 프로젝트를 다시 빌드합니다.

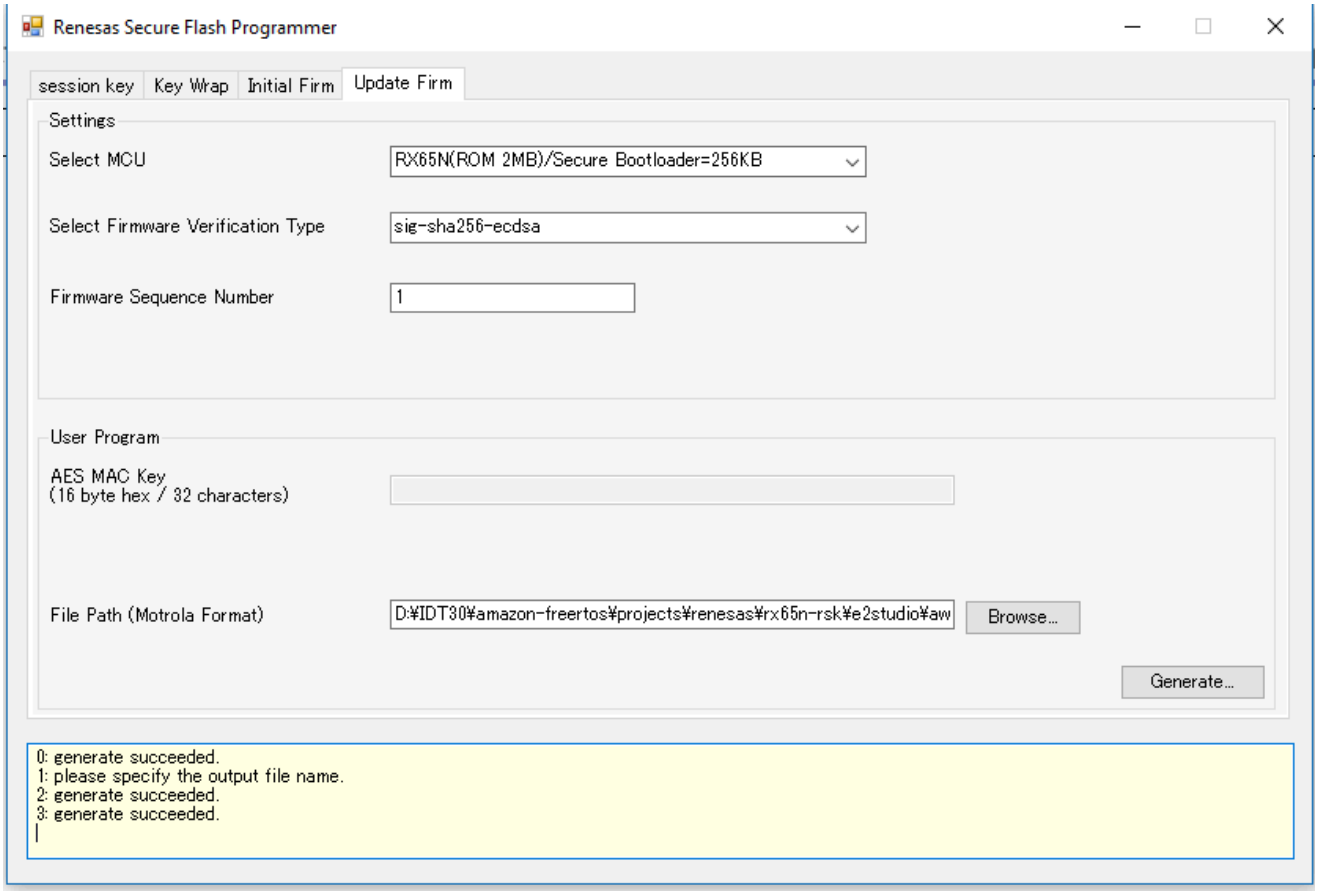
18. Renesas Secure Flash Programmer로 `userprog.rsu` 파일을 생성하여 펌웨어의 버전을 업데이트합니다.

    a. `Amazon-FreeRTOS-Tools\Renesas Secure Flash Programmer.exe` 파일을 엽니다.

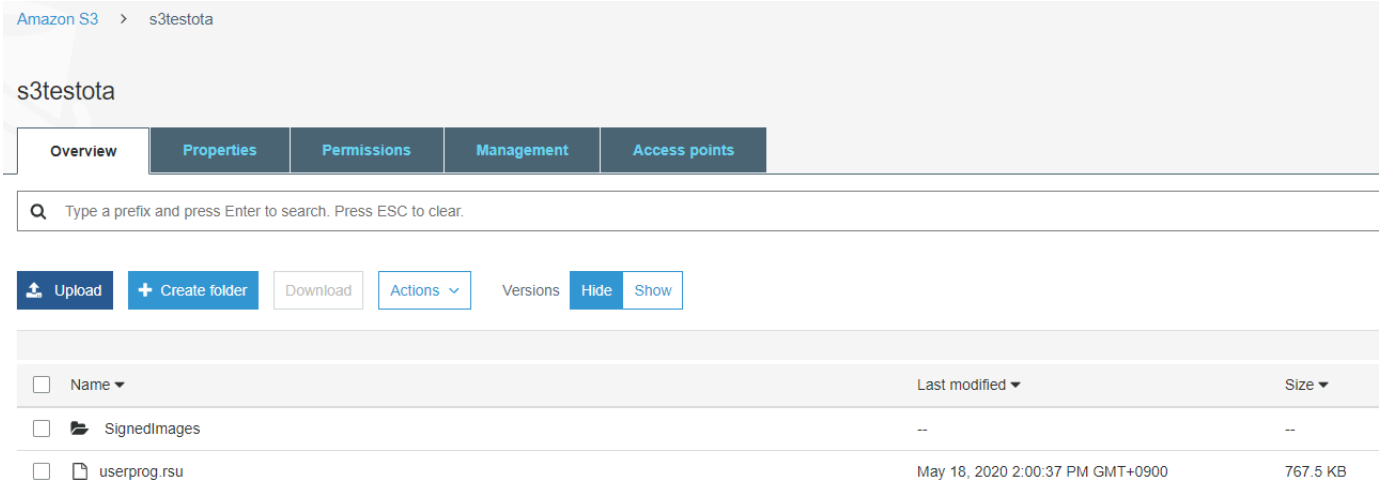
    b. 펌웨어 업데이트 탭을 선택한 후 다음 파라미터를 설정합니다.

    - 파일 경로 - `aws_demos.mot` 파일(`projects\renesas\rx65n-rsk\e2studio\aws_demos\HardwareDebug`)의 위치입니다.

    c. `update _firmware`이라는 디렉터리를 생성합니다. `userprog.rsu`를 생성하여 `update_firmware` 디렉터리에 저장합니다. 생성이 성공했는지 확인합니다.



19. 업데이트를 저장할 Amazon S3 버킷 생성에 설명된 대로 펌웨어 업데이트 `userproj.rsu`를 Amazon S3 버킷에 업로드합니다.



20. RX65N-RSK 펌웨어를 업데이트하는 작업을 생성합니다.

    AWS IoT 작업은 하나 이상의 연결된 디바이스에 보류 중인 작업을 알리는 서비스입니다. 작업을 사용하여 디바이스 플릿을 관리하거나, 디바이스의 펌웨어 및 보안 인증서를 업데이트하거나, 디바이스 재시작 및 진단과 같은 관리 작업을 수행할 수 있습니다.

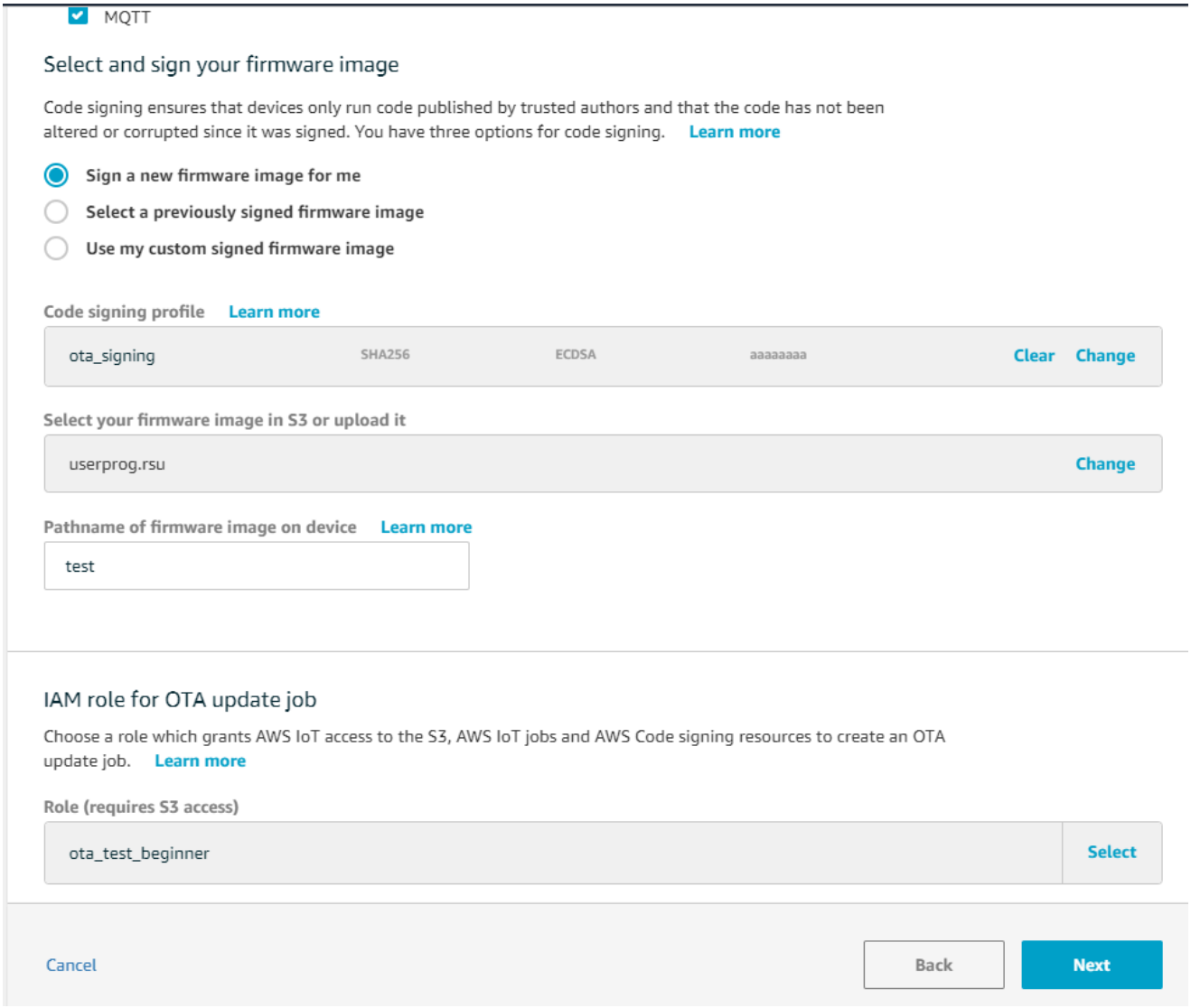
    a. AWS IoT 콘솔에 로그인합니다. 탐색 창에서 관리를 선택하고 작업을 선택합니다.

    b. 작업 생성을 선택하고 OTA 업데이트 작업 생성을 선택합니다. 사물을 선택하고 다음을 선택합니다.

    c. 다음과 같이 FreeRTOS OTA 업데이트 작업을 생성합니다.

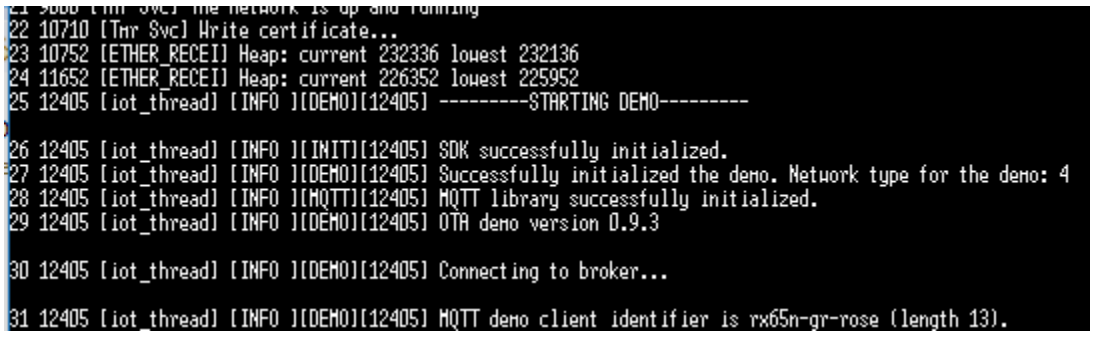
    - MQTT를 선택합니다.
    - 이전 섹션에서 생성한 코드 서명 프로필을 선택합니다.
    - Amazon S3 버킷에 업로드한 펌웨어 이미지를 선택합니다.
    - 디바이스에 있는 펌웨어 이미지의 경로 이름에 **`test`**를 입력합니다.
    - 이전 섹션에서 생성한 IAM 역할을 선택합니다.

    d. Next(다음)를 선택합니다.

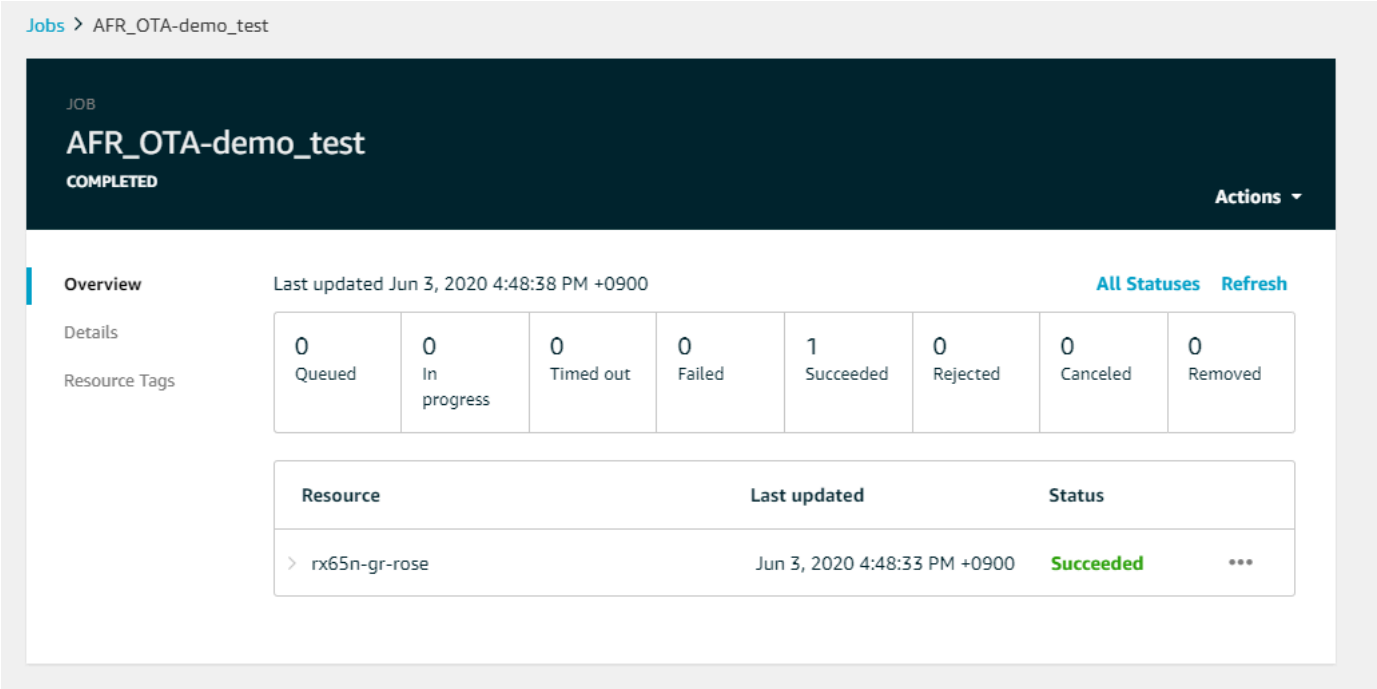


e. ID를 입력한 다음 생성을 선택합니다.

21. Tera Term을 다시 열어 펌웨어가 OTA 데모 버전 0.9.3으로 성공적으로 업데이트되었는지 확인합니다.



22. AWS IoT 콘솔에서 작업 상태가 성공인지 확인합니다.



AWS IoT 디바이스 섀도우 데모 애플리케이션

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

소개

이 데모에서는 AWS IoT Device Shadow 라이브러리를 사용하여 AWS Device Shadow 서비스에 연결하는 방법을 보여줍니다. coreMQTT 라이브러리를 사용하여 MQTT 브로커 및 coreJSON 라이브러리 구문 분석기에 대한 TLS(상호 인증)와 AWS IoT MQTT 연결을 설정하여 AWS 섀도우 서비스에서 받은 섀도우 문서를 구문 분석합니다. 데모에서는 섀도우 문서 업데이트 방법, 섀도우 문서 삭제 방법 등 기본적인 섀도우 작업을 보여줍니다. 또한 coreMQTT 라이브러리에 콜백 함수를 등록하여 AWS IoT 디바이스 섀도우 서비스에서 전송되는 섀도우 `/update` 및 `/update/delta` 메시지와 같은 메시지를 처리하는 방법도 보여줍니다.

이 데모는 섀도우 문서(상태) 업데이트 요청과 업데이트 응답이 동일한 애플리케이션에서 수행되므로 학습용으로만 사용됩니다. 실제 프로덕션 시나리오에서는 디바이스가 현재 연결되어 있지 않더라도

외부 애플리케이션이 원격으로 디바이스 상태 업데이트를 요청합니다. 디바이스가 연결되어 있으면 업데이트 요청을 확인합니다.

> **Note**
>
> FreeRTOS 데모를 설정하고 실행하려면 FreeRTOS 시작하기의 단계를 따릅니다.

기능

이 데모에서는 원격 디바이스 상태 전환을 시뮬레이션하는 섀도우 /update 및 /update/delta 콜백을 보여주는 일련의 예제를 반복하는 단일 애플리케이션 태스크를 생성합니다. 새 desired 상태가 포함된 섀도우 업데이트를 전송하고 디바이스가 새 desired 상태에 대한 응답으로 reported 상태를 변경할 때까지 기다립니다. 또한 섀도우 /update 콜백은 변화하는 섀도우 상태를 인쇄하는 데 사용됩니다. 또한이 데모에서는 MQTT 브로커에 대한 보안 AWS IoT MQTT 연결을 사용하고 디바이스 섀도우에 powerOn 상태가 있다고 가정합니다.

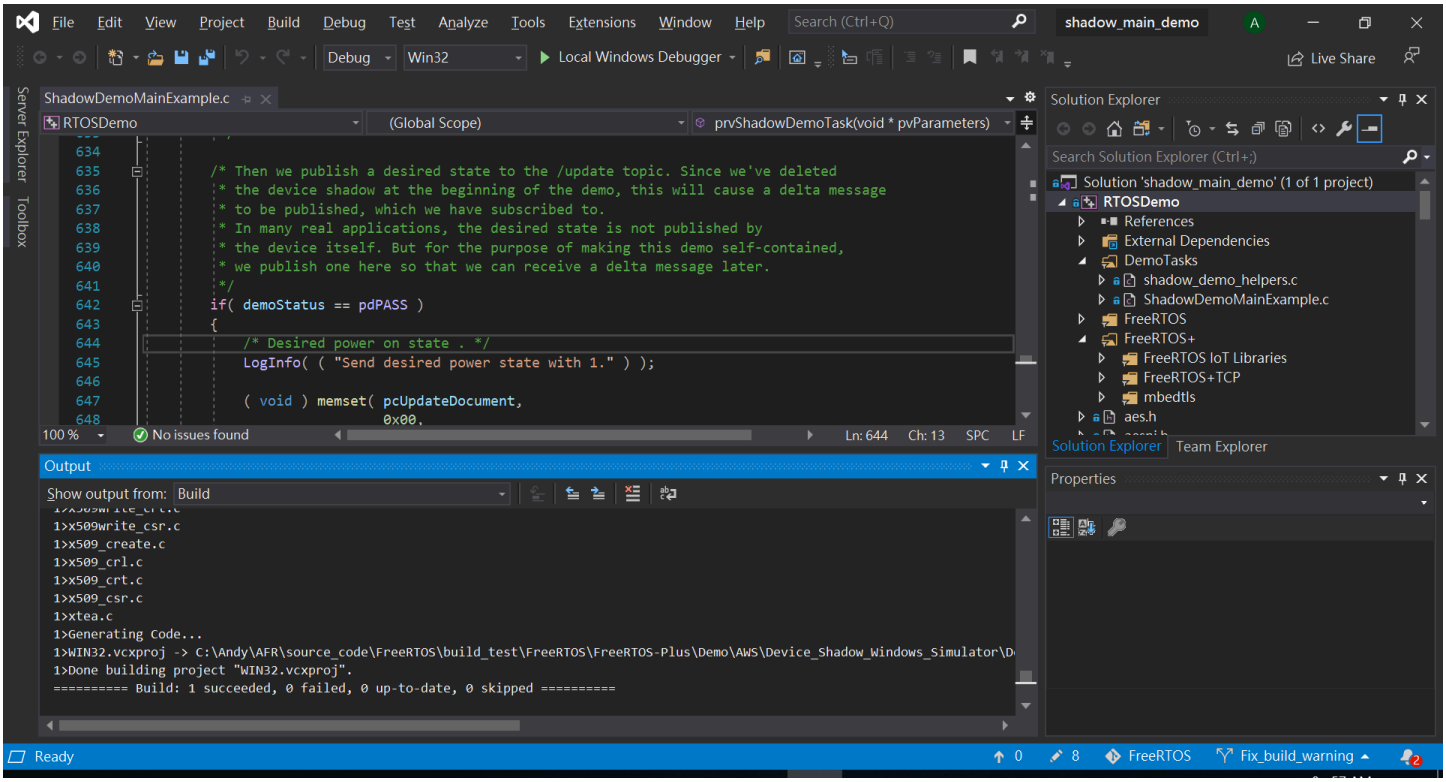
데모는 다음 작업을 수행합니다.

1. shadow_demo_helpers.c의 헬퍼 함수를 사용하여 MQTT 연결을 설정합니다.
2. AWS IoT 디바이스 섀도우 라이브러리에 정의된 매크로를 사용하여 디바이스 섀도우 작업에 사용할 MQTT 주제 문자열을 어셈블합니다.
3. 디바이스 섀도우를 삭제하는 데 사용되는 MQTT 주제에 게시하여 기존 디바이스 섀도우를 삭제합니다.
4. shadow_demo_helpers.c의 헬퍼 함수를 사용하여 /update/delta, /update/accepted 및 /update/rejected에 대한 MQTT 주제를 구독합니다.
5. powerOn의 헬퍼 함수를 사용하여 원하는 shadow_demo_helpers.c 상태를 게시합니다. 그러면 /update/delta 메시지가 디바이스로 전송됩니다.
6. 에서 수신 MQTT 메시지를 처리하고 Device Shadow 라이브러리()에서 정의한 함수를 사용하여 메시지가 AWS IoT 디바이스 섀도우와 관련이 있는지 prvEventCallback확인합니다Shadow_MatchTopic. 메시지가 디바이스 섀도우 /update/delta 메시지인 경우 메인 데모 함수는 보고된 상태를 업데이트하는 두 번째 메시지를 powerOn에 게시합니다. /update/accepted 메시지가 수신되면 clientToken이 이전에 업데이트 메시지에 게시된 것과 동일한지 확인합니다. 그러면 데모가 종료됩니다.

```
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]
```

데모는 *freertos*/demos/device_shadow_for_aws/shadow_demo_main.c 파일 또는 GitHub에서 찾을 수 있습니다.

다음 스크린샷은 데모가 성공할 경우 예상되는 출력을 보여줍니다.



AWS IoT MQTT 브로커에 연결

AWS IoT MQTT 브로커에 연결하려면 `MQTT_Connect()`에서와 동일한 방법을 사용합니다coreMQTT 상호 인증 데모.

섀도우 문서 삭제

섀도우 문서를 삭제하려면 AWS IoT 디바이스 섀도우 라이브러리에서 정의한 매크로를 사용하여 빈 메시지`xPublishToTopic`로를 호출합니다. 그러면 `MQTT_Publish`를 사용하여 `/delete` 주제에 게시합니다. 다음 코드 섹션은 `prvShadowDemoTask` 함수에서 이 작업이 어떻게 이루어지는지 보여줍니다.

```
/* 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 );
```

섀도우 주제 구독

디바이스 섀도우 주제를 구독하여 브로 AWS IoT 커로부터 섀도우 변경에 대한 알림을 받습니다. 디바이스 섀도우 주제는 디바이스 섀도우 라이브러리에 정의된 매크로에 의해 어셈블됩니다. 다음 코드 섹션은 `prvShadowDemoTask` 함수에서 이 작업이 어떻게 이루어지는지 보여줍니다.

```
/* 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 ) );
}
```

섀도우 업데이트 전송

섀도우 업데이트를 전송하기 위해 이 데모는 디바이스 섀도우 라이브러리에 정의된 매크로를 사용하여 JSON 형식 메시지와 함께 xPublishToTopic을 호출합니다. 그러면 MQTT_Publish를 사용하여 /delete 주제에 게시합니다. 다음 코드 섹션은 prvShadowDemoTask 함수에서 이 작업이 어떻게 이루어지는지 보여줍니다.

```
#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 ) );
```

섀도우 델타 메시지 및 섀도우 업데이트 메시지 처리

MQTT_Init 함수를 사용하여 coreMQTT 클라이언트 라이브러리에 등록된 사용자 콜백 함수는 들어오는 패킷 이벤트를 알려줍니다. GitHub에서 콜백 함수 prvEventCallback을 참조하세요.

이 콜백 함수는 들어오는 패킷이 MQTT_PACKET_TYPE_PUBLISH 유형임을 확인하고 디바이스 섀도우 라이브러리 API Shadow_MatchTopic을 사용하여 들어오는 메시지가 섀도우 메시지임을 확인합니다.

들어오는 메시지가 ShadowMessageTypeUpdateDelta 유형의 섀도우 메시지인 경우에는 prvUpdateDeltaHandler를 직접 호출하여 이 메시지를 처리합니다. 핸들러 prvUpdateDeltaHandler는 coreJSON 라이브러리를 사용하여 메시지를 파싱해 powerOn 상태에 대한 델타 값을 가져오고 이를 로컬에서 유지 관리되는 현재 디바이스 상태와 비교합니다. 두 값이 다를 경우 섀도우 문서의 새 powerOn 상태 값을 반영하도록 로컬 디바이스 상태가 업데이트됩니다.

들어오는 메시지가 ShadowMessageTypeUpdateAccepted 유형의 섀도우 메시지인 경우에는 prvUpdateAcceptedHandler를 직접 호출하여 이 메시지를 처리합니다. 핸들러 prvUpdateAcceptedHandler는 coreJSON 라이브러리를 사용하여 메시지를 파싱해 메시지에서 clientToken을 가져옵니다. 이 핸들러 함수는 JSON 메시지의 클라이언트 토큰이 애플리케이션에서 사용하는 클라이언트 토큰과 일치하는지 확인합니다. 일치하지 않을 경우 함수는 경고 메시지를 로그합니다.

보안 소켓 에코 클라이언트 데모

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

다음 예제에서는 단일 RTOS 작업을 사용합니다. 이 예제의 소스 코드는 demos/tcp/aws_tcp_echo_client_single_task.c에서 찾을 수 있습니다.

시작하기 전에 마이크로컨트롤러에 FreeRTOS를 다운로드하고 FreeRTOS 데모 프로젝트를 빌드 및 실행했는지 확인합니다. GitHub에서 FreeRTOS를 복제하거나 다운로드할 수 있습니다. 자세한 내용은 README.md 파일을 참조하십시오.

데모를 실행하려면

> **Note**
>
> FreeRTOS 데모를 설정하고 실행하려면 FreeRTOS 시작하기의 단계를 따릅니다.

TCP 서버와 클라이언트 데모는 현재 Cypress CYW943907AEVAL1F 및 CYW954907AEVAL1F 개발 키트에서 지원되지 않습니다.

1. FreeRTOS 이식 안내서의 TLS 에코 서버 설정에 나와 있는 지침을 따르세요.

   TLS 에코 서버를 실행하고 포트 9000에서 수신 대기해야 합니다.

   설정 중에 다음 네 파일을 생성해야 합니다.

   - `client.pem`(클라이언트 인증서)
   - `client.key`(클라이언트 프라이빗 키)
   - `server.pem`(서버 인증서)
   - `server.key`(서버 프라이빗 키)
2. `tools/certificate_configuration/CertificateConfigurator.html` 도구를 사용하여 클라이언트 인증서(`client.pem`)와 클라이언트 프라이빗 키(`client.key`)를 `aws_clientcredential_keys.h`에 복사합니다.
3. `FreeRTOSConfig.h` 파일을 엽니다.
4. `configECHO_SERVER_ADDR0`, `configECHO_SERVER_ADDR1`, `configECHO_SERVER_ADDR2`, `configECHO_SERVER_ADDR3` 변수를 TLS Echo Server가 실행되는 IP 주소를 구성하는 정수 네 개로 설정합니다.
5. `configTCP_ECHO_CLIENT_PORT` 변수를 TLS Echo Server가 수신 대기하는 포트인 `9000`으로 설정합니다.
6. `configTCP_ECHO_TASKS_SINGLE_TASK_TLS_ENABLED` 변수를 `1`로 설정합니다.
7. `tools/certificate_configuration/PEMfileToCString.html` 도구를 사용하여 서버 인증서(`server.pem`)를 `aws_tcp_echo_client_single_task.c` 파일의 `cTlsECHO_SERVER_CERTIFICATE_PEM`에 복사합니다.
8. *`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/aws_demo_config.h`를 열고 `#define CONFIG_MQTT_DEMO_ENABLED`를 주석으로 처리한 다음 `CONFIG_TCP_ECHO_CLIENT_DEMO_ENABLED`를 정의합니다.

마이크로컨트롤러와 TLS Echo Server는 동일한 네트워크에 있어야 합니다. 데모가 (`main.c`)를 시작하면 `Received correct string from echo server`라는 로그 메시지가 표시됩니다.

FreeRTOS AWS IoT Device Tester 용 사용

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

FreeRTOS용 AWS IoT Device Tester (IDT)를 사용하여 FreeRTOS 운영 체제가 디바이스에서 로컬로 작동하고 AWS IoT 클라우드와 통신할 수 있는지 확인할 수 있습니다. 특히, FreeRTOS 라이브러리의 이식 계층 인터페이스가 올바르게 구현되었는지 확인합니다. 또한를 사용하여 end-to-end 테스트를 수행합니다 AWS IoT Core. 예를 들어 보드에서 MQTT 메시지를 송수신하고 올바르게 처리할 수 있는지 확인합니다. FreeRTOS용 IDT가 실행하는 테스트는 FreeRTOS GitHub 리포지토리에서 정의됩니다.

테스트는 내장형 애플리케이션으로 실행되어 보드에 플래시됩니다. 애플리케이션 바이너리 이미지에는 FreeRTOS, 반도체 공급업체의 이식된 FreeRTOS 인터페이스, 보드 디바이스 드라이버가 포함되어 있습니다. 테스트의 용도는 이식된 FreeRTOS 인터페이스가 디바이스 드라이버 위에서 올바르게 작동하는지 확인하는 것입니다.

FreeRTOS용 IDT는에 제출하여 AWS 파트너 디바이스 카탈로그에 하드웨어 AWS IoT 를 추가할 수 있는 테스트 보고서를 생성합니다. 자세한 내용은 AWS Device Qualification Program을 참조하세요.

FreeRTOS용 IDT는 테스트할 보드에 연결된 호스트 컴퓨터(Windows, MacOS 또는 Linux)에서 실행됩니다. IDT는 테스트 사례를 실행하고 결과를 집계합니다. 또한 테스트 실행을 관리하기 위한 명령줄 인터페이스를 제공합니다.

FreeRTOS용 IDT는 디바이스 테스트 외에 검증 프로세스를 쉽게 진행하기 위한 리소스(예: AWS IoT 사물, FreeRTOS 그룹, Lambda 함수 등)를 생성합니다.

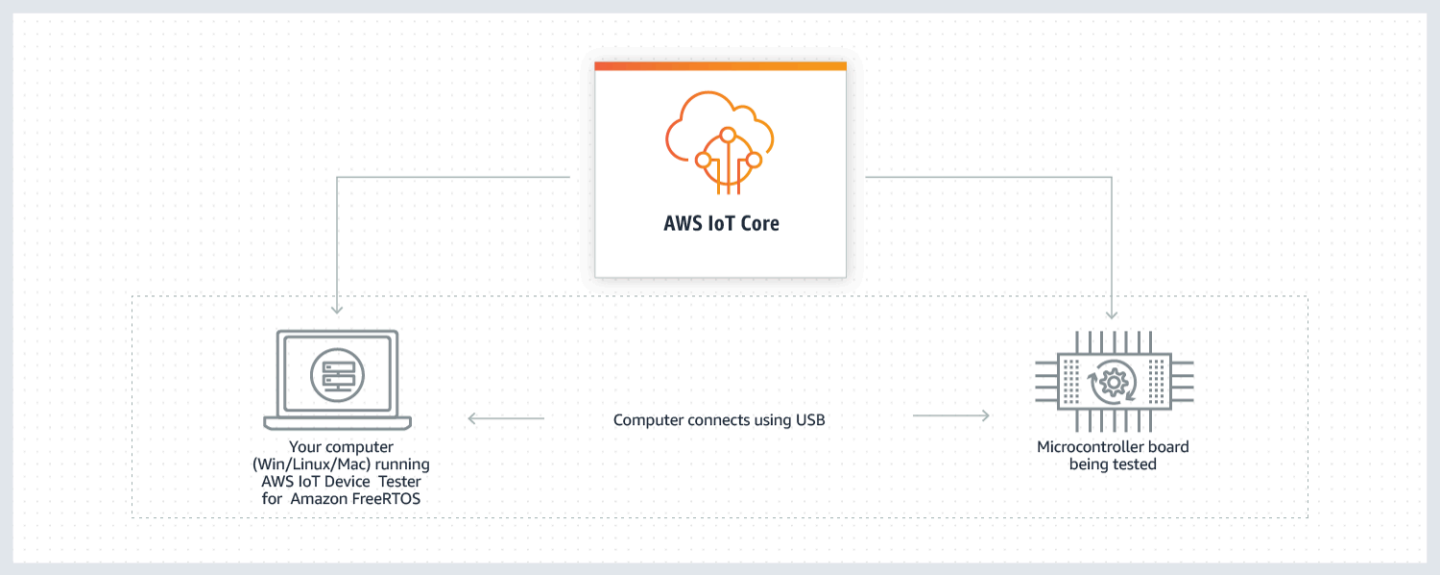
이러한 리소스를 생성하기 위해 FreeRTOS용 IDT는에 구성된 AWS 자격 증명을 사용하여 사용자를 대신하여 `config.json` API를 호출합니다. 이러한 리소스는 테스트 중 다양한 시점에서 프로비저닝됩니다.

FreeRTOS용 IDT를 호스트 컴퓨터에서 실행할 경우 다음 단계를 수행합니다.

1. 디바이스 및 자격 증명 구성을 로드하고 검증합니다.
2. 필수 로컬 및 클라우드 리소스를 사용하여 선택한 테스트를 수행합니다.

3. 로컬 및 클라우드 리소스를 정리합니다.
4. 보드에서 검증에 필요한 테스트를 통과했는지를 나타내는 테스트 보고서를 생성합니다.

다음 다이어그램은 테스트 인프라 설정을 보여 줍니다.



FreeRTOS용 IDT를 실행하기 위해 테스트 리소스를 사용할 수 있습니다. 리소스에는 두 가지 유형이 있습니다.

- 테스트 제품군은 디바이스가 FreeRTOS의 특정 버전에서 작동하는지 확인하는 데 사용되는 테스트 그룹 세트입니다.
- 테스트 그룹은 BLE 및 MQTT 메시징과 같은 특정 기능과 관련된 개별 테스트 집합입니다.

자세한 내용은 AWS IoT Device Tester FreeRTOS 테스트 제품군 버전용 단원을 참조하십시오.

FreeRTOS AWS IoT Device Tester 용의 지원되는 버전

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

이 주제에서는 FreeRTOS용 IDT의 지원되는 버전을 나열합니다. 가장 좋은 방법은 FreeRTOS의 대상 버전을 지원하는 FreeRTOS용 IDT의 최신 버전을 사용하는 것입니다. FreeRTOS용 IDT의 각 버전에

는 해당하는 FreeRTOS 버전이 하나 이상 있습니다. FreeRTOS의 새 릴리스를 사용하려면 새 버전의 FreeRTOS용 IDT를 다운로드해야 합니다.

소프트웨어를 다운로드하면 FreeRTOS용 IDT 라이선스 계약에 동의하는 것입니다.

FreeRTOS AWS IoT Device Tester 용의 최신 버전

다음 링크를 사용하여 최신 버전의 FreeRTOS용 IDT를 다운로드합니다.

FreeRTOS 202012.00용 IDT v4.0.3 및 테스트 제품군 1.5.0(FreeRTOS 202012.00 LTS 라이브러리 사용)

- Linux용 테스트 제품군 FRQ_1.5.0이 포함된 IDT v4.0.3
- macOS용 테스트 제품군 FRQ_1.5.0이 포함된 IDT v4.0.3
- Windows용 테스트 제품군 FRQ_1.5.0이 포함된 IDT v4.0.3

> **Note**
>
> 여러 사용자가 NFS 디렉터리 또는 Windows 네트워크 공유 폴더와 같은 공유 위치에서 IDT를 실행하지 않는 것이 좋습니다. 이로 인해 충돌이나 데이터 손상이 발생할 수 있습니다. 로컬 드라이브에 IDT 패키지의 압축을 풀고 로컬 워크스테이션에서 IDT 바이너리를 실행하는 것이 좋습니다.

릴리스 정보

- 하드웨어 보안 모듈에서 잠긴 보안 인증 정보를 사용하여 디바이스를 검증할 수 있도록 지원합니다.
- 사소한 버그 수정과 개선 사항입니다.

테스트 제품군 버전

- FRQ_1.5.0
  - 2021년 4월 16일에 릴리스되었습니다.

FreeRTOS용 IDT의 이전 버전

다음과 같은 이전 버전의 FreeRTOS용 IDT도 지원됩니다.

FreeRTOS 202012.00용 IDT v4.0.1 및 테스트 제품군 1.4.1(FreeRTOS 202012.00 LTS 라이브러리 사용)

- Linux용 테스트 제품군 FRQ_1.4.1이 포함된 IDT v4.0.1
- macOS용 테스트 제품군 FRQ_1.4.1이 포함된 IDT v4.0.1
- Windows용 테스트 제품군 FRQ_1.4.1이 포함된 IDT v4.0.1

릴리스 정보

- FreeRTOS LTS 라이브러리를 사용하는 FreeRTOS 202012.00을 지원합니다. FreeRTOS 202012.00 릴리스에 포함된 항목에 대한 자세한 내용은 GitHub의 CHANGELOG.md 파일을 참조하세요.
- OTA(Over-the-Air) E2E(엔드투엔드) 테스트 사례를 추가로 도입했습니다.
- FreeRTOS LTS 라이브러리를 사용하는 FreeRTOS 202012.00을 실행하는 개발 보드의 검증을 지원합니다.
- 셀룰러 연결을 사용하는 FreeRTOS 개발 보드 검증에 대한 지원을 추가했습니다.
- 에코 서버 구성의 버그를 수정했습니다.
- FreeRTOS AWS IoT Device Tester 용를 사용하여 자체 사용자 지정 테스트 제품군을 개발하고 실행할 수 있습니다. 자세한 내용은 IDT를 사용하여 자체 테스트 제품군 개발 및 실행 단원을 참조하십시오.
- 코드 서명된 IDT 애플리케이션을 제공하므로 Windows 또는 macOS에서 실행할 때 권한을 부여할 필요가 없습니다.
- BLE 테스트 결과 파싱 로직을 개선했습니다.

테스트 제품군 버전

- FRQ_1.4.1
  - 2021년 1월 19일에 릴리스되었습니다.

FreeRTOS 202011.01용 IDT v3.4.0 및 테스트 제품군 버전 1.3.0

- Linux용 테스트 제품군 FRQ_1.3.0이 포함된 IDT v3.4.0
- macOS용 테스트 제품군 FRQ_1.3.0이 포함된 IDT v3.4.0
- Windows용 테스트 제품군 FRQ_1.3.0이 포함된 IDT v3.4.0

릴리스 정보

- FreeRTOS 202011.01을 지원합니다. FreeRTOS 202011.01 릴리스에 포함된 항목에 대한 자세한 내용은 GitHub의 CHANGELOG.md 파일을 참조하세요.
- 'RSA'가 유효한 PKCS11 구성 옵션이 아닌 버그를 수정했습니다.
- OTA 테스트 후 Amazon S3 버킷이 제대로 정리되지 않는 버그를 수정했습니다.
- FullMQTT 테스트 그룹 내의 새 테스트 사례를 지원하도록 업데이트했습니다.

테스트 제품군 버전

- FRQ_1.3.0
  - 2020년 11월 5일에 릴리스되었습니다.

FreeRTOS202007.00용 IDT v3.3.0 및 테스트 제품군 버전 1.2.0

- Linux용 테스트 제품군 FRQ_1.2.0이 포함된 IDT v3.3.0
- macOS용 테스트 제품군 FRQ_1.2.0이 포함된 IDT v3.3.0
- Windows용 테스트 제품군 FRQ_1.2.0이 포함된 IDT v3.3.0

릴리스 정보

- FreeRTOS 202007.00을 지원합니다. FreeRTOS 202007.00 릴리스에 포함된 항목에 대한 자세한 내용은 GitHub의 CHANGELOG.md 파일을 참조하세요.
- 무선 업데이트(OTA) 일시 중지 및 재개 기능을 검증하기 위한 새로운 엔드투엔드 테스트를 추가했습니다.
- eu-central-1 리전의 사용자가 OTA 테스트에 대한 구성 검증을 통과하지 못하는 버그를 수정했습니다.
- `run-suite` 명령에 `--update-idt` 파라미터를 추가했습니다. 이 옵션을 사용하여 IDT 업데이트 프롬프트에 대한 응답을 설정할 수 있습니다.
- `run-suite` 명령에 `--update-managed-policy` 파라미터를 추가했습니다. 이 옵션을 사용하여 관리형 정책 업데이트 프롬프트에 대한 응답을 설정할 수 있습니다.
- 다음을 포함한 내부 개선 및 버그 수정:
  - 자동 테스트 제품군 업데이트의 경우 구성 파일 업그레이드를 개선했습니다.

테스트 제품군 버전

- FRQ_1.2.0
  - 2020년 9월 17일에 릴리스되었습니다.
  - 다음을 포함한 내부 개선 및 버그 수정:
    - OTA의 경우 이제 MQTT가 구성된 경우 연결 해제 및 재개 테스트에서 MQTT 데이터 영역을 제대로 사용합니다.
    - BLE의 경우 이제 Raspberry Pi 이미지의 암호를 변경할 수 있습니다.

자세한 내용은 FreeRTOS AWS IoT Device Tester 용 지원 정책 단원을 참조하십시오.

FreeRTOS용 IDT의 지원되지 않는 버전

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

이 섹션에서는 지원되지 않는 FreeRTOS용 IDT 버전을 나열합니다. 지원되지 않는 버전에는 버그 수정 또는 업데이트가 제공되지 않습니다. 자세한 내용은 FreeRTOS AWS IoT Device Tester 용 지원 정책 단원을 참조하십시오.

IDT-FreeRTOS의 다음 버전은 더 이상 지원되지 않습니다.

FreeRTOS 202002.00용 IDT v3.0.2

릴리스 정보

- FreeRTOS 202002.00을 지원합니다. FreeRTOS 202002.00 릴리스에 포함된 항목에 대한 자세한 내용은 GitHub의 CHANGELOG.md 파일을 참조하세요.
- IDT 내에서 테스트 제품군의 자동 업데이트를 추가합니다. IDT는 이제 해당 FreeRTOS 버전에서 사용할 수 있는 최신 테스트 제품군을 다운로드할 수 있습니다. 이 기능을 사용하여 다음을 수행할 수 있습니다.
  - `upgrade-test-suite` 명령을 사용하여 최신 테스트 제품군을 다운로드합니다.
  - IDT를 시작할 때 플래그를 설정하여 최신 테스트 제품군을 다운로드합니다.

`-u flag` 옵션을 사용해 항상 다운로드를 하려면 `###`를 'y'로, 기존 버전을 사용하려면 'n'로 지정합니다.

사용 가능한 테스트 제품군 버전이 여러 개인 경우 IDT를 시작할 때 테스트 제품군 ID를 지정하지 않으면 최신 버전이 사용됩니다.

- 새 `list-supported-versions` 옵션을 사용하여 설치된 IDT 버전에서 지원하는 FreeRTOS 및 테스트 제품군 버전을 나열합니다.
- 그룹의 테스트 케이스를 나열하고 개별 테스트를 실행합니다.

테스트 제품군은 `major.minor.patch` 형식을 사용하여 버전이 관리됩니다(1.0.0부터 시작).

- `list-supported-products` 명령 추가 - 설치된 IDT 버전에서 지원하는 FreeRTOS 및 테스트 제품군 버전을 나열합니다.
- `list-test-cases` 명령 추가 - 테스트 그룹에서 사용할 수 있는 테스트 사례를 나열합니다.
- `run-suite` 명령에 `test-id` 옵션 추가 - 이 옵션을 사용하여 테스트 그룹에서 개별 테스트 사례를 실행합니다.

테스트 제품군 버전

- FRQ_1.0.1

FreeRTOS 202002.00용 IDT v1.7.1

릴리스 정보

- FreeRTOS 202002.00을 지원합니다. FreeRTOS 202002.00 릴리스에 포함된 항목에 대한 자세한 내용은 GitHub의 CHANGELOG.md 파일을 참조하세요.
- OTA(Over-the-Air) 종단 간 테스트 케이스에 대한 사용자 정의 코드 서명 방법을 지원하므로 자체 코드 서명 명령 및 스크립트를 사용하여 OTA 페이로드에 서명할 수 있습니다.
- 테스트를 시작하기 전에 직렬 포트에 대한 사전 검사를 추가합니다. 직렬 포트가 `device.json` 파일에 잘못 구성된 경우 향상된 오류 메시지와 함께 테스트가 빠르게 실패합니다.
- AWS IoT Device Tester실행에 필요한 권한이 있는 AWS 관리형 정책 `AWSIoTDeviceTesterForFreeRTOSFullAccess`를 추가했습니다. 새 릴리스에 추가 권한이 필요한 경우 해당 권한을 이 관리형 정책에 추가하므로 사용자가 IAM 권한을 업데이트할 필요가 없습니다.
- 결과 디렉터리에 있는 `AFQ_Report.xml` 파일이 이제 `FRQ_Report.xml`입니다

FreeRTOS 201912.00용 IDT v1.6.2

릴리스 정보

- FreeRTOS 201912.00을 지원합니다.
- HTTPS를 통한 OTA에 대한 선택적 테스트를 지원하여 FreeRTOS 개발 보드를 검증합니다.
- 테스트에서 AWS IoT ATS 엔드포인트를 지원합니다.
- 테스트 제품군 시작 전에 사용자에게 최신 IDT 버전을 알리는 기능을 지원합니다.

FreeRTOS 201910.00용 IDT v1.5.2

릴리스 정보

- 보안 요소(온보드 키)가 있는 FreeRTOS 디바이스의 검증을 지원합니다.
- SecureSocket 및 WiFi 테스트 그룹을 위한 구성 가능한 에코 서버 포트를 지원합니다.
- 제한 시간 관련 오류를 해결할 때 제한 시간을 늘리는 제한 시간 승수 플래그를 지원합니다.
- 로그 구문 분석에 대한 버그 수정이 추가되었습니다.
- 테스트 시 IoT ATS 엔드포인트를 지원합니다.

FreeRTOS 201908.00용 IDT v1.4.1

릴리스 정보

- 새 PKCS11 라이브러리와 테스트 사례 업데이트에 대한 지원을 추가했습니다.
- 실행 가능한 오류 코드를 도입했습니다. 자세한 내용은 IDT 오류 코드 섹션을 참조하세요.
- IDT를 실행하는 데 사용된 IAM 정책이 업데이트되었습니다.

FreeRTOS 201906.00 Major용 IDT v1.3.2

릴리스 정보

- Bluetooth Low Energy(BLE) 테스트에 대한 지원이 추가되었습니다.
- IDT 명령줄 인터페이스(CLI) 명령에 대한 사용자 환경이 개선되었습니다.
- IDT를 실행하는 데 사용된 IAM 정책이 업데이트되었습니다.

IDT-FreeRTOS v1.2

- FreeRTOS 버전:
  - FreeRTOS v1.4.9
  - FreeRTOS v1.4.8
- 출시 정보:
  - FreeRTOS v1.4.8 및 v1.4.9에 대한 지원을 추가했습니다.
  - CMAKE 빌드 시스템으로 FreeRTOS 디바이스를 테스트할 수 있도록 지원을 추가했습니다.

IDT-FreeRTOS v1.1

- FreeRTOS 버전:
  - FreeRTOS v1.4.7

IDT-FreeRTOS v1.0

- FreeRTOS 버전:
  - FreeRTOS v1.4.6
  - FreeRTOS v1.4.5
  - FreeRTOS v1.4.4
  - FreeRTOS v1.4.3
  - FreeRTOS v1.4.2

사전 조건

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

이 섹션에서는를 사용하여 마이크로컨트롤러를 테스트하기 위한 사전 조건을 설명합니다 AWS IoT Device Tester.

FreeRTOS 다운로드

다음 명령을 사용하여 GitHub에서 FreeRTOS 릴리스를 다운로드할 수 있습니다.

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

여기서, <FREERTOS_RELEASE_VERSION>은 FreeRTOS AWS IoT Device Tester 용의 지원되는 버전에 나열된 IDT 버전에 해당하는 FreeRTOS 버전(예: 202007.00)입니다. 이렇게 하면 하위 모듈을 포함한 전체 소스 코드를 사용하고 FreeRTOS 버전에 해당하는 올바른 버전의 IDT를 사용할 수 있으며 그 반대의 경우도 마찬가지입니다.

Windows의 경우 260자의 경로 길이 제한이 있습니다. FreeRTOS의 경로 구조는 깊이가 여러 수준이기 때문에 Windows를 사용하는 경우 파일 경로를 260자 제한 미만으로 유지해야 합니다. 예를 들어 FreeRTOS를 `C:\Users\username\programs\projects\myproj\FreeRTOS\` 대신 `C:\FreeRTOS`로 복제하세요.

LTS 검증(LTS 라이브러리를 사용하는 FreeRTOS에 대한 검증)

- 마이크로컨트롤러를 AWS 파트너 디바이스 카탈로그에서 장기 지원(LTS) 기반 버전의 FreeRTOS 지원으로 지정하려면 매니페스트 파일을 제공해야 합니다. 자세한 내용은 FreeRTOS 검증 가이드의 FreeRTOS 검증 체크리스트를 참조하세요.
- 마이크로컨트롤러가 FreeRTOS의 LTS 기반 버전을 지원하는지 확인하고 AWS 파트너 디바이스 카탈로그에 제출할 수 있도록 검증하려면 FreeRTOS 검증 AWS IoT Device Tester (FRQ) 테스트 제품군 버전 v1.4.x와 함께 (IDT)를 사용해야 합니다.
- FreeRTOS의 LTS 기반 버전에 대한 지원은 FreeRTOS 202012.xx 버전으로 제한됩니다.

FreeRTOS용 IDT 다운로드

FreeRTOS의 모든 버전에는 검증 테스트를 수행하기 위한 해당 버전의 FreeRTOS용 IDT가 있습니다. FreeRTOS AWS IoT Device Tester 용의 지원되는 버전에서 적절한 버전의 FreeRTOS용 IDT를 다운로드합니다.

읽기 및 쓰기 권한이 있는 파일 시스템의 위치에 FreeRTOS용 IDT의 압축을 풉니다. Microsoft Windows에는 경로 길이에 문자 제한이 있으므로 FreeRTOS용 IDT를 `C:\` 또는 `D:\`와 같은 루트 디렉터리에 추출합니다.

> **Note**
>
> 여러 사용자가 NFS 디렉터리 또는 Windows 네트워크 공유 폴더와 같은 공유 위치에서 IDT를 실행하지 않는 것이 좋습니다. 이로 인해 충돌이나 데이터 손상이 발생할 수 있습니다. IDT 패키지를 로컬 드라이브에 추출하는 것이 좋습니다.

에 가입 AWS 계정

이 없는 경우 다음 단계를 AWS 계정완료하여 생성합니다.

에 가입하려면 AWS 계정

1. https://portal.aws.amazon.com/billing/signup을 엽니다.
2. 온라인 지시 사항을 따릅니다.

   가입 절차의 일부로는 전화 또는 문자 메시지를 수신하고 전화 키패드에 확인 코드를 입력하는 것이 포함됩니다.

   에 가입하면 AWS 계정AWS 계정 루트 사용자이 생성됩니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스에 액세스할 권한이 있습니다. 보안 모범 사례는 사용자에게 관리 액세스 권한을 할당하고, 루트 사용자만 사용하여 루트 사용자 액세스 권한이 필요한 작업을 수행하는 것입니다.

AWS 는 가입 프로세스가 완료된 후 확인 이메일을 보냅니다. 언제든지 https://aws.amazon.com/으로 이동하고 내 계정을 선택하여 현재 계정 활동을 보고 계정을 관리할 수 있습니다.

관리자 액세스 권한이 있는 사용자 생성

에 가입한 후 일상적인 작업에 루트 사용자를 사용하지 않도록 관리 사용자를 AWS 계정보호 AWS IAM Identity Center, AWS 계정 루트 사용자활성화 및 생성합니다.

보안 AWS 계정 루트 사용자

1. 루트 사용자를 선택하고 AWS 계정 이메일 주소를 입력하여 계정 소유자AWS Management Console로에 로그인합니다. 다음 페이지에서 비밀번호를 입력합니다.

   루트 사용자를 사용하여 로그인하는 데 도움이 필요하면 AWS 로그인 User Guide의 루트 사용자로 로그인을 참조하세요.

2. 루트 사용자의 다중 인증(MFA)을 활성화합니다.

   지침은 IAM 사용 설명서의 AWS 계정 루트 사용자(콘솔)에 대한 가상 MFA 디바이스 활성화를 참조하세요.

관리자 액세스 권한이 있는 사용자 생성

1. IAM Identity Center를 활성화합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 AWS IAM Identity Center설정을 참조하세요.
2. IAM Identity Center에서 사용자에게 관리 액세스 권한을 부여합니다.

   를 자격 증명 소스 IAM Identity Center 디렉터리 로 사용하는 방법에 대한 자습서는 사용 AWS IAM Identity Center 설명서의 기본값으로 사용자 액세스 구성을 IAM Identity Center 디렉터리 참조하세요.

관리 액세스 권한이 있는 사용자로 로그인

- IAM IDentity Center 사용자로 로그인하려면 IAM Identity Center 사용자를 생성할 때 이메일 주소로 전송된 로그인 URL을 사용합니다.

  IAM Identity Center 사용자를 사용하여 로그인하는 데 도움이 필요하면 AWS 로그인 사용 설명서의 AWS 액세스 포털에 로그인을 참조하세요.

추가 사용자에게 액세스 권한 할당

1. IAM Identity Center에서 최소 권한 적용 모범 사례를 따르는 권한 세트를 생성합니다.

   지침은AWS IAM Identity Center 사용 설명서의 Create a permission set를 참조하세요.
2. 사용자를 그룹에 할당하고, 그룹에 Single Sign-On 액세스 권한을 할당합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 Add groups를 참조하세요.

AWS IoT Device Tester 관리형 정책

`AWSIoTDeviceTesterForFreeRTOSFullAccess` 관리형 정책에는 디바이스 테스터가 지표를 실행하고 수집할 수 있는 다음 권한이 포함되어 있습니다.

- `iot-device-tester:SupportedVersion`

  AWS CLI에서 사용할 수 있도록 IDT에서 지원하는 FreeRTOS 버전 및 테스트 제품군 버전의 목록을 가져올 수 있는 권한을 부여합니다.
- `iot-device-tester:LatestIdt`

  다운로드할 수 있는 최신 AWS IoT Device Tester 버전을 가져올 수 있는 권한을 부여합니다.
- `iot-device-tester:CheckVersion`

  제품, 테스트 제품군 및 AWS IoT Device Tester 버전의 조합이 호환되는지 확인할 수 있는 권한을 부여합니다.
- `iot-device-tester:DownloadTestSuite`

  테스트 제품군을 다운로드할 AWS IoT Device Tester 수 있는 권한을 부여합니다.
- `iot-device-tester:SendMetrics`

  AWS IoT Device Tester 사용량 지표 데이터를 게시할 수 있는 권한을 부여합니다.

(선택 사항) 설치 AWS Command Line Interface

를 사용하여 일부 작업을 수행하는 AWS CLI 것이 좋습니다. 가 AWS CLI 설치되어 있지 않은 경우 설치 AWS CLI의 지침을 따릅니다.

명령줄aws configure에서를 실행하여 사용하려는 AWS 리전에 AWS CLI 대해를 구성합니다. FreeRTOS용 IDT를 지원하는 AWS 리전에 대한 자세한 내용은 AWS 리전 및 엔드포인트를 참조하세요. aws configure에 대한 자세한 내용은 aws configure를 사용한 빠른 구성을 참조하세요.

처음으로 마이크로 컨트롤러 보드의 테스트 준비

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

FreeRTOS 인터페이스를 이식할 때 FreeRTOS용 IDT를 사용하여 테스트할 수 있습니다. 보드의 디바이스 드라이버에 대한 FreeRTOS 인터페이스를 이식한 후 AWS IoT Device Tester 를 사용하여 마이크로컨트롤러 보드에서 검증 테스트를 실행합니다.

라이브러리 이식 계층 추가

FreeRTOS를 디바이스에 이식하려면 FreeRTOS 이식 안내서의 지침을 따르세요.

자격 AWS 증명 구성

AWS 클라우드와 통신하려면 AWS IoT Device Tester 에 대한 AWS 자격 증명을 구성해야 합니다. 자세한 내용은 개발을 위한 AWS 보안 인증 정보 및 리전 설정을 참조하세요. *devicetester_extract_location*/devicetester_afreertos_*[win|mac|linux]*/configs/config.json 구성 파일에 유효한 AWS 자격 증명을 지정해야 합니다.

FreeRTOS용 IDT에서 디바이스 풀 생성

테스트할 디바이스는 디바이스 풀에서 구성됩니다. 각 디바이스 풀은 하나 이상의 동일한 디바이스로 구성됩니다. 풀에서 단일 디바이스를 테스트하거나 풀에서 여러 디바이스를 테스트하도록 FreeRTOS용 IDT를 구성할 수 있습니다. 검증 프로세스를 가속화하기 위해 FreeRTOS용 IDT는 동일한 사양이 있는 디바이스를 병렬로 테스트할 수 있습니다. 라운드 로빈 방법을 사용하여 디바이스 풀에 있는 각 디바이스에 대해 다른 테스트 그룹을 실행합니다.

configs 폴더에서 device.json 템플릿의 devices 섹션을 편집하여 디바이스 풀에 하나 이상의 디바이스를 추가할 수 있습니다.

> **Note**
>
> 동일한 풀에 있는 모든 디바이스는 기술 사양과 SKU가 동일해야 합니다.

다른 테스트 그룹에 대한 소스 코드의 병렬 빌드를 활성화하기 위해 FreeRTOS용 IDT는 FreeRTOS용 IDT 압축 해제 폴더 내부의 결과 폴더에 소스 코드를 복사합니다. 빌드 또는 플래시 명령에 있는 소스 코드 경로는 testdata.sourcePath 또는 sdkPath 변수를 사용하여 참조해야 합니다. FreeRTOS용 IDT는 이 변수를 복사된 소스 코드의 임시 경로로 바꿉니다. 자세한 내용은 FreeRTOS용 IDT 변수 섹션을 참조하세요.

다음은 여러 디바이스가 있는 디바이스 풀을 생성하는 데 사용되는 예제 device.json 파일입니다.

```
[
    {
        "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 | Both"
                    }
                ]
            },
            {
                "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"
            }
          ]
        }
      ]
  }
]
```

device.json 파일에서 사용되는 속성은 다음과 같습니다.

id

디바이스 풀을 고유하게 식별하는 사용자 정의 영숫자 ID입니다. 풀에 속한 디바이스는 동일한 유형이어야 합니다. 테스트 제품군을 실행할 때 풀에 있는 디바이스는 워크로드를 병렬화하는 데 사용됩니다.

sku

테스트하는 보드를 고유하게 식별하는 영숫자 값입니다. SKU는 적격 보드를 추적하는 데 사용됩니다.

> **Note**
>
> AWS Partner Device Catalog에 보드를 나열하려면 여기에서 지정하는 SKU가 나열 프로세스에서 사용하는 SKU와 일치해야 합니다.

`features`

디바이스의 지원되는 기능이 포함된 배열입니다.이 정보를 AWS IoT Device Tester 사용하여 실행할 검증 테스트를 선택합니다.

지원되는 값은 다음과 같습니다.

`TCP/IP`

보드가 TCP/IP 스택을 지원하는지 여부 및 보드가 온칩에서 지원되는지(MCU) 또는 다른 모듈로 오프로드되는지를 나타냅니다. 검증에는 TCP/IP가 필요합니다.

`WIFI`

보드에 Wi-Fi 기능이 있는지 여부를 나타냅니다. `Cellular`가 `Yes`로 설정된 경우 `No`로 설정되어야 합니다.

`Cellular`

보드에 셀룰러 기능이 있는지 여부를 나타냅니다. `WIFI`가 `Yes`로 설정된 경우 `No`로 설정되어야 합니다. 이 기능을 로 설정하면 `Yes` FullSecureSockets 테스트가 AWS t2.micro EC2 인스턴스를 사용하여 실행되며 이로 인해 계정에 추가 비용이 발생할 수 있습니다. 자세한 설명은 Amazon EC2 요금을 참조하세요.

`TLS`

보드가 TLS를 지원하는지 여부를 나타냅니다. 검증에는 TLS가 필요합니다.

`PKCS11`

보드가 지원하는 퍼블릭 키 암호화 알고리즘을 나타냅니다. 검증에는 PKCS11이 필요합니다. 지원되는 값은 `ECC`, `RSA`, `Both`, `No`입니다. `Both`는 `ECC` 및 `RSA` 알고리즘을 모두 지원하는 보드를 가리킵니다.

`KeyProvisioning`

신뢰할 수 있는 X.509 클라이언트 인증서를 보드에 작성하는 방법을 나타냅니다. 유효 값은 `Import`, `Onboard` 및 `No`입니다. 검증에는 키 프로비저닝이 필요합니다.

- 보드에서 프라이빗 키 가져오기를 허용하는 경우에는 `Import`을 사용합니다. IDT는 프라이빗 키를 생성하고 이를 FreeRTOS 소스 코드에 빌드합니다.
- 보드가 온보드 프라이빗 키 생성을 지원하는 경우(예: 디바이스에 보안 요소가 있거나 자체적인 디바이스 키 페어 및 인증서를 생성하려는 경우)에는 `Onboard`을 사용합니다. 각 디바이스 섹션에 `secureElementConfig` 요소를 추가하고 퍼블릭 키 파일에 대한 절대 경로를 `publicKeyAsciiHexFilePath` 필드에 입력합니다.

- 보드가 키 프로비저닝을 지원하지 않는 경우 No를 사용합니다.

`OTA`

보드가 무선(Over-the-Air) 업데이트 기능을 지원하는지 여부를 나타냅니다. `OtaDataPlaneProtocol` 속성은 디바이스가 지원하는 OTA 데이터 영역 프로토콜을 나타냅니다. 디바이스에서 OTA 기능을 지원하지 않으면 이 속성은 무시됩니다. `"Both"`를 선택하면 MQTT, HTTP 둘 다와 혼합된 테스트가 실행되어 OTA 테스트 실행 시간이 길어집니다.

`BLE`

보드가 Bluetooth Low Energy(BLE)를 지원하는지 여부를 나타냅니다.

`devices.id`

테스트 대상 장치의 고유한 사용자 정의 식별자입니다.

`devices.connectivity.protocol`

이러한 장치와 통신하는 데 사용되는 통신 프로토콜입니다. 지원되는 값: `uart`.

`devices.connectivity.serialPort`

테스트할 디바이스에 연결하는 데 사용되는 호스트 컴퓨터의 직렬 포트입니다.

`devices.secureElementConfig.PublicKeyAsciiHexFilePath`

온보드 프라이빗 키에서 추출한 16진수 바이트 퍼블릭 키를 포함하는 파일에 대한 절대 경로입니다.

형식의 예:

```
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
```

퍼블릭 키가 .der 형식인 경우 퍼블릭 키를 직접 16진수 인코딩하여 16진수 파일을 생성할 수 있습니다.

.der 퍼블릭 키를 사용하여 16진수 파일을 생성하기 위한 명령 예제:

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

퍼블릭 키가.pem 형식인 경우 base64 인코딩 부분을 추출하여 바이너리 형식으로 디코딩한 다음 16진수 인코딩하여 16진수 파일을 생성할 수 있습니다.

예를 들어, 다음 명령을 사용하여 .pem 퍼블릭 키용 16진수 파일을 생성할 수 있습니다.

1. 키의 base64 인코딩 부분만 빼내어(헤더 및 푸터를 제거) 파일에 저장하고(예를 들어 base64key로 이름을 지정) 다음 명령을 실행하여 .der 형식으로 변환합니다.

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

2. xxd 명령을 실행하여 이를 16진수 형식으로 변환합니다.

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

`devices.secureElementConfig.SecureElementSerialNumber`

(선택 사항) 보안 요소의 일련 번호입니다. FreeRTOS 데모/테스트 프로젝트를 실행할 때 디바이스 퍼블릭 키와 더불어 일련 번호가 출력되는 경우 이 필드를 제공합니다.

`devices.secureElementConfig.preProvisioned`

(선택 사항) 디바이스에 잠긴 보안 인증 정보를 가진 사전 프로비저닝된 보안 요소가 있고 객체를 가져오거나 생성하거나 삭제할 수 없는 경우 '예'로 설정합니다. 이 구성은 `features`에서 `KeyProvisioning`이 '온보드'로 설정되고 `PKCS11`이 'ECC'로 설정된 경우에만 적용됩니다.

`identifiers`

(선택 사항) 임의 이름-값 페어의 배열입니다. 다음 단원에서 설명하는 빌드 및 플래시 명령에 이러한 값을 사용할 수 있습니다.

빌드, 플래시 및 테스트 설정 구성

FreeRTOS용 IDT를 사용하여 보드에서 자동으로 테스트를 빌드 및 플래시하는 경우, 하드웨어에 대한 빌드 및 플래시 명령을 실행하도록 IDT를 구성해야 합니다. 빌드 및 플래시 명령 설정은 `config` 폴더에 있는 `userdata.json` 템플릿 파일에서 구성됩니다.

디바이스 테스트를 위한 설정 구성

빌드, 플래시 및 테스트 설정은 `configs/userdata.json` 파일에서 수행됩니다. 클라이언트 및 서버 인증서와 키를 모두 `customPath`에서 로드하는 에코 서버 구성을 지원합니다. 자세한 내용은

FreeRTOS 이식 가이드의 에코 서버 설정을 참조하세요. 다음 JSON 예제에서는 여러 디바이스를 테스트하도록 FreeRTOS용 IDT를 구성하는 방법을 보여 줍니다.

```
{
    "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": [
            "/absolute-path-to/build-parallel.sh {{testData.sourcePath}}
 {{enableTests}}"
        ]
    },
    "flashTool": {
        "name": "your-flash-tool-name",
        "version": "your-flash-tool-version",
        "command": [
            "/absolute-path-to/flash-parallel.sh {{testData.sourcePath}}
 {{device.connectivity.serialPort}} {{buildImageName}}"
        ],
        "buildImageInfo" : {
            "testsImageName": "tests-image-name",
            "demosImageName": "demos-image-name"
        }
    },
    "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-
path/aws_tests/config_files/core_pkcs11_config.h"
            },
            {
                "configName": "pkcs11TestConfig",
                "filePath": "{{testData.sourcePath}}/vendors/vendor-name/boards/board-
path/aws_tests/config_files/iot_test_pkcs11_config.h"
            }
        ],
        "optional": [
            {
                "configName": "otaAgentTestsConfig",
                "filePath": "{{testData.sourcePath}}/vendors/vendor-name/boards/board-
path/aws_tests/config_files/aws_ota_agent_config.h"
```

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

다음 목록에서는 userdata.json에서 사용되는 속성을 나열합니다.

sourcePath

이식된 Echo Server 소스 코드의 루트에 대한 경로입니다. SDK를 사용하는 병렬 테스트의 경우 {{userData.sdkConfiguration.path}} 자리 표시자를 사용하여 sourcePath를 설정할 수 있습니다. 예시:

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

vendorPath

공급업체별 FreeRTOS 코드에 대한 경로입니다. 직렬 테스트의 경우 vendorPath를 절대 경로로 설정할 수 있습니다. 예시:

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

병렬 테스트의 경우 vendorPath를 {{testData.sourcePath}} 자리 표시자를 사용하여 설정할 수 있습니다. 예시:

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

vendorPath 변수는 SDK 없이 실행할 때만 필요하며, 그렇지 않으면 제거할 수 있습니다.

> **Note**
>
> SDK를 사용하지 않고 테스트를 병렬로 실행하는 경우 vendorPath, buildTool, flashTool 필드에 {{testData.sourcePath}} 자리 표시자를 사용해야 합니다. 단일 디바이스로 테스트를 실행하는 경우 vendorPath, buildTool, flashTool 필드에 절

대 경로를 사용해야 합니다. SDK를 사용하여 실행하는 경우 `sourcePath`, `buildTool` 및 `flashTool` 명령에 `{{sdkPath}}` 자리 표시자를 사용해야 합니다.

`sdkConfiguration`

이식에 필요한 것 이상으로 파일 및 폴더 구조를 수정하여 FreeRTOS를 검증하려면 이 블록에서 SDK 정보를 구성해야 합니다. SDK 내에서 이식된 FreeRTOS를 사용할 수 없다면 이 블록을 완전히 생략해야 합니다.

`sdkConfiguration.name`

FreeRTOS와 함께 사용 중인 SDK의 이름입니다. SDK를 사용하지 않는 경우 전체 `sdkConfiguration` 블록을 생략해야 합니다.

`sdkConfiguration.version`

FreeRTOS와 함께 사용 중인 SDK의 버전입니다. SDK를 사용하지 않는 경우 전체 `sdkConfiguration` 블록을 생략해야 합니다.

`sdkConfiguration.path`

FreeRTOS 코드가 포함된 SDK 디렉터리의 절대 경로입니다. SDK를 사용하지 않는 경우 전체 `sdkConfiguration` 블록을 생략해야 합니다.

`buildTool`

소스 코드를 빌드하기 위한 명령이 포함된 빌드 스크립트(.bat 또는 .sh)의 전체 경로입니다. 빌드 명령의 소스 코드 경로에 대한 모든 참조는 AWS IoT Device Tester 변수로 대체되고 SDK 경로에 대한 `{{testdata.sourcePath}}` 참조는 로 대체되어야 합니다`{{sdkPath}}`.

`buildImageInfo`

`testsImageName`

`freertos-source``/tests` 폴더에서 테스트를 빌드할 때 빌드 명령으로 인해 생성되는 파일의 이름입니다.

`demosImageName`

`freertos-source``/demos` 폴더에서 테스트를 빌드할 때 빌드 명령으로 인해 생성되는 파일의 이름입니다.

`flashTool`

디바이스에 대한 플래시 명령이 포함된 플래시 스크립트(.sh 또는 .bat)의 전체 경로입니다. 플래시 명령에서 소스 코드 경로에 대한 모든 참조는 FreeRTOS용 IDT 변수

{{testdata.sourcePath}}로 바꿔야 하고 SDK 경로에 대한 모든 참조는 FreeRTOS용 IDT 변수 {{sdkPath}}로 바꿔야 합니다.

`clientWifiConfig`

클라이언트 Wi-Fi 구성입니다. Wi-Fi 라이브러리 테스트를 수행하려면 MCU 보드를 두 개의 액세스 포인트에 연결해야 합니다. (두 액세스 포인트는 동일할 수 있습니다.). 이 속성은 첫 번째 액세스 포인트에 대한 Wi-Fi 설정을 구성합니다. 일부 Wi-Fi 테스트 사례에서는 액세스 포인트가 보안을 갖추고 있고 열리지 않아야 합니다. 두 액세스 포인트가 모두 IDT를 실행하는 호스트 컴퓨터와 동일한 서브넷에 있는지 확인하세요.

`wifi_ssid`

Wi-Fi SSID입니다.

`wifi_password`

Wi-Fi 암호입니다.

`wifiSecurityType`

사용되는 Wi-Fi 보안 유형입니다. 값 중 하나는 다음과 같습니다.

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

> **Note**
>
> 보드가 Wi-Fi를 지원하지 않는 경우에도 `clientWifiConfig` 섹션을 `device.json` 파일에 포함시켜야 하지만 이러한 속성의 값을 생략할 수 있습니다.

`testWifiConfig`

테스트 Wi-Fi 구성입니다. Wi-Fi 라이브러리 테스트를 수행하려면 MCU 보드를 두 개의 액세스 포인트에 연결해야 합니다. (두 액세스 포인트는 동일할 수 있습니다.). 이 속성은 두 번째 액세스 포인트에 대한 Wi-Fi 설정을 구성합니다. 일부 Wi-Fi 테스트 사례에서는 액세스 포인트가 보안을 갖추고

있고 열리지 않아야 합니다. 두 액세스 포인트가 모두 IDT를 실행하는 호스트 컴퓨터와 동일한 서브넷에 있는지 확인하세요.

`wifiSSID`

Wi-Fi SSID입니다.

`wifiPassword`

Wi-Fi 암호입니다.

`wifiSecurityType`

사용되는 Wi-Fi 보안 유형입니다. 값 중 하나는 다음과 같습니다.

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

> **Note**
>
> 보드가 Wi-Fi를 지원하지 않는 경우에도 `testWifiConfig` 섹션을 `device.json` 파일에 포함시켜야 하지만 이러한 속성의 값을 생략할 수 있습니다.

`echoServerCertificateConfiguration`

보안 소켓 테스트를 위한 구성 가능한 에코 서버 인증서 생성 자리 표시자입니다. 이 필드는 필수 항목입니다.

`certificateGenerationMethod`

서버 인증서를 자동으로 생성할지 수동으로 제공할지를 지정합니다.

`customPath`

`certificateGenerationMethod`가 '사용자 지정'인 경우 `certificatePath` 및 `privateKeyPath`는 필수입니다.

`certificatePath`

서버 인증서의 파일 경로를 지정합니다.

`privateKeyPath`

프라이빗 키의 파일 경로를 지정합니다.

`eccCurveFormat`

보드에서 지원하는 곡선 형식을 지정합니다. `device.json`에서 `PKCS11`이 'ecc'로 설정된 경우 필수입니다. 유효한 값은 'P224', 'P256', 'P384' 또는 'P521'입니다.

`echoServerConfiguration`

WiFi 및 SecureSocket 테스트를 위한 구성 가능한 에코 서버 포트입니다. 이 필드는 선택 사항입니다.

`securePortForSecureSocket`

SecureSocket 테스트를 위해 TLS를 사용하여 에코 서버를 설정하는 데 사용되는 포트입니다. 기본값은 33333입니다. 구성된 포트가 방화벽이나 회사 네트워크에 의해 차단되지 않는지 확인합니다.

`insecurePortForSecureSocket`

SecureSocket 테스트를 위해 TLS를 사용하지 않고 에코 서버를 설정하는 데 사용되는 포트입니다. 테스트에 사용되는 기본값은 33334입니다. 구성된 포트가 방화벽이나 회사 네트워크에 의해 차단되지 않는지 확인합니다.

`insecurePortForWiFi`

WiFi 테스트를 위해 TLS를 사용하지 않고 에코 서버를 설정하는 데 사용되는 포트입니다. 테스트에 사용되는 기본값은 33335입니다. 구성된 포트가 방화벽이나 회사 네트워크에 의해 차단되지 않는지 확인합니다.

`otaConfiguration`

OTA 구성입니다. [선택 사항]

`otaFirmwareFilePath`

빌드 후 생성된 OTA 이미지의 전체 경로입니다. 예를 들어 `{{testData.sourcePath}}/`*`relative-path/to/ota/image/from/source/root`*입니다.

`deviceFirmwareFileName`

MCU 디바이스에 OTA 펌웨어가 위치하는 전체 파일 경로입니다. 일부 디바이스는 이 필드를 사용하지 않지만 그래도 값을 입력해야 합니다.

`otaDemoConfigFilePath`

`aws_demo_config.h`에 대한 전체 경로이며 *`afr-source`*`/vendors/vendor/boards/board/aws_demos/config_files/`에 있습니다. 이러한 파일은 FreeRTOS에서 제공한 이식 코드 템플릿에 포함되어 있습니다.

`codeSigningConfiguration`

코드 서명 구성입니다.

`signingMethod`

코드 서명 방법입니다. 가능한 값은 `AWS` 또는 `Custom`입니다.

> **Note**
>
> 베이징 및 닝샤 리전의 경우 `Custom`을 사용합니다. `AWS` 코드 서명은 이러한 리전에서 지원되지 않습니다.

`signerHashingAlgorithm`

디바이스에서 지원되는 해싱 알고리즘입니다. 가능한 값은 `SHA1` 또는 `SHA256`입니다.

`signerSigningAlgorithm`

디바이스에서 지원되는 서명 알고리즘입니다. 가능한 값은 `RSA` 또는 `ECDSA`입니다.

`signerCertificate`

OTA에 사용되는 신뢰할 수 있는 인증서입니다.

AWS 코드 서명 방법의 경우에 업로드된 신뢰할 수 있는 인증서의 Amazon 리소스 이름(ARN)을 사용합니다 AWS Certificate Manager.

사용자 정의 코드 서명 방법의 경우 서명자 인증서 파일의 절대 경로를 사용합니다.

신뢰할 수 있는 인증서 생성에 대한 자세한 내용은 코드 서명 인증서 생성 단원을 참조하십시오.

`signerCertificateFileName`

디바이스의 코드 서명 인증서의 위치입니다.

`compileSignerCertificate`

코드 서명자 서명 확인 인증서가 프로비저닝되거나 플래시되지 않은 `true` 경우 로 설정되므로 프로젝트로 컴파일해야 합니다. 신뢰할 수 있는 인증서를 AWS IoT Device Tester 가져와서 로 컴파일합니다`aws_codesigner_certifiate.h`.

`untrustedSignerCertificateArn`

ACM에 업로드된 코드 서명 인증서의 ARN입니다.

`signerPlatform`

AWS Code Signer가 OTA 업데이트 작업을 생성하는 동안 사용하는 서명 및 해싱 알고리즘입니다. 현재 이 필드에 가능한 값은 `AmazonFreeRTOS-TI-CC3220SF` 및 `AmazonFreeRTOS-Default`입니다.

- `SHA1` 및 `RSA`인 경우 `AmazonFreeRTOS-TI-CC3220SF`를 선택합니다.
- `SHA256` 및 `ECDSA`인 경우 `AmazonFreeRTOS-Default`를 선택합니다.

구성에 `SHA256 | RSA` 또는 `SHA1 | ECDSA`가 필요한 경우 추가 지원을 요청하십시오.

`signingMethod`에 `Custom`을 선택한 경우 `signCommand`를 구성합니다

`signCommand`

사용자 정의 코드 서명을 수행하는 데 사용되는 명령입니다. `/configs/script_templates` 디렉터리에서 템플릿을 찾을 수 있습니다.

명령에 두 자리 표시자 `{{inputImageFilePath}}` 및 `{{outputSignatureFilePath}}`가 필요합니다. `{{inputImageFilePath}}`는 서명하기 위해 IDT에서 만든 이미지의 파일 경로입니다. `{{outputSignatureFilePath}}`는 스크립트에서 생성하는 서명의 파일 경로입니다.

`otaDemoConfigFilePath`

`aws_demo_config.h`에 대한 전체 경로이며, *`afr-source`*`/vendors/vendor/boards/board/aws_demos/config_files/` 내에 있습니다. 이러한 파일은 FreeRTOS에서 제공한 이식 코드 템플릿에 포함되어 있습니다.

`cmakeConfiguration`

CMake 구성[선택 사항]

> **Note**
>
> CMake 테스트 사례를 실행하려면 보드 이름, 공급업체 이름 및 `frToolchainPath` 또는 `compilerName`을 제공해야 합니다. CMake 도구 체인에 대한 사용자 지정 경로가 있다면 `cmakeToolchainPath`를 제공할 수도 있습니다.

`boardName`

테스트 중인 보드의 이름입니다. 보드 이름은 *path/to/afr/source/code*/vendors/*vendor*/boards/*board* 아래의 폴더 이름과 같아야 합니다.

`vendorName`

테스트 중인 보드의 공급업체 이름입니다. 공급업체는 *path/to/afr/source/code*/vendors/*vendor* 아래의 폴더 이름과 같아야 합니다.

`compilerName`

컴파일러 이름입니다.

`frToolchainPath`

컴파일러 도구 체인의 정규화된 경로입니다.

`cmakeToolchainPath`

CMake 도구 체인의 정규화된 경로입니다. 이 필드는 선택 사항입니다.

`freertosFileConfiguration`

테스트를 실행하기 전에 IDT가 수정하는 FreeRTOS 파일의 구성입니다.

`required`

이 섹션에서는 구성 파일을 이동한 필수 테스트(예: PKCS11, TLS 등)를 지정합니다.

`configName`

구성하는 테스트의 이름입니다.

`filePath`

freertos 리포지토리 내에서 구성 파일의 절대 경로입니다. `{{testData.sourcePath}}` 변수를 사용하여 경로를 정의합니다.

`optional`

이 섹션에서는 구성 파일을 이동한 선택적 테스트(예: OTA, WiFi 등)를 지정합니다.

`configName`

구성하는 테스트의 이름입니다.

`filePath`

`freertos` 리포지토리 내에서 구성 파일의 절대 경로입니다. `{{testData.sourcePath}}` 변수를 사용하여 경로를 정의합니다.

> **Note**
>
> CMake 테스트 사례를 실행하려면 보드 이름, 공급업체 이름 및 `afrToolchainPath` 또는 `compilerName`을 제공해야 합니다. CMake 도구 체인에 대한 사용자 지정 경로가 있다면 `cmakeToolchainPath`를 제공할 수도 있습니다.

FreeRTOS용 IDT 변수

코드를 빌드하고 디바이스를 플래시하는 명령을 실행하려면 연결 또는 디바이스에 대한 기타 정보가 필요할 수 있습니다.를 AWS IoT Device Tester 사용하면 플래시에서 디바이스 정보를 참조하고 JsonPath를 사용하여 명령을 빌드할 수 있습니다. 단순 JsonPath 표현식을 사용하여 `device.json` 파일에 지정된 필수 정보를 가져올 수 있습니다.

경로 변수

FreeRTOS용 IDT는 명령줄과 구성 파일에 사용할 수 있는 다음 경로 변수를 정의합니다.

`{{testData.sourcePath}}`

소스 코드 경로로 확장됩니다. 이 변수를 사용하는 경우 플래시 명령과 빌드 명령에서 이 변수를 모두 사용해야 합니다.

`{{sdkPath}}`

빌드 및 플래시 명령에서 사용할 경우 `userData.sdkConfiguration.path`의 값으로 확장됩니다.

`{{device.connectivity.serialPort}}`

직렬 포트로 확장됩니다.

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

디바이스의 일련 번호로 확장됩니다.

`{{enableTests}}`

빌드가 테스트용(값 1)인지 데모용(값 0)인지를 나타내는 정수 값입니다.

`{{buildImageName}}`

빌드 명령에 따라 빌드되는 이미지의 파일 이름입니다.

`{{otaCodeSignerPemFile}}`

OTA 코드 서명자용 PEM 파일입니다.

Bluetooth Low Energy 테스트 실행

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

이 섹션에서는 FreeRTOS AWS IoT Device Tester 용를 사용하여 Bluetooth 테스트를 설정하고 실행하는 방법을 설명합니다. 코어 검증에는 Bluetooth 테스트가 필요하지 않습니다. FreeRTOS Bluetooth를 지원하는 디바이스를 테스트하지 않으려는 경우 이 설정을 건너뛸 수 있으며 device.json의 BLE 기능을 No로 설정된 상태로 그대로 두어야 합니다.

사전 조건

- 처음으로 마이크로 컨트롤러 보드의 테스트 준비의 지침을 따르세요.
- Raspberry Pi 4B 또는 3B+. (Raspberry Pi BLE 컴패니언 애플리케이션을 실행하는 데 필요함)
- Raspberry Pi 소프트웨어용 마이크로 SD 카드 및 SD 카드 어댑터.

Raspberry Pi 설정

DUT(테스트 대상 디바이스)의 BLE 기능을 테스트하려면 Raspberry Pi Model 4B 또는 3B+가 있어야 합니다.

BLE 테스트를 실행하도록 Raspberry Pi를 설정하려면

1. 테스트를 수행하는 데 필요한 소프트웨어가 포함된 사용자 지정 Yocto 이미지 중 하나를 다운로드합니다.

   - Raspberry Pi 4B용 이미지
   - Raspberry Pi 3B+용 이미지

   > **Note**
   >
   > Yocto 이미지는 FreeRTOS AWS IoT Device Tester 용를 사용한 테스트에만 사용해야 하며 다른 용도로는 사용할 수 없습니다.

2. yocto 이미지를 Raspberry Pi용 SD 카드로 플래시합니다.

   - Etcher와 같은 SD 카드 쓰기 도구를 사용하여 다운로드된 *image-name*`.rpi-sd.img` 파일을 SD 카드로 플래시합니다. 운영 체제 이미지가 크기 때문에 이 단계는 시간이 약간 걸릴 수 있습니다. 그런 다음 컴퓨터에서 SD 카드를 꺼내고 Raspberry Pi에 microSD 카드를 삽입합니다.

3. Raspberry Pi를 구성합니다.

   a. 처음으로 부팅하는 경우 Raspberry Pi를 모니터, 키보드 및 마우스에 연결하는 것이 좋습니다.

   b. Raspberry Pi를 마이크로 USB 전원에 연결합니다.

   c. 기본 자격 증명을 사용하여 로그인합니다. 사용자 ID에 **`root`**를 입력합니다. 암호에 **`idtafr`**을 입력합니다.

   d. 이더넷 또는 Wi-Fi 연결을 사용하여 Raspberry Pi를 네트워크에 연결합니다.

      i. Wi-Fi를 통해 Raspberry Pi를 연결하려면 Raspberry Pi에서 `/etc/wpa_supplicant.conf`를 열고 Wi-Fi 자격 증명을 `Network` 구성에 추가합니다.

```
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. `ifup wlan0`을 실행하여 Wi-Fi 연결을 시작합니다. Wi-Fi 네트워크에 연결하려면 1분 정도 걸릴 수 있습니다.

e. 이더넷 연결의 경우 `ifconfig eth0`을 실행합니다. Wi-Fi 연결의 경우 `ifconfig wlan0`을 실행합니다. IP 주소를 기록해 둡니다. 이 주소는 명령 출력에 `inet addr`로 나타납니다. 이 절차의 뒷 부분에서 IP 주소가 필요합니다.

f. (선택 사항) 테스트는 yocto 이미지에 대한 기본 자격 증명을 사용하여 SSH를 통해 Raspberry Pi에서 명령을 실행합니다. 보안을 강화하기 위해 SSH에 대한 퍼블릭 키 인증을 설정하고 암호 기반 SSH를 비활성화하는 것이 좋습니다.

i. OpenSSL `ssh-keygen` 명령을 사용하여 SSH 키를 생성합니다. 호스트 컴퓨터에 이미 SSK 키 페어가 있는 경우 AWS IoT Device Tester FreeRTOS가 Raspberry Pi에 로그인할 수 있도록 새 키 페어를 생성하는 것이 좋습니다.

> **Note**
>
> Windows에는 SSH 클라이언트가 설치되어 있지 않습니다. Windows에 SSH 클라이언트를 설치하는 방법에 대한 자세한 내용은 SSH 소프트웨어 다운로드를 참조하십시오.

ii. `ssh-keygen` 명령은 키 페어 저장 이름과 경로를 입력하라는 메시지를 표시합니다. 기본적으로 키 페어 파일은 `id_rsa`(프라이빗 키) 및 `id_rsa.pub`(퍼블릭 키)로 이름 지정됩니다. macOS와 Linux에서 이러한 파일의 기본 위치는 `~/.ssh/`입니다. Windows에서 기본 위치는 `C:\Users\`*`user-name`*입니다.

iii. 키 구문을 묻는 메시지가 표시되면 Enter를 눌러 계속합니다.

iv. FreeRTOS AWS IoT Device Tester 용가 디바이스에 로그인할 수 있도록 Raspberry Pi에 SSH 키를 추가하려면 호스트 컴퓨터의 `ssh-copy-id` 명령을 사용합니다. 이 명령은 Raspberry Pi에서 `~/.ssh/authorized_keys` 파일에 퍼블릭 키를 추가합니다.

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

v. 암호를 묻는 화면이 나타나면 **`idtafr`**을 입력합니다. 이 값은 yocto 이미지의 기본 암호입니다.

> **Note**
>
> ssh-copy-id 명령은 퍼블릭 키에 id_rsa.pub라는 이름이 지정된다고 가정합니다. macOS와 Linux에서 기본 위치는 ~/.ssh/입니다. Windows에서 기본 위치는 C:\Users*user-name*\.ssh입니다. 퍼블릭 키의 이름을 다르게 지정하거나 다른 위치에 저장한 경우, ssh-copy-id에 -i 옵션을 사용하여 SSH 퍼블릭 키의 정규화된 경로를 지정해야 합니다(예: ssh-copy-id -i ~/my/path/myKey.pub). SSH 키 생성 및 퍼블릭 키 복사에 대한 자세한 내용은 SSH-COPY-ID를 참조하십시오.

vi. 퍼블릭 키 인증이 작동하는지 테스트하려면 ssh -i */my/path/myKey* root@*raspberry-pi-device-ip*를 실행합니다.

암호를 묻는 화면이 나타나지 않으면 퍼블릭 키 인증이 작동하는 것입니다.

vii. 퍼블릭 키를 사용하여 Raspberry Pi에 로그인할 수 있는지 확인한 다음, 암호 기반 SSH를 비활성화합니다.

A. Raspberry Pi에서 /etc/ssh/sshd_config 파일을 편집합니다.

B. PasswordAuthentication 속성을 no로 설정합니다.

C. sshd_config 파일을 저장하고 닫습니다.

D. /etc/init.d/sshd reload를 실행하여 SSH 서버를 다시 로드합니다.

g. resource.json 파일을 생성합니다.

i. AWS IoT Device Tester를 추출한 디렉터리에서 라는 파일을 생성합니다resource.json.

ii. Raspberry Pi에 대한 다음 정보를 파일에 추가하여 *rasp-pi-ip-address*를 Raspberry Pi의 IP 주소로 바꿉니다.

```
[
    {
        "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. (선택 사항) SSH에 퍼블릭 키 인증을 사용하도록 선택하지 않은 경우 `resource.json` 파일의 `connectivity` 섹션에 다음을 추가합니다.

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

iv. (선택 사항) SSH에 퍼블릭 키 인증을 사용하도록 선택한 경우 `resource.json` 파일의 `connectivity` 섹션에 다음을 추가합니다.

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

FreeRTOS 파일 설정

`device.json` 파일에서 BLE 기능을 Yes로 설정합니다. Bluetooth 테스트가 사용 가능하기 전에 `device.json` 파일로 시작하는 경우 BLE용 기능을 `features` 배열에 추가해야 합니다.

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

BLE 테스트 실행

`device.json`에서 BLE 기능을 활성화한 후 그룹 ID를 지정하지 않고 `devicetester_`*`[linux | mac | win_x86-64]`* `run-suite`를 실행하면 BLE 테스트가 실행됩니다.

BLE 테스트를 별도로 실행하려면 BLE의 그룹 ID(`devicetester_`*`[linux | mac | win_x86-64]`* `run-suite --userdata` *`path-to-userdata`*`/userdata.json --group-id FullBLE`)를 지정할 수 있습니다.

가장 신뢰성 있는 성능을 얻으려면 Raspberry Pi를 DUT(테스트 대상 디바이스)와 가깝게 배치하십시오.

BLE 테스트 문제 해결

처음으로 마이크로 컨트롤러 보드의 테스트 준비의 단계를 따랐는지 확인합니다. BLE 이외의 다른 테스트가 실패하면 문제가 Bluetooth 구성으로 인한 것이 아닐 가능성이 높습니다.

FreeRTOS 검증 테스트 제품군 실행

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

FreeRTOS AWS IoT Device Tester 용를 사용하여 FreeRTOS용 IDT와 상호 작용합니다. 다음 명령줄에서는 디바이스 풀(동일한 디바이스의 집합)에 대한 검증을 실행하는 방법을 보여 줍니다.

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
```

장치의 풀에 대해 테스트 제품군을 실행합니다. `userdata.json` 파일은 *devicetester_extract_location*`/devicetester_afreertos_`*[win|mac|linux]*`/configs/` 디렉터리에 위치해야 합니다.

> **Note**
>
> Windows에서 FreeRTOS용 IDT를 실행하는 경우 슬래시(/)를 사용하여 `userdata.json` 파일의 경로를 지정합니다.

다음 명령을 사용하여 특정 테스트 그룹을 실행합니다.

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

단일 디바이스 풀에 대해 단일 테스트 집합을 실행하는 경우(즉, `device.json` 파일에 디바이스 풀이 하나만 정의되어 있는 경우) `suite-id` 및 `pool-id`는 선택 사항입니다.

다음 명령을 사용하여 테스트 그룹의 특정 케이스를 실행합니다.

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

```
    --test-id test-id
```

`list-test-cases` 명령을 사용하여 테스트 그룹의 테스트 케이스를 나열할 수 있습니다.

FreeRTOS용 IDT 명령줄 옵션

group-id

(선택 사항) 쉼표로 구분된 목록으로 실행할 테스트 그룹입니다. 지정하지 않으면 IDT는 테스트 제품군의 모든 테스트 그룹을 실행합니다.

pool-id

(선택 사항) 테스트할 디바이스 풀입니다. 이 작업은 `device.json`에서 여러 디바이스 풀을 정의하는 경우에 필요합니다. 디바이스 풀이 하나만 있는 경우 이 옵션을 생략할 수 있습니다.

suite-id

(선택 사항) 실행할 테스트 제품군 버전입니다. 지정하지 않으면 IDT는 시스템의 테스트 디렉터리에서 최신 버전을 사용합니다.

> **Note**
>
> IDT v3.0.0부터 IDT는 최신 테스트 제품군을 온라인으로 확인합니다. 자세한 내용은 테스트 제품군 버전 단원을 참조하십시오.

test-id

(선택 사항) 쉼표로 구분된 목록으로 실행할 테스트입니다. 지정된 경우 `group-id`은 단일 그룹을 지정해야 합니다.

Example

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

update-idt

(선택 사항) 이 파라미터를 설정하지 않으면 최신 IDT 버전을 사용할 수 있는 경우 IDT를 업데이트하라는 메시지가 표시됩니다. 이 파라미터를 Y로 설정할 경우 IDT는 새 버전을 사용할 수 있

음을 감지하면 테스트 실행을 중지합니다. 이 파라미터를 N으로 설정할 경우 IDT는 테스트 실행을 계속합니다.

update-managed-policy

(선택 사항) 이 파라미터를 사용하지 않으면 IDT가 관리형 정책이 최신 상태가 아님을 감지할 경우 관리형 정책을 업데이트하라는 메시지가 표시됩니다. 이 파라미터를 Y로 설정할 경우 IDT는 관리형 정책이 최신 상태가 아님을 감지하면 테스트 실행을 중단합니다. 이 파라미터를 N으로 설정할 경우 IDT는 테스트 실행을 계속합니다.

upgrade-test-suite

(선택 사항) 사용하고 있지 않지만 최신 테스트 제품군 버전이 제공되는 경우에는 다운로드하라는 메시지가 표시됩니다. 프롬프트를 숨기려면 항상 최신 테스트 제품군을 다운로드하도록 y을 지정하거나, 시스템에서 지정된 테스트 제품군 또는 최신 버전을 사용하도록 n를 지정합니다.

Example

항상 최신 테스트 제품군을 다운로드하여 사용하려면 다음 명령을 사용합니다.

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

시스템에서 최신 테스트 제품군은 다음 명령을 사용합니다.

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

h

도움말 옵션을 사용하여 `run-suite` 옵션에 대해 자세히 알아봅니다.

Example

```
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
```

userdata.json 파일은 *devicetester_extract_location*/devicetester_afreertos_*[win|mac|linux]*/configs/ 디렉터리에 위치해야 합니다.

> **Note**
>
> Windows에서 FreeRTOS용 IDT를 실행하는 경우 슬래시(/)를 사용하여 userdata.json 파일의 경로를 지정합니다.

다음 명령을 사용하여 특정 테스트 그룹을 실행합니다.

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

단일 디바이스 풀에 대해 단일 테스트 세트를 실행하는 경우(즉, device.json 파일에 정의된 디바이스 풀이 하나만 있는 경우) suite-id 및 pool-id는 선택 사항입니다.

FreeRTOS용 IDT 명령줄 옵션

group-id

(선택 사항) 테스트 그룹을 지정합니다.

pool-id

테스트할 디바이스 풀을 지정합니다. 디바이스 풀이 하나만 있는 경우 이 옵션을 생략할 수 있습니다.

suite-id

(선택 사항) 실행할 테스트 제품군을 지정합니다.

FreeRTOS용 IDT 명령

FreeRTOS용 IDT 명령은 다음과 같은 작업을 지원합니다.

IDT v3.0.0 and later

`help`

지정된 명령에 대한 정보를 나열합니다.

`list-groups`

지정된 제품군에 있는 그룹을 나열합니다.

`list-suites`

사용 가능한 제품군을 나열합니다.

`list-supported-products`

지원되는 제품 및 테스트 제품군 버전을 나열합니다.

`list-supported-versions`

현재 IDT 버전에서 지원하는 FreeRTOS 및 테스트 제품군 버전을 나열합니다.

`list-test-cases`

지정된 그룹의 테스트 케이스를 나열합니다.

`run-suite`

장치의 풀에 대해 테스트 제품군을 실행합니다.

`--suite-id` 옵션을 사용하여 테스트 제품군 버전을 지정하거나, 시스템에서 최신 버전을 사용하도록 이를 생략합니다.

`--test-id`를 사용하여 개별 테스트 케이스를 실행합니다.

Example

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

전체 옵션 목록은 FreeRTOS 검증 테스트 제품군 실행 단원을 참조하십시오.

> **Note**
>
> IDT v3.0.0부터 IDT는 최신 테스트 제품군을 온라인으로 확인합니다. 자세한 내용은 테스트 제품군 버전 단원을 참조하십시오.

IDT v1.7.0 and earlier

`help`

지정된 명령에 대한 정보를 나열합니다.

`list-groups`

지정된 제품군에 있는 그룹을 나열합니다.

`list-suites`

사용 가능한 제품군을 나열합니다.

`run-suite`

장치의 풀에 대해 테스트 제품군을 실행합니다.

재검증을 위한 테스트

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

새로운 버전의 FreeRTOS용 IDT 검증 테스트가 릴리스되거나 보드별 패키지 또는 디바이스 드라이버를 업데이트하면 FreeRTOS용 IDT를 사용하여 마이크로컨트롤러 보드를 테스트할 수 있습니다. 이후 검증에서는 최신 버전의 FreeRTOS 및 FreeRTOS용 IDT가 있는지 확인하고 검증 테스트를 다시 실행해야 합니다.

AWS IoT Device Tester FreeRTOS 테스트 제품군 버전용

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

FreeRTOS용 IDT는 테스트 제품군 및 테스트 그룹으로 테스트를 구성합니다.

- 테스트 제품군은 디바이스가 FreeRTOS의 특정 버전에서 작동하는지 확인하는 데 사용되는 테스트 그룹 세트입니다.
- 테스트 그룹은 PKCS, MQTT, OTA와 같은 특정 기능과 관련된 개별 테스트 세트입니다.

IDT v3.0.0부터는 `major.minor.patch` 형식(1.0.0부터 시작)을 사용하여 테스트 제품군의 버전이 관리됩니다. IDT를 다운로드하면 패키지에 최신 테스트 제품군 버전이 포함됩니다.

명령줄 인터페이스에서 IDT를 시작하면 IDT는 최신 테스트 집합 버전을 사용할 수 있는지 여부를 확인합니다. 이 경우 새 버전으로 업데이트하라는 메시지가 표시됩니다. 현재 테스트의 업데이트 또는 계속 진행을 선택할 수 있습니다.

> **Note**
>
> IDT는 검증을 위해 세 가지 최신 테스트 제품군 버전을 지원합니다. 자세한 내용은 FreeRTOS AWS IoT Device Tester 용 지원 정책 단원을 참조하십시오.

`upgrade-test-suite` 명령을 사용하여 테스트 제품군을 다운로드할 수 있습니다. 또는 IDT를 시작할 때 옵션으로 파라미터 `-upgrade-test-suite` *`flag`*를 사용할 수 있습니다. 여기에서 항상 최신 버전을 다운로드하려면 *`###`*를 'y'로, 기존 버전을 사용하려면 'n'으로 지정합니다.

`list-supported-versions` 명령을 실행하여 현재 버전의 IDT에서 지원하는 FreeRTOS 및 테스트 제품군 버전을 나열할 수도 있습니다.

새로운 테스트에서는 새로운 IDT 구성 설정을 도입할 수 있습니다. 설정이 선택 사항인 경우 IDT는 사용자에게 이를 알리고 테스트를 계속 실행합니다. 설정이 필요한 경우 IDT는 사용자에게 이를 알리고 실행을 중지합니다. 설정을 구성한 후에는 테스트를 계속 실행할 수 있습니다.

결과 및 로그 이해

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

이 단원에서는 IDT 결과 보고서 및 로그를 보고 해석하는 방법을 설명합니다.

결과 보기

실행하는 동안 IDT는 콘솔, 로그 파일 및 테스트 보고서에 오류를 작성합니다. IDT는 일련의 검증 테스트를 완료한 후 콘솔에 테스트 실행 요약을 작성하고 두 개의 테스트 보고서를 생성합니다. 이러한 보고서는 *devicetester-extract-location*/results/*execution-id*/에서 확인할 수 있습니다. 두 보고서 모두 검증 테스트 세트의 실행 결과를 캡처합니다.

awsiotdevicetester_report.xml는 AWS Partner Device Catalog에 디바이스를 나열 AWS 하기 위해 제출하는 검증 테스트 보고서입니다. 보고서에는 다음 요소가 포함됩니다.

- FreeRTOS용 IDT 버전.
- 테스트한 FreeRTOS 버전.
- 테스트 통과를 기반으로 디바이스에서 지원되는 FreeRTOS 기능.
- device.json 파일에 지정된 SKU 및 디바이스 이름
- device.json 파일에 지정된 디바이스의 기능.
- 테스트 사례 결과의 집계 요약입니다.
- 디바이스 기능(예: FullWiFi, FullMQTT 등)을 기반으로 테스트된 라이브러리별 테스트 사례 결과의 분석입니다.
- 이 FreeRTOS 검증이 LTS 라이브러리를 사용하는 버전 202012.00에 대한 것인지 여부.

FRQ_Report.xml은 표준 JUnit XML 형식의 보고서입니다. Jenkins, Bamboo 등과 같은 CI/CD 플랫폼에 이를 통합할 수 있습니다. 보고서에는 다음 요소가 포함됩니다.

- 테스트 사례 결과의 집계 요약
- 디바이스 기능을 기반으로 테스트된 라이브러리별 테스트 사례 결과의 분석

FreeRTOS용 IDT 결과 해석

awsiotdevicetester_report.xml 또는 FRQ_Report.xml의 보고서 섹션에는 실행된 테스트 결과가 나열됩니다.

첫 번째 XML 태그 <testsuites>에는 테스트 실행의 전체 요약이 포함됩니다. 예시:

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

<testsuites> 태그에 사용되는 속성

`name`

테스트 제품군의 이름입니다.

`time`

검증 세트를 실행하는 데 걸린 시간(초)

`tests`

실행된 테스트 사례의 수입니다.

`failures`

실행되었지만 통과하지 못한 테스트 사례의 수입니다.

`errors`

FreeRTOS용 IDT에서 실행할 수 없는 테스트 사례의 수입니다.

`disabled`

이 속성은 사용되지 않으므로 무시해도 좋습니다.

테스트 사례 실패 또는 오류가 없는 경우 디바이스는 FreeRTOS를 실행하기 위한 기술 요구 사항을 충족하고 AWS IoT 서비스와 상호 작용할 수 있습니다. AWS Partner Device Catalog에 디바이스를 나열하도록 선택한 경우이 보고서를 검증 증거로 사용할 수 있습니다.

테스트 사례 실패 또는 오류의 경우 `<testsuites>` XML 태그를 검토하여 실패한 테스트 사례를 식별할 수 있습니다. `<testsuites>` 태그 내부의 `<testsuite>` XML 태그는 테스트 그룹에 대한 테스트 사례 결과 요약을 보여 줍니다.

```
<testsuite name="FullMQTT" package="" tests="16" failures="0" time="76"
disabled="0" errors="0" skipped="0">
```

형식은 `<testsuites>` 태그와 비슷하지만, 사용되지 않고 무시할 수 있는 `skipped`라는 속성이 있습니다. 각 `<testsuite>` XML 태그 내부에는 테스트 그룹에 실행된 각 테스트 사례에 대한 `<testcase>` 태그가 있습니다. 예시:

```
<testcase classname="mcu.Full_MQTT" name="AFQP_MQTT_Connect_HappyCase"
attempts="1"></testcase>
```

<awsproduct> 태그에 사용되는 속성

`name`

테스트하는 제품의 이름입니다.

`version`

테스트하는 제품의 버전입니다.

`sdk`

SDK를 사용하여 IDT를 실행한 경우 이 블록에는 SDK의 이름 및 버전이 포함됩니다. SDK를 사용하여 IDT를 실행하지 않은 경우 이 블록에는 다음이 포함됩니다.

```
<sdk>
    <name>N/A</vame>
    <version>N/A</version>
</sdk>
```

`features`

확인된 기능입니다. `required`로 표시된 기능은 자격에 대한 보드를 제출하는 데 필요합니다. 다음 코드 조각은 `awsiotdevicetester_report.xml` 파일에 이 정보가 나타나는 방식을 보여줍니다.

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

`optional`로 표시된 기능은 자격에 필수 기능이 아닙니다. 다음 코드 조각은 선택적 기능을 보여줍니다.

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

필요한 기능에 대한 테스트 실패 또는 오류가 없는 경우 디바이스는 FreeRTOS를 실행하기 위한 기술 요구 사항을 충족하고 AWS IoT 서비스와 상호 작용할 수 있습니다. AWS Partner Device Catalog에 디바이스를 나열하려는 경우 이 보고서를 검증 증거로 사용할 수 있습니다.

테스트 실패 또는 오류의 경우 `<testsuites>` XML 태그를 검토하여 실패한 테스트를 식별할 수 있습니다. `<testsuites>` 태그 내부의 `<testsuite>` XML 태그는 테스트 그룹에 대한 테스트 결과 요약을 보여 줍니다. 예시:

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

형식은 <testsuites> 태그와 비슷하지만, 사용되지 않고 무시할 수 있는 skipped 속성이 있습니다. 각 <testsuite> XML 태그 내부에는 테스트 그룹에 실행된 각 테스트에 대한 <testcase> 태그가 있습니다. 예시:

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

lts

LTS 라이브러리를 사용하는 FreeRTOS 버전에 대한 검증이면 true이고 그렇지 않으면 false입니다.

<testcase> 태그에 사용되는 속성

name

테스트 사례의 이름입니다.

attempts

FreeRTOS용 IDT가 테스트 사례를 실행한 횟수입니다.

테스트가 실패하거나 오류가 발생하는 경우 문제 해결에 대한 정보와 함께 <failure> 또는 <error> 태그가 <testcase> 태그에 추가됩니다. 예시:

```
<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>
```

자세한 내용은 문제 해결 단원을 참조하십시오.

로그 보기

FreeRTOS용 IDT가 테스트 실행에서 생성하는 로그는 *devicetester-extract-location*/results/*execution-id*/logs에서 확인할 수 있습니다. 두 개의 로그 세트가 생성됩니다.

`test_manager.log`

FreeRTOS용 IDT에서 생성된 로그를 포함합니다(예: 로그 관련 구성 및 보고서 생성).

`test_group_id``__`*`test_case_id`*`.log`(예: `FullMQTT__Full_MQTT.log`)

테스트 중인 디바이스의 출력을 포함하는 테스트 케이스용 로그 파일입니다. 로그 파일은 실행한 테스트 그룹 및 테스트 케이스에 따라 명명됩니다.

IDT를 사용하여 자체 테스트 제품군 개발 및 실행

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

IDT v4.0.0부터 FreeRTOS용 IDT는 표준화된 구성 설정 및 결과 형식을 디바이스 및 디바이스 소프트웨어에 대한 사용자 지정 테스트 제품군을 개발할 수 있는 테스트 제품군 환경과 결합합니다. 사용자 지정 테스트를 자체 내부 검증을 위해 추가하거나 디바이스 검증을 위해 고객에게 제공할 수 있습니다.

IDT를 사용하여 다음과 같이 사용자 지정 테스트 도구 모음을 개발하고 실행하십시오.

사용자 지정 테스트 제품군을 개발하려면

- 테스트하려는 디바이스에 대해 사용자 지정 테스트 로직이 포함된 테스트 제품군을 생성합니다.
- IDT에 테스트 실행기를 위한 사용자 지정 테스트 제품군을 제공합니다. 테스트 도구 모음의 특정 설정 구성에 대한 정보를 포함해야 합니다.

사용자 지정 테스트 도구 모음을 실행하려면

- 테스트하려는 장치를 설정합니다.
- 사용하려는 테스트 도구 모음의 필요에 따라 설정 구성을 구현하십시오.
- IDT를 사용하여 사용자 지정 테스트 도구 모음을 실행하세요.
- IDT에서 실행한 테스트의 테스트 결과 및 실행 로그를 확인합니다.

최신 버전의 FreeRTOS용 AWS IoT 디바이스 테스터 다운로드

최신 버전의 IDT를 다운로드하여 읽기 및 쓰기 권한이 있는 파일 시스템의 위치에 소프트웨어의 압축을 풉니다.

> **Note**
>
> 여러 사용자가 NFS 디렉터리 또는 Windows 네트워크 공유 폴더와 같은 공유 위치에서 IDT를 실행하는 것은 지원되지 않습니다. 로컬 드라이브에 IDT 패키지의 압축을 풀고 로컬 워크스테이션에서 IDT 바이너리를 실행하는 것이 좋습니다.
> Windows의 경우 260자의 경로 길이 제한이 있습니다. Windows를 사용하는 경우 경로를 260자 제한 아래로 유지하도록 IDT 압축을 `C:\` 또는 `D:\` 같은 루트 디렉터리에 풉니다.

테스트 도구 모음 생성 워크플로

테스트 도구 모음은 세 가지 유형의 파일로 구성됩니다.

- 테스트 도구 모음 실행 방법에 대한 정보를 IDT에 제공하는 JSON 구성 파일.
- IDT가 테스트 사례를 실행하는 데 사용하는 테스트 실행 파일.
- 테스트를 실행하는 데 필요한 추가 파일.

다음 기본 단계를 완료하여 사용자 지정 IDT 테스트를 생성하세요.

1. 테스트 도구 모음의 JSON 구성 파일을 만드세요.
2. 테스트 도구 모음의 테스트 로직이 포함된 테스트 사례 실행 파일을 만드세요.
3. 테스트 도구 모음을 실행하는 데 테스트 러너에게 필요한 구성 정보를 확인하고 문서화하세요.
4. IDT가 예상대로 테스트 도구 모음을 실행하고 테스트 결과를 생성할 수 있는지 확인하십시오.

샘플 사용자 지정 도구 모음을 빠르게 구축하고 실행하려면 자습서: 샘플 IDT 테스트 제품군 빌드 및 실행의 지침을 따르십시오.

Python으로 사용자 지정 테스트 제품군 생성을 시작하려면 자습서: 간단한 IDT 테스트 제품군 개발 섹션을 참조하세요.

자습서: 샘플 IDT 테스트 제품군 빌드 및 실행

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

AWS IoT Device Tester 다운로드에는 샘플 테스트 제품군의 소스 코드가 포함됩니다. 이 자습서를 완료하여 샘플 테스트 제품군을 빌드하고 실행하여 FreeRTOS AWS IoT Device Tester 에서를 사용하여 사용자 지정 테스트 제품군을 실행하는 방법을 이해할 수 있습니다. 이 자습서에서는 SSH를 사용하지만 FreeRTOS 디바이스와 AWS IoT Device Tester 함께를 사용하는 방법을 배우는 것이 유용합니다.

이 자습서에서는 다음 단계를 완료합니다.

1. 샘플 테스트 도구 모음 구축
2. IDT를 사용하여 샘플 테스트 도구 모음을 실행하세요.

사전 조건

이 자습서를 완료하려면 다음이 필요합니다.

- 호스트 컴퓨터 요구 사항
  - 의 최신 버전 AWS IoT Device Tester
  - Python 3.7 이상

    컴퓨터에 설치된 Python 버전 번호를 확인하려면 인스턴스에서 다음 명령을 실행합니다.

    ```
    python3 --version
    ```

    Windows에서 이 명령 사용시 오류가 반환되면 `python --version`을(를) 대신 사용하십시오. 반환된 버전 번호가 3.7 이상인 경우 Powershell 터미널에서 다음 명령을 실행하여 `python` 명령의 별칭으로 `python3`을(를) 설정합니다.

    ```
    Set-Alias -Name "python3" -Value "python"
    ```

    버전 정보가 반환되지 않았거나 버전 번호가 3.7 미만이면 Python 다운로드의 지침에 따라 Python 3.7 이상을 설치합니다. 자세한 내용은 Python 설명서를 참조하세요.
  - urllib3

    `urllib3`이 제대로 설치되었는지 확인하려면 다음 명령을 실행합니다.

    ```
    python3 -c 'import urllib3'
    ```

    `urllib3`가 설치되지 않은 경우에는 다음 명령을 실행하여 설치합니다.

```
python3 -m pip install urllib3
```

- 장치 요구 사항
  - Linux 운영 체제를 사용하고 호스트 컴퓨터와 동일한 네트워크에 네트워크로 연결된 장치입니다.

    Raspberry Pi OS와 함께 Raspberry Pi를 사용하는 것이 좋습니다. Raspberry Pi에 원격으로 연결하려면 Pi에서 SSH를 설정해야 합니다.

IDT용 장치 정보 구성

IDT가 테스트를 실행할 수 있도록 장치 정보를 구성하십시오. *<device-tester-extract-location>*/configs 폴더에 있는 device.json 템플릿을 다음 정보로 업데이트해야 합니다.

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

devices 개체에 다음 정보를 제공하시기 바랍니다.

id

테스트 대상 장치의 고유한 사용자 정의 식별자입니다.

connectivity.ip

장치의 IP 주소입니다.

connectivity.port

선택 사항. 장치에 SSH 연결에 사용할 포트 번호입니다.

connectivity.auth

연결에 대한 인증 정보입니다.

이 속성은 connectivity.protocol이 ssh로 설정된 경우에만 적용됩니다.

connectivity.auth.method

지정된 연결 프로토콜을 통해 장치에 액세스하는 데 사용되는 인증 방법입니다.

지원되는 값은 다음과 같습니다.

- pki
- password

connectivity.auth.credentials

인증에 사용되는 자격 증명입니다.

connectivity.auth.credentials.user

장치에 로그인하는 데 사용되는 사용자 이름.

connectivity.auth.credentials.privKeyPath

장치에 로그인하는 데 사용하는 프라이빗 키의 전체 경로입니다.

이 값은 connectivity.auth.method가 pki로 설정된 경우에만 적용됩니다.

devices.connectivity.auth.credentials.password

장치에 로그인하기 위해 사용하는 암호입니다.

이 값은 connectivity.auth.method가 password로 설정된 경우에만 적용됩니다.

> **Note**
>
> `method`가 `pki`로 설정된 경우에만 `privKeyPath`를 지정합니다.
> `method`가 `password`로 설정된 경우에만 `password`를 지정합니다.

샘플 테스트 도구 모음 구축

<device-tester-extract-location>/samples/python 폴더에는 제공된 구축 스크립트를 사용하여 테스트 도구 모음에 결합할 수 있는 샘플 구성 파일, 소스 코드 및 IDT Client SDK가 포함되어 있습니다. 다음 디렉터리 트리는 이러한 샘플 파일의 위치를 보여줍니다.

```
<device-tester-extract-location>
### ...
### tests
### samples
#   ### ...
#   ### python
#       ### configuration
#       ### src
#       ### build-scripts
#           ### build.sh
#           ### build.ps1
### sdks
    ### ...
    ### python
        ### idt_client
```

테스트 도구 모음을 구축하려면 호스트 컴퓨터에서 다음 명령을 실행합니다.

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
```

그러면 IDTSampleSuitePython_1.0.0 폴더 내 *<device-tester-extract-location>*/tests 폴더에 샘플 테스트 도구 모음이 만들어집니다. IDTSampleSuitePython_1.0.0 폴더의 파일을 검토하여 샘플 테스트 도구 모음이 어떻게 구성되어 있는지 이해하고 테스트 사례 실행 파일 및 테스트 구성 JSON 파일의 다양한 예제를 확인하세요.

다음 단계: IDT를 사용하여 생성한 샘플 테스트 도구 모음을 실행하십시오.

IDT를 사용하여 샘플 테스트 도구 모음을 실행하세요.

샘플 테스트 도구 모음을 실행하려면 호스트 컴퓨터에서 다음 명령을 실행하세요.

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id IDTSampleSuitePython
```

IDT는 샘플 테스트 도구 모음을 실행하고 결과를 콘솔로 스트리밍합니다. 테스트 실행이 완료되면 다음 정보가 표시됩니다.

```
========== 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
```

문제 해결

다음 정보를 사용하면 튜토리얼 완료와 관련된 문제를 해결하는 데 도움이 됩니다.

테스트 케이스가 성공적으로 실행되지 않습니다

테스트가 성공적으로 실행되지 않을 경우 IDT는 오류 로그를 콘솔로 스트리밍하여 테스트 실행 문제를 해결하는 데 도움을 줍니다. 이 튜토리얼의 모든 사전 요구 사항을 충족하는지 확인하세요.

테스트 중인 장치에 연결할 수 없습니다.

다음을 확인합니다.

- `device.json` 파일에는 올바른 IP 주소, 포트 및 인증 정보가 들어 있습니다.
- 호스트 컴퓨터에서 SSH를 통해 장치에 연결할 수 있습니다.

자습서: 간단한 IDT 테스트 제품군 개발

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

테스트 제품군은 다음을 결합합니다.

- 테스트 로직이 포함된 테스트 실행 파일
- 테스트 제품군을 설명하는 JSON 구성 파일

이 자습서에서는 FreeRTOS용 IDT를 사용하여 단일 테스트 사례가 포함된 Python 테스트 제품군을 개발하는 방법을 보여줍니다. 이 자습서에서는 SSH를 사용하지만 FreeRTOS 디바이스와 AWS IoT Device Tester 함께를 사용하는 방법을 배우는 것이 유용합니다.

이 자습서에서는 다음 단계를 완료합니다.

1. 테스트 도구 모음 디렉터리 생성
2. JSON 구성 파일 생성
3. 테스트 사례 실행 파일 생성
4. 테스트 도구 모음 실행

사전 조건

이 자습서를 완료하려면 다음이 필요합니다.

- 호스트 컴퓨터 요구 사항
  - 의 최신 버전 AWS IoT Device Tester

- Python 3.7 이상

  컴퓨터에 설치된 Python 버전 번호를 확인하려면 인스턴스에서 다음 명령을 실행합니다.

  ```
  python3 --version
  ```

  Windows에서 이 명령 사용시 오류가 반환되면 `python --version`을(를) 대신 사용하십시오. 반환된 버전 번호가 3.7 이상인 경우 Powershell 터미널에서 다음 명령을 실행하여 `python` 명령의 별칭으로 `python3`을(를) 설정합니다.

  ```
  Set-Alias -Name "python3" -Value "python"
  ```

  버전 정보가 반환되지 않았거나 버전 번호가 3.7 미만이면 Python 다운로드의 지침에 따라 Python 3.7 이상을 설치합니다. 자세한 내용은 Python 설명서를 참조하세요.

- urllib3

  `urllib3`이 제대로 설치되었는지 확인하려면 다음 명령을 실행합니다.

  ```
  python3 -c 'import urllib3'
  ```

  `urllib3`가 설치되지 않은 경우에는 다음 명령을 실행하여 설치합니다.

  ```
  python3 -m pip install urllib3
  ```

- 장치 요구 사항
  - Linux 운영 체제를 사용하고 호스트 컴퓨터와 동일한 네트워크에 네트워크로 연결된 장치입니다.

    Raspberry Pi OS와 함께 Raspberry Pi를 사용하는 것이 좋습니다. Raspberry Pi에 원격으로 연결하려면 Pi에서 SSH를 설정해야 합니다.

테스트 도구 모음 디렉터리 생성

IDT는 각 테스트 도구 모음 내의 테스트 그룹에 테스트 사례를 논리적으로 분리합니다. 각 테스트 사례는 테스트 그룹 내에 있어야 합니다. 이 튜토리얼을 위해 `MyTestSuite_1.0.0`(이)라는 폴더를 만들고 이 폴더 내에 다음 디렉터리 트리를 생성하십시오.

```
MyTestSuite_1.0.0
### suite
```

```
    ### myTestGroup
        ### myTestCase
```

JSON 구성 파일 생성

테스트 도구 모음에는 다음과 같은 필수 JSON 구성 파일이 포함되어야 합니다.

필수 JSON 파일

`suite.json`

테스트 제품군 정보가 포함되어 있습니다. suite.json을 구성하십시오.을(를) 참조하세요.

`group.json`

테스트 그룹에 대한 정보를 포함합니다. 테스트 도구 모음의 각 테스트 그룹에 대한 `group.json` 파일을 만들어야 합니다. group.json을 구성하십시오.을(를) 참조하세요.

`test.json`

테스트 케이스에 대한 정보가 들어 있습니다. 테스트 도구 모음의 각 테스트 케이스에 대한 `test.json` 파일을 만들어야 합니다. test.json을 구성하십시오.을(를) 참조하세요.

1. `MyTestSuite_1.0.0/suite` 폴더에서 다음 폴더 구조로 `suite.json` 파일을 안에 생성합니다.

   ```
   {
       "id": "MyTestSuite_1.0.0",
       "title": "My Test Suite",
       "details": "This is my test suite.",
       "userDataRequired": false
   }
   ```

2. `MyTestSuite_1.0.0/myTestGroup` 폴더에서 다음 폴더 구조로 `group.json` 파일을 안에 생성합니다.

   ```
   {
       "id": "MyTestGroup",
       "title": "My Test Group",
       "details": "This is my test group.",
       "optional": false
   }
   ```

3. MyTestSuite_1.0.0/myTestGroup/myTestCase 폴더에서 다음 폴더 구조로 test.json 파일을 안에 생성합니다.

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

이제 MyTestSuite_1.0.0 폴더의 디렉터리 트리가 다음과 같이 표시되어야 합니다.

```
MyTestSuite_1.0.0
### suite
    ### suite.json
    ### myTestGroup
        ### group.json
        ### myTestCase
            ### test.json
```

IDT 클라이언트 SDK 다운로드

IDT 클라이언트 SDK를 사용하여 IDT가 테스트 대상 장치와 상호 작용하고 테스트 결과를 보고할 수 있도록 합니다. 이 튜토리얼에서는 Python 버전의 SDK를 사용합니다.

<device-tester-extract-location>/sdks/python/ 폴더에서 idt_client 폴더를 MyTestSuite_1.0.0/suite/myTestGroup/myTestCase 폴더로 복사합니다.

SDK가 성공적으로 복사되었는지 확인하려면 다음 명령을 실행합니다.

```
cd MyTestSuite_1.0.0/suite/myTestGroup/myTestCase
python3 -c 'import idt_client'
```

테스트 사례 실행 파일 생성

테스트 사례 실행 파일에는 실행하려는 테스트 로직이 포함되어 있습니다. 테스트 도구 모음에는 여러 테스트 사례 실행 파일이 포함될 수 있습니다. 이 튜토리얼에서는 하나의 테스트 사례 실행 파일을 생성합니다.

1. 테스트 도구 모음 파일을 만드세요.

   MyTestSuite_1.0.0/suite/myTestGroup/myTestCase 폴더 안에 다음 내용으로 myTestCase.py라는 파일을 만듭니다.

   ```
   from idt_client import *

   def main():
       # Use the client SDK to communicate with IDT
       client = Client()

   if __name__ == "__main__":
       main()
   ```

2. 클라이언트 SDK 함수를 사용하여 myTestCase.py 파일에 다음 테스트 로직을 추가합니다.

   a. 테스트 중인 장치에서 SSH 명령어를 실행합니다.

      ```
      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. 테스트 결과를 IDT로 전송합니다.

```
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()
```

IDT용 장치 정보 구성

IDT가 테스트를 실행할 수 있도록 장치 정보를 구성하십시오. *<device-tester-extract-location>*/configs 폴더에 있는 device.json 템플릿을 다음 정보로 업데이트해야 합니다.

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

devices 개체에 다음 정보를 제공하시기 바랍니다.

`id`

테스트 대상 장치의 고유한 사용자 정의 식별자입니다.

`connectivity.ip`

장치의 IP 주소입니다.

`connectivity.port`

선택 사항. 장치에 SSH 연결에 사용할 포트 번호입니다.

`connectivity.auth`

연결에 대한 인증 정보입니다.

이 속성은 `connectivity.protocol`이 `ssh`로 설정된 경우에만 적용됩니다.

`connectivity.auth.method`

지정된 연결 프로토콜을 통해 장치에 액세스하는 데 사용되는 인증 방법입니다.

지원되는 값은 다음과 같습니다.

- `pki`
- `password`

`connectivity.auth.credentials`

인증에 사용되는 자격 증명입니다.

`connectivity.auth.credentials.user`

장치에 로그인하는 데 사용되는 사용자 이름.

`connectivity.auth.credentials.privKeyPath`

장치에 로그인하는 데 사용하는 프라이빗 키의 전체 경로입니다.

이 값은 `connectivity.auth.method`가 `pki`로 설정된 경우에만 적용됩니다.

`devices.connectivity.auth.credentials.password`

장치에 로그인하기 위해 사용하는 암호입니다.

이 값은 `connectivity.auth.method`가 `password`로 설정된 경우에만 적용됩니다.

> **Note**
>
> `method`가 `pki`로 설정된 경우에만 `privKeyPath`를 지정합니다.
> `method`가 `password`로 설정된 경우에만 `password`를 지정합니다.

테스트 도구 모음 실행

테스트 도구 모음을 만든 후에는 예상대로 작동하는지 확인해야 합니다. 이를 위해 기존 장치 풀로 테스트 도구 모음을 실행하려면 다음 단계를 완료하세요.

1. MyTestSuite_1.0.0 폴더를 *<device-tester-extract-location>*/tests에 복사하세요.
2. 다음 명령을 실행합니다.

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id MyTestSuite
```

IDT는 테스트 도구 모음을 실행하고 결과를 콘솔로 스트리밍합니다. 테스트 실행이 완료되면 다음 정보가 표시됩니다.

```
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
```

문제 해결

다음 정보를 사용하면 튜토리얼 완료와 관련된 문제를 해결하는 데 도움이 됩니다.

테스트 케이스가 성공적으로 실행되지 않습니다

테스트가 성공적으로 실행되지 않을 경우 IDT는 오류 로그를 콘솔로 스트리밍하여 테스트 실행 문제를 해결하는 데 도움을 줍니다. 오류 로그를 확인하기 전에 다음 사항을 확인합니다.

- IDT 클라이언트 SDK는 이 단계에서 설명한 대로 올바른 폴더에 있습니다.
- 이 튜토리얼의 모든 사전 요구 사항을 충족합니다.

테스트 중인 장치에 연결할 수 없습니다.

다음을 확인합니다.

- `device.json` 파일에는 올바른 IP 주소, 포트 및 인증 정보가 들어 있습니다.
- 호스트 컴퓨터에서 SSH를 통해 장치에 연결할 수 있습니다.

IDT 테스트 제품군 구성 파일 생성

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

이 섹션에서는 사용자 지정 테스트 제품군을 작성할 때 포함하는 JSON 구성 파일을 만드는 형식을 설명합니다.

필수 JSON 파일

`suite.json`

테스트 제품군 정보가 포함되어 있습니다. suite.json을 구성하십시오.을(를) 참조하세요.

`group.json`

테스트 그룹에 대한 정보를 포함합니다. 테스트 도구 모음의 각 테스트 그룹에 대한 `group.json` 파일을 만들어야 합니다. group.json을 구성하십시오.을(를) 참조하세요.

`test.json`

테스트 케이스에 대한 정보가 들어 있습니다. 테스트 도구 모음의 각 테스트 케이스에 대한 `test.json` 파일을 만들어야 합니다. test.json을 구성하십시오.을(를) 참조하세요.

JSON 파일(선택 사항)

`state_machine.json`

IDT가 테스트 도구 모음을 실행할 때 테스트가 실행되는 방법을 정의합니다. state_machine.json을 구성합니다.을(를) 참조하세요.

`userdata_schema.json`

테스트 실행기가 설정 구성에 포함할 수 있는 `userdata.json` 파일의 스키마를 정의합니다. 이 `userdata.json` 파일은 테스트를 실행하는 데 필요하지만 `device.json` 파일에는 없는 추가 구성 정보에 사용됩니다. userdata_schema.json을 구성합니다.을(를) 참조하세요.

JSON 구성 파일은 다음과 같이 *<custom-test-suite-folder>* 파일에 저장됩니다.

```
<custom-test-suite-folder>
### suite
    ### suite.json
    ### state_machine.json
    ### userdata_schema.json
    ### <test-group-folder>
        ### group.json
        ### <test-case-folder>
            ### test.json
```

suite.json을 구성하십시오.

`suite.json` 파일은 환경 변수를 설정하고 테스트 도구 모음을 실행하는 데 사용자 데이터가 필요한지 여부를 결정합니다. 다음 템플릿을 사용하여 *<custom-test-suite-folder>*`/suite/suite.json` 파일을 구성하십시오.

```
{
    "id": "<suite-name>_<suite-version>",
    "title": "<suite-title>",
    "details": "<suite-details>",
    "userDataRequired": true | false,
    "environmentVariables": [
        {
            "key": "<name>",
            "value": "<value>",
        },
        ...
```

```
        {
            "key": "<name>",
            "value": "<value>",
        }
    ]
}
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

`id`

테스트 도구 모음의 고유한 사용자 정의 ID. `id`의 값은 `suite.json` 파일이 있는 테스트 도구 모음 폴더의 이름과 일치해야 합니다. 세트 이름과 세트 버전도 다음 요구 사항을 충족해야 합니다.

- *<suite-name>*은(는) 언더바를 포함할 수 없습니다.
- *<suite-version>*은(는) 다음과 같이 *x.x.x*(으)로 표시됩니다. 여기서 x은(는) 숫자입니다.

ID는 IDT에서 생성한 테스트 보고서에 표시됩니다.

`title`

이 테스트 도구 모음에서 테스트 중인 제품 또는 기능의 사용자 정의 이름입니다. 테스트 실행기의 IDT CLI에 이름이 표시됩니다.

`details`

테스트 도구 모음의 용도에 대한 짧은 설명입니다.

`userDataRequired`

테스트 실행기가 `userdata.json` 파일에 사용자 지정 정보를 포함해야 하는지 여부를 정의합니다. 이 값을 `true`으(로) 설정하는 경우, 테스트 도구 모음 폴더에도 [userdata_schema.json 파일](#)을 포함해야 합니다.

`environmentVariables`

선택 사항. 이 테스트 도구 모음에 설정할 환경 변수 배열입니다.

`environmentVariables.key`

환경 변수의 이름입니다.

`environmentVariables.value`

환경 변수의 값입니다.

group.json을 구성하십시오.

group.json 파일은 테스트 그룹이 필수인지 옵션인지 여부를 정의합니다. 다음 템플릿을 사용하여 *<custom-test-suite-folder>*/suite/*<test-group>*/group.json 파일을 구성하십시오.

```
{
    "id": "<group-id>",
    "title": "<group-title>",
    "details": "<group-details>",
    "optional": true | false,
}
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

`id`

테스트 그룹의 고유한 사용자 정의 ID입니다. `id`의 값은 `group.json` 파일이 있는 테스트 그룹 폴더의 이름과 일치해야 합니다. ID는 IDT에서 생성한 테스트 보고서에 사용됩니다.

`title`

테스트 그룹을 나타내는 서술형 이름입니다. 테스트 실행기의 IDT CLI에 이름이 표시됩니다.

`details`

테스트 그룹의 용도에 대한 짧은 설명입니다.

`optional`

선택 사항. IDT에서 필수 테스트 실행을 완료한 후 이 테스트 그룹을 선택적 그룹으로 표시하도록 `true`(으)로 설정합니다. 기본값은 `false`입니다.

test.json을 구성하십시오.

`test.json` 파일은 테스트 케이스 실행 파일 및 테스트 케이스에서 사용되는 환경 변수를 결정합니다. 테스트 케이스 실행 파일 생성에 대한 자세한 내용은 IDT 테스트 사례 실행 파일 생성 단원을 참조하세요.

다음 템플릿을 사용하여 *<custom-test-suite-folder>*/suite/*<test-group>*/*<test-case>*/test.json 파일을 구성하십시오.

```
{
    "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>",
        }
    ]
}
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

`id`

테스트 사례의 고유한 사용자 정의 ID입니다. `id`의 값은 `test.json` 파일이 있는 테스트 사례 폴더의 이름과 일치해야 합니다. ID는 IDT에서 생성한 테스트 보고서에 사용됩니다.

`title`

테스트 케이스를 나타내는 서술형 이름입니다. 테스트 실행기의 IDT CLI에 이름이 표시됩니다.

`details`

테스트 케이스의 용도에 대한 짧은 설명입니다.

`requireDUT`

선택 사항. 이 테스트를 실행하는 데 기기가 필요한 경우, `true`(으)로 설정하고, 그렇지 않으면 `false`(으)로 설정합니다. 기본값은 `true`입니다. 테스트 실행기는 `device.json` 파일에서 테스트를 실행하는 데 사용할 기기를 구성합니다.

`requiredResources`

선택 사항. 이 테스트를 실행하는 데 필요한 리소스 기기에 대한 정보를 제공하는 배열입니다.

`requiredResources.name`

이 테스트를 실행할 때 리소스 기기에 부여하는 고유한 이름입니다.

`requiredResources.features`

사용자 정의 리소스 장치 기능의 배열.

`requiredResources.features.name`

기능의 이름입니다. 이 장치를 사용하려는 장치 기능. 이 이름은 테스트 실행기가 `resource.json` 파일에 제공한 기능 이름과 일치합니다.

`requiredResources.features.version`

선택 사항. 기능의 버전. 이 값은 테스트 실행기가 `resource.json` 파일에서 제공한 기능 버전과 일치합니다. 버전이 제공되지 않으면 기능이 확인되지 않습니다. 기능에 버전 번호가 필요하지 않은 경우, 이 필드를 비워 둡니다.

`requiredResources.features.jobSlots`

선택 사항. 이 기능이 지원할 수 있는 동시 테스트 수입니다. 기본값은 1입니다. IDT에서 개별 기능에 대해 서로 다른 기기를 사용하도록 하려면 이 값을 1(으)로 설정하는 것이 좋습니다.

`execution.timeout`

IDT가 테스트 실행이 완료될 때까지 기다리는 시간(밀리초) 입니다. 이 값 설정에 대한 자세한 내용을 알아보려면 IDT 테스트 사례 실행 파일 생성 섹션을 참조하세요.

`execution.os`

IDT를 실행하는 호스트 컴퓨터의 운영 체제에 따라 실행할 테스트 케이스 실행 파일입니다. 지원되는 값은 `linux`, `mac` 및 `win`입니다.

`execution.os.cmd`

지정된 운영 체제에서 실행하려는 테스트 케이스 실행 파일의 경로입니다. 이 위치는 시스템 경로에 있어야 합니다.

`execution.os.args`

선택 사항. 테스트 케이스 실행 파일을 실행하기 위해 제공할 인수입니다.

`environmentVariables`

선택 사항. 이 테스트 케이스에 설정된 환경 변수 배열입니다.

`environmentVariables.key`

환경 변수의 이름입니다.

`environmentVariables.value`

환경 변수의 값입니다.

> **Note**
>
> `test.json` 파일과 `suite.json` 파일에서 동일한 환경 변수를 지정하는 경우, `test.json` 파일의 값이 우선합니다.

state_machine.json을 구성합니다.

상태 머신은 테스트 제품군 실행 흐름을 제어하는 구조입니다. 테스트 도구 모음의 시작 상태를 결정하고, 사용자 정의 규칙을 기반으로 상태 전환을 관리하며, 최종 상태에 도달할 때까지 해당 상태를 계속 전환합니다.

테스트 제품군에 사용자 정의 상태 머신이 포함되지 않은 경우 IDT가 상태 머신을 자동으로 생성합니다. 기본 상태 머신은 다음 기능을 수행합니다.

- 테스트 실행기에 전체 테스트 제품군 대신 특정 테스트 그룹을 선택하고 실행할 수 있는 기능을 제공합니다.
- 특정 테스트 그룹을 선택하지 않은 경우, 테스트 제품군의 모든 테스트 그룹을 무작위 순서로 실행합니다.
- 보고서를 생성하고 각 테스트 그룹 및 테스트 사례의 테스트 결과를 보여주는 콘솔 요약을 인쇄합니다.

IDT 상태 머신이 작동하는 자세한 방법은 IDT 상태 시스템 구성을(를) 참조하십시오.

userdata_schema.json을 구성합니다.

`userdata_schema.json` 파일은 테스트 실행기가 사용자 데이터를 제공하는 스키마를 결정합니다. 테스트 도구 모음에 `device.json` 파일에 없는 정보가 필요한 경우, 사용자 데이터가 필요합니다. 예를 들어 테스트에는 Wi-Fi 네트워크 자격 증명, 특정 오픈 포트 또는 사용자가 제공해야 하는 인증서가 필요할 수 있습니다. 이 정보는 `userdata`(이)라는 입력 파라미터로 IDT에 제공될 수 있습니다. 이때 값은 *`<device-tester-extract-location>`*`/config` 폴더에 생성한 `userdata.json` 파일입니다. `userdata.json` 파일 형식은 테스트 도구 모음에 포함된 `userdata_schema.json` 파일을 기반으로 합니다.

테스트 실행기가 `userdata.json` 파일을 제공해야 함을 나타내려면:

1. `suite.json` 파일에서 `userDataRequired`을(를) `true`(으)로 설정합니다.
2. *`<custom-test-suite-folder>`*에서 `userdata_schema.json` 파일을 생성하십시오.
3. `userdata_schema.json` 파일을 편집하여 유효한 IETF 초안 v4 JSON 스키마를 생성하십시오.

IDT는 테스트 도구 모음을 실행하면 자동으로 스키마를 읽고 이를 사용하여 테스트 실행기가 제공한 `userdata.json` 파일을 검증합니다. 유효한 경우, `userdata.json` 파일의 내용은 IDT 컨텍스트와 상태 시스템 컨텍스트 모두에서 사용할 수 있습니다.

IDT 상태 시스템 구성

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

상태 시스템은 테스트 제품군 실행 흐름을 제어하는 구조입니다. 테스트 도구 모음의 시작 상태를 결정하고, 사용자 정의 규칙을 기반으로 상태 전환을 관리하며, 최종 상태에 도달할 때까지 해당 상태를 계속 전환합니다.

테스트 제품군에 사용자 정의 상태 머신이 포함되지 않은 경우 IDT가 상태 머신을 자동으로 생성합니다. 기본 상태 머신은 다음 기능을 수행합니다.

- 테스트 실행기에 전체 테스트 제품군 대신 특정 테스트 그룹을 선택하고 실행할 수 있는 기능을 제공합니다.
- 특정 테스트 그룹을 선택하지 않은 경우, 테스트 제품군의 모든 테스트 그룹을 무작위 순서로 실행합니다.
- 보고서를 생성하고 각 테스트 그룹 및 테스트 사례의 테스트 결과를 보여주는 콘솔 요약을 인쇄합니다.

IDT 테스트 제품군의 상태 머신은 다음 기준을 충족해야 합니다.

- 각 상태는 테스트 그룹 실행 또는 보고서 파일 생성과 같이 IDT가 취할 수 있는 작업에 해당합니다.
- 상태로 전환하면 상태와 관련된 작업이 실행됩니다.
- 각 상태는 다음 상태의 전환 규칙을 정의합니다.
- 종료 상태는 `Succeed` 또는 `Fail` 중 하나여야 합니다.

상태 머신 형식

다음 템플릿을 사용하여 *<custom-test-suite-folder>*/suite/state_machine.json 파일을 직접 구성할 수 있습니다.

```
{
  "Comment": "<description>",
  "StartAt": "<state-name>",
  "States": {
    "<state-name>": {
      "Type": "<state-type>",
      // Additional state configuration
    }

    // Required states
    "Succeed": {
      "Type": "Succeed"
```

```
    },
    "Fail": {
      "Type": "Fail"
    }
  }
}
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

`Comment`

상태 머신의 설명입니다.

`StartAt`

IDT가 테스트 제품군을 실행하기 시작하는 상태의 이름. `StartAt`의 값은 `States` 객체에 나열된 상태 중 하나로 설정해야 합니다.

`States`

사용자 정의 상태 이름을 유효한 IDT 상태에 매핑하는 객체입니다. 각 상태.*`state-name`* 객체에는 상태 *`state-name`*에 매핑된 유효한 상태의 정의가 포함됩니다.

`States` 객체에는 `Succeed` 및 `Fail` 상태가 포함되어야 합니다. 유효한 상태에 대한 자세한 내용은 유효한 상태 및 상태 정의 섹션을 참조하세요.

유효한 상태 및 상태 정의

이 섹션에서는 IDT 상태 머신에서 사용할 수 있는 모든 유효한 상태의 상태 정의를 설명합니다. 다음 상태 중 일부는 테스트 케이스 수준의 구성을 지원합니다. 하지만 꼭 필요한 경우가 아니면 테스트 케이스 수준 대신 테스트 그룹 수준에서 상태 전환 규칙을 구성하는 것이 좋습니다.

상태 정의

- RunTask
- Choice
- Parallel
- AddProductFeatures
- Report
- LogMessage
- SelectGroup

- Fail
- Succeed

RunTask

RunTask 상태는 테스트 제품군에 정의된 테스트 그룹에서 테스트 케이스를 실행합니다.

```
{
    "Type": "RunTask",
    "Next": "<state-name>",
    "TestGroup": "<group-id>",
    "TestCases": [
        "<test-id>"
    ],
    "ResultVar": "<result-name>"
}
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

`Next`

현재 상태에서 작업을 실행한 후 전환할 상태의 이름입니다.

`TestGroup`

선택 사항. 실행할 테스트 그룹의 ID. 이 값을 지정하지 않으면 IDT는 테스트 실행기가 선택한 테스트 그룹을 실행합니다.

`TestCases`

선택 사항. `TestGroup`에서 지정한 그룹의 테스트 케이스 ID 배열. IDT는 `TestGroup` 및 `TestCases` 값을 기반으로 다음과 같이 테스트 실행 동작을 결정합니다.

- `TestGroup`과 `TestCases`가 모두 지정된 경우 IDT는 테스트 그룹에서 지정된 테스트 케이스를 실행합니다.
- `TestCases`가 지정되었지만 `TestGroup`이 지정되지 않은 경우 IDT는 지정된 테스트 케이스를 실행합니다.
- `TestGroup`이 지정되었지만 `TestCases`가 지정되지 않은 경우 IDT는 지정된 테스트 그룹 내의 모든 테스트 케이스를 실행합니다.
- `TestGroup` 또는 `TestCases` 둘 다 지정되지 않은 경우 IDT는 테스트 실행기가 IDT CLI에서 선택한 테스트 그룹에서 모든 테스트 케이스를 실행합니다. 테스트 실행기에 대한 그룹 선택을 활

성화하려면 `statemachine.json` 파일에 `RunTask` 및 `Choice` 상태를 모두 포함해야 합니다. 작동 방식에 대한 예제는 상태 시스템 예시: 사용자가 선택한 테스트 그룹 실행을 참조하십시오.

테스트 실행기의 IDT CLI 명령 활성화에 대한 자세한 내용은 the section called “IDT CLI 명령을 활성화합니다.” 섹션을 참조하세요.

`ResultVar`

테스트 실행 결과와 함께 설정할 컨텍스트 변수의 이름. `TestGroup`에 대한 값을 지정하지 않은 경우 이 값을 지정하지 마십시오. IDT는 다음에 따라 `ResultVar`에서 `true` 또는 `false`로 정의한 변수 값을 설정합니다.

- 변수 이름의 *`text`*_*`text`*_`passed` 형식인 경우 값은 첫 번째 테스트 그룹의 모든 테스트를 통과했는지 아니면 건너뛰었는지로 설정됩니다.
- 다른 모든 경우에는 값이 모든 테스트 그룹의 모든 테스트를 통과했는지 아니면 건너뛰었는지로 설정됩니다.

일반적으로 `RunTask` 상태를 사용하여 개별 테스트 케이스 ID를 지정하지 않고 테스트 그룹 ID를 지정하면 IDT가 지정된 테스트 그룹의 모든 테스트 케이스를 실행하게 됩니다. 이 상태에서 실행되는 모든 테스트 케이스는 무작위 순서로 병렬로 실행됩니다. 그러나 모든 테스트 케이스를 실행하는 데 장치가 필요하고 사용 가능한 장치가 하나뿐인 경우에는 테스트 케이스가 대신 순차적으로 실행됩니다.

오류 처리

지정된 테스트 그룹 또는 테스트 케이스 ID가 유효하지 않은 경우 이 상태에서는 `RunTaskError` 실행 오류가 발생합니다. 상태에서 실행 오류가 발생하는 경우 상태 머신 컨텍스트의 `hasExecutionError` 변수도 `true`로 설정됩니다.

Choice

`Choice` 상태를 사용하면 사용자 정의 조건에 따라 전환할 다음 상태를 동적으로 설정할 수 있습니다.

```
{
    "Type": "Choice",
    "Default": "<state-name>",
    "FallthroughOnError": true | false,
    "Choices": [
        {
            "Expression": "<expression>",
            "Next": "<state-name>"
        }
```

```
    ]
}
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

`Default`

Choices에 정의된 표현식을 true로 평가할 수 없는 경우 전환할 기본 상태입니다.

`FallthroughOnError`

선택 사항. 상태에서 표현식을 평가할 때 오류가 발생할 경우의 동작을 지정합니다. 평가 결과 오류가 발생할 경우 표현식을 건너뛰려면 true로 설정합니다. 일치하는 표현식이 없는 경우 상태 머신은 해당 Default 상태로 전환됩니다. false 값이 지정하지 않은 경우 기본값은 FallthroughOnError입니다.

`Choices`

현재 상태에서 동작을 실행한 후 전환할 상태를 결정하는 표현식 및 상태의 배열입니다.

`Choices.Expression`

부울 값으로 평가되어야 하는 표현식 문자열입니다. 표현식이 true로 평가되면 상태 머신은 Choices.Next에 정의된 상태로 전환됩니다. 표현식 문자열은 상태 시스템 컨텍스트에서 값을 검색한 다음 해당 값에 대한 연산을 수행하여 부울 값에 도달합니다. 상태 머신 컨텍스트 액세스에 대한 자세한 내용은 상태 머신 컨텍스트 섹션을 참조하세요.

`Choices.Next`

Choices.Expression에 정의된 표현식이 true로 평가되면 전환할 상태의 이름.

오류 처리

Choice 상태는 다음과 같은 경우에 오류 처리를 요구할 수 있습니다.

- 선택 표현식의 일부 변수는 상태 머신 컨텍스트에 존재하지 않습니다.
- 표현식의 결과는 부울 값이 아닙니다.
- JSON 조회 결과는 문자열, 숫자 또는 부울이 아닙니다.

이 상태에서는 Catch 블록을 사용하여 오류를 처리할 수 없습니다. 오류가 발생했을 때 상태 머신 실행을 중지하려면 FallthroughOnError를 false로 설정해야 합니다. 하지만 사용 사례에 따라 다음 중 하나를 수행하도록 FallthroughOnError를 true로 설정하는 것이 좋습니다.

- 액세스하는 변수가 존재하지 않을 것으로 예상되는 경우가 있다면, `Default`의 값과 추가 `Choices` 블록을 사용하여 다음 상태를 지정하십시오.
- 액세스 중인 변수가 항상 존재해야 하는 경우 `Default` 상태를 `Fail`로 설정하십시오.

Parallel

`Parallel` 상태를 사용하면 새 상태 머신을 서로 병렬로 정의하고 실행할 수 있습니다.

```
{
    "Type": "Parallel",
    "Next": "<state-name>",
    "Branches": [
        <state-machine-definition>
    ]
}
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

`Next`

현재 상태에서 작업을 실행한 후 전환할 상태의 이름입니다.

`Branches`

실행할 상태 머신 정의의 배열입니다. 각 스테이트 머신 정의에는 고유한 `StartAt`, `Succeed`, `Fail` 상태가 포함되어야 합니다. 이 배열의 상태 머신 정의는 자체 정의 이외의 상태를 참조할 수 없습니다.

> **Note**
>
> 각 브랜치 상태 머신은 동일한 상태 머신 컨텍스트를 공유하므로 한 브랜치에서 변수를 설정한 다음 다른 브랜치에서 해당 변수를 읽으면 예상치 못한 동작이 발생할 수 있습니다.

브랜치 상태 머신을 모두 실행한 후에만 `Parallel` 상태가 다음 상태로 이동합니다. 장치가 필요한 각 상태는 장치를 사용할 수 있을 때까지 실행을 대기합니다. 여러 장치를 사용할 수 있는 경우 이 상태는 여러 그룹의 테스트 케이스를 병렬로 실행합니다. 사용할 수 있는 장치가 충분하지 않으면 테스트 케이스가 순차적으로 실행됩니다. 테스트 케이스는 병렬로 실행될 때 무작위 순서로 실행되므로 동일한 테스트 그룹에서 테스트를 실행하는 데 여러 장치가 사용될 수 있습니다.

오류 처리

실행 오류를 처리하려면 브랜치 상태 머신과 부모 상태 머신이 모두 해당 `Fail` 상태로 전환되는지 확인하십시오.

브랜치 상태 머신은 부모 상태 머신에 실행 오류를 전송하지 않으므로 `Catch` 블록을 사용하여 브랜치 상태 머신의 실행 오류를 처리할 수 없습니다. 대신 공유 상태 머신 컨텍스트의 `hasExecutionErrors` 값을 사용하십시오. 이렇게 하는 방법의 예는 상태 머신 예제: 두 테스트 그룹을 병렬로 실행 섹션을 참조하세요.

AddProductFeatures

`AddProductFeatures` 상태에서는 IDT에서 생성한 `awsiotdevicetester_report.xml` 파일에 제품 기능을 추가할 수 있습니다.

제품 기능은 장치가 충족할 수 있는 특정 기준에 대한 사용자 정의 정보입니다. 예를 들어, MQTT 제품 기능은 장치가 MQTT 메시지를 올바르게 게시하도록 지정할 수 있습니다. 보고서에서 제품 기능은 지정된 테스트의 통과 여부에 따라 `supported`, `not-supported` 또는 사용자 지정 값으로 설정됩니다.

> **Note**
>
> `AddProductFeatures` 상태는 자체적으로 보고서를 생성하지 않습니다. 보고서를 생성하려면 이 상태를 `Report` 상태로 전환해야 합니다.

```
{
    "Type": "Parallel",
    "Next": "<state-name>",
    "Features": [
        {
            "Feature": "<feature-name>",
            "Groups": [
                "<group-id>"
            ],
            "OneOfGroups": [
                "<group-id>"
            ],
            "TestCases": [
                "<test-id>"
            ],
            "IsRequired": true | false,
            "ExecutionMethods": [
```

```
                "<execution-method>"
            ]
        }
    ]
}
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

`Next`

현재 상태에서 작업을 실행한 후 전환할 상태의 이름입니다.

`Features`

`awsiotdevicetester_report.xml` 파일에 표시할 제품 기능의 배열.

`Feature`

기능의 이름입니다.

`FeatureValue`

선택 사항. 보고서에서 `supported` 대신 사용할 사용자 지정 값입니다. 이 값을 지정하지 않으면 테스트 결과에 따라 기능 값이 `supported` 또는 `not-supported`로 설정됩니다.

`FeatureValue`에 사용자 지정 값을 사용하면 동일한 기능을 다른 조건으로 테스트할 수 있으며, IDT는 지원되는 조건에 대한 기능 값을 연결합니다. 예를 들어, 다음 발췌문은 두 개의 개별 기능 값이 있는 `MyFeature` 기능을 보여줍니다.

```
...
{
    "Feature": "MyFeature",
    "FeatureValue": "first-feature-supported",
    "Groups": ["first-feature-group"]
},
{
    "Feature": "MyFeature",
    "FeatureValue": "second-feature-supported",
    "Groups": ["second-feature-group"]
},
...
```

두 테스트 그룹이 모두 통과하면 기능 값이 `first-feature-supported, second-feature-supported`로 설정됩니다.

`Groups`

선택 사항. 테스트 그룹 ID의 배열입니다. 기능을 지원하려면 지정된 각 테스트 그룹 내의 모든 테스트를 통과해야 합니다.

`OneOfGroups`

선택 사항. 테스트 그룹 ID의 배열입니다. 기능이 지원되려면 지정된 테스트 그룹 중 하나 이상의 모든 테스트를 통과해야 합니다.

`TestCases`

선택 사항. 테스트 케이스 ID의 배열입니다. 이 값을 지정하면 다음이 적용됩니다.

- 기능이 지원되려면 지정된 모든 테스트 케이스를 통과해야 합니다.
- `Groups`은 테스트 그룹 ID는 하나만 포함해야 합니다.
- `OneOfGroups`은 지정하지 않아야 합니다.

`IsRequired`

선택 사항. 보고서에서 이 기능을 선택적 기능으로 표시하려면 `false`로 설정합니다. 기본값은 `true`입니다.

`ExecutionMethods`

선택 사항. `device.json` 파일에 지정된 `protocol` 값과 일치하는 실행 메서드의 배열. 이 값을 지정하는 경우 테스트 실행기는 이 배열의 값 중 하나와 일치하는 `protocol` 값을 지정하여 보고서에 기능을 포함해야 합니다. 이 값을 지정하지 않으면 기능이 항상 보고서에 포함됩니다.

`AddProductFeatures` 상태를 사용하려면 `RunTask` 상태의 `ResultVar` 값을 다음 값 중 하나로 설정해야 합니다.

- 개별 테스트 케이스 ID를 지정한 경우 `ResultVar`을(를) *`group-id_test-id`*_passed(으)로 설정합니다.
- 개별 테스트 케이스 ID를 지정하지 않은 경우 `ResultVar`을(를) *`group-id`*_passed(으)로 설정합니다.

`AddProductFeatures` 상태에서는 다음과 같은 방식으로 테스트 결과를 확인합니다.

- 테스트 케이스 ID를 지정하지 않은 경우 각 테스트 그룹의 결과는 상태 머신 컨텍스트의 *`group-id`*_passed 변수 값에 따라 결정됩니다.

- 테스트 케이스 ID를 지정한 경우 각 테스트의 결과는 상태 머신 컨텍스트의 *group-id_test-id*_passed 변수 값을 기반으로 결정됩니다.

오류 처리

이 상태에서 제공된 그룹 ID가 유효한 그룹 ID가 아닌 경우 이 상태로 인해 AddProductFeaturesError 실행 오류가 발생합니다. 상태에서 실행 오류가 발생하는 경우 상태 머신 컨텍스트의 hasExecutionErrors 변수도 true로 설정됩니다.

Report

Report 상태는 *suite-name*_Report.xml 및 awsiotdevicetester_report.xml 파일을 생성합니다. 또한 이 상태는 보고서를 콘솔로 스트리밍합니다.

```
{
    "Type": "Report",
    "Next": "<state-name>"
}
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

Next

현재 상태에서 작업을 실행한 후 전환할 상태의 이름입니다.

테스트 실행기가 테스트 결과를 볼 수 있도록 테스트 실행 흐름이 끝날 무렵에는 항상 Report 상태로 전환해야 합니다. 일반적으로 이 상태 이후의 다음 상태는 Succeed입니다.

오류 처리

이 상태에서 보고서를 생성하는 데 문제가 발생하면 ReportError 실행 오류가 발생합니다.

LogMessage

LogMessage 상태는 test_manager.log 파일을 생성하고 로그 메시지를 콘솔로 스트리밍합니다.

```
{
    "Type": "LogMessage",
    "Next": "<state-name>"
    "Level": "info | warn | error"
```

```
    "Message": "<message>"
}
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

`Next`

현재 상태에서 작업을 실행한 후 전환할 상태의 이름입니다.

`Level`

로그 메시지를 생성할 때의 오류 수준입니다. 유효하지 않은 수준을 지정하면 이 상태가 오류 메시지를 생성하고 삭제합니다.

`Message`

로그할 메시지.

SelectGroup

`SelectGroup` 상태는 상태 머신 컨텍스트를 업데이트하여 선택된 그룹을 나타냅니다. 이 상태에서 설정된 값은 이후의 모든 `Choice` 상태에서 사용됩니다.

```
{
    "Type": "SelectGroup",
    "Next": "<state-name>"
    "TestGroups": [
        <group-id>"
    ]
}
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

`Next`

현재 상태에서 작업을 실행한 후 전환할 상태의 이름입니다.

`TestGroups`

선택된 것으로 표시될 테스트 그룹 배열입니다. 이 배열의 각 테스트 그룹 ID에 대해 *group-id*_selected 변수는 컨텍스트에서 `true`로 설정됩니다. IDT는 지정된 그룹이 존재하는지 여부를 확인하지 않으므로 유효한 테스트 그룹 ID를 제공해야 합니다.

Fail

`Fail` 상태는 상태 머신이 제대로 실행되지 않았음을 나타냅니다. 이는 상태 머신의 종료 상태이며 각 상태 머신 정의에는 이 상태가 포함되어야 합니다.

```
{
    "Type": "Fail"
}
```

Succeed

`Succeed` 상태는 상태 머신이 올바르게 실행되었음을 나타냅니다. 이는 상태 머신의 종료 상태이며 각 상태 머신 정의에는 이 상태가 포함되어야 합니다.

```
{
    "Type": "Succeed"
}
```

상태 머신 컨텍스트

상태 머신 컨텍스트는 실행 중에 상태 머신이 사용할 수 있는 데이터를 포함하는 읽기 전용 JSON 문서입니다. 상태 머신 컨텍스트는 상태 머신에서만 액세스할 수 있으며 테스트 흐름을 결정하는 정보를 포함합니다. 예를 들어 `userdata.json` 파일에서 테스트 실행기가 구성한 정보를 사용하여 특정 테스트 실행이 필요한지 여부를 결정할 수 있습니다.

상태 머신 컨텍스트는 다음 형식을 사용합니다.

```
{
    "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`

테스트 실행을 위해 선택한 장치 풀에 대한 정보. 선택한 장치 풀의 경우 이 정보는 `device.json` 파일에 정의된 해당 최상위 장치 풀 배열 요소에서 검색됩니다.

`userData`

`userdata.json` 파일의 정보.

`config`

정보는 `config.json` 파일을 정의합니다.

`suiteFailed`

값은 상태 머신이 시작될 때 `false`로 설정됩니다. 테스트 그룹이 `RunTask` 상태로 실패하는 경우 이 값은 상태 머신의 남은 실행 기간 동안으로 `true`로 설정됩니다.

`specificTestGroups`

테스트 실행기가 전체 테스트 제품군 대신 실행할 특정 테스트 그룹을 선택하면 이 키가 생성되고 여기에는 특정 테스트 그룹 ID 목록이 포함됩니다.

`specificTestCases`

테스트 실행기가 전체 테스트 제품군 대신 실행할 특정 테스트 케이스를 선택하면 이 키가 생성되고 여기에는 특정 테스트 케이스 ID 목록이 포함됩니다.

`hasExecutionErrors`

상태 머신이 시작될 때 종료되지 않습니다. 어떤 상태에서든 실행 오류가 발생하는 경우 이 변수가 생성되고 남은 상태 머신 실행 기간 동안 `true`로 설정됩니다.

JSONPath 표기법을 사용하여 컨텍스트를 쿼리할 수 있습니다. 상태 정의의 JSONPath 쿼리 구문은 {{$.*query*}}입니다. 일부 상태에서는 JSONPath 쿼리를 자리 표시자 문자열로 사용할 수 있습니다. IDT는 자리 표시자 문자열을 컨텍스트에서 평가된 JSONPath 쿼리의 값으로 대체합니다. 다음 값에 자리 표시자를 사용할 수 있습니다.

- `RunTask` 상태의 `TestCases` 값.

- `Expression` 값 `Choice` 상태.

상태 머신 컨텍스트에서 데이터에 액세스할 때 다음 조건이 충족되는지 확인하십시오.

- JSON 경로는 `$.`로 시작해야 합니다.
- 각 값은 문자열, 숫자 또는 부울로 평가되어야 합니다.

JSONPath 표기법을 사용하여 컨텍스트에서 데이터에 액세스하는 방법에 대한 자세한 내용은 IDT 컨텍스트 사용 섹션을 참조하십시오.

실행 오류

실행 오류는 상태 머신이 상태를 실행할 때 발생하는 상태 머신 정의의 오류입니다. IDT는 `test_manager.log` 파일의 각 오류에 대한 정보를 기록하고 로그 메시지를 콘솔로 스트리밍합니다.

다음 방법을 사용하여 실행 오류를 처리할 수 있습니다.

- 상태 정의에 `Catch` 블록을 추가합니다.
- 상태 머신 컨텍스트에서 `hasExecutionErrors` 값의 값을 확인합니다.

Catch

`Catch`를 사용하려면 상태 정의에 다음을 추가하십시오.

```
"Catch": [
    {
        "ErrorEquals": [
            "<error-type>"
        ]
        "Next": "<state-name>"
    }
]
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

`Catch.ErrorEquals`

캐치할 오류 유형의 배열입니다. 실행 오류가 지정된 값 중 하나와 일치하면 상태 머신이 `Catch.Next`에서 지정된 상태로 전환됩니다. 발생하는 오류 유형에 대한 자세한 내용은 각 상태 정의를 참조하십시오.

`Catch.Next`

현재 상태에서 `Catch.ErrorEquals`에서 지정한 값 중 하나와 일치하는 실행 오류가 발생하는 경우 전환할 다음 상태입니다.

Catch 블록은 둘 중 하나가 일치할 때까지 순차적으로 처리됩니다. Catch 블록에 나열된 오류와 일치하는 오류가 없으면 상태 머신이 계속 실행됩니다. 실행 오류는 잘못된 상태 정의로 인해 발생하므로 상태에 실행 오류가 발생하면 Fail 상태로 전환하는 것이 좋습니다.

hasExecutionError

일부 상태에서는 실행 오류가 발생하는 경우 오류가 발생하는 것 외에도 상태 시스템 컨텍스트에서 `hasExecutionError` 값을 `true`로 설정합니다. 이 값을 사용하여 오류 발생 시기를 감지한 다음 `Choice` 상태를 사용하여 상태 머신을 해당 `Fail` 상태로 전환할 수 있습니다.

이 메서드는 다음과 특징이 있습니다.

- 상태 머신은 `hasExecutionError`에 할당된 어떤 값으로도 시작되지 않으며 특정 상태가 값을 설정하기 전까지는 이 값을 사용할 수 없습니다. 즉, 실행 오류가 발생하지 않을 경우 상태 머신이 중지되지 않도록 이 값에 액세스하는 `Choice` 상태에 대해 명시적으로 `FallthroughOnError`를 `false`로 설정해야 합니다.
- `true`로 설정한 후에는 `hasExecutionError`가 false로 설정되거나 컨텍스트에서 제거되지 않습니다. 즉, 이 값은 `true`로 처음 설정할 때만 유용하며 이후의 모든 상태에서는 의미 있는 값을 제공하지 않습니다.
- `hasExecutionError` 값은 `Parallel` 상태의 모든 브랜치 상태 머신과 공유되므로 액세스 순서에 따라 예상치 못한 결과가 발생할 수 있습니다.

이러한 특성 때문에 Catch 블록을 대신 사용할 수 있는 경우에는 이 방법을 사용하지 않는 것이 좋습니다.

상태 머신 예제

이 섹션에서는 몇 가지 상태 시스템 구성의 예제를 제공합니다.

예시

- 상태 머신 예제: 단일 테스트 그룹 실행
- 상태 머신 예제: 사용자가 선택한 테스트 그룹 실행
- 상태 머신 예제: 제품 기능이 포함된 단일 테스트 그룹 실행

- 상태 머신 예제: 두 테스트 그룹을 병렬로 실행

상태 머신 예제: 단일 테스트 그룹 실행

이 상태 머신:

- ID GroupA를 사용하여 테스트 그룹을 실행하며 이 ID는 group.json 파일에 있어야 합니다.
- 실행 오류가 있는지 확인하고 오류가 있는 경우, Fail로 전환합니다.
- 보고서를 생성하고 오류가 없는 경우 Succeed로 전환하고 그렇지 않으면 Fail로 전환합니다.

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

상태 머신 예제: 사용자가 선택한 테스트 그룹 실행

이 상태 머신:

- 테스트 실행기가 특정 테스트 그룹을 선택했는지 확인합니다. 테스트 실행기가 테스트 그룹을 선택하지 않으면 테스트 케이스를 선택할 수 없기 때문에 상태 머신은 특정 테스트 케이스를 확인하지 않습니다.
- 테스트 그룹을 선택한 경우:
  - 선택한 테스트 그룹 내에서 테스트 케이스를 실행합니다. 이를 위해 상태 머신은 RunTask 상태의 테스트 그룹이나 테스트 케이스를 명시적으로 지정하지 않습니다.
  - 모든 테스트를 실행한 후 보고서를 생성하고 종료합니다.
- 테스트 그룹을 선택하지 않은 경우:
  - 테스트 그룹 GroupA에서 테스트를 실행합니다.
  - 보고서를 생성하고 종료합니다.

```
{
    "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"
        }
    }
```

```
}
```

상태 머신 예제: 제품 기능이 포함된 단일 테스트 그룹 실행

이 상태 머신:

- 테스트 그룹 GroupA를 실행합니다.
- 실행 오류가 있는지 확인하고 오류가 있는 경우, Fail로 전환합니다.
- awsiotdevicetester_report.xml 파일에 FeatureThatDependsOnGroupA 기능을 추가합니다.
  - GroupA가 통과하면 기능이 supported로 설정됩니다.
  - 이 기능은 보고서에서 선택 사항으로 표시되어 있지 않습니다.
- 보고서를 생성하고 오류가 없는 경우 Succeed로 전환하고 그렇지 않으면 Fail로 전환합니다.

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

상태 머신 예제: 두 테스트 그룹을 병렬로 실행

이 상태 머신:

- GroupA및 GroupB 테스트 그룹을 병렬로 실행합니다. 브랜치 상태 머신의 RunTask 상태에 의해 컨텍스트에 저장된 ResultVar 변수는 AddProductFeatures 상태에서 사용할 수 있습니다.
- 실행 오류가 있는지 확인하고 오류가 있는 경우, Fail로 전환합니다. 이 상태 머신은 Catch 블록을 사용하지 않습니다. 해당 메서드는 브랜치 상태 머신의 실행 오류를 감지하지 못하기 때문입니다.
- 통과한 그룹을 기반으로 awsiotdevicetester_report.xml 파일에 기능을 추가합니다.
  - GroupA가 통과하면 기능이 supported로 설정됩니다.
  - 이 기능은 보고서에서 선택 사항으로 표시되어 있지 않습니다.
- 보고서를 생성하고 오류가 없는 경우 Succeed로 전환하고 그렇지 않으면 Fail로 전환합니다.

장치 풀에 두 개의 장치가 구성된 경우 GroupA 및 GroupB 둘 다 동시에 실행할 수 있습니다. 그러나 GroupA 또는 GroupB 둘 중 하나에 여러 테스트가 포함된 경우에는 두 장치 모두 해당 테스트에 할당될 수 있습니다. 장치가 하나만 구성된 경우 테스트 그룹은 순차적으로 실행됩니다.

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

IDT 테스트 사례 실행 파일 생성

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

다음과 같은 방법으로 테스트 제품군 폴더에 테스트 사례 실행 파일을 만들고 배치할 수 있습니다.

- `test.json` 파일의 인수나 환경 변수를 사용하여 실행할 테스트를 결정하는 테스트 세트의 경우, 전체 테스트 세트에 대해 단일 테스트 사례 실행 파일을 만들거나 테스트 세트의 각 테스트 그룹에 대해 테스트 실행 파일을 만들 수 있습니다.
- 지정된 명령을 기반으로 특정 테스트를 실행하려는 테스트 세트의 경우, 테스트 세트의 각 테스트 사례에 대해 테스트 사례 실행 파일을 하나씩 만듭니다.

테스트 작성자는 사용 사례에 적합한 접근 방식을 결정하고 그에 따라 테스트 사례 실행 파일을 구성할 수 있습니다. 각 `test.json` 파일에 올바른 테스트 케이스 실행 파일 경로를 제공하고 지정된 실행 파일이 올바르게 실행되는지 확인합니다.

모든 장치에서 테스트 케이스를 실행할 준비가 되면 IDT는 다음 파일을 읽습니다.

- 선택한 테스트 사례에 대한 `test.json`에 따라 시작할 프로세스와 설정할 환경 변수가 결정됩니다.
- 테스트 세트용 `suite.json`에 따라 설정할 환경 변수가 결정됩니다.

IDT는 `test.json` 파일에 지정된 명령과 인수를 기반으로 필수 테스트 실행 파일 프로세스를 시작하고 필요한 환경 변수를 프로세스에 전달합니다.

IDT 클라이언트 SDK 사용

IDT 클라이언트 SDK를 사용하면 IDT 및 테스트 대상 장치와 상호 작용하는 데 사용할 수 있는 API 명령을 사용하여 테스트 실행 파일에 테스트 로직을 작성하는 방법을 간소화할 수 있습니다. IDT는 현재 다음과 같은 SDK를 제공합니다.

- Python용 IDT 클라이언트 SDK
- Go용 IDT 클라이언트 SDK

이 SDK는 *`<device-tester-extract-location>`*`/sdks` 폴더에 있습니다. 새 테스트 케이스 실행 파일을 만들 때는 사용할 SDK를 테스트 케이스 실행 파일이 들어 있는 폴더에 복사하고 코드에서 SDK를 참조해야 합니다. 이 섹션에서는 테스트 케이스 실행 파일에서 사용할 수 있는 사용 가능한 API 명령에 대한 간략한 설명을 제공합니다.

이 섹션의 내용

- 장치 상호작용
- IDT 상호 작용
- 호스트 상호작용

장치 상호작용

다음 명령을 사용하면 추가 장치 상호 작용 및 연결 관리 기능을 구현하지 않고도 테스트 중인 장치와 통신할 수 있습니다.

`ExecuteOnDevice`

테스트 세트가 SSH 또는 Docker 셸 연결을 지원하는 장치에서 셸 명령을 실행할 수 있습니다.

`CopyToDevice`

테스트 세트가 IDT를 실행하는 호스트 컴퓨터의 로컬 파일을 SSH 또는 Docker 셸 연결을 지원하는 장치의 지정된 위치로 복사할 수 있습니다.

`ReadFromDevice`

테스트 세트가 UART 연결을 지원하는 장치의 직렬 포트에서 읽을 수 있도록 허용합니다.

> **Note**
>
> IDT는 컨텍스트의 장치 액세스 정보를 사용하여 만든 장치에 대한 직접 연결을 관리하지 않으므로 테스트 사례 실행 파일에서 이러한 장치 상호 작용 API 명령을 사용하는 것이 좋습니다. 하지만 이러한 명령이 테스트 사례 요구 사항을 충족하지 않는 경우, IDT 컨텍스트에서 장치 액세스 정보를 검색하고 이를 사용하여 테스트 세트에서 장치에 직접 연결할 수 있습니다. 직접 연결하려면 테스트 중인 장치와 리소스 장치에 대해 각각 `device.connectivity` 및 `resource.devices.connectivity` 필드에서 정보를 검색합니다. IDT 컨텍스트 사용에 관한 자세한 내용은 IDT 컨텍스트 사용 섹션을 참조합니다.

IDT 상호 작용

다음 명령을 사용하면 테스트 세트가 IDT와 통신할 수 있습니다.

`PollForNotifications`

테스트 세트에서 IDT의 알림을 확인할 수 있습니다.

`GetContextValue` 및 `GetContextString`

테스트 세트가 IDT 컨텍스트에서 값을 검색할 수 있도록 합니다. 자세한 내용은 IDT 컨텍스트 사용 단원을 참조하십시오.

`SendResult`

테스트 세트에서 테스트 사례 결과를 IDT에 보고할 수 있습니다. 이 명령은 테스트 세트의 각 테스트 케이스 끝에서 직접적으로 호출해야 합니다.

호스트 상호작용

다음 명령을 사용하면 테스트 세트가 호스트 머신과 통신할 수 있습니다.

`PollForNotifications`

테스트 세트에서 IDT의 알림을 확인할 수 있습니다.

`GetContextValue` 및 `GetContextString`

테스트 세트가 IDT 컨텍스트에서 값을 검색할 수 있도록 합니다. 자세한 내용은 IDT 컨텍스트 사용 단원을 참조하십시오.

`ExecuteOnHost`

테스트 세트가 로컬 머신에서 명령을 실행할 수 있도록 하고 IDT가 테스트 케이스 실행 수명 주기를 관리할 수 있도록 합니다.

IDT CLI 명령을 활성화합니다.

`run-suite` 명령 IDT CLI는 테스트 실행기가 테스트 실행을 사용자 지정할 수 있는 몇 가지 옵션을 제공합니다. 테스트 실행기가 이러한 옵션을 사용하여 사용자 지정 테스트 세트를 실행할 수 있도록 하려면 IDT CLI에 대한 지원을 구현해야 합니다. 지원을 구현하지 않는 경우, 테스트 실행기는 여전히 테스트를 실행할 수 있지만 일부 CLI 옵션은 제대로 작동하지 않습니다. 이상적인 고객 경험을 제공하려면 IDT CLI에서 `run-suite` 명령에 대한 다음 인수에 대한 지원을 구현하는 것이 좋습니다.

`timeout-multiplier`

테스트를 실행하는 동안 모든 제한 시간에 적용할 1.0보다 큰 값을 지정합니다.

테스트 실행기는 이 인수를 사용하여 실행하려는 테스트 사례의 제한 시간을 늘릴 수 있습니다. 테스트 실행기가 `run-suite` 명령에서 이 인수를 지정하면 IDT는 이 인수를 사용하여 IDT_TEST_TIMEOUT 환경 변수의 값을 계산하고 IDT 컨텍스트에서 `config.timeoutMultiplier` 필드를 설정합니다. 이 인수를 뒷받침하려면 다음을 수행하여야 합니다.

- test.json 파일의 제한 시간 값을 직접 사용하는 대신 IDT_TEST_TIMEOUT 환경 변수를 읽고 올바르게 계산된 제한 시간 값을 구합니다.
- IDT 컨텍스트에서 config.timeoutMultiplier 값을 검색하여 장기 실행 제한 시간에 적용합니다.

제한 시간 이벤트로 인한 조기 종료에 대한 자세한 내용은 종료 동작을 지정합니다.을(를) 참조하세요.

stop-on-first-failure

장애가 발생할 경우, IDT에서 모든 테스트 실행을 중지하도록 지정합니다.

테스트 실행기가 run-suite 명령에 이 인수를 지정하면 IDT는 장애가 발생하는 즉시 테스트 실행을 중단합니다. 그러나 테스트 케이스가 병렬로 실행되는 경우, 예상치 못한 결과가 발생할 수 있습니다. 지원을 구현하려면 IDT에서 이 이벤트가 발생할 경우, 테스트 로직에서 실행 중인 모든 테스트 사례를 중지하고, 임시 리소스를 정리하고, 테스트 결과를 IDT에 보고하도록 지시해야 합니다. 실패 시 조기 종료에 대한 자세한 내용은 종료 동작을 지정합니다. 섹션을 참조하세요.

group-id 및 test-id

IDT가 선택한 테스트 그룹 또는 테스트 케이스만 실행하도록 지정합니다.

테스트 실행기는 run-suite 명령과 함께 이러한 인수를 사용하여 다음과 같은 테스트 실행 동작을 지정할 수 있습니다.

- 지정된 테스트 제품군에 있는 모든 테스트 그룹을 실행합니다.
- 지정된 테스트 그룹 내에서 엄선된 테스트를 실행합니다.

이러한 인수를 지원하려면 테스트 세트의 상태 머신이 상태 머신의 특정 RunTask 및 Choice 상태 세트를 포함해야 합니다. 사용자 지정 상태 머신을 사용하지 않는 경우, 기본 IDT 상태 머신에 필요한 상태가 포함되므로 추가 조치를 취할 필요가 없습니다. 하지만 사용자 지정 상태 머신을 사용하는 경우에는 상태 머신 예제: 사용자가 선택한 테스트 그룹 실행을(를) 샘플로 사용하여 상태 머신에 필요한 상태를 추가합니다.

IDT CLI 명령에 대한 자세한 내용은 사용자 지정 테스트 제품군 디버그 및 실행 단원을 참조합니다.

이벤트 로그 작성

테스트가 실행되는 동안 stdout 및 stderr에 데이터를 보내 콘솔에 이벤트 로그와 오류 메시지를 기록합니다. 콘솔 메시지의 형식에 대한 자세한 정보는 콘솔 메시지 형식에서 확인하세요.

IDT가 테스트 세트 실행을 마치면 *<devicetester-extract-location>*/results/*<execution-id>*/logs 폴더에 있는 test_manager.log 파일에서도 이 정보를 확인할 수 있습니다.

테스트 대상 장치의 로그를 포함하여 테스트 실행의 로그를 *<device-tester-extract-location>*/results/*execution-id*/logs 폴더에 있는 *<group-id>*_*<test-id>* 파일에 기록하도록 각 테스트 사례를 구성할 수 있습니다. 이렇게 하려면 testData.logFilePath 쿼리를 사용하여 IDT 컨텍스트에서 로그 파일의 경로를 검색하고 해당 경로에 파일을 만든 다음 원하는 콘텐츠를 작성해야 합니다. IDT는 실행 중인 테스트 케이스를 기반으로 경로를 자동으로 업데이트합니다. 테스트 사례에 대한 로그 파일을 만들지 않기로 선택하면 해당 테스트 사례에 대한 파일이 생성되지 않습니다.

필요에 따라 *<device-tester-extract-location>*/logs 폴더에 추가 로그 파일을 생성하도록 테스트 실행 파일을 설정할 수도 있습니다. 파일을 덮어쓰지 않도록 로그 파일 이름에 고유한 접두사를 지정하는 것이 좋습니다.

결과를 IDT에 보고합니다.

IDT는 테스트 결과를 awsiotdevicetester_report.xml 및 *suite-name*_report.xml 파일에 기록합니다. 이 보고서 파일은 *<device-tester-extract-location>*/results/*<execution-id>*/에 위치합니다. 두 보고서 모두 테스트 세트의 실행 결과를 캡처합니다. IDT에서 이러한 보고서에 사용하는 스키마에 대한 자세한 내용은 IDT 테스트 결과 및 로그 검토을(를) 참조합니다.

*suite-name*_report.xml 파일 내용을 채우려면 테스트 실행이 완료되기 전에 SendResult 명령을 사용하여 테스트 결과를 IDT에 보고해야 합니다. IDT에서 테스트 결과를 찾을 수 없는 경우, 테스트 사례에 오류가 발생합니다. 다음 Python 발췌문은 테스트 결과를 IDT로 보내는 명령을 보여줍니다.

```
request-variable = SendResultRequest(TestResult(result))
client.send_result(request-variable)
```

API를 통해 결과를 보고하지 않는 경우, IDT는 테스트 아티팩트 폴더에서 테스트 결과를 찾습니다. 이 폴더의 경로는 IDT 컨텍스트의 testData.testArtifactsPath 파일에 저장됩니다. 이 폴더에서 IDT는 찾은 첫 번째 알파벳순으로 정렬된 XML 파일을 테스트 결과로 사용합니다.

테스트 로직이 JUnit XML 결과를 생성하는 경우, 결과를 파싱한 다음 API를 사용하여 IDT에 제출하는 대신 아티팩트 폴더의 XML 파일에 테스트 결과를 기록하여 결과를 IDT에 직접 제공할 수 있습니다.

이 방법을 사용하는 경우, 테스트 로직이 테스트 결과를 정확하게 요약하고 결과 파일의 형식을 *suite-name*_report.xml 파일과 동일한 형식으로 지정해야 합니다. IDT는 제공된 데이터에 대한 검증을 수행하지 않습니다. 단, 다음과 같은 경우는 예외입니다.

- IDT는 `testsuites` 태그의 모든 속성을 무시합니다. 대신 보고된 다른 테스트 그룹 결과에서 태그 속성을 계산합니다.
- 하나 이상의 `testsuite` 태그가 `testsuites` 내에 있어야 합니다.

IDT는 모든 테스트 사례에 동일한 아티팩트 폴더를 사용하고 테스트 실행 사이에 결과 파일을 삭제하지 않기 때문에 IDT에서 잘못된 파일을 읽을 경우, 이 방법을 사용하면 잘못된 보고가 발생할 수도 있습니다. 생성된 XML 결과 파일은 모든 테스트 사례에서 동일한 이름을 사용하여 각 테스트 사례의 결과를 덮어쓰고 IDT에서 사용할 수 있는 올바른 결과가 있는지 확인하는 것이 좋습니다. 테스트 세트의 보고에는 혼합된 접근 방식을 사용할 수 있습니다. 즉, 일부 테스트 사례에는 XML 결과 파일을 사용하고 다른 테스트 사례에는 API를 통해 결과를 제출하는 등 혼합된 접근 방식을 사용할 수 있지만 이 방법은 권장되지 않습니다.

종료 동작을 지정합니다.

테스트 케이스에서 실패 또는 오류 결과가 보고되더라도 텍스트 실행 파일이 항상 종료 코드 0으로 종료되도록 구성합니다. 0이 아닌 종료 코드는 테스트 케이스가 실행되지 않았음을 나타내거나 테스트 케이스 실행 파일이 IDT에 결과를 전달할 수 없는 경우에만 사용합니다. IDT가 0이 아닌 종료 코드를 받으면 테스트 케이스에 오류가 발생하여 테스트 케이스를 실행할 수 없게 된 것으로 표시합니다.

IDT는 다음 이벤트에서 테스트 케이스가 완료되기 전에 테스트 케이스의 실행을 중단하도록 요청하거나 예상할 수 있습니다. 이 정보를 사용하여 테스트 케이스에서 이러한 각 이벤트를 감지하도록 테스트 케이스 실행 파일을 구성합니다.

제한 시간

테스트 케이스가 `test.json` 파일에 지정된 제한 시간 값보다 오래 실행될 때 발생합니다. 테스트 실행기가 `timeout-multiplier` 인수를 사용하여 제한 시간 승수를 지정한 경우, IDT는 승수로 제한 시간 값을 계산합니다.

이 이벤트를 탐지하려면 IDT_TEST_TIMEOUT 환경 변수를 사용합니다. 테스트 실행기가 테스트를 시작하면 IDT는 IDT_TEST_TIMEOUT 환경 변수의 값을 계산된 제한 시간 값(초)으로 설정하고 변수를 테스트 케이스 실행 파일로 전달합니다. 변수 값을 읽어 적절한 타이머를 설정할 수 있습니다.

인터럽트

테스트 러너가 IDT를 인터럽트할 때 발생합니다. 예를 들어, Ctrl+C 키를 누르면 됩니다.

터미널은 신호를 모든 하위 프로세스에 전파하므로 인터럽트 신호를 감지하도록 테스트 케이스에 신호 처리기를 구성하기만 하면 됩니다.

또는 정기적으로 API를 폴링하여 `PollForNotifications` API 응답의 `CancellationRequested` 부울 값을 확인할 수도 있습니다. IDT는 인터럽트 신호를 수신하면 `CancellationRequested` 부울 값을 `true`(으)로 설정합니다.

첫 번째 실패 시 중지

현재 테스트 케이스와 병렬로 실행 중인 테스트 케이스가 실패하고 테스트 실행기가 `stop-on-first-failure` 인수를 사용하여 IDT에 오류가 발생할 경우, 중지하도록 지정하는 경우, 발생합니다.

이 이벤트를 탐지하기 위해 주기적으로 API를 폴링하여 `PollForNotifications` API 응답의 `CancellationRequested` 부울 값을 확인할 수 있습니다. IDT에서 장애가 발생하고 첫 번째 실패 시 중지되도록 구성된 경우, IDT는 `CancellationRequested` 부울 값을 `true`(으)로 설정합니다.

이러한 이벤트가 발생하면 IDT는 현재 실행 중인 테스트 케이스의 실행이 완료될 때까지 5분 동안 기다립니다. 실행 중인 모든 테스트 케이스가 5분 내에 종료되지 않으면 IDT는 각 프로세스를 강제로 중지합니다. 프로세스가 종료되기 전에 IDT에서 테스트 결과를 받지 못한 경우, 테스트 케이스가 제한 시간이 초과된 것으로 표시됩니다. 가장 좋은 방법은 테스트 케이스에서 이벤트 중 하나가 발생할 때 다음 작업을 수행하도록 하는 것입니다.

1. 일반 테스트 로직 실행을 중단합니다.
2. 테스트 대상 장치의 테스트 아티팩트와 같은 임시 리소스를 모두 정리합니다.
3. 테스트 실패 또는 오류와 같은 테스트 결과를 IDT에 보고합니다.
4. 종료합니다.

IDT 컨텍스트 사용

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

IDT가 테스트 제품군을 실행할 때, 테스트 제품군은 각 테스트 실행 방식을 결정하는 데 사용할 수 있는 데이터 세트에 액세스할 수 있습니다. 이 데이터를 IDT 컨텍스트라고 합니다. 예를 들어 테스트 러

너가 `userdata.json` 파일로 제공한 사용자 데이터 구성은 IDT 컨텍스트의 테스트 도구 모음에서 사용할 수 있도록 만들어졌습니다.

IDT 컨텍스트는 읽기 전용 JSON 문서로 간주할 수 있습니다. 테스트 도구 모음은 객체, 배열, 숫자 등과 같은 표준 JSON 데이터 유형을 사용하여 컨텍스트에서 데이터를 검색하고, 컨텍스트에 데이터를 쓸 수 있습니다.

컨텍스트 스키마

IDT 컨텍스트는 다음 형식을 사용합니다.

```
{
    "config": {
        <config-json-content>
        "timeoutMultiplier": timeout-multiplier
    },
    "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`

config.json 파일의 정보. config 필드에는 다음과 같은 추가 필드도 포함됩니다.

`config.timeoutMultiplier`

테스트 도구 모음에서 사용하는 모든 시간 제한 값의 승수입니다. 이 값은 IDT CLI에서 테스트 러너에 의해 지정됩니다. 기본값은 1입니다.

`device`

테스트 실행을 위해 선택한 장치에 대한 정보. 이 정보는 선택한 장치의 device.json 파일에 있는 devices 배열 요소와 동일합니다.

`devicePool`

테스트 실행을 위해 선택한 장치 풀에 대한 정보. 이 정보는 선택한 장치 풀에 대해 device.json 파일에 정의된 최상위 장치 풀 배열 요소와 동일합니다.

`resource`

resource.json 파일의 리소스 장치에 대한 정보.

`resource.devices`

이 정보는 resource.json 파일에 정의된 devices 배열과 동일합니다. 각 devices 요소에는 다음과 같은 추가 필드가 포함됩니다.

`resource.device.name`

리소스 장치의 이름입니다. 이 값은 test.json 파일의 requiredResource.name 값으로 설정됩니다.

`testData.awsCredentials`

테스트에서 AWS 클라우드에 연결하는 데 사용하는 AWS 자격 증명입니다. 이 정보는 config.json 파일에서 가져옵니다.

`testData.logFilePath`

테스트 사례가 로그 메시지를 기록하는 로그 파일의 경로입니다. 이 파일이 존재하지 않는 경우 테스트 도구 모음에서 해당 파일을 생성합니다.

`userData`

테스트 러너가 userdata.json 파일에서 제공한 정보.

컨텍스트에서 데이터 액세스

JSON 파일 및 `GetContextValue` 및 `GetContextString` API로 실행 가능한 텍스트 파일에서 JSONPath 표기법을 사용하여 컨텍스트를 쿼리할 수 있습니다. IDT 컨텍스트에 액세스하기 위한 JSONPath 문자열의 구문은 다음과 같이 다양합니다.

- `suite.json`와(과) `test.json`에서는 `{{`*`query`*`}}`을(를) 사용합니다. 즉, 루트 요소 `$.`을(를) 사용하여 표현식을 시작하지 마십시오.
- `statemachine.json`에서는 `{{$.`*`query`*`}}`을(를) 사용합니다.
- API 명령에서는 명령에 따라 *`query`* 또는 `{{$.`*`query`*`}}`을(를) 사용합니다. 자세한 내용을 알아보려면 SDK에서 인라인 설명서를 참조하세요.

다음 표에서는 일반적인 foobar JSONPath 표현식의 연산자를 설명합니다.

연산자	설명
`$`	루트 요소. IDT의 최상위 컨텍스트 값은 객체이므로 일반적으로 `$.`를 사용하여 쿼리를 시작합니다.
`.childName`	객체의 이름 `childName` 을 사용하여 하위 요소에 액세스합니다. 배열에 적용하면 이 연산자가 각 요소에 적용된 새 배열이 생성됩니다. 요소 이름은 대/소문자를 구분합니다. 예를 들어, `config` 객체의 `awsRegion` 값에 액세스하기 위한 쿼리는 `$.config.awsRegion` 입니다.
`[start:end]`	배열에서 요소를 필터링하여 `start` 인덱스부터 `end` 인덱스까지의 항목을 검색합니다.
`[index1, index2, ... , indexN]`	배열에서 요소를 필터링하여 지정된 인덱스에서만 항목을 검색합니다.
`[?(expr)]`	`expr` 표현식을 사용하여 배열에서 요소를 필터링합니다. 이 표현식은 부울 값으로 평가되어야 합니다.

필터 표현식을 생성하려면 다음 구문을 사용합니다.

```
<jsonpath> | <value> operator <jsonpath> | <value>
```

이 구문에서:

- `jsonpath`은(는) 표준 JSON 구문을 사용하는 JSONPath입니다.
- `value`은(는) 표준 JSON 구문을 사용하는 모든 사용자 지정 값입니다.
- `operator`는 다음 작업 중 하나를 호출합니다.
  - < (미만)
  - <= (이하)
  - == (같음)

    표현식의 JSONPath 또는 값이 배열, 부울 또는 객체 값인 경우 이것이 사용할 수 있는 유일하게 지원되는 바이너리 연산자입니다.
  - >= (이상)
  - > (초과)
  - =~(정규 표현식 일치). 필터 표현식에서 이 연산자를 사용하려면 표현식 왼쪽의 JSONPath 또는 값이 문자열로 평가되어야 하고, 오른쪽은 RE2 구문을 따르는 패턴 값이어야 합니다.

{{*query*}} 형식의 JSONPath 쿼리를 `test.json` 파일의 `args` 및 `environmentVariables` 필드, `suite.json` 파일의 `environmentVariables` 필드 내에서 자리 표시자 문자열로 사용할 수 있습니다. IDT는 컨텍스트 조회를 수행하고 쿼리의 평가된 값으로 필드를 채웁니다. 예를 들어 `suite.json` 파일에서 자리 표시자 문자열을 사용하여 각 테스트 사례에 따라 변경되는 환경 변수 값을 지정할 수 있으며, IDT는 환경 변수를 각 테스트 사례에 맞는 값으로 채웁니다. 하지만 `test.json` 및 `suite.json` 파일에서 자리 표시자 문자열을 사용하는 경우 쿼리에 다음 사항을 고려해야 합니다.

- 쿼리에서 나타나는 `devicePool` 키는 모두 소문자로 입력해야 합니다. 즉, `devicepool`을(를) 대신 사용하십시오.
- 배열의 경우 문자열 배열만 사용할 수 있습니다. 또한 배열은 비표준 `item1, item2,...,itemN` 형식을 사용합니다. 배열에 요소가 하나만 포함된 경우 이 배열은 `item`(으)로 직렬화되어 문자열 필드와 구분할 수 없게 됩니다.
- 자리 표시자를 사용하여 컨텍스트에서 객체를 검색할 수 없습니다.

이러한 고려 사항 때문에 가능하면 `test.json` 및 `suite.json` 파일의 자리 표시자 문자열 대신 API를 사용하여 테스트 로직의 컨텍스트에 액세스하는 것이 좋습니다. 하지만 경우에 따라 JSONPath 자리 표시자를 사용하여 단일 문자열을 가져와서 환경 변수로 설정하는 것이 더 편리할 수 있습니다.

테스트 실행기 설정 구성

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

사용자 지정 테스트 제품군을 실행하려면 테스트 실행기가 실행하려는 테스트 제품군을 기반으로 설정을 구성해야 합니다. 설정은 *<device-tester-extract-location>*`/configs/` 폴더에 있는 JSON 구성 파일 템플릿을 기반으로 지정됩니다. 필요한 경우 테스트 실행기는 IDT가 AWS 클라우드에 연결하는 데 사용할 AWS 자격 증명도 설정해야 합니다.

테스트 작성자는 테스트 도구 모음을 디버깅하도록 이러한 파일을 구성해야 합니다. 테스트 러너가 테스트 도구 모음을 실행하는 데 필요한 대로 다음 설정을 구성할 수 있도록 지침을 제공해야 합니다.

device.json 구성

`device.json` 파일은 테스트가 실행되는 장치에 대한 정보가 필요합니다(예: IP 주소, 로그인 정보, 운영 체제, CPU 아키텍처).

테스트 러너는 *<device-tester-extract-location>*`/configs/` 폴더에 있는 다음 템플릿 `device.json` 파일을 사용하여 이 정보를 제공할 수 있습니다.

```
[
    {
        "id": "<pool-id>",
        "sku": "<pool-sku>",
        "features": [
            {
                "name": "<feature-name>",
                "value": "<feature-value>",
                "configs": [
                    {
                        "name": "<config-name>",
                        "value": "<config-value>"
```

```
                    }
                ],
            }
        ],
        "devices": [
            {
                "id": "<device-id>",
                "connectivity": {
                    "protocol": "ssh | uart | docker",
                    // ssh
                    "ip": "<ip-address>",
                    "port": <port-number>,
                    "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>",
                }
            }
        ]
    }
]
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

`id`

장치 풀이라고 하는 장치 모음을 고유하게 식별하는 사용자 정의 영숫자 ID입니다. 풀에 속한 장치의 하드웨어는 서로 동일해야 합니다. 테스트 제품군을 실행할 때 풀에 있는 장치는 워크로드를 병렬화하는 데 사용됩니다. 다양한 테스트를 실행하기 위해 여러 장치가 사용됩니다.

`sku`

테스트 대상 장치를 고유하게 식별하는 영숫자 값입니다. SKU는 정규화된 장치를 추적하는 데 사용됩니다.

> **Note**
>
> AWS Partner Device Catalog에 보드를 나열하려면 여기에서 지정하는 SKU가 나열 프로세스에서 사용하는 SKU와 일치해야 합니다.

`features`

선택 사항. 장치의 지원되는 기능이 포함된 배열입니다. 장치 기능은 테스트 도구 모음에서 구성한 사용자 정의 값입니다. `device.json` 파일에 포함할 기능 이름 및 값에 대한 정보를 테스트 러너에게 제공해야 합니다. 예를 들어, 다른 장치의 MQTT 서버 역할을 하는 장치를 테스트하려는 경우 이름이 `MQTT_QOS`로 지정된 기능에 대해 지원되는 특정 수준을 검증하도록 테스트 로직을 구성할 수 있습니다. 테스트 러너는 이 기능 이름을 제공하고 기능 값을 장치에서 지원하는 QOS 수준으로 설정합니다. 제공된 정보는 `devicePool.features` 쿼리를 사용하여 IDT 컨텍스트에서 검색하거나 `pool.features` 쿼리를 사용하여 상태 머신 컨텍스트에서 검색할 수 있습니다.

`features.name`

기능의 이름입니다.

`features.value`

지원되는 기능 값.

`features.configs`

필요한 경우 기능에 대한 구성 설정.

`features.config.name`

구성 설정의 이름입니다.

`features.config.value`

지원되는 설정값.

`devices`

테스트할 풀의 장치 배열입니다. 최소 1개 이상의 장치가 필요합니다.

`devices.id`

테스트 대상 장치의 고유한 사용자 정의 식별자입니다.

`connectivity.protocol`

이러한 장치와 통신하는 데 사용되는 통신 프로토콜입니다. 풀의 각 장치는 동일한 프로토콜을 사용해야 합니다.

현재 지원되는 값은 `uart` 물리적 장치의 경우 및, `docker` Docker 컨테이너의 경우 `ssh` 입니다.

`connectivity.ip`

테스트 대상 장치의 IP 입니다.

이 속성은 `connectivity.protocol`이 `ssh`로 설정된 경우에만 적용됩니다.

`connectivity.port`

선택 사항. SSH 연결하는 데 사용하는 포트 번호입니다.

기본값은 4입니다.

이 속성은 `connectivity.protocol`이 `ssh`로 설정된 경우에만 적용됩니다.

`connectivity.auth`

연결에 대한 인증 정보입니다.

이 속성은 `connectivity.protocol`이 `ssh`로 설정된 경우에만 적용됩니다.

`connectivity.auth.method`

지정된 연결 프로토콜을 통해 장치에 액세스하는 데 사용되는 인증 방법입니다.

지원되는 값은 다음과 같습니다.

- `pki`
- `password`

`connectivity.auth.credentials`

인증에 사용되는 자격 증명입니다.

`connectivity.auth.credentials.password`

테스트 대상 장치에 로그인하기 위해 사용하는 암호입니다.

이 값은 `connectivity.auth.method`가 `password`로 설정된 경우에만 적용됩니다.

`connectivity.auth.credentials.privKeyPath`

테스트 대상 장치에 로그인하는 데 사용하는 프라이빗 키의 전체 경로입니다.

이 값은 `connectivity.auth.method`가 `pki`로 설정된 경우에만 적용됩니다.

`connectivity.auth.credentials.user`

테스트 대상 장치에 로그인하기 위한 사용자 이름입니다.

`connectivity.serialPort`

선택 사항. 장치가 연결된 직렬 포트입니다.

이 속성은 `connectivity.protocol`이 `uart`로 설정된 경우에만 적용됩니다.

`connectivity.containerId`

테스트 대상 Docker 컨테이너의 컨테이너 ID 또는 이름입니다.

이 속성은 `connectivity.protocol`이 `docker`로 설정된 경우에만 적용됩니다.

`connectivity.containerUser`

선택 사항. 컨테이너 내부 사용자의 사용자 이름입니다. 기본값은 Dockerfile에 제공된 사용자입니다.

기본값은 4입니다.

이 속성은 `connectivity.protocol`이 `docker`로 설정된 경우에만 적용됩니다.

> **Note**
>
> 테스트 러너가 테스트를 위해 잘못된 장치 연결을 구성했는지 확인하기 위해 상태 머신 컨텍스트에서 `pool.Devices[0].Connectivity.Protocol`을 검색하고 이를 `Choice` 상태에서 예상한 값과 비교할 수 있습니다. 잘못된 프로토콜이 사용된 경우 `LogMessage` 상태를 사용하여 메시지를 인쇄하고 `Fail` 상태로 전환하세요.
> 또는 오류 처리 코드를 사용하여 잘못된 장치 유형에 대한 테스트 실패를 보고할 수 있습니다.

(선택 사항) userdata.json 구성

`userdata.json` 파일에는 테스트 도구 모음에 필요하지만 `device.json` 파일에 지정되지 않은 모든 추가 정보가 들어 있습니다. 이 파일의 형식은 테스트 도구 모음에 정의된 `userdata_scheme.json` 파일에 따라 달라집니다. 테스트 작성자인 경우, 이 정보를 작성한 테스트 도구 모음을 실행할 사용자에게 제공해야 합니다.

(선택 사항) resource.json 구성

`resource.json` 파일에는 리소스 장치로 사용될 모든 장치에 대한 정보가 들어 있습니다. 리소스 장치는 테스트 대상 장치의 특정 기능을 테스트하는 데 필요한 장치입니다. 예를 들어 장치의 Bluetooth 기능을 테스트하려면 리소스 장치를 사용하여 장치가 제대로 연결될 수 있는지 테스트할 수 있습니다. 리소스 장치는 선택 사항이며 필요한 만큼 리소스 장치를 요청할 수 있습니다. 테스트 작성자는 test.json 파일을 사용하여 테스트에 필요한 리소스 장치 기능을 정의합니다. 그런 다음 테스트 러너는 `resource.json` 파일을 사용하여 필수 기능을 갖춘 리소스 장치 풀을 제공합니다. 이 정보를 작성한 테스트 도구 모음을 실행할 사용자에게 제공해야 합니다.

테스트 러너는 *<device-tester-extract-location>*/configs/ 폴더에 있는 다음 템플릿 `resource.json` 파일을 사용하여 이 정보를 제공할 수 있습니다.

```
[
    {
        "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>,
                    "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>",
            }
        }
    ]
}
]
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

`id`

장치 풀이라고 하는 장치 모음을 고유하게 식별하는 사용자 정의 영숫자 ID입니다. 풀에 속한 장치의 하드웨어는 서로 동일해야 합니다. 테스트 제품군을 실행할 때 풀에 있는 장치는 워크로드를 병렬화하는 데 사용됩니다. 다양한 테스트를 실행하기 위해 여러 장치가 사용됩니다.

`features`

선택 사항. 장치의 지원되는 기능이 포함된 배열입니다. 이 필드에 필요한 정보는 테스트 도구 모음의 test.json 파일에 정의되어 있으며, 실행할 테스트와 이 테스트를 실행하는 방법을 결정합니다. 테스트 도구 모음에 기능이 필요하지 않은 경우에는 이 필드가 필요하지 않습니다.

`features.name`

기능의 이름입니다.

`features.version`

기능 버전.

`features.jobSlots`

장치를 동시에 사용할 수 있는 테스트 수를 나타내는 설정입니다. 기본값은 1입니다.

`devices`

테스트할 풀의 장치 배열입니다. 최소 1개 이상의 장치가 필요합니다.

`devices.id`

테스트 대상 장치의 고유한 사용자 정의 식별자입니다.

`connectivity.protocol`

이러한 장치와 통신하는 데 사용되는 통신 프로토콜입니다. 풀의 각 장치는 동일한 프로토콜을 사용해야 합니다.

현재 지원되는 값은 `uart` 물리적 장치의 경우 및, `docker` Docker 컨테이너의 경우 `ssh` 입니다.

`connectivity.ip`

테스트 대상 장치의 IP 입니다.

이 속성은 `connectivity.protocol`이 `ssh`로 설정된 경우에만 적용됩니다.

`connectivity.port`

선택 사항. SSH 연결하는 데 사용하는 포트 번호입니다.

기본값은 4입니다.

이 속성은 `connectivity.protocol`이 `ssh`로 설정된 경우에만 적용됩니다.

`connectivity.auth`

연결에 대한 인증 정보입니다.

이 속성은 `connectivity.protocol`이 `ssh`로 설정된 경우에만 적용됩니다.

`connectivity.auth.method`

지정된 연결 프로토콜을 통해 장치에 액세스하는 데 사용되는 인증 방법입니다.

지원되는 값은 다음과 같습니다.

- `pki`
- `password`

`connectivity.auth.credentials`

인증에 사용되는 자격 증명입니다.

`connectivity.auth.credentials.password`

테스트 대상 장치에 로그인하기 위해 사용하는 암호입니다.

이 값은 `connectivity.auth.method`가 `password`로 설정된 경우에만 적용됩니다.

`connectivity.auth.credentials.privKeyPath`

테스트 대상 장치에 로그인하는 데 사용하는 프라이빗 키의 전체 경로입니다.

이 값은 `connectivity.auth.method`가 `pki`로 설정된 경우에만 적용됩니다.

`connectivity.auth.credentials.user`

테스트 대상 장치에 로그인하기 위한 사용자 이름입니다.

`connectivity.serialPort`

선택 사항. 장치가 연결된 직렬 포트입니다.

이 속성은 `connectivity.protocol`이 `uart`로 설정된 경우에만 적용됩니다.

`connectivity.containerId`

테스트 대상 Docker 컨테이너의 컨테이너 ID 또는 이름입니다.

이 속성은 `connectivity.protocol`이 `docker`로 설정된 경우에만 적용됩니다.

`connectivity.containerUser`

선택 사항. 컨테이너 내부 사용자의 사용자 이름입니다. 기본값은 Dockerfile에 제공된 사용자입니다.

기본값은 4입니다.

이 속성은 `connectivity.protocol`이 `docker`로 설정된 경우에만 적용됩니다.

(선택 사항) config.json 구성

`config.json` 파일 형식의 IDT용 구성 정보가 들어 있습니다. 일반적으로 테스트 실행기는 IDT 및 선택적으로 AWS 리전에 대한 AWS 사용자 자격 증명을 제공하는 경우를 제외하고이 파일을 수정할

필요가 없습니다. 필요한 권한이 있는 자격 증명이 제공되면 AWS 가 사용량 지표를 AWS IoT Device Tester 수집하여에 제출합니다 AWS. 이는 옵트인 기능이며 IDT 기능을 개선하는 데 사용됩니다. 자세한 내용은 IDT 사용량 지표 단원을 참조하십시오.

테스트 실행기는 다음 방법 중 하나로 AWS 자격 증명을 구성할 수 있습니다.

- 보안 인증 파일

  IDT는 AWS CLI와 동일한 자격 증명 파일을 사용합니다. 자세한 내용은 구성 및 자격 증명 파일을 참조하십시오.

  자격 증명 파일의 위치는 사용하는 운영 체제에 따라 달라집니다.

  - macOS, Linux의 경우: `~/.aws/credentials`
  - Windows: `C:\Users\UserName\.aws\credentials`
- 환경 변수

  환경 변수는 운영 체제에서 유지 관리하고 시스템 명령에서 사용하는 변수입니다. SSH 세션 중에 정의된 변수는 해당 세션이 종료된 후에는 사용할 수 없습니다. IDT는 `AWS_ACCESS_KEY_ID` 및 `AWS_SECRET_ACCESS_KEY` 환경 변수를 사용하여 AWS 자격 증명을 저장할 수 있습니다.

  Linux, macOS 또는 Unix에서 이러한 변수를 설정하려면 export를 사용합니다.

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

  Windows에서 이러한 변수를 설정하려면 set을 사용합니다.

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

IDT에 대한 AWS 자격 증명을 구성하려면 테스트 실행기는 *<device-tester-extract-location>*`/configs/` 폴더에 있는 `config.json` 파일의 `auth` 섹션을 편집합니다.

```
{
    "log": {
        "location": "logs"
    },
    "configFiles": {
        "root": "configs",
```

```
        "device": "configs/device.json"
    },
    "testPath": "tests",
    "reportPath": "results",
    "awsRegion": "<region>",
    "auth": {
        "method": "file | environment",
        "credentials": {
            "profile": "<profile-name>"
        }
    }
}
]
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

> **Note**
>
> 이 파일의 모든 경로는 *<device-tester-extract-location>*을 기준으로 정의됩니다.

`log.location`

*<device-tester-extract-location>*에 있는 로그 폴더의 경로.

`configFiles.root`

구성 파일이 포함된 폴더 경로입니다.

`configFiles.device`

device.json 파일 경로입니다.

`testPath`

테스트 도구 모음이 들어 있는 폴더의 경로.

`reportPath`

IDT에서 테스트 도구 모음을 실행한 후 테스트 결과를 포함할 폴더의 경로입니다.

`awsRegion`

선택 사항. 테스트 제품군에서 사용할 AWS 리전입니다. 설정하지 않으면 테스트 도구 모음은 각 테스트 도구 모음에 지정된 기본 리전을 사용합니다.

`auth.method`

IDT가 자격 AWS 증명을 검색하는 데 사용하는 메서드입니다. 지원되는 값은 보안 인증 파일에서 보안 인증을 검색하는 `file`와(과) 환경 변수를 사용하여 보안 인증을 검색하는 `environment`입니다.

`auth.credentials.profile`

보안 인증 파일에서 사용할 보안 인증 프로필. 이 속성은 `auth.method`이 `file`로 설정된 경우에만 적용됩니다.

사용자 지정 테스트 제품군 디버그 및 실행

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

필요한 구성이 설정되면 IDT에서 테스트 제품군을 실행할 수 있습니다. 전체 테스트 제품군의 런타임은 하드웨어와 테스트 제품군의 구성에 따라 달라집니다. 참고로, Raspberry Pi 3B에서 전체 FreeRTOS 검증 테스트 제품군을 완료하는 데 약 30분이 걸립니다.

테스트 제품군을 작성할 때 IDT를 사용하여 테스트 제품군을 디버그 모드에서 실행하여 코드를 실행하기 전에 검사하거나 테스트 실행기에 제공할 수 있습니다.

IDT를 디버그 모드에서 실행합니다.

테스트 도구 모음은 장치와 상호 작용하고, 컨텍스트를 제공하고, 결과를 받기 위해 IDT를 사용하기 때문에 IDT 상호 작용 없이 IDE에서 테스트 도구 모음을 간단히 디버깅할 수는 없습니다. 이를 위해 IDT CLI는 IDT를 디버그 모드에서 실행할 수 있는 `debug-test-suite` 명령을 제공합니다. 다음 명령을 실행하여 `debug-test-suite`에 대해 사용 가능한 옵션을 봅니다.

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

IDT를 디버그 모드에서 실행할 때 IDT는 실제로 테스트 도구 모음을 시작하거나 상태 머신을 실행하지 않습니다. 대신 IDE와 상호 작용하여 IDE에서 실행 중인 테스트 도구 모음의 요청에 응답하고 콘솔에 로그를 인쇄합니다. IDT는 타임아웃이 발생하지 않으며 수동으로 중단될 때까지 종료를 기다립니다.

디버그 모드에서도 IDT는 상태 머신을 실행하지 않으며 보고서 파일을 생성하지 않습니다. 테스트 도구 모음을 디버깅하려면 IDE를 사용하여 IDT가 일반적으로 구성 JSON 파일에서 얻는 일부 정보를 제공해야 합니다. 다음 정보를 제공하는지 확인합니다.

- 각 테스트의 환경 변수 및 인수 IDT는 `test.json` 또는 `suite.json`에서 이 정보를 읽지 않습니다.
- 리소스 장치를 선택하기 위한 인수. IDT는 `test.json`에서 이 정보를 읽지 않습니다.

테스트 도구 모음을 디버깅하려면 다음 단계를 완료하세요.

1. 테스트 도구 모음을 실행하는 데 필요한 설정 구성 파일을 생성합니다. 예를 들어, 테스트 도구 모음에 `device.json`, `resource.json`, 및 `user data.json`이(가) 필요한 경우, 필요에 따라 모두 구성해야 합니다.
2. 다음 명령을 실행하여 IDT를 디버그 모드로 설정하고 테스트를 실행하는 데 필요한 장치를 선택합니다.

   ```
   devicetester_[linux | mac | win_x86-64] debug-test-suite [options]
   ```

   이 명령을 실행하면 IDT는 테스트 도구 모음의 요청을 기다린 다음 요청에 응답합니다. 또한 IDT는 IDT Client SDK의 케이스 프로세스에 필요한 환경 변수를 생성합니다.
3. IDE에서 `run` 또는 `debug` 구성을 사용하여 다음을 수행합니다.

   a. IDT에서 생성한 환경 변수의 값을 설정합니다.

   b. `test.json` 및 `suite.json` 파일에 지정한 모든 환경 변수 또는 인수의 값을 설정합니다.

   c. 필요에 따라 중단점을 설정합니다.
4. IDE에서 테스트 도구 모음을 실행합니다.

   필요한 횟수만큼 테스트 도구 모음을 디버깅하고 다시 실행할 수 있습니다. 디버그 모드에서는 IDT가 타임아웃되지 않습니다.
5. 디버깅을 완료한 후 IDT를 중단하여 디버그 모드를 종료하십시오.

테스트를 실행하기 위한 IDT CLI 명령

다음 단원에서는 IDT CLI 명령에 대해 설명합니다.

IDT v4.0.0

`help`

지정된 명령에 대한 정보를 나열합니다.

`list-groups`

지정된 테스트 제품군에 있는 그룹을 나열합니다.

`list-suites`

사용 가능한 테스트 제품군을 나열합니다.

`list-supported-products`

IDT 버전에 대해 지원되는 제품(이 경우, FreeRTOS 버전)과 현재 IDT 버전에 사용 가능한 FreeRTOS 검증 테스트 제품군 버전을 나열합니다.

`list-test-cases`

주어진 테스트 그룹의 테스트 사례를 나열합니다. 다음 옵션이 지원됩니다.

- `group-id`. 검색할 테스트 그룹입니다. 이 옵션은 필수이며 단일 그룹을 지정해야 합니다.

`run-suite`

장치의 풀에 대해 테스트 제품군을 실행합니다. 다음은 몇 가지 일반적으로 사용되는 옵션입니다:

- `suite-id`. 실행할 테스트 제품군 버전입니다. 지정하지 않으면 IDT는 `tests` 폴더의 최신 버전을 사용합니다.
- `group-id`. 실행할 테스트 그룹(쉼표로 구분된 목록). 지정하지 않으면 IDT는 테스트 제품군의 모든 테스트 그룹을 실행합니다.
- `test-id`. 실행할 테스트 케이스(쉼표로 구분된 목록). 지정된 경우, `group-id`은(는) 단일 그룹을 지정해야 합니다.
- `pool-id`. 테스트할 장치 풀. `device.json` 파일에 여러 장치 풀이 정의되어 있는 경우, 테스트 실행기는 하나의 풀을 지정해야 합니다.
- `timeout-multiplier`. 사용자 정의 승수를 사용하여 테스트에 대해 `test.json` 파일에 지정된 테스트 실행 제한 시간을 수정하도록 IDT를 구성합니다.
- `stop-on-first-failure`. 첫 번째 실패 시 실행을 중지하도록 IDT를 구성합니다. 이 옵션은 지정된 테스트 그룹을 디버깅하는 데 `group-id`와(과) 함께 사용해야 합니다.

- `userdata`. 테스트 도구 모음을 실행하는 데 필요한 사용자 데이터 정보가 포함된 파일을 설정합니다. 이는 테스트 도구 모음 `suite.json` 파일에서 `userdataRequired`이(가) true로 설정된 경우에만 필요합니다.

`run-suite` 옵션에 대한 자세한 내용은 다음 `help` 옵션을 사용하십시오.

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

`debug-test-suite`

테스트 도구 모음을 디버그 모드에서 실행합니다. 자세한 내용은 IDT를 디버그 모드에서 실행합니다. 단원을 참조하십시오.

IDT 테스트 결과 및 로그 검토

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

이 섹션에서는 IDT가 콘솔 로그와 테스트 보고서를 생성하는 형식을 설명합니다.

콘솔 메시지 형식

AWS IoT Device Tester 는 테스트 제품군을 시작할 때 콘솔에 메시지를 인쇄하기 위해 표준 형식을 사용합니다. 다음 발췌문은 IDT에서 생성한 콘솔 메시지의 한 가지 예를 보여줍니다.

```
time="2000-01-02T03:04:05-07:00" level=info msg=Using suite: MyTestSuite_1.0.0
executionId=9a52f362-1227-11eb-86c9-8c8590419f30
```

대부분의 콘솔 메시지는 다음 필드로 구성되어 있습니다.

`time`

로깅된 이벤트의 전체 ISO 8601 타임스탬프.

`level`

로깅된 이벤트의 메시지 수준. 일반적으로 로깅된 메시지 수준은 `info`, `warn` 또는 `error`중 하나입니다. IDT는 예상 이벤트가 발생하여 이벤트가 일찍 종료되는 경우 `fatal` 또는 `panic` 메시지를 표시합니다.

`msg`

로깅된 메시지.

`executionId`

현재 IDT 프로세스의 고유 ID 문자열입니다. 이 ID는 개별 IDT 실행을 구분하는 데 사용됩니다.

테스트 제품군에서 생성된 콘솔 메시지는 테스트 대상 장치와 테스트 제품군, 테스트 그룹 및 IDT가 실행하는 테스트 케이스에 대한 추가 정보를 제공합니다. 다음 발췌문은 테스트 제품군에서 생성한 콘솔 메시지의 한 가지 예를 보여줍니다:

```
time="2000-01-02T03:04:05-07:00" level=info msg=Hello world! suiteId=MyTestSuite
groupId=myTestGroup testCaseId=myTestCase deviceId=my-device
executionId=9a52f362-1227-11eb-86c9-8c8590419f30
```

콘솔 메시지의 테스트 제품군 관련 부분에는 다음 필드가 포함됩니다.

`suiteId`

현재 실행 중인 테스트 제품군의 이름.

`groupId`

현재 실행 중인 테스트 그룹의 ID.

`testCaseId`

현재 실행 중인 테스트 케이스의 ID.

`deviceId`

현재 테스트 케이스에서 사용 중인 테스트 대상 장치의 ID.

IDT에서 테스트 실행을 완료했을 때 콘솔에 테스트 요약을 인쇄하려면 상태 시스템에 `Report 상태`를 포함해야 합니다. 테스트 요약에는 테스트 제품군, 실행된 각 그룹의 테스트 결과, 생성된 로그 및 보고서 파일의 위치에 대한 정보가 포함됩니다. 다음 예제에서는 테스트 요약 메시지를 보여줍니다.

```
========== 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 스키마 보고

`awsiotdevicetester_report.xml`은 다음 정보가 포함된 서명된 보고서입니다.

- IDT 버전
- 테스트 제품군 버전입니다.
- 보고서에 서명하는 데 사용된 보고서 서명 및 키.
- `device.json` 파일에 지정된 SKU 및 장치 풀 이름.
- 테스트된 제품 버전 및 장치 특성.
- 테스트 결과의 집계 요약 이 정보는 *`suite-name`*`_report.xml` 파일에 포함된 정보와 동일합니다.

```
<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>
```

awsiotdevicetester_report.xml 파일에는 테스트하는 제품에 대한 정보와 테스트 제품군을 실행한 후 확인된 제품 기능에 대한 정보를 포함하는 <awsproduct> 태그가 포함되어 있습니다.

<awsproduct> 태그에 사용되는 속성

name

테스트하는 제품의 이름입니다.

version

테스트하는 제품의 버전입니다.

features

확인된 기능입니다. 필수로 required 표시된 특성은 테스트 제품군에서 장치를 검증하는 데 필요합니다. 다음 코드 조각은 awsiotdevicetester_report.xml 파일에 이 정보가 나타나는 방식을 보여 줍니다.

```
<feature name="ssh" value="supported" type="required"></feature>
```

optional 표시된 특성은 검증에 필요하지 않습니다. 다음 코드 조각은 선택적 기능을 보여 줍니다.

```
<feature name="hsi" value="supported" type="optional"></feature>
<feature name="mqtt" value="not-supported" type="optional"></feature>
```

테스트 제품군 보고서 스키마

*suite-name*_Result.xml 보고서는 JUnit XML 형식입니다. Jenkins, Bamboo 등과 같은 지속적 통합 및 배포 플랫폼에 이 보고서를 통합할 수 있습니다. 보고서는 테스트 결과의 집계 요약을 포함합니다.

```
<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>
```

awsiotdevicetester_report.xml 또는 *suite-name*_report.xml의 보고서 섹션에는 실행된 테스트 및 결과가 나열됩니다.

첫 번째 XML 태그 <testsuites>에는 테스트 실행의 요약이 포함됩니다. 예시:

```
<testsuites name="MyTestSuite results" time="2299" tests="28" failures="0" errors="0"
 disabled="0">
```

<testsuites> 태그에 사용되는 속성

name

테스트 제품군의 이름입니다.

time

테스트 제품군를 실행하는 데 걸린 시간(초).

tests

실행된 테스트의 수입니다.

failures

실행되었지만 통과하지 못한 테스트의 수입니다.

errors

IDT에서 실행하지 못한 테스트의 수입니다.

disabled

이 속성은 사용되지 않으므로 무시해도 좋습니다.

테스트 실패 또는 오류의 경우 <testsuites> XML 태그를 검토하여 실패한 테스트를 식별할 수 있습니다. <testsuites> 태그 내부의 <testsuite> XML 태그는 테스트 그룹에 대한 테스트 결과 요약을 보여 줍니다. 예시:

```
<testsuite name="combination" package="" tests="1" failures="0" time="161" disabled="0"
 errors="0" skipped="0">
```

형식은 <testsuites> 태그와 비슷하지만, 사용되지 않고 무시할 수 있는 skipped 속성이 있습니다. 각 <testsuite> XML 태그 내부에는 테스트 그룹에 실행된 각 테스트에 대한 <testcase> 태그가 있습니다. 예시:

```
<testcase classname="Security Test" name="IP Change Tests" attempts="1"></testcase>
```

<testcase> 태그에 사용되는 속성

name

테스트의 이름입니다.

attempts

IDT에서 테스트 사례를 실행한 횟수입니다.

테스트가 실패하거나 오류가 발생하는 경우 문제 해결에 대한 정보와 함께 <failure> 또는 <error> 태그가 <testcase> 태그에 추가됩니다. 예시:

```
<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>
```

IDT 사용량 지표

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

필요한 권한이 포함된 AWS 자격 증명을 제공하는 경우는 사용량 지표를 AWS IoT Device Tester 수집하여에 제출합니다 AWS. 이는 옵트인 기능이며 IDT 기능을 개선하는 데 사용됩니다. IDT는 다음과 같은 정보를 수집합니다.

- IDT를 실행하는 데 사용되는 AWS 계정 ID
- 테스트 실행에 사용된 IDT CLI 명령

- 실행되는 테스트 제품군
- *<device-tester-extract-location>* 폴더의 테스트 제품군
- 장치 풀에 구성된 장치 수
- 테스트 케이스 이름 및 런타임
- 테스트 결과 정보(예: 테스트 통과, 실패, 오류 발생 또는 건너뛰었는지 여부)
- 제품 기능 테스트
- IDT 종료 행동(예: 예상치 못한 종료 또는 조기 종료)

IDT가 전송하는 모든 정보는 *<device-tester-extract-location>*/results/*<execution-id>*/ 폴더의 metrics.log 파일에도 기록됩니다. 로그 파일을 보면 테스트 실행 중에 수집된 정보를 볼 수 있습니다. 이 파일은 사용량 지표를 수집하도록 선택한 경우에만 생성됩니다.

지표 수집을 비활성화하기 위해 추가 조치를 취할 필요가 없습니다. 자격 AWS 증명을 저장하지 말고 자격 증명이 저장된 경우 해당 AWS 자격 증명에 액세스하도록 config.json 파일을 구성하지 마십시오.

자격 AWS 증명 구성

AWS 계정이 아직 없는 경우 계정을 생성해야 합니다. 이미 AWS 계정이 있는 경우 IDT가 사용자를 대신하여에 사용량 지표를 전송할 수 있도록 계정에 필요한 권한을 구성하기만 하면 AWS 됩니다.

에 가입 AWS 계정

이 없는 경우 다음 단계를 AWS 계정완료하여 생성합니다.

에 가입하려면 AWS 계정

1. https://portal.aws.amazon.com/billing/signup을 엽니다.
2. 온라인 지시 사항을 따릅니다.

   가입 절차의 일부로는 전화 또는 문자 메시지를 수신하고 전화 키패드에 확인 코드를 입력하는 것이 포함됩니다.

   에 가입하면 AWS 계정AWS 계정 루트 사용자이 생성됩니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스에 액세스할 권한이 있습니다. 보안 모범 사례는 사용자에게 관리 액세스 권한을 할당하고, 루트 사용자만 사용하여 루트 사용자 액세스 권한이 필요한 작업을 수행하는 것입니다.

AWS 는 가입 프로세스가 완료된 후 확인 이메일을 보냅니다. 언제든지 https://aws.amazon.com/으로 이동하고 내 계정을 선택하여 현재 계정 활동을 보고 계정을 관리할 수 있습니다.

관리자 액세스 권한이 있는 사용자 생성

에 가입한 후 일상적인 작업에 루트 사용자를 사용하지 않도록 관리 사용자를 AWS 계정보호 AWS IAM Identity Center, AWS 계정 루트 사용자활성화 및 생성합니다.

보안 AWS 계정 루트 사용자

1. 루트 사용자를 선택하고 AWS 계정 이메일 주소를 입력하여 계정 소유자AWS Management Console로에 로그인합니다. 다음 페이지에서 비밀번호를 입력합니다.

   루트 사용자를 사용하여 로그인하는 데 도움이 필요하면 AWS 로그인 User Guide의 루트 사용자로 로그인을 참조하세요.
2. 루트 사용자의 다중 인증(MFA)을 활성화합니다.

   지침은 IAM 사용 설명서의 AWS 계정 루트 사용자(콘솔)에 대한 가상 MFA 디바이스 활성화를 참조하세요.

관리자 액세스 권한이 있는 사용자 생성

1. IAM Identity Center를 활성화합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 AWS IAM Identity Center설정을 참조하세요.
2. IAM Identity Center에서 사용자에게 관리 액세스 권한을 부여합니다.

   를 자격 증명 소스 IAM Identity Center 디렉터리 로 사용하는 방법에 대한 자습서는 사용 AWS IAM Identity Center 설명서의 기본값으로 사용자 액세스 구성을 IAM Identity Center 디렉터리 참조하세요.

관리 액세스 권한이 있는 사용자로 로그인

- IAM IDentity Center 사용자로 로그인하려면 IAM Identity Center 사용자를 생성할 때 이메일 주소로 전송된 로그인 URL을 사용합니다.

  IAM Identity Center 사용자를 사용하여 로그인하는 데 도움이 필요하면 AWS 로그인 사용 설명서의 AWS 액세스 포털에 로그인을 참조하세요.

추가 사용자에게 액세스 권한 할당

1. IAM Identity Center에서 최소 권한 적용 모범 사례를 따르는 권한 세트를 생성합니다.

   지침은AWS IAM Identity Center 사용 설명서의 Create a permission set를 참조하세요.
2. 사용자를 그룹에 할당하고, 그룹에 Single Sign-On 액세스 권한을 할당합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 Add groups를 참조하세요.

IDT 권한 구성

이 단계에서는 IDT가 테스트를 실행하고 IDT 사용 데이터를 수집하는 데 사용하는 권한을 구성합니다. AWS Management Console 또는 AWS Command Line Interface (AWS CLI)를 사용하여 IDT에 대한 IAM 정책과 사용자를 생성한 다음 사용자에게 정책을 연결할 수 있습니다.

- IDT에 대한 권한 구성(콘솔)
- IDT에 대한 권한 구성(AWS CLI)

IDT에 대한 권한을 구성하려면(콘솔)

콘솔을 사용하여 FreeRTOS용 IDT의 권한을 구성하려면 다음 단계를 수행합니다.

1. IAM 콘솔에 로그인합니다.
2. 특정 권한으로 역할을 생성하는 권한을 부여하는 고객 관리형 정책을 만듭니다.

   a. 탐색 창에서 정책을 선택한 후 정책 생성을 선택합니다.

   b. JSON 탭에서 자리 표시자 콘텐츠를 다음 정책으로 바꿉니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot-device-tester:SendMetrics"
            ],
            "Resource": "*"
        }
    ]
```

```
}
```

c. 정책 검토를 선택합니다.

d. 이름에서 **`IDTUsageMetricsIAMPermissions`**을 입력합니다. Summary(요약) 아래에서 정책에 의해 부여된 권한을 검토합니다.

e. 정책 생성을 선택합니다.

3. IAM 사용자를 생성하고 사용자에 권한을 연결합니다.

   액세스 권한을 제공하려면 사용자, 그룹 또는 역할에 권한을 추가하세요:

   - 의 사용자 및 그룹 AWS IAM Identity Center:

     권한 세트를 생성합니다. AWS IAM Identity Center 사용 설명서의 권한 세트 생성의 지침을 따릅니다.

   - 보안 인증 공급자를 통해 IAM에서 관리되는 사용자:

     ID 페더레이션을 위한 역할을 생성합니다. IAM 사용 설명서의 Create a role for a third-party identity provider (federation)의 지침을 따릅니다.

   - IAM 사용자:
     - 사용자가 맡을 수 있는 역할을 생성합니다. IAM 사용 설명서에서 Create a role for an IAM user의 지침을 따릅니다.
     - (권장되지 않음)정책을 사용자에게 직접 연결하거나 사용자를 사용자 그룹에 추가합니다. IAM 사용 설명서에서 사용자(콘솔)에 권한 추가의 지침을 따르세요.

IDT에 대한 권한을 구성하려면(AWS CLI)

다음 단계에 따라 AWS CLI 를 사용하여 FreeRTOS용 IDT에 대한 권한을 구성합니다.

1. 컴퓨터에 아직 설치되지 않은 AWS CLI 경우를 설치하고 구성합니다. AWS Command Line Interface 사용 설명서 AWS CLI설치 단계를 따르십시오.

   > **Note**
   >
   > AWS CLI 는 명령줄 셸의 AWS 서비스와 상호 작용하는 데 사용할 수 있는 오픈 소스 도구입니다.

2. IDT 및 FreeRTOS 역할을 관리할 수 있는 권한을 부여하는 다음과 같은 고객 관리형 정책을 생성합니다.

Linux, macOS, or Unix

```
aws iam create-policy --policy-name IDTUsageMetricsIAMPermissions --policy-
document '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot-device-tester:SendMetrics"
            ],
            "Resource": "*"
        }
    ]
}'
```

Windows command prompt

```
aws iam create-policy --policy-name IDTUsageMetricsIAMPermissions --policy-
document
                                          '{\"Version\": \"2012-10-17\",
 \"Statement\": [{\"Effect\": \"Allow\", \"Action\": [\"iot-device-
tester:SendMetrics\"], \"Resource\": \"*\"}]}'
```

> **Note**
>
> Linux, macOS 또는 Unix 터미널 명령과 다른 JSON 구문을 사용하기 때문에 이 단계에는 Windows 명령 프롬프트 예제가 포함되어 있습니다.

3. IAM 사용자를 생성하고 FreeRTOS용 IDT에 필요한 권한을 연결합니다.

   a. IAM 사용자를 생성합니다.

   ```
   aws iam create-user --user-name user-name
   ```

   b. 생성한 `IDTUsageMetricsIAMPermissions` 정책을 새 IAM 사용자에 연결합니다. 명령에서 *user-name*을 IAM 사용자 이름으로 바꾸고 *<account-id>*를 AWS 계정의 ID로 바꿉니다.

```
aws iam attach-user-policy --user-name user-name --policy-arn
 arn:aws:iam::<account-id>:policy/IDTFreeRTOSIAMPermissions
```

4. 사용자에 대한 비밀 액세스 키를 만듭니다.

```
aws iam create-access-key --user-name user-name
```

출력을 안전한 위치에 저장합니다. 나중에이 정보를 사용하여 AWS 자격 증명 파일을 구성합니다.

IDT에 AWS 자격 증명 제공

IDT가 자격 AWS 증명에 액세스하고 지표를 제출하도록 허용하려면 다음을 AWS수행합니다.

1. IAM 사용자의 AWS 자격 증명을 환경 변수 또는 자격 증명 파일에 저장합니다.

   a. 다음 명령을 실행하여 환경 변수를 설정합니다.

```
AWS_ACCESS_KEY_ID=access-key
AWS_SECRET_ACCESS_KEY=secret-access-key
```

   b. 자격 증명 파일을 사용하려면 다음 정보를 `.aws/credentials file:`에 추가합니다.

```
[profile-name]
aws_access_key_id=access-key
aws_secret_access_key=secret-access-key
```

2. `config.json` 파일의 `auth` 섹션을 구성합니다. 자세한 내용은 (선택 사항) config.json 구성 단원을 참조하십시오.

문제 해결

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

각 테스트 제품군 실행 시 `results` 디렉터리에서 `results/`*`execution-id`*라는 폴더를 생성하는 데 사용되는 고유 실행 ID가 있습니다. 개별 테스트 그룹 로그는 `results/`*`execution-id`*`/logs` 디렉터리 아래에 있습니다. FreeRTOS용 IDT 콘솔 출력을 사용해 실행 ID, 테스트 사례 ID, 실패한 테스트 사례의 테스트 그룹을 찾은 뒤 해당 테스트 사례의 로그 파일 `results/`*`execution-id`*`/logs/`*`test_group_id__test_case_id`*`.log`를 엽니다. 이 파일에는 있는 정보는 다음과 같습니다.

- 전체 빌드 및 플래시 명령 출력.
- 테스트 실행 결과
- 보다 상세한 FreeRTOS용 IDT 콘솔 출력.

다음 문제 해결 워크플로우를 사용하는 것이 좋습니다.

1. "*`###/##`*이 리소스에 액세스할 수 있는 권한이 없습니다." 라는 오류가 표시되면 ???에 지정된 대로 사용 권한을 구성해야 합니다.
2. 실행 UUID 및 현재 실행 중인 작업과 같은 정보를 찾으려면 콘솔 출력을 읽습니다.
3. `FRQ_Report.xml` 파일에서 각 테스트의 오류 문을 찾습니다. 이 디렉터리에는 각 테스트 그룹의 실행 로그가 포함되어 있습니다.
4. `/results/`*`execution-id`*`/logs` 아래에 있는 로그 파일을 살펴봅니다.
5. 다음 문제 영역 중 하나를 조사합니다.
   - `/configs/` 폴더의 JSON 구성 파일과 같은 디바이스 구성
   - 디바이스 인터페이스. 로그를 통해 실패한 인터페이스를 확인합니다.
   - 디바이스 도구. 디바이스를 빌드하고 플래시하기 위한 도구 체인이 올바르게 설치 및 구성되었는지 확인합니다.
   - 정리되고 복제된 버전의 FreeRTOS 소스 코드가 있는지 확인합니다. FreeRTOS 릴리스는 FreeRTOS 버전에 따라 태그가 지정됩니다. 특정 코드 버전을 복제하려면 다음 명령을 사용합니다.

```
git clone --branch version-number https://github.com/aws/amazon-freertos.git
cd amazon-freertos
git submodule update --checkout --init --recursive
```

디바이스 구성 문제 해결

FreeRTOS용 IDT를 사용할 때는 바이너리를 실행하기 전에 올바른 구성 파일을 준비해야 합니다. 구문 분석 및 구성 오류가 발생할 경우 첫 번째 단계는 환경에 적합한 구성 템플릿을 찾아서 사용하는 것입니다. 이러한 템플릿은 *IDT_ROOT*/configs 디렉터리에 위치합니다.

그래도 문제가 발생할 경우 다음 디버깅 프로세스를 참조하십시오.

어디를 살펴봐야 합니까?

먼저 콘솔 출력을 읽고 이 설명서에서 execution-id로 참조되는 실행 UUID와 같은 정보를 찾습니다.

그런 다음 /results/*execution-id* 디렉터리에서 FRQ_Report.xml 파일을 찾습니다. 이 파일에는 실행된 모든 테스트 사례와 각 실패에 대한 오류 코드 조각이 포함되어 있습니다. 모든 실행 로그를 가져오려면 각 테스트 케이스에 대한 /results/*execution-id*/logs/*test_group_id*__*test_case_id*.log 파일을 찾아봅니다.

IDT 오류 코드

다음 표에서는 FreeRTOS용 IDT에서 생성되는 오류 코드에 대해 설명합니다.

오류 코드	오류 코드 이름	가능한 근본 원인	문제 해결
201	InvalidInputError	device.json , config.json 또는 userdata.json 의 필드가 누락되었거나 잘못된 형식입니다.	필수 필드가 누락되지 않았으며 나열된 파일에서 필요한 형식인지 확인합니다. 자세한 내용은 처음으로 마이크로 컨트롤러 보드의 테스트 준비 단원을 참조하십시오.
202	ValidationError	device.json , config.json 또는 userdata.json 의 필드에 잘못된 값이 포함되어 있습니다.	보고서에서 오류 코드의 오른쪽에 표시되는 오류 메시지를 확인합니다. • 잘못된 AWS 리전 - config.json 파

오류 코드	오류 코드 이름	가능한 근본 원인	문제 해결
			일에 유효한 AWS 리전을 지정합니다. AWS 리전에 대한 자세한 내용은 리전 및 엔드포인트를 참조하세요. • 잘못된 AWS 자격 증명 - 테스트 시스템에서 유효한 AWS 자격 증명을 설정합니다(환경 변수 또는 자격 증명 파일을 통해). 인증 필드가 올바르게 구성되었는지 확인합니다. 자세한 내용은 ??? 단원을 참조하십시오.
203	CopySourceCodeError	FreeRTOS 소스 코드를 지정된 디렉터리에 복사할 수 없습니다.	다음 항목을 확인합니다. • `userdata.json` 파일에서 유효한 `sourcePath` 가 지정되었는지 확인합니다. • FreeRTOS 소스 코드 디렉터리에서 `build` 폴더를 삭제합니다(해당 폴더가 있는 경우). 자세한 내용은 빌드, 플래시 및 테스트 설정 구성 단원을 참조하십시오.

오류 코드	오류 코드 이름	가능한 근본 원인	문제 해결
204	BuildSourceError	FreeRTOS 소스 코드를 컴파일할 수 없습니다.	다음 항목을 확인합니다. • `userdata.json` 파일에서 `buildTool` 아래 있는 정보가 올바른지 확인합니다. • `cmake`를 빌드 도구로 사용하는 경우 `buildTool` 명령에서 `{{enableTests}}` 가 지정되었는지 확인합니다. 자세한 내용은 빌드, 플래시 및 테스트 설정 구성 단원을 참조하십시오. • `C:\Users\My Name\Desktop\`을 예로 들어, 공백이 포함된 시스템의 파일 경로로 FreeRTOS용 IDT를 추출한 경우 경로가 제대로 파싱되었는지 확인하기 위해 빌드 명령 안에 추가 따옴표가 필요할 수 있습니다. 플래시 명령도 마찬가지일 수 있습니다.

오류 코드	오류 코드 이름	가능한 근본 원인	문제 해결
205	FlashOrRunTestError	IDT FreeRTOS가 DUT에서 FreeRTOS를 플래시하거나 실행할 수 없습니다.	`userdata.json` 파일에서 `flashTool` 아래 있는 정보가 올바른지 확인합니다. 자세한 내용은 빌드, 플래시 및 테스트 설정 구성 단원을 참조하십시오.
206	StartEchoServerError	IDT FreeRTO가 WiFi 또는 SecureSocket 테스트를 위해 에코 서버를 시작할 수 없습니다.	`userdata.json` 파일의 `echoServerConfiguration` 에 구성된 포트가 사용 중이 아닌지 또는 방화벽이나 네트워크 설정에 의해 차단되지 않았는지 확인합니다.

구문 분석 오류 디버깅

경우에 따라 JSON 구성의 오타로 인해 구문 분석 오류가 발생할 수 있습니다. 대부분의 경우 문제는 JSON 파일에서 대괄호, 쉼표 또는 따옴표를 생략한 결과입니다. FreeRTOS용 IDT는 JSON 확인을 수행하고 디버깅 정보를 인쇄합니다. IDT는 오류가 발생한 줄, 줄 번호, 구문 오류의 열 번호를 인쇄합니다. 이 정보만 있으면 오류를 수정할 수 있지만, 그래도 오류를 찾는 데 문제가 있는 경우 IDE, Atom이나 Sublime과 같은 텍스트 편집기에서 또는 JSONLint와 같은 온라인 도구를 통해 수동으로 확인을 수행할 수 있습니다.

무결성 검사 실패 디버깅

FreeRToIntegrity 테스트 그룹을 실행할 때 오류가 발생하면 먼저 *`freertos`* 디렉터리 파일을 수정하지 않았는지 확인합니다. 수정하지 않았는데도 여전히 문제가 발생하는 경우 올바른 브랜치를 사용하고 있는지 확인합니다. IDT의 `list-supported-products` 명령을 실행하면 *`freertos`* 리포지토리에서 어떤 태그가 지정된 브랜치를 사용해야 하는지 찾을 수 있습니다.

freertos 리포지토리에서 올바른 태그가 지정된 브랜치를 복제했는데도 여전히 문제가 있는 경우 submodule update 명령을 실행했는지 확인합니다. freertos 리포지토리의 복제 워크플로는 다음과 같습니다.

```
git clone --branch version-number https://github.com/aws/amazon-freertos.git
cd amazon-freertos
git submodule update --checkout –init –recursive
```

무결성 검사기가 찾는 파일 목록은 *freertos* 디렉터리의 checksums.json 파일에 있습니다. 파일 및 폴더 구조를 수정하지 않고 FreeRTOS 포트를 검증하려면 checksums.json 파일의 'exhaustive' 및 'minimal' 섹션에 나열된 파일이 수정되지 않았는지 확인합니다. 구성된 SDK를 사용하여 실행하려면 'minimal' 섹션 아래의 파일이 수정되지 않았는지 확인합니다.

SDK를 사용하여 IDT를 실행하고 *freertos* 디렉터리에서 일부 파일을 수정한 경우 userdata 파일에 SDK를 올바르게 구성했는지 확인합니다. 그렇지 않으면 무결성 검사기가 *freertos* 디렉터리의 모든 파일을 검사합니다.

FullWiFi 테스트 그룹 실패 디버깅

FullWiFi 테스트 그룹에서 실패가 표시되고 'AFQP_WiFiConnectMultipleAP' 테스트가 실패하는 경우 이는 두 액세스 포인트가 IDT를 실행하는 호스트 컴퓨터와 동일한 서브넷에 있지 않기 때문일 수 있습니다. 두 액세스 포인트 모두 IDT를 실행하는 호스트 컴퓨터와 동일한 서브넷에 있는지 확인합니다.

"필수 파라미터 누락" 오류 디버깅

FreeRTOS용 IDT에 새로운 기능이 추가되고 있으므로 구성 파일에 대한 변경 사항이 도입될 수 있습니다. 기존 구성 파일을 사용하면 구성이 손상될 수 있습니다. 이러한 문제가 발생할 경우, results/*execution-id*/logs 디렉터리 아래의 *test_group_id*__*test_case_id*.log 파일에 모든 누락 파라미터가 명시적으로 나열됩니다. 또한 FreeRTOS용 IDT는 JSON 구성 파일 스키마를 검사하여 지원되는 최신 버전이 사용되었는지 확인합니다.

"테스트를 시작할 수 없음" 오류 디버깅

테스트 시작 중에 실패를 가리키는 오류가 발생할 수 있습니다. 이 오류에는 여러 원인이 있을 수 있으므로 다음 영역이 정확한지 확인하십시오.

- 실행 명령에 포함된 풀 이름이 실제로 존재하는지 확인합니다. 이 이름은 device.json 파일에서 직접 참조됩니다.
- 풀에 있는 디바이스에 올바른 구성 파라미터가 있는지 확인합니다.

“리소스 액세스 권한 없음” 오류 디버깅

터미널 출력 또는 `/results/execution-id/logs` 아래 `test_manager.log` 파일에 “*###/##*이 이 리소스에 액세스할 권한이 없습니다.”라는 오류가 표시될 수 있습니다. 이 문제를 해결하려면 `AWSIoTDeviceTesterForFreeRTOSFullAccess` 관리형 정책을 테스트 사용자에게 연결합니다. 자세한 내용은 ??? 단원을 참조하십시오.

네트워크 테스트 오류 디버깅

네트워크 기반 테스트의 경우 IDT는 호스트 시스템의 비 예약 포트에 바인딩하는 에코 서버를 시작합니다. WiFi 또는 SecureSocket 테스트에서 제한 시간 또는 사용할 수 없는 연결로 인해 오류가 발생하는 경우, 네트워크가 1024 - 49151 범위로 구성된 포트에 대한 트래픽을 허용하도록 구성되어 있는지 확인하십시오.

SecureSocket 테스트는 기본적으로 포트 33333 및 33334를 사용합니다. WiFi 테스트는 기본적으로 포트 33335를 사용합니다. 이 3개의 포트가 사용 중이거나 방화벽 또는 네트워크에 의해 차단된 경우 userdata.json의 다른 포트를 테스트에 사용하도록 선택할 수 있습니다. 자세한 내용은 빌드, 플래시 및 테스트 설정 구성 단원을 참조하십시오. 다음 명령을 사용하여 특정 포트가 사용 중인지 확인할 수 있습니다.

- Windows: `netsh advfirewall firewall show rule name=all | grep port`
- Linux: `sudo netstat -pan | grep port`
- macOS: `netstat -nat | grep port`

동일한 버전 페이로드로 인해 OTA 업데이트 실패

OTA가 수행된 후 디바이스에 동일한 버전이 있어 OTA 테스트 사례가 실패한 경우 빌드 시스템(예: cmake)이 FreeRTOS 소스 코드에 대한 IDT의 변경 사항을 알지 못하고 업데이트된 바이너리를 빌드하지 않았기 때문일 수 있습니다. 이로 인해 OTA가 현재 디바이스에 있는 동일한 바이너리로 수행되고 테스트가 실패합니다. OTA 업데이트 실패에 대한 문제를 해결하려면 먼저 지원되는 최신 버전의 빌드 시스템을 사용하고 있는지 확인하십시오.

`PresignedUrlExpired` 테스트 케이스에 대한 OTA 테스트 실패

이 테스트의 한 가지 전제 조건은 OTA 업데이트 시간이 60초 이상이어야 한다는 것입니다. 그렇지 않으면 테스트가 실패합니다. 이 경우 로그에 "Test takes less than 60 seconds (url expired time) to finish. Please reach out to us." 오류 메시지가 표시됩니다.

디바이스 인터페이스 및 포트 오류 디버깅

이 단원에는 IDT가 디바이스에 연결하기 위해 사용하는 디바이스 인터페이스에 대한 정보가 포함되어 있습니다.

지원하는 플랫폼

IDT는 Linux, macOS 및 Windows를 지원합니다. 세 가지 플랫폼은 연결되는 직렬 디바이스에 대한 이름 지정 체계가 모두 다릅니다.

- Linux: `/dev/tty*`
- macOS: `/dev/tty.*` 또는 `/dev/cu.*`
- Windows: COM*

디바이스 포트를 확인하려면,

- Linux/macOS의 경우 터미널을 열고 `ls /dev/tty*`를 실행합니다.
- macOS의 경우 터미널을 열고 `ls /dev/tty.*` 또는 `ls /dev/cu.*`를 실행합니다.
- Windows의 경우 Device Manager를 열고 직렬 디바이스 그룹을 확장합니다.

포트에 연결된 디바이스를 확인하려면,

- Linux의 경우 udev 패키지가 설치되었는지 확인하고 `udevadm info -name=`*`PORT`*를 실행합니다. 이 유틸리티는 올바른 포트를 사용하고 있는지 확인하는 데 도움이 되는 디바이스 드라이버 정보를 인쇄합니다.
- macOS의 경우 Launchpad를 열고 **`System Information`**을 검색합니다.
- Windows의 경우 Device Manager를 열고 직렬 디바이스 그룹을 확장합니다.

디바이스 인터페이스

포함된 각 디바이스가 다르므로 하나 이상의 직렬 포트가 있을 수 있습니다. 일반적으로 시스템에 연결될 때 디바이스에는 두 개의 포트가 있습니다.

- 디바이스를 플래시하기 위한 데이터 포트
- 출력을 읽기 위한 읽기 포트.

device.json 파일에서 올바른 읽기 포트를 설정해야 합니다. 그렇지 않으면 디바이스에서 출력을 읽지 못할 수 있습니다.

여러 포트가 있는 경우 device.json 파일에서 디바이스의 읽기 포트를 사용하는지 확인합니다. 예를 들어, Espressif WRover 디바이스를 플러그인하고 이 디바이스에 할당된 두 개의 포트가 /dev/ttyUSB0 및 /dev/ttyUSB1인 경우 device.json 파일에서 /dev/ttyUSB1을 사용합니다.

Windows의 경우 동일한 로직을 따릅니다.

디바이스 데이터 읽기

FreeRTOS용 IDT는 개별 디바이스 빌드 및 플래시 도구를 사용하여 포트 구성을 지정합니다. 디바이스를 테스트하는 동안 출력을 얻을 수 없는 경우 다음 기본 설정을 사용해 봅니다.

- 전송 속도: 115200
- 데이터 비트: 8
- 패리티: 없음
- 정지 비트: 1
- 흐름 제어: 없음

이러한 설정은 FreeRTOS용 IDT에서 처리됩니다. 직접 설정할 필요가 없습니다. 하지만 동일한 방법을 사용하여 디바이스 출력을 수동으로 읽을 수 있습니다. Linux 또는 macOS에서는 screen 명령을 사용하여 이 작업을 수행할 수 있습니다. Windows에서는 TeraTerm과 같은 프로그램을 사용할 수 있습니다.

```
Screen: screen /dev/cu.usbserial 115200
```

```
TeraTerm: Use the above-provided settings to set the fields explicitly in
the GUI.
```

개발 도구 체인 문제

이 단원에서는 도구 체인에 발생할 수 있는 문제를 설명합니다.

Ubuntu의 Code Composer Studio

최신 버전의 Ubuntu(17.10 및 18.04)에는 Code Composer Studio 7.x 버전과 호환되지 않는 glibc 패키지 버전이 있습니다. Code Composer Studio 버전 8.2 이상을 설치하는 것이 좋습니다.

비호환성을 나타내는 증상에는 다음이 포함될 수 있습니다.

- FreeRTOS가 디바이스에 빌드하거나 플래시하지 못합니다.
- Code Composer Studio 설치 관리자가 동결될 수 있습니다.
- 빌드 또는 플래시 프로세스 중에 로그 출력이 콘솔에 표시되지 않습니다.
- 헤드리스로 호출하더라도 빌드 명령이 GUI 모드에서 시작하려고 시도합니다.

로깅

FreeRTOS용 IDT 로그는 단일 위치에 배치됩니다. 루트 IDT 디렉터리에서 다음과 같은 파일을 `results/`*`execution-id`*`/`에서 사용할 수 있습니다.

- `FRQ_Report.xml`
- `awsiotdevicetester_report.xml`
- `logs/`*`test_group_id`*`__`*`test_case_id`*`.log`

`FRQ_Report.xml` 및 `logs/`*`test_group_id`*`__`*`test_case_id`*`.log`는 검사해야 하는 가장 중요한 로그입니다. `FRQ_Report.xml`에는 특정 오류 메시지와 함께 실패한 테스트 케이스에 대한 정보가 포함되어 있습니다. 그런 다음 컨텍스트를 더 잘 이해하기 위해 `logs/`*`test_group_id`*`__`*`test_case_id`*`.log`를 사용하여 문제를 심층 분석할 수 있습니다.

콘솔 오류

AWS IoT Device Tester 가 실행되면 오류가 간략한 메시지와 함께 콘솔에 보고됩니다. 오류에 대해 자세히 알아보려면 `results/`*`execution-id`*`/logs/`*`test_group_id`*`__`*`test_case_id`*`.log`를 살펴봅니다.

오류 로그

각 테스트 제품군 실행에는 `results/`*`execution-id`*라는 폴더를 생성하는 데 사용되는 고유 실행 ID가 있습니다. 개별 테스트 케이스 로그는 `results/`*`execution-id`*`/logs` 디렉터리에 있습니다. FreeRTOS용 IDT 콘솔의 출력을 사용해 실행 ID, 테스트 사례 ID, 실패한 테스트 사례의 테스트 그룹 ID를 찾습니다. 그런 다음이 정보를 사용하여 라는 테스트 사례에 대한 로그 파일을 찾아 엽니다`results/`*`execution-id`*`/logs/`*`test_group_id`*`__`*`test_case_id`*`.log`.이 파일의 정보에는 전체 빌드 및 플래시 명령 출력, 테스트 실행 출력 및 더 자세한 AWS IoT Device Tester 콘솔 출력이 포함됩니다.

S3 버킷 문제

IDT를 실행하는 동안 CTRL+C 키를 누르면 IDT가 정리 프로세스를 시작합니다. 이 정리 작업 중에 IDT 테스트의 일환으로 생성된 Amazon S3 리소스가 제거됩니다. 정리를 완료할 수 없는 경우 생성된 Amazon S3 버킷이 너무 많아 문제가 발생한 것일 수 있습니다. 즉, 다음에 IDT를 실행하면 테스트가 실패하기 시작합니다.

CTRL+C 키를 눌러 IDT를 중지한 경우 정리 프로세스가 완료되도록 해야 이 문제가 발생하지 않습니다. 수동으로 생성한 Amazon S3 버킷을 계정에서 삭제할 수도 있습니다.

제한 시간 오류 해결

테스트 제품군이 실행되는 동안 제한 시간 오류가 발생하면 제한 시간 승수 계수를 지정하여 제한 시간을 늘립니다. 이 계수는 기본 제한 시간 값에 적용됩니다. 이 플래그에 대해 구성된 값은 1.0보다 크거나 같아야 합니다. 제한 시간 승수를 사용하려면 테스트 제품군을 실행할 때 `--timeout-multiplier` 플래그를 사용합니다.

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
```

셀룰러 기능 및 AWS 요금

`device.JSON` 파일`Yes`에서 `Cellular` 기능이 로 설정된 경우 FullSecureSockets는 테스트를 실행하는 데 t.micro EC2 인스턴스를 사용하므로 AWS 계정에 추가 비용이 발생할 수 있습니다. 자세한 설명은 Amazon EC2 요금을 참조하세요.

검증 보고서 생성 정책

검증 보고서는 지난 2년 이내에 릴리스된 FreeRTOS 버전을 지원하는 AWS IoT Device Tester (IDT) 버전에서만 생성됩니다. 지원 정책에 대해 궁금한 점이 있으면 AWS Support에 문의하세요.

FreeRTOS AWS IoT Device Tester 용 지원 정책

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

AWS IoT Device Tester FreeRTOS용는 AWS 파트너 디바이스 카탈로그에 포함할 FreeRTOS 디바이스를 검증하고 검증하는 테스트 자동화 도구입니다. 최신 버전의 FreeRTOS 및 AWS IoT 디바이스 테스터를 사용하여 디바이스를 테스트하거나 검증하는 것이 좋습니다. 최신 버전의 FreeRTOS 및 이전 6개월 이내에 릴리스된 FreeRTOS 버전에 대해 AWS IoT 디바이스 테스터를 지원합니다. FreeRTOS의 LTS 기반 버전을 AWS IoT Device Tester 지원하는 버전은 2년 동안 지원됩니다. 현재 FreeRTOS 202012.00을 AWS IoT Device Tester 지원하는 버전만 2년 동안 지원됩니다. FreeRTOS의 최신 버전은 GitHub에서 사용할 수 있습니다. 지원되는 AWS IoT 디바이스 테스터 버전은 섹션을 참조하세요FreeRTOS AWS IoT Device Tester 용의 지원되는 버전.

IDT 프레임워크의 각 버전에서는 디바이스 검증을 위해 세 가지 버전의 테스트 제품군이 지원됩니다.

지원되는 버전 중 하나를 해당 버전의 FreeRTOS AWS IoT Device Tester 와 함께 사용하여 디바이스를 테스트하거나 검증할 수도 있습니다. FreeRTOS용 IDT의 지원되지 않는 버전은 계속 사용할 수 있지만 버그 수정 또는 업데이트가 제공되지 않습니다.

지원 정책에 대해 궁금한 점이 있는 경우 AWS 고객 지원 센터에 문의하십시오.

의 보안 AWS

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

의 클라우드 보안 AWS 이 최우선 순위입니다. AWS 고객은 보안에 가장 민감한 조직의 요구 사항을 충족하도록 구축된 데이터 센터 및 네트워크 아키텍처의 이점을 누릴 수 있습니다.

보안은 AWS 와 사용자 간의 공동 책임입니다. 공동 책임 모델은 이 사항을 클라우드 내 보안 및 클라우드의 보안으로 설명합니다.

- 클라우드 보안 - AWS 는 AWS 클라우드에서 AWS 서비스를 실행하는 인프라를 보호할 책임이 있습니다. AWS 또한는 안전하게 사용할 수 있는 서비스를 제공합니다. 서드 파티 감사자는 정기적으로 AWS 규정 준수 프로그램의 일환으로 보안 효과를 테스트하고 검증합니다. AWS 서비스에 적용되는 규정 준수 프로그램에 대한 자세한 내용은 AWS 규정 준수 프로그램 제공 범위 내 서비스를 참조하세요.
- 클라우드의 보안 - 사용자의 책임은 사용하는 AWS 서비스에 따라 결정됩니다. 또한 데이터의 민감도, 조직의 요건 및 관련 법률 및 규정을 비롯한 기타 요소에 대해서도 책임이 있습니다.

이 설명서는를 사용할 때 공동 책임 모델을 적용하는 방법을 이해하는 데 도움이 됩니다 AWS. 다음 주제에서는 보안 및 규정 준수 목표를 충족하도록 AWS 를 구성하는 방법을 보여줍니다. 또한 AWS 리소스를 모니터링하고 보호하는 데 도움이 되는 AWS 서비스를 사용하는 방법을 알아봅니다.

AWS IoT 보안에 대한 자세한 내용은 보안 및 자격 증명을 참조하세요 AWS IoT.

주제

- FreeRTOS의 Identity and Access Management
- 규정 준수 확인
- 의 복원력 AWS
- FreeRTOS의 인프라 보안

FreeRTOS의 Identity and Access Management

AWS Identity and Access Management (IAM)는 관리자가 AWS 리소스에 대한 액세스를 안전하게 제어하는 데 도움이 AWS 서비스 되는 입니다. IAM 관리자는 누가 FreeRTOS 리소스를 사용하도록 인증되고(로그인됨) 권한 부여되는지(권한 있음)를 제어합니다. IAM은 추가 비용 없이 사용할 수 AWS 서비스 있는 입니다.

주제

- 대상
- ID를 통한 인증
- 정책을 사용하여 액세스 관리
- FreeRTOS에서 IAM을 사용하는 방법
- FreeRTOS 자격 증명 기반 정책 예제
- FreeRTOS 자격 증명 및 액세스 문제 해결

대상

AWS Identity and Access Management (IAM)를 사용하는 방법은 FreeRTOS에서 수행하는 작업에 따라 다릅니다.

서비스 사용자 - FreeRTOS 서비스를 사용하여 작업을 수행하는 경우 필요한 보안 인증 정보와 권한을 관리자가 제공합니다. 더 많은 FreeRTOS 기능을 사용하여 작업을 수행하게 되면 추가 권한이 필요할 수 있습니다. 액세스 권한 관리 방법을 이해하면 관리자에게 올바른 권한을 요청하는 데 도움이 됩니다. FreeRTOS의 기능에 액세스할 수 없는 경우 FreeRTOS 자격 증명 및 액세스 문제 해결 단원을 참조하세요.

서비스 관리자 - 회사에서 FreeRTOS 리소스를 책임지고 있는 경우 FreeRTOS에 대한 전체 액세스 권한을 가지고 있을 것입니다. 서비스 관리자는 서비스 사용자가 액세스해야 하는 FreeRTOS 기능 및 리소스를 결정합니다. 그런 다음 IAM 관리자에게 요청을 제출하여 서비스 사용자의 권한을 변경해야 합니다. 이 페이지의 정보를 검토하여 IAM의 기본 개념을 이해하세요. 회사가 FreeRTOS에서 IAM을 사용하는 방법에 대해 자세히 알아보려면 FreeRTOS에서 IAM을 사용하는 방법 단원을 참조하세요.

IAM 관리자 - IAM 관리자라면 FreeRTOS에 대한 액세스 권한 관리 정책 작성 방법을 자세히 알아야 할 것입니다. IAM에서 사용할 수 있는 FreeRTOS 자격 증명 기반 정책 예제를 보려면 FreeRTOS 자격 증명 기반 정책 예제 단원을 참조하세요.

ID를 통한 인증

인증은 자격 증명 자격 증명을 AWS 사용하여에 로그인하는 방법입니다. , AWS 계정 루트 사용자 IAM 사용자 또는 IAM 역할을 수임하여 인증(로그인 AWS)되어야 합니다.

자격 증명 소스를 통해 제공된 자격 증명을 사용하여 페더레이션 자격 증명 AWS 으로에 로그인할 수 있습니다. AWS IAM Identity Center (IAM Identity Center) 사용자, 회사의 Single Sign-On 인증 및 Google 또는 Facebook 자격 증명은 페더레이션 자격 증명의 예입니다. 페더레이션형 ID로 로그인할 때 관리자가 이전에 IAM 역할을 사용하여 ID 페더레이션을 설정했습니다. 페더레이션을 사용하여 AWS 에 액세스하면 간접적으로 역할을 수임하게 됩니다.

사용자 유형에 따라 AWS Management Console 또는 AWS 액세스 포털에 로그인할 수 있습니다. 로그인에 대한 자세한 내용은 AWS 로그인 사용 설명서의에 로그인하는 방법을 AWS참조하세요. AWS 계정

AWS 프로그래밍 방식으로에 액세스하는 경우는 자격 증명을 사용하여 요청에 암호화 방식으로 서명할 수 있는 소프트웨어 개발 키트(SDK)와 명령줄 인터페이스(CLI)를 AWS 제공합니다. AWS 도구를 사용하지 않는 경우 직접 요청에 서명해야 합니다. 권장 방법을 사용하여 요청에 직접 서명하는 자세한 방법은 IAM 사용 설명서에서 API 요청용AWS Signature Version 4를 참조하세요.

사용하는 인증 방법에 상관없이 추가 보안 정보를 제공해야 할 수도 있습니다. 예를 들어는 멀티 팩터 인증(MFA)을 사용하여 계정의 보안을 강화할 것을 AWS 권장합니다. 자세한 내용은 AWS IAM Identity Center 사용 설명서에서 다중 인증 및 IAM 사용 설명서에서 IAM의AWS 다중 인증을 참조하세요.

AWS 계정 루트 사용자

를 생성할 때 계정의 모든 AWS 서비스 및 리소스에 대한 완전한 액세스 권한이 있는 하나의 로그인 자격 증명으로 AWS 계정시작합니다. 이 자격 증명을 AWS 계정 루트 사용자라고 하며 계정을 생성하는 데 사용한 이메일 주소와 암호로 로그인하여 액세스합니다. 일상적인 작업에 루트 사용자를 사용하지 않을 것을 강력히 권장합니다. 루트 사용자 자격 증명을 보호하고 루트 사용자만 수행할 수 있는 작업을 수행하는 데 사용합니다. 루트 사용자로 로그인해야 하는 전체 작업 목록은 IAM 사용 설명서의 루트 사용자 보안 인증이 필요한 작업을 참조하세요.

페더레이션 자격 증명

가장 좋은 방법은 관리자 액세스가 필요한 사용자를 포함한 인간 사용자에게 자격 증명 공급자와의 페더레이션을 사용하여 임시 자격 증명을 사용하여 AWS 서비스 에 액세스하도록 요구하는 것입니다.

페더레이션 자격 증명은 엔터프라이즈 사용자 디렉터리, 웹 자격 증명 공급자, AWS Directory Service, Identity Center 디렉터리 또는 자격 증명 소스를 통해 제공된 자격 증명을 사용하여 AWS 서비스 에 액

세스하는 모든 사용자의 사용자입니다. 페더레이션 자격 증명에 액세스할 때 역할을 AWS 계정수임하고 역할은 임시 자격 증명을 제공합니다.

중앙 집중식 액세스 관리를 위해 AWS IAM Identity Center을(를) 사용하는 것이 좋습니다. IAM Identity Center에서 사용자 및 그룹을 생성하거나 모든 및 애플리케이션에서 사용할 수 있도록 자체 ID 소스의 사용자 AWS 계정 및 그룹 집합에 연결하고 동기화할 수 있습니다. IAM Identity Center에 대한 자세한 내용은 AWS IAM Identity Center 사용 설명서에서 IAM Identity Center란 무엇인가요?를 참조하세요.

IAM 사용자 및 그룹

IAM 사용자는 단일 사용자 또는 애플리케이션에 대한 특정 권한이 AWS 계정 있는 내의 자격 증명입니다. 가능하면 암호 및 액세스 키와 같은 장기 자격 증명이 있는 IAM 사용자를 생성하는 대신 임시 자격 증명을 사용하는 것이 좋습니다. 하지만 IAM 사용자의 장기 자격 증명이 필요한 특정 사용 사례가 있는 경우, 액세스 키를 교체하는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 장기 보안 인증이 필요한 사용 사례의 경우, 정기적으로 액세스 키 교체를 참조하세요.

IAM 그룹은 IAM 사용자 컬렉션을 지정하는 자격 증명입니다. 사용자는 그룹으로 로그인할 수 없습니다. 그룹을 사용하여 여러 사용자의 권한을 한 번에 지정할 수 있습니다. 그룹을 사용하면 대규모 사용자 집합의 권한을 더 쉽게 관리할 수 있습니다. 예를 들어, IAMAdmins라는 그룹이 있고 이 그룹에 IAM 리소스를 관리할 권한을 부여할 수 있습니다.

사용자는 역할과 다릅니다. 사용자는 한 사람 또는 애플리케이션과 고유하게 연결되지만, 역할은 해당 역할이 필요한 사람이라면 누구나 수임할 수 있습니다. 사용자는 영구적인 장기 자격 증명을 가지고 있지만, 역할은 임시 보안 인증만 제공합니다. 자세한 내용은 IAM 사용 설명서에서 IAM 사용자 사용 사례를 참조하세요.

IAM 역할

IAM 역할은 특정 권한이 AWS 계정 있는 내의 자격 증명입니다. IAM 사용자와 유사하지만, 특정 개인과 연결되지 않습니다. 에서 IAM 역할을 일시적으로 수임하려면 사용자에서 IAM 역할(콘솔)로 전환할 AWS Management Console수 있습니다. https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_use_switch-role-console.html 또는 AWS API 작업을 호출하거나 사용자 지정 URL을 AWS CLI 사용하여 역할을 수임할 수 있습니다. 역할 사용 방법에 대한 자세한 내용은 IAM 사용 설명서의 역할 수임 방법을 참조하세요.

임시 보안 인증이 있는 IAM 역할은 다음과 같은 상황에서 유용합니다.

- 페더레이션 사용자 액세스 - 페더레이션 ID에 권한을 부여하려면 역할을 생성하고 해당 역할의 권한을 정의합니다. 페더레이션 ID가 인증되면 역할이 연결되고 역할에 정의된 권한이 부여됩니다. 페

더레이션 관련 역할에 대한 자세한 내용은 IAM 사용 설명서의 Create a role for a third-party identity provider (federation)를 참조하세요. IAM Identity Center를 사용하는 경우, 권한 집합을 구성합니다. 인증 후 ID가 액세스할 수 있는 항목을 제어하기 위해 IAM Identity Center는 권한 집합을 IAM의 역할과 연관짓습니다. 권한 집합에 대한 자세한 내용은 AWS IAM Identity Center 사용 설명서의 권한 집합을 참조하세요.

- 임시 IAM 사용자 권한 - IAM 사용자 또는 역할은 IAM 역할을 수임하여 특정 작업에 대한 다양한 권한을 임시로 받을 수 있습니다.
- 교차 계정 액세스 - IAM 역할을 사용하여 다른 계정의 사용자(신뢰할 수 있는 보안 주체)가 내 계정의 리소스에 액세스하도록 허용할 수 있습니다. 역할은 계정 간 액세스를 부여하는 기본적인 방법입니다. 그러나 일부 에서는 정책을 리소스에 직접 연결할 AWS 서비스수 있습니다(역할을 프록시로 사용하는 대신). 교차 계정 액세스에 대한 역할과 리소스 기반 정책의 차이점을 알아보려면 IAM 사용 설명서의 IAM의 교차 계정 리소스 액세스를 참조하세요.
- 교차 서비스 액세스 - 일부는 다른에서 기능을 AWS 서비스 사용합니다 AWS 서비스. 예를 들어, 서비스에서 호출하면 일반적으로 해당 서비스는 Amazon EC2에서 애플리케이션을 실행하거나 Amazon S3에 객체를 저장합니다. 서비스는 직접적으로 호출하는 위탁자의 권한을 사용하거나, 서비스 역할을 사용하거나, 또는 서비스 연결 역할을 사용하여 이 작업을 수행할 수 있습니다.
  - 전달 액세스 세션(FAS) - IAM 사용자 또는 역할을 사용하여에서 작업을 수행하는 경우 AWS보안 주체로 간주됩니다. 일부 서비스를 사용하는 경우, 다른 서비스에서 다른 작업을 시작하는 작업을 수행할 수 있습니다. FAS는를 호출하는 보안 주체의 권한을 다운스트림 서비스에 AWS 서비스 대한 요청과 AWS 서비스함께 사용합니다. FAS 요청은 서비스가 완료하기 위해 다른 AWS 서비스 또는 리소스와의 상호 작용이 필요한 요청을 수신할 때만 이루어집니다. 이 경우, 두 작업을 모두 수행할 수 있는 권한이 있어야 합니다. FAS 요청 시 정책 세부 정보는 전달 액세스 세션을 참조하세요.
  - 서비스 역할 - 서비스 역할은 서비스가 사용자를 대신하여 작업을 수행하기 위해 맡는 IAM 역할입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 정보는 IAM 사용 설명서의 Create a role to delegate permissions to an AWS 서비스를 참조하세요.
  - 서비스 연결 역할 - 서비스 연결 역할은에 연결된 서비스 역할의 한 유형입니다 AWS 서비스. 서비스는 사용자를 대신하여 작업을 수행하기 위해 역할을 수임할 수 있습니다. 서비스 연결 역할은에 표시 AWS 계정 되며 서비스가 소유합니다. IAM 관리자는 서비스 링크 역할의 권한을 볼 수 있지만 편집은 할 수 없습니다.
- Amazon EC2에서 실행되는 애플리케이션 - IAM 역할을 사용하여 EC2 인스턴스에서 실행되고 AWS CLI 또는 AWS API 요청을 수행하는 애플리케이션의 임시 자격 증명을 관리할 수 있습니다. 이는 EC2 인스턴스 내에 액세스 키를 저장할 때 권장되는 방법입니다. EC2 인스턴스에 AWS 역할을 할당하고 모든 애플리케이션에서 사용할 수 있도록 하려면 인스턴스에 연결된 인스턴스 프로파일을 생성합니다. 인스턴스 프로필에는 역할이 포함되어 있으며 EC2 인스턴스에서 실행되는 프로그

램이 임시 보안 인증을 얻을 수 있습니다. 자세한 정보는 IAM 사용 설명서의 IAM 역할을 사용하여 Amazon EC2 인스턴스에서 실행되는 애플리케이션에 권한 부여를 참조하세요.

정책을 사용하여 액세스 관리

정책을 AWS 생성하고 자격 증명 또는 리소스에 연결하여 AWS 에서 액세스를 제어합니다. 정책은 자격 증명 또는 리소스와 연결될 AWS 때 권한을 정의하는의 객체입니다.는 보안 주체(사용자, 루트 사용자 또는 역할 세션)가 요청할 때 이러한 정책을 AWS 평가합니다. 정책에서 권한은 요청이 허용되거나 거부되는 지를 결정합니다. 대부분의 정책은에 JSON 문서 AWS 로 저장됩니다. JSON 정책 문서의 구조와 콘텐츠에 대한 자세한 정보는 IAM 사용 설명서의 JSON 정책 개요를 참조하세요.

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

기본적으로, 사용자 및 역할에는 어떠한 권한도 없습니다. 사용자에게 사용자가 필요한 리소스에서 작업을 수행할 권한을 부여하려면 IAM 관리자가 IAM 정책을 생성하면 됩니다. 그런 다음 관리자가 IAM 정책을 역할에 추가하고, 사용자가 역할을 수임할 수 있습니다.

IAM 정책은 작업을 수행하기 위해 사용하는 방법과 상관없이 작업에 대한 권한을 정의합니다. 예를 들어, `iam:GetRole` 작업을 허용하는 정책이 있다고 가정합니다. 해당 정책이 있는 사용자는 AWS Management Console AWS CLI, 또는 API에서 역할 정보를 가져올 수 있습니다 AWS .

ID 기반 정책

ID 기반 정책은 IAM 사용자, 사용자 그룹 또는 역할과 같은 ID에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 사용자 및 역할이 어떤 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 제어합니다. 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서에서 고객 관리형 정책으로 사용자 지정 IAM 권한 정의를 참조하세요.

ID 기반 정책은 인라인 정책 또는 관리형 정책으로 한층 더 분류할 수 있습니다. 인라인 정책은 단일 사용자, 그룹 또는 역할에 직접 포함됩니다. 관리형 정책은의 여러 사용자, 그룹 및 역할에 연결할 수 있는 독립 실행형 정책입니다 AWS 계정. 관리형 정책에는 AWS 관리형 정책 및 고객 관리형 정책이 포함됩니다. 관리형 정책 또는 인라인 정책을 선택하는 방법을 알아보려면 IAM 사용 설명서의 관리형 정책 및 인라인 정책 중에서 선택을 참조하세요.

리소스 기반 정책

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 리소스 기반 정책의 예제는 IAM 역할 신뢰 정책과 Amazon S3 버킷 정책입니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자

는 이러한 정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 정책이 연결된 리소스의 경우 정책은 지정된 위탁자가 해당 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 정의합니다. 리소스 기반 정책에서 위탁자를 지정해야 합니다. 보안 주체에는 계정, 사용자, 역할, 페더레이션 사용자 또는가 포함될 수 있습니다 AWS 서비스.

리소스 기반 정책은 해당 서비스에 있는 인라인 정책입니다. 리소스 기반 정책에서는 IAM의 AWS 관리형 정책을 사용할 수 없습니다.

액세스 제어 목록(ACL)

액세스 제어 목록(ACL)은 어떤 보안 주체(계정 멤버, 사용자 또는 역할)가 리소스에 액세스할 수 있는 권한을 가지고 있는지를 제어합니다. ACL은 JSON 정책 문서 형식을 사용하지 않지만 리소스 기반 정책과 유사합니다.

Amazon S3 AWS WAF및 Amazon VPC는 ACLs. ACL에 관한 자세한 내용은 Amazon Simple Storage Service 개발자 가이드의 액세스 제어 목록(ACL) 개요를 참조하세요.

기타 정책 타입

AWS 는 덜 일반적인 추가 정책 유형을 지원합니다. 이러한 정책 타입은 더 일반적인 정책 유형에 따라 사용자에게 부여되는 최대 권한을 설정할 수 있습니다.

- 권한 경계 – 권한 경계는 ID 기반 정책에 따라 IAM 엔터티(IAM 사용자 또는 역할)에 부여할 수 있는 최대 권한을 설정하는 고급 기능입니다. 개체에 대한 권한 경계를 설정할 수 있습니다. 그 결과로 얻는 권한은 객체의 ID 기반 정책과 그 권한 경계의 교집합입니다. `Principal` 필드에서 사용자나 역할을 지정하는 리소스 기반 정책은 권한 경계를 통해 제한되지 않습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 권한 경계에 대한 자세한 정보는 IAM 사용 설명서의 IAM 엔티티에 대한 권한 경계를 참조하세요.
- 서비스 제어 정책(SCPs) - SCPs는의 조직 또는 조직 단위(OU)에 대한 최대 권한을 지정하는 JSON 정책입니다 AWS Organizations. AWS Organizations 는 비즈니스가 소유 AWS 계정 한 여러를 그룹화하고 중앙에서 관리하기 위한 서비스입니다. 조직에서 모든 기능을 활성화할 경우, 서비스 제어 정책(SCP)을 임의의 또는 모든 계정에 적용할 수 있습니다. SCP는 각각을 포함하여 멤버 계정의 엔터티에 대한 권한을 제한합니다 AWS 계정 루트 사용자. 조직 및 SCP에 대한 자세한 내용은 AWS Organizations 사용 설명서에서 Service control policies을 참조하세요.
- 리소스 제어 정책(RCP) - RCP는 소유한 각 리소스에 연결된 IAM 정책을 업데이트하지 않고 계정의 리소스에 대해 사용 가능한 최대 권한을 설정하는 데 사용할 수 있는 JSON 정책입니다. RCP는 멤버 계정의 리소스에 대한 권한을 제한하며 조직에 속하는지 여부에 AWS 계정 루트 사용자관계없이를 포함한 자격 증명에 대한 유효 권한에 영향을 미칠 수 있습니다. RCP를 AWS 서비스 지원하는 목록

을 포함하여 조직 및 RCPs에 대한 자세한 내용은 AWS Organizations 사용 설명서의 리소스 제어 정책(RCPs)을 참조하세요.

- 세션 정책 – 세션 정책은 역할 또는 페더레이션 사용자에 대해 임시 세션을 프로그래밍 방식으로 생성할 때 파라미터로 전달하는 고급 정책입니다. 결과적으로 얻는 세션의 권한은 사용자 또는 역할의 ID 기반 정책의 교차와 세션 정책입니다. 또한 권한을 리소스 기반 정책에서 가져올 수도 있습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 자세한 정보는 IAM 사용 설명서의 세션 정책을 참조하세요.

여러 정책 유형

여러 정책 유형이 요청에 적용되는 경우, 결과 권한은 이해하기가 더 복잡합니다. 에서 여러 정책 유형이 관련될 때 요청을 허용할지 여부를 AWS 결정하는 방법을 알아보려면 IAM 사용 설명서의 정책 평가 로직을 참조하세요.

FreeRTOS에서 IAM을 사용하는 방법

IAM을 사용하여 FreeRTOS에 대한 액세스를 관리하기 전에 FreeRTOS와 함께 사용할 수 있는 IAM 기능을 알아봅니다.

FreeRTOS에서 사용할 수 있는 IAM 기능

IAM 기능	FreerTOS 지원
ID 기반 정책	예
리소스 기반 정책	아니요
정책 작업	예
정책 리소스	예
정책 조건 키(서비스별)	예
ACLs	아니요
ABAC(정책 내 태그)	부분
임시 자격 증명	예

IAM 기능	FreerTOS 지원
보안 주체 권한	예
서비스 역할	예
서비스 연결 역할	아니요

FreeRTOS 및 기타 AWS 서비스가 대부분의 IAM 기능과 작동하는 방식을 개괄적으로 알아보려면 IAM 사용 설명서의 AWS IAM으로 작업하는 서비스를 참조하세요.

FreeRTOS에서 사용되는 자격 증명 기반 정책

ID 기반 정책 지원: 예

ID 기반 정책은 IAM 사용자, 사용자 그룹 또는 역할과 같은 ID에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 사용자 및 역할이 어떤 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 제어합니다. 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서에서 고객 관리형 정책으로 사용자 지정 IAM 권한 정의를 참조하세요.

IAM ID 기반 정책을 사용하면 허용되거나 거부되는 작업과 리소스뿐 아니라 작업이 허용되거나 거부되는 조건을 지정할 수 있습니다. ID 기반 정책에서는 위탁자가 연결된 사용자 또는 역할에 적용되므로 위탁자를 지정할 수 없습니다. JSON 정책에서 사용하는 모든 요소에 대해 알아보려면 IAM 사용 설명서의 IAM JSON 정책 요소 참조를 참조하세요.

FreeRTOS 자격 증명 기반 정책 예제

FreeRTOS 자격 증명 기반 정책의 예를 보려면 FreeRTOS 자격 증명 기반 정책 예제 단원을 참조하세요.

FreeRTOS 내 리소스 기반 정책

리소스 기반 정책 지원: 아니요

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 리소스 기반 정책의 예제는 IAM 역할 신뢰 정책과 Amazon S3 버킷 정책입니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자는 이러한 정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 정책이 연결된 리소스의 경우 정책은 지정된 위탁자가 해당 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 정의합니

다. 리소스 기반 정책에서 위탁자를 지정해야 합니다. 보안 주체에는 계정, 사용자, 역할, 페더레이션 사용자 또는가 포함될 수 있습니다 AWS 서비스.

교차 계정 액세스를 활성화하려는 경우, 전체 계정이나 다른 계정의 IAM 개체를 리소스 기반 정책의 위탁자로 지정할 수 있습니다. 리소스 기반 정책에 크로스 계정 보안 주체를 추가하는 것은 트러스트 관계 설정의 절반밖에 되지 않는다는 것을 유념하세요. 보안 주체와 리소스가 다른 경우 신뢰할 수 AWS 계정있는 계정의 IAM 관리자는 보안 주체 엔터티(사용자 또는 역할)에게 리소스에 액세스할 수 있는 권한도 부여해야 합니다. 엔터티에 ID 기반 정책을 연결하여 권한을 부여합니다. 하지만 리소스 기반 정책이 동일 계정의 위탁자에 액세스를 부여하는 경우, 추가 자격 증명 기반 정책이 필요하지 않습니다. 자세한 내용은 IAM 사용 설명서의 교차 계정 리소스 액세스를 참조하세요.

FreeRTOS에서 사용되는 정책 작업

정책 작업 지원: 예

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 위탁자가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

JSON 정책의 `Action` 요소는 정책에서 액세스를 허용하거나 거부하는 데 사용할 수 있는 작업을 설명합니다. 정책 작업은 일반적으로 연결된 AWS API 작업과 이름이 동일합니다. 일치하는 API 작업이 없는 권한 전용 작업 같은 몇 가지 예외도 있습니다. 정책에서 여러 작업이 필요한 몇 가지 작업도 있습니다. 이러한 추가 작업을 일컬어 종속 작업이라고 합니다.

연결된 작업을 수행할 수 있는 권한을 부여하기 위한 정책에 작업을 포함하세요.

FreeRTOS 작업의 목록을 보려면 서비스 권한 부여 참조의 FreeRTOS에서 정의한 작업을 참조하세요.

FreeRTOS의 정책 작업은 작업 앞에 다음 접두사를 사용합니다.

```
awes
```

단일 문에서 여러 작업을 지정하려면 다음과 같이 쉼표로 구분합니다.

```
"Action": [
      "awes:action1",
      "awes:action2"
         ]
```

FreeRTOS 자격 증명 기반 정책의 예를 보려면 FreeRTOS 자격 증명 기반 정책 예제 단원을 참조하세요.

FreeRTOS용 정책 리소스

정책 리소스 지원: 예

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

`Resource` JSON 정책 요소는 작업이 적용되는 하나 이상의 객체를 지정합니다. 문에는 `Resource`또는 `NotResource`요소가 반드시 추가되어야 합니다. 모범 사례에 따라 Amazon 리소스 이름(ARN)을 사용하여 리소스를 지정합니다. 리소스 수준 권한이라고 하는 특정 리소스 유형을 지원하는 작업에 대해 이를 수행할 수 있습니다.

작업 나열과 같이 리소스 수준 권한을 지원하지 않는 작업의 경우, 와일드카드(*)를 사용하여 해당 문이 모든 리소스에 적용됨을 나타냅니다.

```
"Resource": "*"
```

FreeRTOS 리소스 유형 및 해당 ARN의 목록을 보려면 서비스 권한 부여 참조의 FreeRTOS에서 정의한 리소스를 참조하세요. 각 리소스의 ARN을 지정할 수 있는 작업을 알아보려면 FreeRTOS에서 정의한 작업을 참조하세요.

FreeRTOS 자격 증명 기반 정책의 예를 보려면 FreeRTOS 자격 증명 기반 정책 예제 단원을 참조하세요.

FreeRTOS에서 사용되는 정책 조건 키

서비스별 정책 조건 키 지원: 예

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

`Condition` 요소(또는 `Condition` 블록)를 사용하면 정책이 발효되는 조건을 지정할 수 있습니다. `Condition` 요소는 옵션입니다. 같거나 작음과 같은 조건 연산자를 사용하여 정책의 조건을 요청의 값과 일치시키는 조건식을 생성할 수 있습니다.

한 문에서 여러 Condition 요소를 지정하거나 단일 Condition 요소에서 여러 키를 지정하는 경우, AWS 는 논리적 AND 작업을 사용하여 평가합니다. 단일 조건 키에 여러 값을 지정하는 경우는 논리적 OR 작업을 사용하여 조건을 AWS 평가합니다. 문의 권한을 부여하기 전에 모든 조건을 충족해야 합니다.

조건을 지정할 때 자리 표시자 변수를 사용할 수도 있습니다. 예를 들어, IAM 사용자에게 IAM 사용자 이름으로 태그가 지정된 경우에만 리소스에 액세스할 수 있는 권한을 부여할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 IAM 정책 요소: 변수 및 태그를 참조하세요.

AWS 는 전역 조건 키와 서비스별 조건 키를 지원합니다. 모든 AWS 전역 조건 키를 보려면 IAM 사용 설명서의 AWS 전역 조건 컨텍스트 키를 참조하세요.

FreeRTOS 조건 키의 목록을 보려면 서비스 권한 부여 참조의 FreeRTOS에서 사용되는 조건 키를 참조하세요. 조건 키를 사용할 수 있는 작업 및 리소스를 알아보려면 FreeRTOS에서 정의한 작업을 참조하세요.

FreeRTOS 자격 증명 기반 정책의 예를 보려면 FreeRTOS 자격 증명 기반 정책 예제 단원을 참조하세요.

FreeRTOS의 ACL

ACL 지원: 아니요

액세스 제어 목록(ACL)은 어떤 위탁자(계정 멤버, 사용자 또는 역할)가 리소스에 액세스할 수 있는 권한을 가지고 있는지를 제어합니다. ACL은 JSON 정책 문서 형식을 사용하지 않지만 리소스 기반 정책과 유사합니다.

FreeRTOS에서의 ABAC

ABAC 지원(정책의 태그): 부분적

속성 기반 액세스 제어(ABAC)는 속성에 근거하여 권한을 정의하는 권한 부여 전략입니다. 에서는 AWS이러한 속성을 태그라고 합니다. IAM 엔터티(사용자 또는 역할) 및 많은 AWS 리소스에 태그를 연결할 수 있습니다. ABAC의 첫 번째 단계로 개체 및 리소스에 태그를 지정합니다. 그런 다음 위탁자의 태그가 액세스하려는 리소스의 태그와 일치할 때 작업을 허용하도록 ABAC 정책을 설계합니다.

ABAC는 빠르게 성장하는 환경에서 유용하며 정책 관리가 번거로운 상황에 도움이 됩니다.

태그에 근거하여 액세스를 제어하려면 aws:ResourceTag/*key-name*, aws:RequestTag/*key-name* 또는 aws:TagKeys 조건 키를 사용하여 정책의 조건 요소에 태그 정보를 제공합니다.

서비스가 모든 리소스 유형에 대해 세 가지 조건 키를 모두 지원하는 경우, 값은 서비스에 대해 예입니다. 서비스가 일부 리소스 유형에 대해서만 세 가지 조건 키를 모두 지원하는 경우, 값은 부분적입니다.

ABAC에 대한 자세한 내용은 IAM 사용 설명서의 ABAC 권한 부여를 통한 권한 정의를 참조하세요. ABAC 설정 단계가 포함된 자습서를 보려면 IAM 사용 설명서의 속성 기반 액세스 제어(ABAC) 사용을 참조하세요.

FreeRTOS에서 임시 보안 인증 정보 사용

임시 자격 증명 지원: 예

일부 AWS 서비스 는 임시 자격 증명을 사용하여 로그인할 때 작동하지 않습니다. 임시 자격 증명으로 AWS 서비스 작업하는를 포함한 추가 정보는 AWS 서비스 IAM 사용 설명서의 IAM으로 작업하는 섹션을 참조하세요.

사용자 이름과 암호를 제외한 방법을 AWS Management Console 사용하여에 로그인하는 경우 임시 자격 증명을 사용합니다. 예를 들어 회사의 SSO(Single Sign-On) 링크를 AWS 사용하여에 액세스하면 해당 프로세스가 임시 자격 증명을 자동으로 생성합니다. 또한 콘솔에 사용자로 로그인한 다음 역할을 전환할 때 임시 자격 증명을 자동으로 생성합니다. 역할 전환에 대한 자세한 내용은 IAM 사용 설명서의 사용자에서 IAM 역할로 전환(콘솔)을 참조하세요.

AWS CLI 또는 AWS API를 사용하여 임시 자격 증명을 수동으로 생성할 수 있습니다. 그런 다음 이러한 임시 자격 증명을 사용하여 장기 액세스 키를 사용하는 대신 임시 자격 증명을 동적으로 생성하는 access AWS. AWS recommends에 액세스할 수 있습니다. 자세한 정보는 IAM의 임시 보안 자격 증명 섹션을 참조하세요.

FreeRTOS에서 사용되는 교차 서비스 보안 주체 권한

전달 액세스 세션(FAS) 지원: 예

IAM 사용자 또는 역할을 사용하여에서 작업을 수행하는 경우 AWS보안 주체로 간주됩니다. 일부 서비스를 사용하는 경우, 다른 서비스에서 다른 작업을 시작하는 작업을 수행할 수 있습니다. FAS는를 호출하는 보안 주체의 권한을 다운스트림 서비스에 AWS 서비스 대한 요청과 AWS 서비스함께 사용합니다. FAS 요청은 서비스가 완료하기 위해 다른 AWS 서비스 또는 리소스와의 상호 작용이 필요한 요청을 수신할 때만 이루어집니다. 이 경우, 두 작업을 모두 수행할 수 있는 권한이 있어야 합니다. FAS 요청 시 정책 세부 정보는 전달 액세스 세션을 참조하세요.

FreeRTOS용 서비스 역할

서비스 역할 지원: 예

서비스 역할은 서비스가 사용자를 대신하여 작업을 수행하는 것으로 가정하는 IAM 역할입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 정보는 IAM 사용 설명서의 Create a role to delegate permissions to an AWS 서비스를 참조하세요.

> **Warning**
>
> 서비스 역할에 대한 권한을 변경하면 FreeRTOS 기능이 중단될 수 있습니다. FreeRTOS에서 관련 지침을 제공하는 경우에만 서비스 역할을 편집하세요.

FreeRTOS용 서비스 연결 역할

서비스 링크 역할 지원: 아니요

서비스 연결 역할은에 연결된 서비스 역할의 한 유형입니다 AWS 서비스. 서비스는 사용자를 대신하여 작업을 수행하기 위해 역할을 수임할 수 있습니다. 서비스 연결 역할은에 표시 AWS 계정 되며 서비스가 소유합니다. IAM 관리자는 서비스 링크 역할의 권한을 볼 수 있지만 편집은 할 수 없습니다.

서비스 연결 역할 생성 또는 관리에 대한 자세한 내용은 IAM으로 작업하는AWS 서비스를 참조하세요. 서비스 연결 역할 열에서 `Yes`이(가) 포함된 서비스를 테이블에서 찾습니다. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 예 링크를 선택합니다.

FreeRTOS 자격 증명 기반 정책 예제

기본적으로 사용자 및 역할에는 FreeRTOS 리소스를 생성하거나 수정할 수 있는 권한이 없습니다. 또한 AWS Management Console, AWS Command Line Interface (AWS CLI) 또는 AWS API를 사용하여 작업을 수행할 수 없습니다. 사용자에게 사용자가 필요한 리소스에서 작업을 수행할 권한을 부여하려면 IAM 관리자가 IAM 정책을 생성하면 됩니다. 그런 다음 관리자가 IAM 정책을 역할에 추가하고, 사용자가 역할을 맡을 수 있습니다.

이러한 예제 JSON 정책 문서를 사용하여 IAM 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 IAM 정책 생성(콘솔)을 참조하세요.

각 리소스 유형에 대한 ARN 형식을 포함하여 FreeRTOS에서 정의한 작업 및 리소스 유형에 대한 자세한 내용은 서비스 권한 부여 참조에서 FreeRTOS에서 사용되는 작업, 리소스 및 조건 키를 참조하세요.

주제

- 정책 모범 사례
- FreeRTOS 콘솔 사용
- 사용자가 자신의 고유한 권한을 볼 수 있도록 허용

정책 모범 사례

ID 기반 정책에 따라 계정에서 사용자가 FreeRTOS 리소스를 생성, 액세스 또는 삭제할 수 있는지 여부가 결정됩니다. 이 작업으로 인해 AWS 계정에 비용이 발생할 수 있습니다. ID 기반 정책을 생성하거나 편집할 때는 다음 지침과 권장 사항을 따릅니다.

- AWS 관리형 정책을 시작하고 최소 권한으로 전환 - 사용자 및 워크로드에 권한 부여를 시작하려면 많은 일반적인 사용 사례에 대한 권한을 부여하는 AWS 관리형 정책을 사용합니다. 에서 사용할 수 있습니다 AWS 계정. 사용 사례에 맞는 AWS 고객 관리형 정책을 정의하여 권한을 추가로 줄이는 것이 좋습니다. 자세한 정보는 IAM 사용 설명서의 AWS 관리형 정책 또는 AWS 직무에 대한 관리형 정책을 참조하세요.
- 최소 권한 적용 – IAM 정책을 사용하여 권한을 설정하는 경우, 작업을 수행하는 데 필요한 권한만 부여합니다. 이렇게 하려면 최소 권한으로 알려진 특정 조건에서 특정 리소스에 대해 수행할 수 있는 작업을 정의합니다. IAM을 사용하여 권한을 적용하는 방법에 대한 자세한 정보는 IAM 사용 설명서에 있는 IAM의 정책 및 권한을 참조하세요.
- IAM 정책의 조건을 사용하여 액세스 추가 제한 – 정책에 조건을 추가하여 작업 및 리소스에 대한 액세스를 제한할 수 있습니다. 예를 들어, SSL을 사용하여 모든 요청을 전송해야 한다고 지정하는 정책 조건을 작성할 수 있습니다. AWS 서비스와 같은 특정를 통해 사용되는 경우 조건을 사용하여 서비스 작업에 대한 액세스 권한을 부여할 수도 있습니다 AWS CloudFormation. 자세한 정보는 IAM 사용 설명서의 IAM JSON 정책 요소: 조건을 참조하세요.
- IAM Access Analyzer를 통해 IAM 정책을 확인하여 안전하고 기능적인 권한 보장 - IAM Access Analyzer에서는 IAM 정책 언어(JSON)와 모범 사례가 정책에서 준수되도록 새로운 및 기존 정책을 확인합니다. IAM Access Analyzer는 100개 이상의 정책 확인 항목과 실행 가능한 추천을 제공하여 안전하고 기능적인 정책을 작성하도록 돕습니다. 자세한 내용은 IAM 사용 설명서의 IAM Access Analyzer에서 정책 검증을 참조하세요.
- 다중 인증(MFA) 필요 -에서 IAM 사용자 또는 루트 사용자가 필요한 시나리오가 있는 경우 추가 보안을 위해 MFA를 AWS 계정켭니다. API 작업을 직접 호출할 때 MFA가 필요하면 정책에 MFA 조건을 추가합니다. 자세한 내용은 IAM 사용 설명서의 MFA를 통한 보안 API 액세스를 참조하세요.

IAM의 모범 사례에 대한 자세한 내용은 IAM 사용 설명서의 IAM의 보안 모범 사례를 참조하세요.

FreeRTOS 콘솔 사용

FreeRTOS 콘솔에 액세스하려면 최소한의 권한 세트가 있어야 합니다. 이러한 권한은에서 FreeRTOS 리소스에 대한 세부 정보를 나열하고 볼 수 있도록 허용해야 합니다 AWS 계정. 최소 필수 권한보다 더 제한적인 ID 기반 정책을 생성하는 경우, 콘솔이 해당 정책에 연결된 엔티티(사용자 또는 역할)에 대해 의도대로 작동하지 않습니다.

AWS CLI 또는 AWS API만 호출하는 사용자에게는 최소 콘솔 권한을 허용할 필요가 없습니다. 대신 수행하려는 API 작업과 일치하는 작업에만 액세스할 수 있도록 합니다.

사용자와 역할이 FreeRTOS 콘솔을 계속 사용할 수 있도록 하려면 FreeRTOS *ConsoleAccess* 또는 *ReadOnly* AWS 관리형 정책도 엔터티에 연결합니다. 자세한 내용은 IAM 사용 설명서의 사용자에게 권한 추가를 참조하세요.

사용자가 자신의 고유한 권한을 볼 수 있도록 허용

이 예제는 IAM 사용자가 자신의 사용자 ID에 연결된 인라인 및 관리형 정책을 볼 수 있도록 허용하는 정책을 생성하는 방법을 보여줍니다. 이 정책에는 콘솔에서 또는 AWS CLI 또는 AWS API를 사용하여 프로그래밍 방식으로이 작업을 완료할 수 있는 권한이 포함됩니다.

```
{
    "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": "*"
        }
```

```
    ]
}
```

FreeRTOS 자격 증명 및 액세스 문제 해결

다음 정보를 사용하여 FreeRTOS 및 IAM에서 발생할 수 있는 공통적인 문제를 진단하고 수정할 수 있습니다.

주제

- FreeRTOS에서 작업을 수행할 권한이 없음
- iam:PassRole을 수행하도록 인증되지 않음
- 내 외부의 사람이 내 FreeRTOS 리소스 AWS 계정 에 액세스하도록 허용하고 싶습니다.

FreeRTOS에서 작업을 수행할 권한이 없음

작업을 수행할 권한이 없다는 오류가 수신되면, 작업을 수행할 수 있도록 정책을 업데이트해야 합니다.

다음의 예제 오류는 `mateojackson` IAM 사용자가 콘솔을 사용하여 가상 *`my-example-widget`* 리소스에 대한 세부 정보를 보려고 하지만 가상 `awes:`*`GetWidget`* 권한이 없을 때 발생합니다.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 awes:GetWidget on resource: my-example-widget
```

이 경우, `awes:`*`GetWidget`* 작업을 사용하여 *`my-example-widget`* 리소스에 액세스할 수 있도록 `mateojackson` 사용자 정책을 업데이트해야 합니다.

도움이 필요한 경우 AWS 관리자에게 문의하세요. 관리자는 로그인 자격 증명을 제공한 사람입니다.

iam:PassRole을 수행하도록 인증되지 않음

`iam:PassRole` 작업을 수행할 수 있는 권한이 없다는 오류가 수신되면 FreeRTOS에 역할을 전달할 수 있도록 정책을 업데이트해야 합니다.

일부 AWS 서비스 에서는 새 서비스 역할 또는 서비스 연결 역할을 생성하는 대신 기존 역할을 해당 서비스에 전달할 수 있습니다. 이렇게 하려면 사용자가 서비스에 역할을 전달할 수 있는 권한을 가지고 있어야 합니다.

다음 예제 오류는 marymajor라는 IAM 사용자가 콘솔을 사용하여 FreeRTOS에서 작업을 수행하려고 하는 경우에 발생합니다. 하지만 작업을 수행하려면 서비스 역할이 부여한 권한이 서비스에 있어야 합니다. Mary는 서비스에 역할을 전달할 수 있는 권한을 가지고 있지 않습니다.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

이 경우, Mary가 iam:PassRole 작업을 수행할 수 있도록 Mary의 정책을 업데이트해야 합니다.

도움이 필요한 경우 AWS 관리자에게 문의하세요. 관리자는 로그인 자격 증명을 제공한 사람입니다.

내 외부의 사람이 내 FreeRTOS 리소스 AWS 계정 에 액세스하도록 허용하고 싶습니다.

다른 계정의 사용자 또는 조직 외부의 사람이 리소스에 액세스할 때 사용할 수 있는 역할을 생성할 수 있습니다. 역할을 수임할 신뢰할 수 있는 사람을 지정할 수 있습니다. 리소스 기반 정책 또는 액세스 제어 목록(ACL)을 지원하는 서비스의 경우, 이러한 정책을 사용하여 다른 사람에게 리소스에 대한 액세스 권한을 부여할 수 있습니다.

자세히 알아보려면 다음을 참조하세요.

- FreeRTOS에서 이러한 기능을 지원하는지 여부를 알아보려면 FreeRTOS에서 IAM을 사용하는 방법 단원을 참조하세요.
- 소유 AWS 계정 한의 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 소유한 다른의 IAM 사용자에게 액세스 권한 제공을 참조 AWS 계정 하세요.
- 타사에 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 타사가 AWS 계정 소유한에 대한 액세스 권한 제공을 AWS 계정참조하세요.
- ID 페더레이션을 통해 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 외부에서 인증된 사용자에게 액세스 권한 제공(ID 페더레이션)을 참조하세요.
- 크로스 계정 액세스에 대한 역할과 리소스 기반 정책 사용의 차이점을 알아보려면 IAM 사용 설명서의 IAM의 크로스 계정 리소스 액세스를 참조하세요.

규정 준수 확인

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

AWS 서비스 가 특정 규정 준수 프로그램의 범위 내에 있는지 알아보려면 AWS 서비스 규정 준수 프로그램 제공 범위 섹션을 참조하고 관심 있는 규정 준수 프로그램을 선택합니다. 일반 정보는 AWS 규정 준수 프로그램.

를 사용하여 타사 감사 보고서를 다운로드할 수 있습니다 AWS Artifact. 자세한 내용은 Downloading Reports inDownloading AWS Artifact을 참조하세요.

사용 시 규정 준수 책임은 데이터의 민감도, 회사의 규정 준수 목표 및 관련 법률과 규정에 따라 AWS 서비스 결정됩니다.는 규정 준수를 지원하기 위해 다음 리소스를 AWS 제공합니다.

- 보안 규정 준수 및 거버넌스 - 이러한 솔루션 구현 가이드에서는 아키텍처 고려 사항을 설명하고 보안 및 규정 준수 기능을 배포하는 단계를 제공합니다.
- HIPAA 적격 서비스 참조 - HIPAA 적격 서비스가 나열되어 있습니다. 모두 HIPAA 자격이 AWS 서비스 있는 것은 아닙니다.
- AWS 규정 준수 리소스 -이 워크북 및 가이드 모음은 산업 및 위치에 적용될 수 있습니다.
- AWS 고객 규정 준수 가이드 - 규정 준수의 관점에서 공동 책임 모델을 이해합니다. 이 가이드에는 여러 프레임워크(미국 국립표준기술연구소(NIST), 결제 카드 산업 보안 표준 위원회(PCI), 국제표준화기구(ISO) 포함)의 보안 제어에 대한 지침을 보호하고 AWS 서비스 매핑하는 모범 사례가 요약되어 있습니다.
- AWS Config 개발자 안내서의 규칙을 사용하여 리소스 평가 -이 AWS Config 서비스는 리소스 구성이 내부 관행, 업계 지침 및 규정을 얼마나 잘 준수하는지 평가합니다.
- AWS Security Hub - 내 보안 상태에 대한 포괄적인 보기를 AWS 서비스 제공합니다 AWS. Security Hub는 보안 컨트롤을 사용하여 AWS 리소스를 평가하고 보안 업계 표준 및 모범 사례에 대한 규정 준수를 확인합니다. 지원되는 서비스 및 제어 목록은 Security Hub 제어 참조를 참조하세요.
- Amazon GuardDuty - 의심스러운 활동과 악의적인 활동이 있는지 환경을 모니터링하여 사용자, AWS 계정워크로드, 컨테이너 및 데이터에 대한 잠재적 위협을 AWS 서비스 탐지합니다. GuardDuty는 특정 규정 준수 프레임워크에서 요구하는 침입 탐지 요구 사항을 충족하여 PCI DSS와 같은 다양한 규정 준수 요구 사항을 따르는 데 도움을 줄 수 있습니다.
- AWS Audit Manager - 이를 AWS 서비스 통해 AWS 사용량을 지속적으로 감사하여 위험과 규정 및 업계 표준 준수를 관리하는 방법을 간소화할 수 있습니다.

의 복원력 AWS

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

AWS 글로벌 인프라는 AWS 리전 및 가용 영역을 중심으로 구축됩니다. AWS 리전은 물리적으로 분리되고 격리된 다수의 가용 리전을 제공하며 이러한 가용 리전은 짧은 지연 시간, 높은 처리량 및 높은 중복성을 갖춘 네트워크에 연결되어 있습니다. 가용 영역을 사용하면 중단 없이 가용 영역 간에 자동으로 장애 조치가 이루어지는 애플리케이션 및 데이터베이스를 설계하고 운영할 수 있습니다. 가용 영역은 기존의 단일 또는 복수 데이터 센터 인프라보다 가용성, 내결함성, 확장성이 뛰어납니다.

AWS 리전 및 가용 영역에 대한 자세한 내용은 AWS 글로벌 인프라를 참조하세요.

FreeRTOS의 인프라 보안

> **Important**
>
> 이 문서는 FreeRTOS 릴리스 202012.00에서 사용하기 위한 FreeRTOS 사용 설명서의 보관 버전입니다. 이 문서의 최신 버전은 FreeRTOS 사용 설명서를 참조하세요.

AWS 관리형 서비스는 Amazon Web Services: 보안 프로세스 개요 백서에 설명된 AWS 글로벌 네트워크 보안 절차로 보호됩니다. https://d0.awsstatic.com/whitepapers/Security/AWS_Security_Whitepaper.pdf

AWS 에서 게시한 API 호출을 사용하여 네트워크를 통해 AWS 서비스에 액세스합니다. 클라이언트가 전송 계층 보안(TLS) 1.2 이상을 지원해야 합니다. TLS 1.3 이상을 권장합니다. 클라이언트는 Ephemeral Diffie-Hellman(DHE) 또는 Elliptic Curve Ephemeral Diffie-Hellman(ECDHE)과 같은 PFS(전달 완전 보안, Perfect Forward Secrecy)가 포함된 암호 제품군도 지원해야 합니다. Java 7 이상의 최신 시스템은 대부분 이러한 모드를 지원합니다.

또한 요청은 액세스 키 ID 및 IAM 위탁자와 관련된 보안 암호 액세스 키를 사용하여 서명해야 합니다. 또는 AWS Security Token Service(AWS STS)를 사용하여 임시 자격 증명을 생성하여 요청에 서명할 수 있습니다.

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.